INTEGRATING AUTOMATED REASONING WITH MACHINE LEARNING FOR STRUCTURED PREDICTION AND SCIENTIFIC DISCOVERY

by

Nan Jiang

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

May 2025

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Yexiang Xue, Chair

Department of Computer Science, Purdue University

Dr. Willem-Jan Van Hoeve

Tepper School of Business, Carnegie Mellon University

Dr. Jean Honorio

School of Computing and Information Systems, University of Melbourne

Dr. Brian Bullins

Department of Computer Science, Purdue University

Approved by:

Dr. Voicu S. Popescu

Dedicated to my girlfriend Xiaoyi Zhu and my parents Xinlin Jiang and Gendi Qian.

ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to all those who have supported me in my Ph.D. academic journey and contributed to the completion of this dissertation.

First and foremost, I am immensely grateful to my advisor, Professor Yexiang Xue, for his invaluable guidance, unwavering support, and patient mentorship. Professor Xue fostered a welcoming and inclusive lab environment for graduate students to study, communicate, and collaborate. Under his mentorship, I embarked on the path of professional scientific research, and he skillfully guided me through the obstacles and challenges in both my personal and research life. His exceptional expertise, timely feedback, and continuous encouragement have been pivotal in the completion of this work. Professor Xue is not only an academic advisor to me but also a role model for my life. His friendly treatment of others, dedication to his career, and devotion to his family have profoundly influenced my value system and will continue to have a positive impact on my future endeavors.

I would like to extend my sincere appreciation to other committee members Professor Willem-Jan Van Hoeve, Professor Jean Honorio, and Professor Brian Bullins for their time, expertise, and constructive feedback. Their critical evaluations have significantly contributed to the refinement and improvement of this dissertation.

I am thankful to my paper collaborators Jinzhao Li, Yi Gu, Md Nasim, Fan Ding, Chen Luo and Maosen Zhang for their efficient collaboration, insightful discussions, and fruitful assistance. Their contributions have enriched the outcomes of this work.

A special thanks go to my lab mates and friends Wenjie Bai, Qinlin Meng, and Wei Deng for their help, understanding, and encouragement. It has been a privilege to work alongside these talented and inspiring individuals, and I feel fortunate to have had the opportunity to get to know their fascinating personalities.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Structural data are ubiquitous in our daily lives. However, decision-making with models learned from such data still remains a significant challenge when Machine Learning (ML) and Automated Reasoning (AR) are applied in isolation. Learning without reasoning often fails to generate output satisfying combinatorial constraints, while reasoning without learning often yields rigid models, that lack flexibility to evolving environments. Integrating ML and AR is essential but remains largely unsolved.

My research focuses on **embedding automated reasoning into machine learning**, to tackle challenging problems in structured prediction and AI-driven scientific discovery. My models are able to generate valid outputs satisfying complex combinatorial constraints, which greatly surpasses pure learning-based approaches. Moreover, these models are adaptive to evolving training data distributions, addressing the limitations of pure reasoning algorithms. My learning algorithms offer tight theoretical guarantees and demonstrate great empirical improvements in accuracy. Specifically, my contributions are:

(a) Combining AR and ML to ensure constraint satisfaction in machine learning: By embedding AR solvers as differentiable layers into neural network-based ML models, my work ensures constraint satisfaction of the predicted output when solving a variety of structural learning problems across operations research, combinatorial optimization, and natural language processing. Notably, in a data-driven vehicle dispatching task, our approach generates routes that 100% satisfy constraints while previous approaches produce less than 1% valid routes.

(b) Combining AR and ML to accelerate AI-driven scientific discovery: Integrating scientific approach-inspired reasoning, my work accelerates the discovery of physical knowledge from experimental data. My approach significantly extended the capabilities of existing methods in solving datasets with multiple independent variables. My approach successfully discovers ground-truth scientific expressions involving up to 50 variables, whereas previous approaches struggle with equations of just three variables.

My vision is to build an AI ecosystem with safety and robustness guarantees, encoding physical knowledge and operational constraints into machine learning models through au-

tomated reasoning. Meanwhile, I aim to enable the discovery of knowledge and constraints automatically from data, creating systems that are adaptive, reliable, and capable of addressing complex real-world challenges.

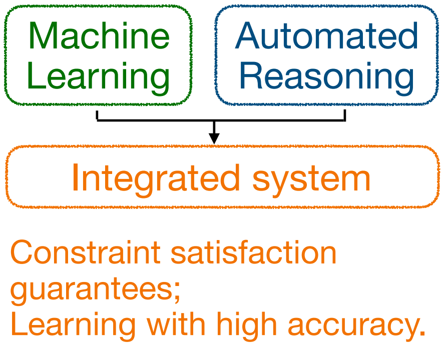
1. INTRODUCTION

1.1 Borad Overview

Structural data are ubiquitous in our daily lives. However, decision-making with models learned from such data still remains a significant challenge when Machine Learning (ML) and Automated Reasoning (AR) are applied in isolation. Learning without reasoning often fails to generate output satisfying combinatorial constraints, while reasoning without learning often yields rigid models, that lack flexibility to evolving environments. Integrating ML and AR is essential but remains largely unsolved.

My research focuses on **embedding automated reasoning into machine learning**, to tackle challenging problems in structured prediction and AI-driven scientific discovery. My models are able to generate valid outputs satisfying complex combinatorial constraints, which greatly surpasses pure learning-based approaches. Moreover, these models are adaptive to evolving training data distributions, addressing the limitations of pure reasoning algorithms. My learning algorithms offer tight theoretical guarantees and demonstrate great empirical improvements in accuracy. Specifically, my contributions are:

Combining AR and ML to Ensure Constraint Satisfaction in Machine Learning


Machine Learning
Automated Reasoning
Integrated system
Constraint satisfaction guarantees;
Learning with high accuracy.

For complex and structured problems that require adherence to physical or operational constraints, state-of-the-art ML models struggle to generate feasible outputs. This challenge cannot be solved by simply improving training algorithms, refining neural network architectures, or expanding training datasets. Conversely, AR tools such as Satisfiability (SAT) solvers and Satisfiability Modulo Theory (SMT) solvers lack flexibility in adapting to evolving data distributions, making them inadequate for solving structured problems on their own.

My research attacks this gap by developing a line of strategies to embed constraint reasoning algorithms into machine learning for tasks with combinatorial constraints, which makes great improvements over pure learning approaches. To summarize, I proposed to **(a)** Augment structured ML models with a constraint reasoning module that represents physical and operational requirements, which is a general formulation to handle a wide list of constraints. **(b)** Augment ML model with a constrained sampler subject to combinatorial logical constraints. This sampler can draw valid samples over hundreds of variables efficiently, greatly surpassing current samplers. **(c)** Augment ML model with a Tree Search-enhanced sampler, for language generation under grammar constraints. The proposed sampler can handle combinatorial hard (logical-valued) and soft (real-valued) constraints, whereas existing baselines only work for simple hard constraints. Each part is explained in detail in the following paragraphs.

Combining AR and ML to accelerate AI-driven Scientific Discovery Learning physical knowledge in closed-form equations from experimental data is vital in AI-driven scientific discovery. State-of-the-art approaches are limited to learning relatively simple expressions involving a few independent variables, struggling with multivariate expressions due to the exponentially large search space of symbolic expressions.

My research attacks this gap by integrating scientific approach-inspired reasoning and proposes an integrated system with (1) customized data acquisition, and (2) customized symbolic regressor, that searches in the constrained space of equations. The developed framework accelerates the discovery greatly over historical work with theoretical justification, enabling them to handle datasets with up to 50 variables, a major leap from the previous limit of fewer than 3 variables. The proposed framework is generalized to a wide list of symbolic regressors, including genetic programming, Monte Carlo Tree Search, and Deep Reinforcement Learning. The proposed framework is generalized to handle time-series data where we use a customized data acquisition triple to acquire the data to accelerate the discovery of dynamical systems.

The idea of combining machine learning models with sophisticated constraint reasoning tools is highly motivated by the real-world problems across multiple domains. To give you a clearer understanding and vision of the exact algorithms and tasks, the following paragraphs describe the exact reasoning algorithms that we use, the concrete applications we can solve, and the integrated learning framework we propose.

1.2 Brief Description of Each Chapter

Chapter 2: Embed Decision Diagram into Structured Prediction

In Chapter 2, we consider those real-world problems with structured outputs, which are also known as structured prediction problems. These problems need machine learning to capture data distribution and constraint reasoning to ensure the structure validity of the predicted output. Nevertheless, constrained structured prediction is still limited in real-world applications because of the lack of tools to bridge constraint satisfaction and machine learning.

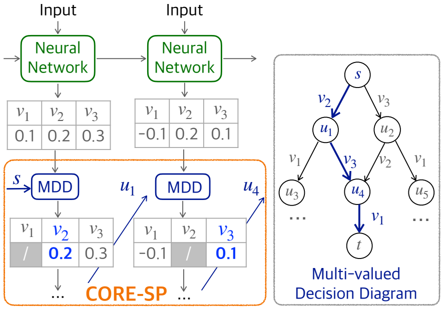
We propose, **CO**nstraint **RE**asoning embedded **S**tructured **P**rediction (Core-Sp), a scalable constraint reasoning and machine learning integrated approach for learning over structured domains [1]. Specifically, we propose to embed decision diagrams, a popular constraint reasoning tool, as a fully differentiable module into deep neural networks for structured prediction. To trade off the memory consumption of the decision diagram with the learning performance of the whole learning model, we also propose an iterative search algorithm to automate the searching process of the best Core-Sp structure within the memory budget.

In experiments, we evaluate Core-Sp on three applications: vehicle dispatching service planning, if-then program synthesis, and text2SQL generation. The proposed CORE-SP module demonstrates superior performance over state-of-the-art approaches in all three applications. The structures generated with Core-Sp satisfy 100% of the constraints when using exact decision diagrams. In addition, Core-Sp boosts learning performance by reducing the modeling space via constraint satisfaction.

Chapter 3: Learning Combinatorial Structures via Markov Random Fields with Sampling through Lovász Local Lemma

Chapter 3 considers those applications where the output has combinatorial structures. Learning to generate complex combinatorial structures satisfying constraints will have transformative impacts in many application domains. However, it is beyond the capabilities of existing approaches due to the highly intractable nature of the embedded probabilistic inference. Prior works spend most of the training time learning to separate valid from invalid structures but do not learn the inductive biases of valid structures.

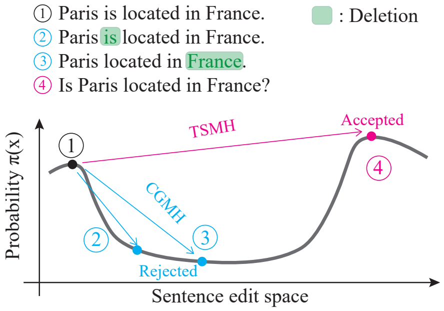
We develop NEural Lovász Sampler (Nelson), which embeds the sampler through Lovász Local Lemma (LLL) as a fully differentiable neural network layer. Our Nelson-CD embeds this sampler into the contrastive divergence learning process of Markov random fields. Nelson allows us to obtain valid samples from the current model distribution. Contrastive divergence is then applied to separate these samples from those in the training set. Nelson is implemented as a fully differentiable neural net, taking advantage of the parallelism of GPUs. Experimental results on several real-world domains reveal that Nelson learns to generate 100% valid structures, while baselines either time out or cannot ensure validity. Nelson also outperforms other approaches in running time, log-likelihood, and MAP scores.



Chapter 4: Controllable Language Generation via Combinatorial Constraint Satisfaction: A Tree Search Enhanced Monte-Carlo Approach

Chapter 4 explores a reasoning paradigm that generates human-like sentences subject to grammar rules, with access to a large-scale language model. Generating *human-like natural language* under combinatorial constraints is a principal milestone towards controllable text generation.

We present a language generation framework with combinatorial constraints, **T**ree **S**earch enhanced **M**etropolis **H**astings approach (TSMH), allowing the specification of grammar rules, keywords, and sentence attitude [2]. TSMH generates high-likelihood sentences with respect to a pre-trained language model while satisfying the given specifications that are required to satisfy. Our approach is highly flexible, requires no task-specific training, and leverages sophisticated constraint satisfaction techniques (i.e., Tree search) to enforce the


① Paris is located in France.
② Paris is located in France.
③ Paris located in France.
④ Is Paris located in France?
🟩 : Deletion
Probability π(x)
Sentence edit space
TSMH
Accepted
CGMH
Rejected
①
②
③
④

given constraints. To better handle the combinatorial constraints, a tree search algorithm is embedded into the proposal process of the Markov chain Monte Carlo (MCMC) to explore candidates that satisfy more constraints.

In experiments, our TSMH approach enjoys a much higher acceptance rate compared to existing MCMC approaches. Further, the TSMH method achieves consistent and significant improvement in language generation tasks, including generating interrogative, imperative sentences with keywords, and sentences with given sentiments.

Chapter 5: Probabilistic Area Loss Minimization for Protein Sequence Alignment

Chapter 5 considers a specific structured prediction problem (i.e., pairwise sequence alignment) where we can develop an efficient and simple reasoning algorithm (i.e., dynamic programming) to sample valid outputs (i.e., a consecutive path) from the model's distribution.

Pairwise sequence alignment is a fundamental problem in computational structural biology and is popular for detecting protein homology. A valid alignment is a consecutive path from the top-left corner to the bottom-right location in the 2D grids with the two protein sequences being the axis. Given training data comparing a set of pairs of protein sequences as well as its corresponding alignments, the learning task is to maximize the likelihood of the ground-truth alignment for every pair of sequences.

The main difficulty we are facing with this application is the dataset itself is noisy. Since most of the developed programs for detecting protein sequence alignments are based on the likelihood information of amino acids and are sensitive to alignment noises in the dataset.

We present, **P**robabilistic **A**rea **L**oss **M**inimization (PALM), a robust method for modeling pairwise protein sequence alignments, with the proposed area distance to reduce the biological measurement noise [3]. PALM learns the alignment of two protein sequences with probabilistic area distance objective, which can denoise the measurement errors and offsets from different biologists. During learning, PALM optimizes the learning objective by estimating the gradients via sampling valid alignments from the model's probability distribution.

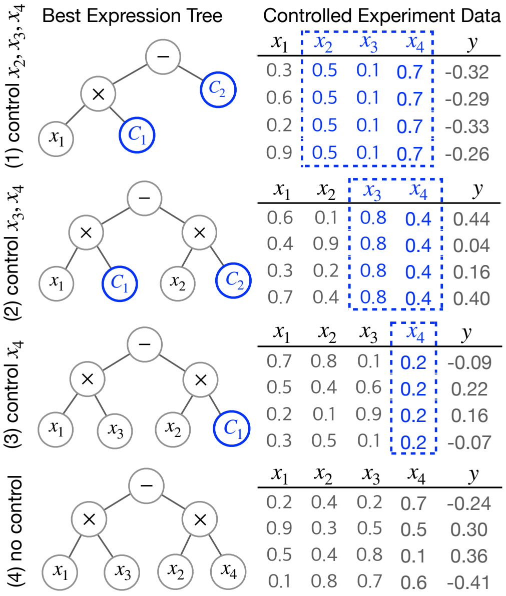
We show the alignment sampling procedure can be laid out as dynamic programming which is computationally efficient.

Empirically, we show that PALM can generate sequence alignments with higher Precision and Recall, as well as F1-score than the competing methods, especially for long protein sequences and remote homologies. This study implies PALM can learn robustly over large-scale protein sequence alignment problems.

Chapter 6: Symbolic Regression via Control Variable Genetic Programming

Learning symbolic expressions directly from experiment data is a vital step in AI-driven scientific discovery. Nevertheless, state-of-the-art approaches are limited to learning simple expressions. Regressing expressions involving many independent variables still remain out of reach.

Motivated by the control variable experiments widely utilized in science, we propose **C**ontrol **V**ariable **G**enetic **P**rogramming (CVGP) for symbolic regression over many independent variables. CVGP expedites symbolic expression discovery via customized experiment design, rather than learning from a fixed dataset collected a priori. CVGP starts by fitting simple expressions involving a small set of independent variables using genetic programming, under controlled experiments where other variables are held as constants. It then extends expressions learned in previous generations by adding new independent variables, using new control variable experiments in which these variables are allowed to vary.



x_1	x_2	x_3	x_4	y
0.3	0.5	0.1	0.7	-0.32
0.6	0.5	0.1	0.7	-0.29
0.2	0.5	0.1	0.7	-0.33
0.9	0.5	0.1	0.7	-0.26

x_1	x_2	x_3	x_4	y
0.6	0.1	0.8	0.4	0.44
0.4	0.9	0.8	0.4	0.04
0.3	0.2	0.8	0.4	0.16
0.7	0.4	0.8	0.4	0.40

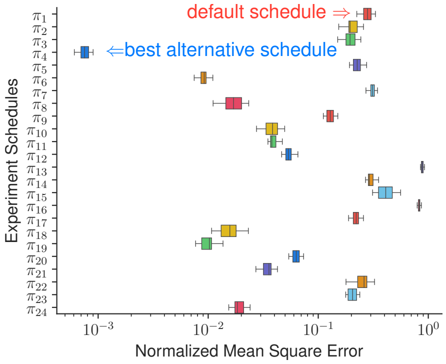
x_1	x_2	x_3	x_4	y
0.7	0.8	0.1	0.2	-0.09
0.5	0.4	0.6	0.2	0.22
0.2	0.1	0.9	0.2	0.16
0.3	0.5	0.1	0.2	-0.07

x_1	x_2	x_3	x_4	y
0.2	0.4	0.2	0.7	-0.24
0.9	0.3	0.5	0.5	0.30
0.5	0.4	0.8	0.1	0.36
0.1	0.8	0.7	0.6	-0.41

Theoretically, we show CVGP as an incremental building approach can yield an exponential reduction in the search space when learning a class of expressions. Experimentally, CVGP outperforms several baselines in learning symbolic expressions involving multiple independent variables.

Chapter 7: Racing Control Variable Genetic Programming for Symbolic Regression

Symbolic regression, as one of the most crucial tasks in AI for science, discovers governing equations from experimental data. Popular approaches based on genetic programming, Monte Carlo tree search, or deep reinforcement learning learn symbolic regression from a fixed dataset. These methods require massive datasets and long training time especially when learning complex equations involving many variables. Recently, Control Variable Genetic Programming (CVGP) has been introduced which accelerates the regression process by discovering equations from designed control variable experiments. However, the set of experiments is fixed a-priori in CVGP and we observe that sub-optimal selection of experiment schedules delay the discovery process significantly.



To overcome this limitation, we propose Racing Control Variable Genetic Programming (Racing-CVGP), which carries out multiple experiment schedules simultaneously. A selection scheme similar to that used in selecting good symbolic equations in genetic programming is implemented to ensure that promising experiment schedules eventually win over the average ones. The unfavorable schedules are terminated early to save time for the promising ones.

We evaluate Racing-CVGP on several synthetic and real-world datasets corresponding to true physics laws. We demonstrate that Racing-CVGP outperforms CVGP and a series of symbolic regressors which discover equations from fixed datasets.

Chapter 8: Vertical Symbolic Regression using Deep Policy Gradient

Vertical Symbolic Regression (VSR) has recently been proposed to expedite the discovery of symbolic equations with many independent variables from experimental data. VSR reduces the search spaces following the vertical discovery path by building from reduced-form equations involving a subset of variables to all variables. While deep neural networks have shown promise in enhancing symbolic regression, directly integrating VSR with deep networks faces challenges such as gradient propagation and engineering complexities due to the tree representation of expressions. We propose **V**ertical **S**ymbolic **R**egression using **D**eep **P**olicy **G**radient (VSR-DPG) and demonstrate that VSR-DPG can recover ground-truth equations involving multiple input variables, significantly beyond both deep reinforcement learning-based approaches and previous VSR variants. Our VSR-DPG models symbolic regression as a sequential decision-making process, in which equations are built from repeated applications of grammar rules. The integrated deep model is trained to maximize a policy gradient objective.

Experimental results demonstrate that our VSR-DPG significantly outperforms popular baselines in identifying both algebraic equations and ordinary differential equations on a series of benchmarks.

Chapter 9: Active Symbolic Discovery of Ordinary Differential Equations via Phase Portrait Sketching

The symbolic discovery of Ordinary Differential Equations (ODEs) from trajectory data plays a pivotal role in AI-driven scientific discovery. Existing symbolic methods predominantly rely on fixed, pre-collected training datasets, which often result in suboptimal performance, as demonstrated in our case study in Figure 9.1. Drawing inspiration from active learning, we investigate strategies to query informative trajectory data that can enhance the evaluation of predicted ODEs. However, the butterfly effect in dynamical systems reveals that small variations in initial conditions can lead to drastically different trajectories, necessitating the storage of vast quantities of trajectory data using conventional active learning.

To address this, we introduce **A**ctive Symbolic Discovery of Ordinary Differential Equations via **P**hase **P**ortrait **S**ketching (APPS). Instead of directly selecting individual initial conditions, our APPS first identifies an informative region within the phase space and then samples a batch of initial conditions from this region. Compared to traditional active learning methods, APPS mitigates the gap of maintaining a large amount of data. Extensive

Table 1.1. Summary of all the reasoning tools and applications in this thesis.

	Reasoning tools			
Applications	Decision Diagram	LLL-based Constrained Sampler	MCMC with Tree Search	Dynamic Programming
Controllable Text Generation			Chapter 4	
Generate Pairwise Sequence Alignment				Chapter 5
Vehicle Dispatching Service Planning	Chapter 2	Chapter 3		
if-then program synthesis	Chapter 2			
text2SQL generation	Chapter 2			
Random solutions for K satisfiability			Chapter 3	
Sink-free graph orientation		Chapter 3		

	Reasoning tools		
Evaluation Datasets	Control Variable Experiment	Racing Experiment Design	Phase Portrait Sketching
Multivariate expression dataset	Chapters 6, 7,8	Chapters 6, 7,8	
Feynman dataset	Chapters 6, 7,8	Chapters 6, 7,8	
SrBench dataset	Chapters 6, 7,8	Chapters 6, 7,8	
ODEBase dataset	Chapters 6, 7,8	Chapter 8	Chapter 9

experiments demonstrate that APPS consistently discovers more accurate ODE expressions than baseline methods using passively collected datasets.

1.3 Summary

This thesis introduces a novel and important computational framework based on diverse strategies of embedding sophisticated constraint reasoning tools with machine learning models, to solve complex real-world applications. My research was mainly motivated by those important problems across multiple scientific domains and wide collaboration with researchers in the areas of operational research, computational sustainability, and combinatorics. Part of the results in this preliminary report has been published in the following peer-reviewed publications:

- **Nan Jiang**, Md. Nasim, and Yexiang Xue. Active symbolic discovery of ordinary differential equations via phase portrait sketching. In Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI), 2025.
- **Nan Jiang***, Md Nasim, and Yexiang Xue. Vertical symbolic regression via deep policy gradient. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI), pages 5891-5899, 8 2024.
- Jinzhao Li, **Nan Jiang**, and Yexiang Xue. Solving satisfiability modulo counting for symbolic and statistical ai integration with provable guarantees. In Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI), pages 20481-20490, 2024.
- **Nan Jiang** and Yexiang Xue. Racing control variable genetic programming for symbolic regression. In Thirty- Eighth AAAI Conference on Artificial Intelligence (AAAI), pages 12901-12909, 2024.
- **Nan Jiang**, Jinzhao Li, and Yexiang Xue. A tighter convergence proof of reverse experience replay. Reinforcement Learning Journal, 1:470-480, 2024.
- Md. Nasim, **Nan Jiang**, Yexiang Xue. Computational Approaches to Scientific Discovery. Lecture Notes in Computer Science. Springer, 2024. Book Chapter.
- **Nan Jiang** and Yexiang Xue. Symbolic regression via control variable genetic programming. In Machine Learning and Knowledge Discovery in Databases: Research Track - European Conference (ECML/PKDD), volume 14172, pages 178-195, 2023.
- **Nan Jiang***, Yi Gu*, and Yexiang Xue. Learning Markov random fields for combinatorial structures via sampling through lovász local lemma. In Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI), pages 4016-4024, 2023.
- Maxwell J. Jacobson, Case Q. Wright, **Nan Jiang**, Gustavo Rodriguez-Rivera, and Yexiang Xue. Task detection in continual learning via familiarity autoencoders. In

Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), pages 1-8, 2022.

- **Nan Jiang**, Chen Luo, Vihan Lakshman, Yesh Dattatreya, and Yexiang Xue. Massive text normalization via an efficient randomized algorithm. In The ACM Web Conference (WWW), pages 2946-2956, 2022.

- **Nan Jiang**, Maosen Zhang, Willem-Jan van Hoeve, and Yexiang Xue. Constraint reasoning embedded structured prediction. Journal of Machine Learning Research, 23:345:1-345:40, 2022.

- **Nan Jiang***, Fan Ding*, Jianzhu Ma, Jian Peng, Jinbo Xu, and Yexiang Xue. PALM: probabilistic area loss minimization for protein sequence alignment. In Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI), volume 161, pages 1100-1109, 2021.

- Libin Shi, Wenge Rong, Shijie Zhou, **Nan Jiang**, and Zhang Xiong. A dual channel class hierarchy-based recurrent language modeling. Neurocomputing, 418:291-299, 2020.

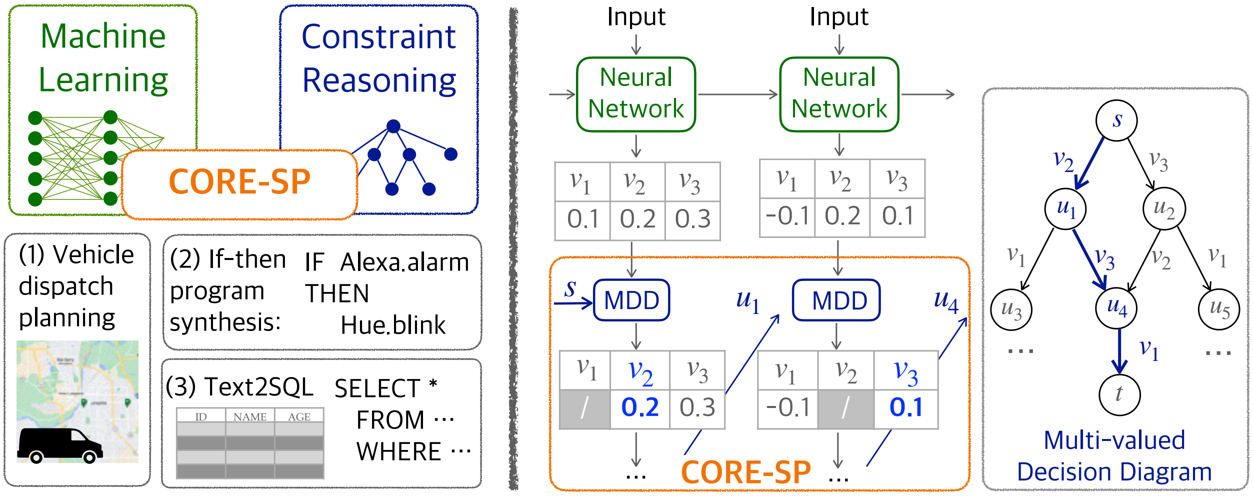
- Maosen Zhang, **Nan Jiang**, Lei Li, and Yexiang Xue. Constraint satisfaction driven natural language generation: A tree search embedded MCMC approach. In Findings of the Association for Computational Linguistics: (EMNLP), pages 1286-1298, 2020.

2. Constraint Reasoning Embedded Structured Prediction

2.1 Introduction

The emergence of large-scale constraint reasoning and machine learning technologies have impacted virtually all application domains, including marketing, linguistics, operations, retail, robotics, and health care. Constraint reasoning has traditionally been applied to building *prescriptive* models that generate solutions for strategic, tactical, or operational use [4]. It requires a precise problem description and is usually difficult to be made flexible to the evolving data distributions. Machine learning, on the other hand, has been applied primarily to build *predictive* models, such as classifications or regressions [5, 6]. While the structure of a machine learning model (like a neural net) must be designed, the actual model parameters are learned automatically via gradient descent algorithms. This gives machine learning models the flexibility to adapt to the evolving data distributions. Nevertheless, it is difficult to enforce constraints on the output of machine learning models. Many real-world applications are beyond the reach of constraint reasoning or machine learning alone.

In this chapter, we focus on structured prediction problems, which is a class of learning problems requiring both constraint reasoning and machine learning. It expands the output space of classification problems into high-dimensional structured space. Structured prediction has diverse application domains, ranging from natural language processing [7], social network analysis [8], and ecological modeling [9, 10]. The applications we consider in this chapter all require tight integration of constraint reasoning and machine learning. Our first application *vehicle dispatching service planning* is to recommend a route that satisfies the daily service needs as well as meeting the drivers' preferences. Historical data may reveal that the drivers do not follow common stylized objectives such as minimizing distance or time. Therefore standard constraint reasoning tools, e.g., solvers for the traveling salesman problem, cannot be applied. While we need machine learning to capture the drivers' objective functions, pure machine learning-based approaches are insufficient because they often generate routes that violate delivery requests. Our second and third applications are *program synthesis from natural language*, which clearly require machine learning to generate



(a) Core-SP framework

(b) Core-SP encodes MDD as a differentiable layer

Figure 2.1. **(a)** Our proposed Core-Sp framework embeds constraint reasoning in machine learning for structured prediction. We demonstrate the effectiveness of Core-Sp on vehicle dispatching service, if-then program synthesis, and Text2SQL generation tasks. **(b)** At a high level, Core-Sp (in orange colored box) is a fully differentiable layer that simulates a path descending in the corresponding decision diagram. Core-Sp filters out the infeasible output from the structured output to ensure constraint satisfaction.

structured programs. Nevertheless, a pure learning approach cannot enforce the syntactic and semantic rules of those programs.

We propose **Co**nstraint **Re**asoning embedded **S**tructured **P**rediction (Core-Sp), a scalable constraint reasoning and machine learning integrated approach for learning over the structured domains. The main idea is to augment structured predictive models with a constraint reasoning module that represents physical and operational requirements. Specifically, we propose to embed decision diagrams [11, 12], a popular constraint reasoning tool, as a fully-differentiable module into deep neural networks. A decision diagram is a compact graphical representation of the constraints. It encodes each solution (an assignment of values to variables satisfying the constraints) as a path from the root to the terminal in the diagram. Core-Sp regards the neural network predictions as the simulation of descending along a path in the decision diagram. To ensure constraint satisfaction, Core-Sp filters out variable assignments from the neural network predictions that violate constraints. With the integration of CORE-SP, we provide structured prediction models with constraint satisfaction assurances. Moreover, structured prediction models with the CORE-SP layer enjoy a smaller prediction space than traditional structured prediction approaches, allowing our approach to learn faster in training and generalize better in testing. See Figure 2.1(a) for our proposed CORE-SP model which integrates constraint reasoning and machine learning for the three application domains. The high-level idea of CORE-SP is illustrated in Figure 2.1(b).

Previous approaches have considered regularizing machine learning with constraint reasoning in various application domains. Within the broader context of learning constrained models, the work of [13–17] have studied automating the constraint acquisition process from historic data or (user-)generated queries. These approaches use partial or complete examples to identify the constraints that can be added to the model. The type of constraints that can be learned depends on the formulation. Several works [18–21] enable learning in a constrained domain via encoding mathematical programming, such as quadratic programming or mixed integer linear programming, as a neural network layer. [22] propose to formulate the output space as an automata. They use the constraints to prune all the invalid transitions in the automata to ensure the validity of the structured outputs. In addition, constraints imposed by a knowledge graph have been embedded into the neural network as differentiable layers [23, 24]. [25] and [26] enforce physical constraints or expert inputs as soft constraints. We will illustrate the difference between our approach and these methods in Section 2.3.2. A different approach is to embed a machine learning model into optimization, e.g., by extending a constraint system with appropriate global constraints. For example, [27] integrate neural networks and decision trees with constraint programming, while [28] and [29] introduce a "Neuron" global constraint that represents a pre-trained neural network. Another series of approaches based on grammar variational autoencoders [30–32] use neural networks to encode and decode from the parse-tree of a context-free grammar to generate discrete structures. Such approaches are used to generate chemical molecule expressions, which represent a structured domain. Machine learning approaches have also been used to solve constraint reasoning and optimization problems. This includes the works of [33] and [34], which use neural networks to extend partial solutions to complete ones. [35] handles the traveling salesman problem by framing it as reinforcement learning. [36] proposes to learn an SAT solver from single-bit supervision. Approaches based on neural Turing machines [37] employ neural networks with external memory for discrete structure generation. More recently, [38] tackles the combinatorial optimization problems in graphs, by employing neural networks to learn the heuristics in the backtrack-free search. There is also a recent trend to synthesize programs using machine learning [39, 40].

In experimental analysis, we demonstrate the effectiveness of Core-Sp on the following three applications: (1) *Vehicle Dispatching Service Planning*: a route planning problem that recommends routes to drivers to meet the service needs while satisfying the drivers' preferences. The implicit preferences of drivers are learned from the historical traveling data. The input of this problem is the daily service requests. The output is the permutations of the service locations, representing the sequential order that the locations should be visited by the drivers. This task requires machine learning models to capture drivers' preferences from the traveling data, and constraint reasoning to ensure the satisfaction of service requests. (2) *If-then Program Synthesis*: the task is to automatically synthesize conditional programs from the natural language. Automatic program synthesis tools are useful for streamlining the program of a few online services such as IFTTT and Zapier. The if-then program is in the form of: if `trigger function` happens in the `trigger service`, then take the `action function` from the `action service`. The machine learning task, therefore, is to predict the quadruple (`trigger service`, `trigger function`, `action service`, `action function`). This application again requires machine learning to understand the semantics of the natural language, as well as constraint reasoning to satisfy the syntactic rules of the programs. (3) *Text2SQL Generation*: our last application is to automatically generate SQL queries that extract information from a database to answer a question posed in natural language. The neural model is used to understand the user's queries in natural language while the constraint reasoning tool is applied to ensure the model generates grammatically-valid SQL queries.

Our proposed Core-Sp framework demonstrates superior performance against the state-of-the-art approaches in all three applications. First, the structures generated by Core-Sp are better in constraint satisfaction. In vehicle service dispatching, all Core-Sp generated routes are valid, while a conditional generative adversarial network (cGAN) without Core-Sp generates on average less than 1% of valid routes when handling medium-sized delivery requests. We also apply a post-processing step [41] to boost cGAN's performance, but it cannot handle the complexity brought by the large combinatorial space of the routing problem. Its performance quickly defaults to the case without post-processing as the number of delivery locations increases. For if-then program synthesis, the percentage of valid programs produced increased from 88% to 100% with the Core-Sp module incorporated into the state-of-the-art LatentAttention model [42]. For Text2SQL, the percentage of valid SQL queries increased from 83.7% to 100% with Core-Sp incorporated into the state-of-the-art SQLNova model [43] on a hard testing set. Core-Sp also improves the learning performance of structured prediction models. We show that the routes generated by Core-Sp better fulfill drivers' preferences than cGAN without Core-Sp. In if-then program synthesis, Core-Sp module leads to approximately 2.0% improvement in accuracy compared with the state-of-the-art LatentAttention model and converges to models with higher accuracy in fewer training epochs. In Text2SQL generation, the Core-Sp module improves around 4.2% in execution accuracy and 1.9% in logical accuracy against SQLNova on a challenging test set.

2.2 Preliminaries

2.2.1 Structured Prediction

Structured prediction expands the output space of classification problems into a high-dimensional combinatorial space. Specifically, given a set of input-output samples $\mathcal{D}^{tr} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ drawn *i.i.d.* from some unknown distribution over the space $\mathcal{X} \times \mathcal{Y}$, a structure prediction model learns a conditional distribution $p_\theta(y|x)$, for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$ from data $\mathcal{D}^{tr}$. Note here the output space $\mathcal{Y} = \{0, 1\}^l$ is a high dimensional space of combinatorial structures. The three applications we consider in this chapter are all structured prediction problems. In vehicle dispatching service planning, the structured outputs are the delivery routes on a map. In if-then program synthesis, the structured outputs are the programs that complete web-service tasks. In Text2SQL generation, the structured outputs are the SQL queries that follow the SQL grammar.

In literature, various approaches have been proposed for structured prediction problems. The classifier chain approach [44] decomposes the joint likelihood into a product of conditionals and reduces the structured prediction problem into a series of binary prediction problems. Nevertheless, the error tends to propagate along the classifier chain, which limits their effectiveness [45]. Energy-based modeling, such as conditional random fields [46, 47] and structured prediction energy networks [48], learn to assign a high likelihood to structures that exist in the training data set while keeping the likelihood low for unseen structures. Constraints can be incorporated into these models as prior terms in the energy function but approximate inference is required to compute the intractable partition function, which often hinders their scalability. Another line of approaches are structured support vector machines [49], which uses hinge loss and row generation approaches for structured prediction; however, they were superseded in performance by later neural-network-based approaches. Recently, generative models, such as conditional generative advarsarial networks [50, 51], flow models [52], and sequence-to-sequence models [53] have become increasingly popular for structured prediction. These models use highly flexible neural networks to increase model capability. The over-parameterized networks with gradient descent-based optimization can learn better representation for the structures than the classic shallow models. However, it is not straightforward to enforce constraints into the neural network-based models.

Constraints in Structured Prediction Often the structured output space $\mathcal{Y}$ is subject to additional constraints $\mathcal{C}$. The conditional probability that y takes values which violate the physical constraints $\mathcal{C}$ given the input x is zero. Such information is known prior to the training of the machine learning model. Put it in mathematical language:

$$p_\theta(y|x) \begin{cases} > 0 & \text{if } y \text{ satisfies constraints } \mathcal{C}, \\ = 0 & \text{if } y \text{ violates constraints } \mathcal{C}. \end{cases} \tag{2.1}$$

Take the first task discussed in this chapter as an example. A valid delivery route should cover all the requested locations and should only visit each location once. Thus, the machine learning model should assign zero probability for those invalid routes. Notice that the constraints are often intricate and the inference problem of finding a valid structure satisfying constraints cannot be decomposed into independent small problems. After learning, the *inference* problem is to predict the structure output y given the input x. Such inference problems can be solved by either Maximum A Posteriori (MAP) inference, e.g., computing $\max_y\ p(y|x)$ or marginal inference, e.g., computing $\mathbb{E}_y[p(y|x)]$. Learning structured prediction models involves solving the inference problems within the learning loop, hence has an even higher complexity.

Combinatorial constraints render both the inference and the learning problems highly intractable. Indeed, much effort has been made to improve the efficiency of both the inference and learning problems ([54, 55]). For example, [56] proposes the sparseMAP function which solves the inference problem by returning a few sparse structures that attain high likelihoods. This inference method sits between the MAP and marginal inference. sparseMAP can be solved via quadratic programming in their problem setup. However, combinatorial constraints considered in this chapter renders the inference problem non-convex, even for a fixed structured prediction model, letting along the more challenging learning problem. Overall, constrained structured prediction presents two main challenges. The first is the *sample complexity*, since massive data is needed to learn an accurate model in an exponentially large space. The second is the *computational complexity*, since it is combinatorially intractable to generate outputs with structured outputs subject to complicated constraints.

Sequence-to-sequence Structured Prediction Sequence-to-sequence models are recently proposed popular structured prediction models. Our proposed Core-Sp builds on top of these models. The sequence-to-sequence model uses the re-parameterization trick to model the conditional probability $p_\theta(y|x)$, where $x \in \mathcal{X}$ denotes the input variables and $y \in \mathcal{Y}$ is the structured output. Here θ denotes the parameters of the neural models. Instead of modeling the probability $p_\theta(y|x)$ directly, the model introduces an additional random variable z and models it as a deterministic transformation from random variable z and evidence x to the output y. In other words, the conditional probability $p_\theta(y|x)$ is an integral over random variable z in the following way:

$$\begin{aligned} p_\theta(y|x) &= \int p_\theta(y|x,z)p(z)\,dz, \\ p_\theta(y|x,z) &= \mathbf{1}\{y = f_\theta(x,z)\}, \end{aligned} \tag{2.2}$$

where we assume z is from a known prior probability distribution $p(z)$. As a result, we only need to model $p_\theta(y|x,z)$ for the overall model $p_\theta(y|x)$. We further assume that $p_\theta(y|x,z)$ is given in the form of a deterministic function. Let $f_\theta(x,z) \in \mathcal{Y}$ be a deterministic mapping from inputs (x,z) to an output in the structured space $\mathcal{Y}$. The indicator function $\mathbf{1}\{\cdot\}$ evaluates to 1 if and only if $y = f_\theta(x,z)$. This formulation is closely related to the generative adversarial network and gives us high flexibility to model multi-modal distributions. Take the vehicle dispatching service planning as an example. Input x is the daily vehicle request and y is the suggested dispatching route. There can be several routes which meet the delivery demands as well as satisfying the driver's underlying preference function. In this case, the conditional probability $p_\theta(y|x)$ may have multiple modes, one for each good route. This formulation allows us to represent the multi-modal distribution effectively. The variable z decides which route to pick. Function $f_\theta(x,z)$ returns one route that meets the demand of input x and is randomly selected by z. If $p_\theta(y|x)$ has k modes, the space of z will be split into k regions where variable z in every region will be mapped to one mode in $p_\theta(y|x)$.

A sequence-to-sequence neural network is used to model the function $f_\theta(x,z)$. Assume the input variables x, z, and the output y are all represented in sequential forms $x = (x_1, x_2, \dots, x_T)$, $z = (z_1, z_2, \dots, z_T)$ and $y = (y_1, y_2, \dots, y_T)$. The sequence-to-sequence model is made of an encoder and a decoder. The sequential encoder takes in x and output a representation vector for input x. The sequential decoder takes in the output of the encoder as well as z and outputs y in T steps, where T refers as the maximum length for variable y. In the k-th step ($1 \leq k \leq T$), the decoder network takes z_k, and the hidden vector from the previous step h_{k-1} as inputs, and outputs a score vector $o_k = (o_{k1}, o_{k2}, \dots, o_{kD_k})$ of length $D_k = |D(y_k)|$. Here, o_k corresponds to the un-normalized likelihoods of each value that variable y_k can take. The softmax function is then applied to get the normalized probability:

$$p_{kj} = p\left(y_k = v_j | x, h_{k-1}\right) = \frac{\exp(o_{kj})}{\sum_{j'=1}^{D_k} \exp(o_{kj'})}, \qquad \text{for } j = 1, 2, \dots, D_k.$$

p_{kj} is the probability that variable y_k takes the j-th value v_j. Assume the prior distribution $p(z_k)$ is uniform distribution in $(0,1)$, noted as $\mathcal{U}(0,1)$. Variable z_k is sampled from $(0,1)$ uniformly at random and is used to determine the value for y_k according to the probability distribution vector $p_k = (p_{k1}, p_{k2}, \dots, p_{kD_k})$. Let $P_{k1}, P_{k2}, \dots, P_{k(D_k+1)}$ be the cumulative probabilities:

$$P_{kj} = \begin{cases} 0 & \text{for } j = 1, \\ \sum_{j'=1}^{j-1} p_{kj'} & \text{for } j = 2, 3, \dots, D_k, \\ 1 & \text{for } j = D_k + 1. \end{cases}$$

y_k is set to the value v_j if and only if $z_k \in \left[P_{kj}, P_{k(j+1)}\right)$. Notice that z_k is sampled from the uniform distribution between $(0, 1)$, the probability that y_k takes the j-th value v_j is exactly p_{kj}. Aside from producing the value for y_k in the k-th step, the sequence-to-sequence neural net also produces the hidden-state vector h_k at the k-th step, which is used by the neural net again in the subsequent $(k+1)$-th step. The overall architecture of the sequence-to-sequence model can be seen in Figure 2.4.

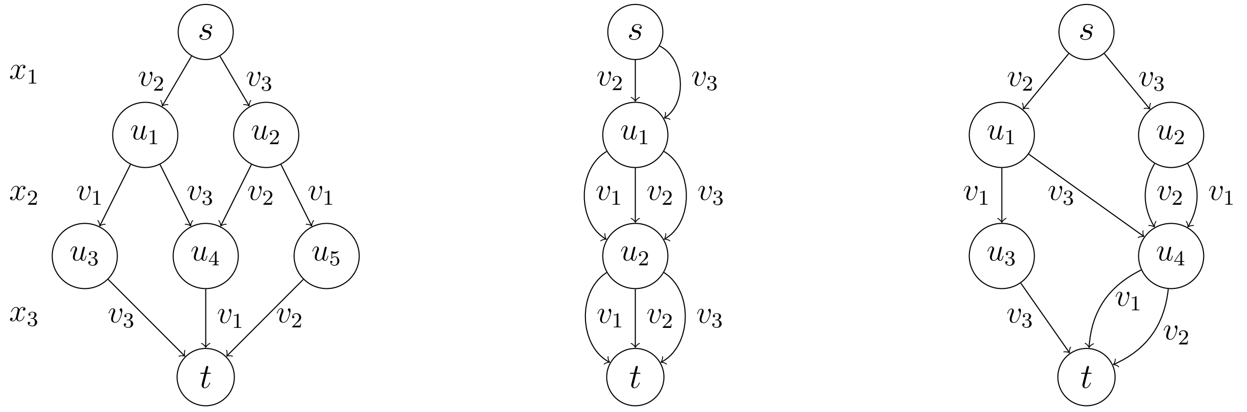
The training process of the sequence-to-sequence model is to minimize a pre-defined loss function, or an additional discriminator neural net, which penalizes the differences of the predicted structure $f_\theta(x, z)$ and the observed structure y. Here $f_\theta(x, z)$ is a predicted sequence obtained from the above process. Given a training dataset $D^{tr} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, the learning objective is to minimize the loss function:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{z^{(i)}} \ell \left(f_\theta \left(x^{(i)}, z^{(i)} \right), y^{(i)} \right). \tag{2.3}$$

Here $\ell(\cdot, \cdot)$ can be a predefined loss function that measures the mismatch between the predicted and observed structures. $\ell(\cdot, \cdot)$ can also be represented as a discriminator network, in which case leads to the training of a generative adversarial network. The parameters θ is updated via gradient descent: $\theta^{t+1} = \theta^t - \eta \nabla \mathcal{L}(\theta)$, where η denotes the learning rate.

2.2.2 Decision Diagrams

Decision diagrams were originally introduced to compactly represent the Boolean functions in a graphical form [11, 12]. Since then, they have been widely used in the context of verification and configuration problems [57]. More recently, they have been used successfully



(a) Exact MDD (b) width-1 relaxed MDD (c) width-2 relaxed MDD

Figure 2.2. Illustration of Multi-valued Decision Diagrams (MDDs) for decision variables x_1, x_2, x_3. **(a)** an exact MDD with all variable assignments satisfying two constraints: `all-diff`(x_1, x_2, x_3) and $x_1 \neq v_1$; **(b)** A width-1 relaxed MDD for the exact MDD in (a); **(c)** A width-2 relaxed MDD, which is formed by combining nodes u_4 and u_5 of the MDD in (a).

as an optimization tool, by representing the set of solutions to combinatorial optimization problems [58, 59].

Specifically, decision diagrams are defined with a sequence of decision variables $x_1, \ldots, x_n$. Variable x_i has a domain of possible values $D(x_i)$, for $i = 1, 2, \ldots, n$. We first introduce the notion of a decision diagram with the fixed variable ordering $x_1, \ldots, x_n$. A decision diagram is a directed acyclic graph, with $n + 1$ layers of nodes. Layer 1 contains a single node s, called the root. Layer $n + 1$ also contains a single node t, called the terminal. An arc from a node in layer i to a node in layer $i + 1$ represents a possible assignment of variable x_i to a value in its domain and is therefore associated with a label in $D(x_i)$. For an arc $e(v, u)$, we use $\texttt{val}(v, u) \in D(x_i)$ to represent the assigned label for variable x_i. For a node v in layer i, we use $\texttt{val}(v) \subseteq D(x_i)$ to represent the union of the values of each arc starting from node v, i.e., $\texttt{val}(v) = \texttt{val}(v, u) \cup \texttt{val}(v, u') \cup \cdots$. In other words, $\texttt{val}(v)$ represents the possible value assignments for the decision variable x_i, at node v. Each path from the root s to the terminal t represents a solution, i.e., a complete variable assignment. in this chapter, we consider variables with domains of categorical values, which is the so-called multi-valued decision diagrams (MDDs) [60]. See Figure 2.2 for an example.

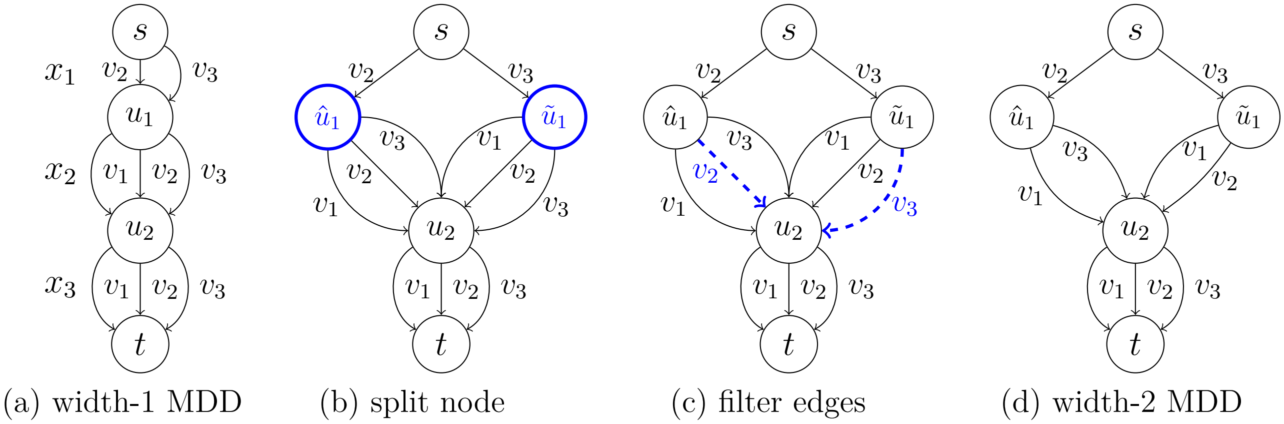


Figure 2.3. Node splitting and arc filtering for MDDs for variables x_1, x_2, x_3. **(a)** A width-1 relaxed MDD as in Figure 2.2(b). **(b)** Split node u_1 into $\hat{u}_1$ and $\tilde{u}_1$. **(c)** Filter arcs $e(\hat{u}_1, u_2) = v_2, e(\tilde{u}_1, u_2) = v_3$ that violate the constraint `all-diff`(x_1, x_2, x_3). The arcs in dashed lines are removed. **(d)** A width-2 relaxed MDD after one iteration of node splitting and arc filtering.

Exact Decision Diagrams. Given a set of constraints $\mathcal{C}$, the MDD $\mathcal{M}$ is said to be *exact* with respect to $\mathcal{C}$ if and only if every path that leads from the root node s to the terminal node t in $\mathcal{M}$ is a variable assignment satisfying all constraints in $\mathcal{C}$. Conversely, every valid variable assignment can be found as a path from s to t in $\mathcal{M}$.

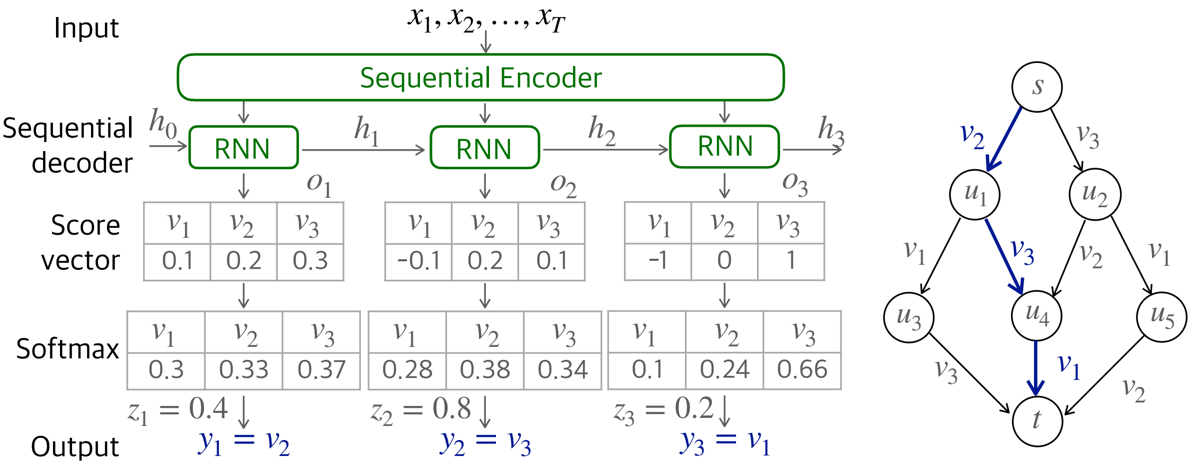
Relaxed Decision Diagrams. Since the exact decision diagrams can grow exponentially large, the *relaxed* decision diagrams are later proposed to limit the size the of MDDs [61]. The set of paths in a relaxed decision diagram forms a superset of the paths in the exact decision diagram. Relaxed MDDs are often defined with respect to the maximum layer width, which is the number of nodes in its largest layer.

Ordering in Decision Diagrams. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a permutation of the index set $\{1, 2, \ldots, n\}$. A MDD is said to have variable ordering π if it expands the value assignment of variable x_{π_i} in the i-th step, i.e., the arcs leading from layer i to layer $i+1$ are labeled with values assigned to variable x_{π_i}. Variable ordering is an important factor to optimize in MDD compilation, due to its impact on the memory consumption of the MDD [62].

Example 2.2.1. Figure 2.2 demonstrates several MDDs. Let x_1, x_2, x_3 be a sequence of decision variables with domain $D(x_1) = D(x_2) = D(x_3) = \{v_1, v_2, v_3\}$. The constraint `all-diff`$(x_1, x_2, x_3)$ restricts the values of x_1, x_2 and x_3 to be all different, i.e., they form a permutation. The other constraint is $x_1 \neq v_1$. (1) Exact MDD. The set of feasible permutations is $\{(v_2, v_1, v_3), (v_2, v_3, v_1), (v_3, v_2, v_1), (v_3, v_1, v_2)\}$. Figure 2.2(a) depicts the exact MDD that encodes all permutations satisfying the two constraints. (2) Relaxed MDD. Figure 2.2(b) is a width-1 relaxed MDD and Figure 2.2(c) is a width-2 relaxed MDD. The set of paths in the relaxed MDD forms a superset of all feasible permutations. To illustrate, Figure 2.2(c) contains two infeasible solutions $\{(v_3, v_1, v_1), (v_2, v_2, v_2)\}$. (3) Variable ordering. All the MDDs in Figure 2.2 have the same variable ordering of $\pi = (1, 2, 3)$, meaning that the MDD first expands on variable x_1, then x_2, finally x_3.

Decision Diagram Compilation. Decision diagrams can be compiled via a repeated process of *node splitting* and *arc filtering* from a width-1 relaxed MDD [61, 63]. Arc filtering removes arcs that lead to infeasible solutions, while node splitting increases the size of the decision diagram by splitting one node into two or more nodes. In practice, one can reach an exact MDD by repeatedly going through the splitting and filtering processes from a width-1 MDD. We refer to [60] for the detailed process of MDD compilation.

Example 2.2.2. Figure 2.3 demonstrates one possible process of applying the node splitting and arc filtering steps. We re-use the example in Figure 2.2(b) as the initial MDD in Figure 2.3(a), which depicts a width-1 relaxed MDD before compilation. The constraint to be applied is `all-diff`(x_1, x_2, x_3), that the assignments of variables x_1, x_2, x_3 should be different. The node u_1 in Figure 2.3(a) is split into two nodes $\hat{u}_1, \tilde{u}_1$ in Figure 2.3(b). The incoming arc $e(s, u_1) = v_2$ is assigned to node $\hat{u}_1$ and the other incoming arc $e(s, u_1) = v_3$ is assigned to node $\tilde{u}_1$. The outgoing arcs of node u_1 are copied for the two nodes. In Figure 2.3(c), the arc filtering process checks if certain variable assignments violate constraints for the two nodes. Arc $e(\hat{u}_1, u_2) = v_2$ is not compatible with the previous arc $e(s, \hat{u}_1) = v_2$ because they violate `all-diff`(x_1, x_2, x_3). Thus it is removed. For the same reason, arc $e(\tilde{u}_1, u_2) = v_3$ is also removed. (d) We get a width-2 relaxed MDD after splitting node u_1 and filtering the arcs.



(a) sequence-to-sequence model

(b) a path in MDD

Figure 2.4. Illustration of (a) a sequence-to-sequence model which generates an output corresponding to (b) a path in the multi-valued decision diagram. **(a)** a sequence-to-sequence model takes in input x and random variables z, and outputs $y_1 = v_2$, $y_2 = v_3$ and $y_3 = v_1$ in three steps. **(b)** the assignment $(y_1, y_2, y_3) = (v_2, v_3, v_1)$ corresponds to path $s \xrightarrow{v_2} u_1 \xrightarrow{v_3} u_4 \xrightarrow{v_1} t$ in the multi-valued decision diagram.

2.3 Constraint Reasoning Embedded in Structured Prediction

Core-Sp is motivated to address the lack of constraint satisfaction in sequence-to-sequence structured prediction models. One key idea that leads to the development of Core-Sp is the correspondence between the predicted outcomes of a sequence-to-sequence model and a path in a multi-valued decision diagram (MDD). Figure 2.4 provides an example. In this example, the sequence-to-sequence model outputs a sequence of variables' assignment $y_1 = v_2$, $y_2 = v_3$, $y_3 = v_1$ in Figure 2.4(a), which exactly corresponds to the highlighted blue path in the MDD in Figure 2.4(b). Nevertheless, the sequence-to-sequence model is also likely to output a variable assignment with no correspondence to the MDD. For example, suppose the neural model in Figure 2.4(a) outputs $y_1 = v_2$, $y_2 = v_3$, $y_3 = v_2$, there are no corresponding path in the MDD in Figure 2.4(b). This is the case where the output of the sequence-to-sequence model violates the `all-diff` constraint. Indeed, neural network-based models for structured prediction problems are not guaranteed to satisfy constraints as in Equation (2.1), which forms a key limitation of the state-of-the-art structured prediction models.

Core-Sp ensures constraint satisfaction of the neural network prediction by limiting the values that each variable can take following the flow of the MDD. Suppose we set $y_1 = v_2$ and $y_2 = v_3$ in Figure 2.4(b) and arrive at node u_4, the only valid option for y_3 is to set $y_3 = v_1$. The rest options $y_3 = v_2$ or $y_3 = v_3$ lead to constraint violations. Hence Core-Sp masks out the choices of $y_3 = v_2$ and $y_3 = v_3$ for the sequence-to-sequence model. In this way, Core-Sp addresses one key limitation of structured prediction models. We provide the details of Core-Sp in the next section.

2.3.1 Constrained Reasoning Embedded in Structured Prediction

The proposed Core-Sp creates an additional layer to the sequence-to-sequence model to enforce constraint satisfaction for the structured problems. It can be plugged into various structured prediction neural networks for different tasks. in this chapter, we demonstrate the Core-Sp layer on the sequence-to-sequence structured prediction network. Core-Sp works by masking out the output which violates constraints, hence providing correctness guarantees. Following the discussions of Section 2.2.1, the sequence-to-sequence structure prediction neural network takes input $x = (x_1, x_2, \dots, x_T)$ and $z = (z_1, z_2, \dots, z_T)$ in sequential format and outputs $y = (y_1, y_2, \dots, y_T)$. In the k-th step, score vector $o_k = (o_{k1}, o_{k2}, \dots, o_{kD_k})$ is produced by the sequence-to-sequence network, in which o_{kj} represents the un-normalized likelihood that y_k takes the value v_j. Vector $p_k = (p_{k1}, p_{k2}, \dots, p_{kD_k})$ is the result after normalizing o_k, where p_{kj} is the probability for variable y_k to take the value v_j. Without the addition of Core-Sp layer, certain p_{kj}'s which lead to constraint violations may be assigned with a positive probability value.

The Core-Sp module enforces constraints by masking out certain entries p_{kj} of the vector p_k which leads to constraint violations. Core-Sp tracks a pivot node in the associated MDD. Initially, the pivot node starts at the source node of the MDD and descends along a path determined by the output of Core-Sp in a sequential way. In the example in Figure 2.4, the pivot node starts at node s, descends along nodes u_1, u_4, and arrives at t, following the output of the sequence-to-sequence model. In each step, Core-Sp maintains a mask vector $c_k = (c_{k1}, c_{k2}, \ldots, c_{kD_k})$ based on the current pivot node. c_k is used to mask out entries in p_k that will lead to constraint violation. c_{kj} is set to 0, if there is no path labeled with v_j leaving the current pivot node. Otherwise, c_{kj} is set to 1. Suppose the pivot node is at u_1 in the example shown in Figure 2.4, c_{22} is set to 0, and c_{21}, c_{23} are set to 1 because the two outgoing edges from u_1 are labeled with v_1 and v_3. The next step of Core-Sp is the element-wise multiplication of p_k and c_k, resulting in p'_k. Denote $\odot$ as the element-wise vector-vector product, the masking step is computed as $p'_k = p_k \odot c_k$. Those entries that lead to constraint violation in p'_k are zeroed out. To make sure that the probabilities sum up to 1, p'_k further goes through a re-normalization step. The re-normalized probability vector is computed as: $\tilde{p}_{kj} = \frac{p'_{kj}}{\sum_{j'} p'_{kj'}}$. Finally, z_k is sampled uniformly at random from $\mathcal{U}(0,1)$ and the output y_k is decided based on the cumulative probabilities $P_{k1}, P_{k2}, \ldots, P_{k(D_k+1)}$ computed from $\tilde{p}_k$: $P_{k1} = 0$, and $P_{kj} = \sum_{j'=1}^{j-1} \tilde{p}_{kj'}$, for $j = 2, 3, \ldots, D_k$ and $P_{k(D_k+1)} = 1$. y_k is set to the value of v_j if and only if $z_k \in \Big[P_{kj}, P_{k(j+1)}\Big)$. Denote assignment indicator vector $q_k = (q_{k1}, q_{k2}, \ldots, q_{kD_k})$, where q_{kj} is an indicator variable for $y_k = v_j$. This implies q_{kj} is 1 if and only if $y_k = v_j$, otherwise $q_{kj} = 0$. After setting the value of y_k, the pivot node descends to a new node along the corresponding arc in the MDD. To conclude, the computational pipeline at the k-th step is reflected in the following equations:

$$p_{kj} = \frac{\exp(o_{kj})}{\sum_{j'=1}^{D_k} \exp(o_{kj'})}, \tag{2.4}$$

$$p'_k = p_k \odot c_k, \tag{2.5}$$

$$\tilde{p}_{kj} = \frac{p'_{kj}}{\sum_{j'=1}^{D_k} p'_{kj'}}, \tag{2.6}$$

$$\tag{2.7}$$

$$P_{kj} = \begin{cases} 0 & \text{for } j = 1, \\ \sum_{j'=1}^{j-1} \tilde{p}_{kj'} & \text{for } j = 2, 3, \ldots, D_k, \\ 1 & \text{for } j = D_k + 1. \end{cases} \tag{2.8}$$

$$\tag{2.9}$$

$$q_{kj} = \begin{cases} 1 & \text{if } z_k \in \left[P_{kj}, P_{k(j+1)}\right), \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

$$\tag{2.11}$$

$$y_k = v_j, \qquad \text{if } q_{kj} = 1, \text{ for } 1 \leq j \leq D_k \tag{2.12}$$

where $\odot$ denotes the element-wise product between two vectors and $z_k \sim \mathcal{U}(0, 1)$. We illustrate how Core-Sp works using the following Example 2.3.1.

Example 2.3.1. We illustrate the procedure of Core-Sp using the example in Figure 2.5. Initially, the pivot node for tracking the MDD is set as root node s. The first step is to set the value for the first variable y_1. Here, the neural network outputs an un-normalized likelihood vector $o_1 = (0.1, 0.2, 0.3)$. The next softmax layer takes in o_1 and outputs normalized probability vector $p_1 = \left(\frac{\exp(0.1)}{\exp(0.1)+\exp(0.2)+\exp(0.3)}, \frac{\exp(0.2)}{\exp(0.1)+\exp(0.2)+\exp(0.3)}, \frac{\exp(0.3)}{\exp(0.1)+\exp(0.2)+\exp(0.3)}\right) \approx (0.30, 0.33, 0.37)$. From the MDD on the right hand side, y_1 has only two valid assignments, v_2 or v_3. Therefore, Core-Sp produces a mask vector $c_1 = (0, 1, 1)$, which forbids y_1 taking the value v_1. As in Equation (2.5), multiplying p_1 with c_1 elementwisely gives us an un-normalized probability vector $p'_1 = (0, 0.33, 0.37)$. After the re-normalization operation in Equation (2.6), we obtain $\tilde{p}_1 = (0, \frac{0.33}{0.33+0.37}, \frac{0.37}{0.33+0.37}) \approx (0, 0.47, 0.53)$. According to Equation (2.10), the cumulative probability vector would be: $P_1 = (0, 0, 0.47, 1)$. We then uniformly sample z_1 with random between 0 and 1. In this example, $z_1 = 0.4 \in [0, 0.47)$, hence we get vector $q_1 = (0, 1, 0)$ and set $y_1 = v_2$. After setting y_1's value, Core-Sp set the pivot node to u_1 following the arc $e(s, u_1) = v_2$. It continues the same process of setting

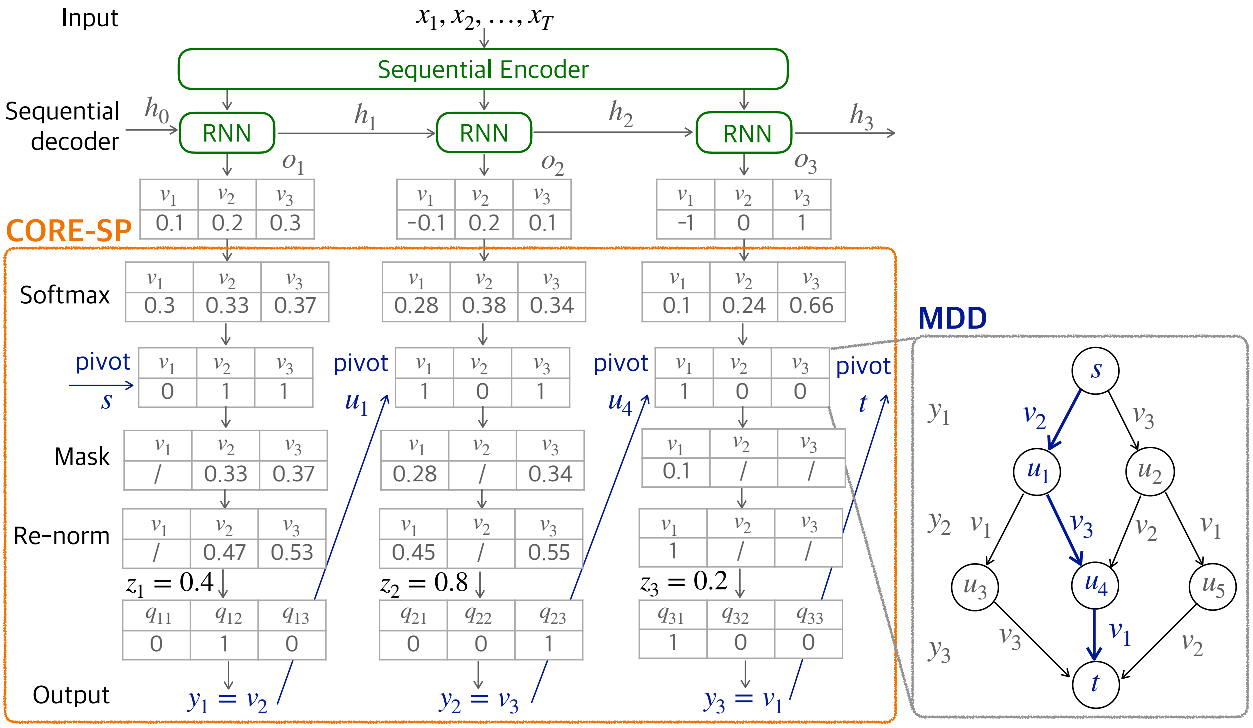


Figure 2.5. Architecture of embedding Core-Sp into a sequence-to-sequence model for the decision variables y_1, y_2, y_3, where the highlighted Core-Sp module encodes the exact MDD in Figure 2.2(a). Core-Sp descends layer-by-layer in the MDD. Initially, the pivot node of Core-Sp is at root s. The node s limits the value of y_1 to be $y_1 \in \mathtt{val}(s) = \{v_2, v_3\}$. If the model picks $y_1 = v_2$, then the pivot node moves to node u_1 following the arc $e(s, u_1) = v_2$. Next, the node u_1 limits the value of y_2 to be $y_2 \in \mathtt{val}(u_1) = \{v_1, v_3\}$. If the neural model picks $y_2 = v_3$, then the pivot node shifts to node u_4 following the arc $e(u_1, u_4) = v_3$. Finally, the pivot node descends to u_4 following the single outgoing arc: $e(u_4, t) = v_1$. Hence the assignment for variable y_3 becomes $y_3 = v_1$.

values for y_2 and y_3. This example sets y_2 to v_3 and y_3 to v_1, which corresponds to the blue path in the decision diagram on the right-hand side.

Proposition 2.3.2. *Let $\mathcal{M}$ be an exact MDD that is compiled from the constraint set $\mathcal{C}$, the sequence-to-sequence model with the addition of Core-Sp is guaranteed to generate structured outputs satisfying all constraints in $\mathcal{C}$.*

Implementation. Core-Sp allows for efficient back-propagation of the gradient of the neural network's parameters. In model training, all computations are differentiable during the gradient backward pass except for the setting of q_{kj} values. q_{kj} is set to 1 if and only if $z_k \in [P_{kj}, P_{k(j+1)})$. In other words, q_{kj}'s value is determined by $q_{kj} = \mathbf{1}\{z_k \geq P_{kj}\}\mathbf{1}\{z_k < P_{k(j+1)}\}$. When computing $\partial q_{kj}/\partial P_{kj}$, we use the sigmoid function with a very large constant to replace the indicator function $\mathbf{1}\{\cdot\}$. This operation allows for gradient propagation and improves the numeric stability of gradient computation, and avoids producing NaN or Infinity gradients. For the cases where the loss function can be directly defined on $\tilde{p}_k = (\tilde{p}_{k1}, \tilde{p}_{k2}, \ldots, \tilde{p}_{kD_k})$, such as the cross-entropy loss, we do not use z variables to sample variables y_k during training. z variables are used during testing. See applications in later text. The MDD inside Core-Sp is implemented with two key-value dictionaries. One dictionary memorizes all the mask vectors. It uses the nodes in the MDD as keys and returns the corresponding mask vectors. The other dictionary saves the connectivity of the MDD, it uses the current node in MDD as the key and returns all of its following nodes in the next step. This dictionary allows for the pivot node to descend along the path and is also used for the node split and arc filtering procedure discussed in the next section.

2.3.2 Discussions

Connection to Existing Works There are several existing works that also use reasoning tools to enforce constraints in neural-network-based models. OptNet [20] and MIPaaL [21] propose to encoding quadratic programming (QP) or mixed integer programming (MIP) to enforce constraints, these methods back propagates the gradients through their optimality conditions. Both approaches require solving a linear programming problem (or MIP problem) in the forward pass. In contrast, our approach pre-computes the feasible set and therefore can be integrated "as is" into the neural net. No LP or MIP solver is needed.

Another line of work encourage sparsity on the structured output. The sparseMAP method ([56]) models the probability distribution using a combination of a few sparse structured outputs. Their sparsity assumption implicitly enforces constraints by assigning invalid solutions zero probabilities. Nevertheless, their overall formulation needs to be convex, which limits the types of combinatorial constraints they can handle. The authors generalize their approaches to handle more general logic constraints in their follow-up work

LP-sparseMAP [64]. Their approach is to decompose the problem in the factor graph, and uses the Alternating Direction Method of Multipliers (ADMM) to enforce the consistent value assignments towards variables. This approach indeed provides a good way to handle constraints in structured prediction. However, ADMM only ascends towards the maximum of the dual problem, albeit the primal-dual gap can be large for non-convex problems. Our approach Core-Sp provides an alternative way to handle constraints beyond problem decomposition and harnessing the primal-dual gap.

[22] propose a strategy to formalize the constraints as an automata. During inference, the outputs are generated by walking step-by-step in the automata. Compared to this work, the MDD we use is similar to the automata since both of them only uses valid paths as valid solutions. However, we enforce Core-Sp during both learning and inference stages while their space-optimized automata can only be applied in inference. We will show in Section 2.6 that constraint satisfaction during learning actually leads to improvement in learning performance because of the reduced modeling space. In addition, we propose a relaxed search algorithm in Section 2.4 for MDD structures to automatically find the sweet point balancing model complexity and learning performance.

2.4 Searching for the Optimal CORE-SP Structure

The exact MDDs for real-world problems could be arbitrarily large, so the exact MDD may consume too much memory overhead. Because of the large space complexity of exact MDDs, it is not practical to deploy the Core-Sp with the exact MDD on several real-world problems. Also a large MDD implicitly implies a complex output space, which requires more data to learn an accurate model. In problems where exact MDDs are not practical, relaxed MDDs can be used to reduce the memory requirement. The set of solutions of a relaxed MDD forms a super-set of those of the exact MDD. In this section, we explore the trade-off between space complexity and learning performance by exploring the usage of relaxed MDDs.

To find the optimal MDD structure which balances memory consumption and learning performance, we propose an iterative search procedure in Algorithm 1. We tune the width

Algorithm 1: Iterative algorithm for searching optimal performance of Core-Sp.

```
Input: Training set $D^{tr}$, validation set $D^{val}$; Parameters of sequence-to-sequence
       neural network $\theta$; Constraints $\mathcal{C}$; Maximum layer width $\omega_{\max}$.
Output: Core-Sp module $(\mathcal{M}, \theta)$ with optimal performance.
$\mathcal{L}_{prev} = +\infty$;
$\mathcal{M}$ = initMDD($\mathcal{C}, w = 1$);                  // initialize the width-1 MDD
for $w = 2$ to $\omega_{\max}$ do
    $\theta$ = train($\theta$, $\mathcal{M}$, $D^{tr}$);          // learn $\theta$ with MDD on training data
    $\mathcal{L}$ = validation($\theta$, $\mathcal{M}$, $D^{val}$);        // evaluate on validation data
    if $\mathcal{L} > \mathcal{L}_{prev}$ then
        break;
    else
        $\mathcal{L}_{prev} = \mathcal{L}$;
    $\mathcal{M}$ = relaxMDD($\mathcal{M}$, $\mathcal{C}$, $w$);          // relax MDD with a larger width
return $\mathcal{M}$, $\theta$;                              // find the optimal Core-Sp
Procedure relaxMDD($\mathcal{M}$, $\mathcal{C}$, $w$):
    for layer = 1 to layerSize($\mathcal{M}$) do
        while |$\mathcal{M}$[layer]|< $w$ do
            do
                u = $\mathcal{M}$[layer].pop();              // pick node u to split
            while |u.in|<=1;
            vi, wi = nodeSplit(u.in);     // split incoming arcs of node u
            v = node(vi, u.out); w = node(wi, u.out);
            arcFilter(v, w, $\mathcal{C}$);     // filter invalid arcs by constraints
            $\mathcal{M}$[layer].add(v, w);         // update the MDD with new nodes
            $\mathcal{M}$[layer].delete(u);
    return $\mathcal{M}$;
```

parameter to find a relaxed Core-Sp that achieves the optimal performance. The algorithm starts increasing the width from 1 to the given hyper-parameter maximum layer with $\omega_{\max}$, iteratively learning Core-Sp model with new MDD model $\mathcal{M}$ on the training set, validating their learning performance on a separated validation data set until finding a good MDD structure. The inputs to Algorithm 1 are training and validation data sets (D^{tr}, D^{val}), parameters of the sequence-to-sequence neural network θ, a set of constraints $\mathcal{C}$ and the maximum width $\omega_{\max}$ of MDD. At beginning, the `initMDD` function initializes a width-1 MDD. At every iteration, the `train` function trains the neural network with the constraints enforced in the relaxed MDD via gradient descent on the training data set. This is detailed in Section 2.3.1. Then the `validation` function evaluates the performance on a separate validation data set. In line $6-9$ of Algorithm 1, we evaluate if the current MDD has a better performance than the previous one. If the loss on the validation set is decreasing, the algorithm would continue the relaxation; otherwise, the algorithm terminates and returns the current Core-Sp as well as the learned parameters.

For the `relaxMDD` procedure, it takes a relaxed MDD as input and relaxes all its layers from top to bottom to the given width. For each layer, the algorithm repeatedly picks a node and then split it into two nodes, which corresponds to the `nodeSplit` function until the width of the layer reaches the given width. Every node split is followed by arc filtering process to enforce constraints, that we use `arcFilter` to denote the process. See Figure 2.3 for an example of `nodeSplit` and `arcFilter` process on a relaxed MDD.

Note that in node splitting (line $14-17$ of Algorithm 1), those nodes with only one incoming arc are skipped for the splitting process, i.e., $u.in \geq 1$. In our paper, the heuristics for splitting is: the set of incoming arcs (noted as $.in$) of the original node is randomly assigned to the two newly created nodes (v, w) and then the outgoing arcs (noted as $.out$) are copied to nodes v, w. There are other heuristic methods for node splitting. We refer the readers to [63] for details. The `arcFilter` function is applied to remove those outgoing arcs that lead to constraints violation.

Limitations While the advantages of Core-Sp have been demonstrated with a few real-world applications, several limitations of Core-Sp present and we leave the development of new methods addressing these limitations as future work. First, the current constraint reasoning module based on MDDs cannot enforce continuous-valued constraints. Indeed, discretization can transform continuous constraints into discrete ones and apply the developed Core-Sp, however such discretization may result in very large decision diagrams. Second, the decision diagram is based on a sequence-to-sequence structured prediction model. Other encoding-decoding structures, for example encoding-decoding on a graph, may require exploring other types of structured prediction models. We leave the work to address these limitations as future works.

2.5 Applications

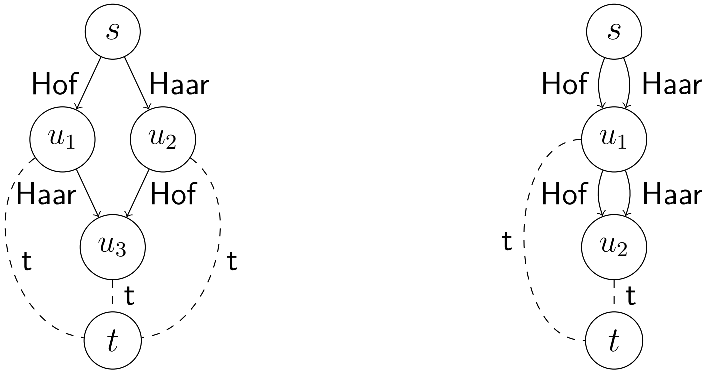
2.5.1 Vehicle Dispatching Service Planning

Task Definition. Consider a routing problem in which one needs to dispatch a service vehicle to perform maintenance at a set of locations. The sets of locations differ per day and are rarely the same. Previous routes indicate that the driver does not follow a clear objective, such as minimizing the distance or time. Instead, historical data suggest that the driver has an underlying route preference, such as visiting a shopping mall after leaving a restaurant. Our task is: given the historic routes and a set of requested locations, determine a path that visits all the locations once and only once while capturing the hidden trends embedded in the historical data. To be specific, given a request to visit a set of locations $x = \{x_1, x_2, \dots, x_{T_i}\}$ in the i-th day, determine $y = (y_1, y_2, \dots, y_{T_i})$, which forms a permutation of x and captures the driver's preferences. For this application, we assume an upper bound T on the number of sites to visit per day. In other words, for all i, $T_i \leq T$.

Traditional optimization methods such as integer programming or constraint programming do not work well in this context since they are unable to represent an appropriate objective function for the latent route preference [65]. Machine learning models can be used to learn the underlying pattern from the historical routes [66]. Nevertheless, the routes generated from pure machine learning models cannot satisfy key operational constraints. They may visit some locations multiple times, or fail to visit all locations. Post-processing steps, such as removing redundant locations and randomly appending unvisited locations, have been tried to fix the output of machine learning models [41]. However, their performance are limited in our experiments (see Section 2.6.1 for details).

Constraint Definition. The input of this application is a set of locations to visit for day i: $x = \{x_1, x_2, \dots, x_{T_i}\}$. The goal is to generate a schedule y to represent the order of visiting the locations, where y is a permutation of x. The route y needs to satisfy the following constraints:

- `full-cover` constraint. The delivery route should visit all and only the locations in x. In other words, the set of locations in y is the same as the set in x.



(a) exact MDD (b) width-1 relaxed MDD

Figure 2.6. The MDDs used in vehicle dispatching service planning. (a) An exact MDD which models the visit to "Hof", or "Haar", or both of them. All arcs of solid lines are of first type and arcs of dashed lines are of second type, which directs the delivery agent to the stop location t. (b) A width-1 relaxed MDD, which is formed by combining nodes u_1 and u_2 of the exact MDD.

- `all-diff` constraint. The route should not visit one place twice. In other words, $y_j \neq y_k$ for all $y_j, y_k \in y$ and $j \neq k$.

MDD Construction. The delivery routes have at most T locations in the data set, so the MDD graph $\mathcal{M}$ would contain $T + 2$ layers. There is a single source node s in the first layer and a single sink node t at the last layer. There are two types of arcs in the MDD. For the first type, an arc $e(u, u') = v_i$ (where $u' \neq t$) in the j-th layer represents that we visit v_i as the j-th location in the schedule. The second type of arcs $e(u, t) = t$ connect every node to the sink node t, allowing the delivery agent to travel to the ending location at any time.

Figure 2.6(a) shows an example MDD. Here, the delivery agent needs to visit "Hof" and "Haar" or both of them. The arcs of the first type are shown in solid lines and the arcs of the second type are shown in dashed lines. The two arcs $e(s, u_1) = Hof, e(s, u_2) = Haar$ leaving the root node s denote the first location that the delivery agent visit, which can be either "Hof" or "Haar". The three arcs $e(u_1, t) = e(u_2, t) = e(u_3, t) = t$ are of second type. The purpose is to bring the delivery agent to the ending location t. In practice, an MDD that represents all valid paths can be exponential with respect to the number of maximum locations. Figure 2.6(b) shows a width-1 relaxed MDD, formed by combining nodes u_1 and

u_2 in Figure 2.6(a). The paths in relaxed MDDs are supersets of all valid paths. As we can see, "Hof → Hof → t" is a path in the relaxed MDD, but it violates the `all-diff` constraint.

MDD Filtering to Process Daily Requests. The previously constructed MDD contains routes up to length T visiting all the locations. The delivery request of each day is a subset of these locations, where the length T_i is less than T. Therefore, we present `FilterDaily` which augments the aforementioned MDD into MDDs encoding schedules that meet daily requests. `FilterDaily` removes arcs that represent the visits to locations outside of each day's requested set. It also removes arcs of the second type that visit too many or too few locations. Then, working from the last layer up to the first layer, `FilterDaily` recursively removes nodes or arcs that are unreachable from node t.

Proposition 2.5.1. *Suppose $\mathcal{M}$ is an exact MDD representing routes up to length T visiting all the locations. The delivery request for day i is x. Then* `FilterDaily`$(\mathcal{M}, x)$ *returns an exact MDD which includes all valid routes that satisfy the delivery need for day i.*

Model Structure. We employ Core-Sp module on a conditional Generative Adversarial Network (cGAN) to generate routes that capture the implicit preferences of the drivers as well as preserving the operational constraints. In the generative adversarial structure, the generator network G is trained to generate routes to mimic the pattern in the training data set. The discriminator network D is trained to separate the generated routes from the actual ones in the training data set. When training converges, the discriminator should not be able to tell the difference between the true outputs and the structures generated by the generator. In return, the generator generates structures that closely look like the ones in the data set. The Core-Sp module is embedded in the generator and filters out those routes that violate the operational constraints. As a result, the generated routes would satisfy all operational constraints. We employ the conditional GAN model structure because the element-wise loss function is not ideal to measure the distance between the predicted route and the ground truth route. Suppose one route $(y_T, y_{T-1}, \ldots, y_1)$ is a circular shift of the optimal route $(y_1, y_2, \ldots, y_T)$. Both of them may be equally good to fit the delivery constraints as well as the driver's underlying preference. However, an element-wise loss function penalizes the shifted route heavily because it is different from the optimal route in every location.

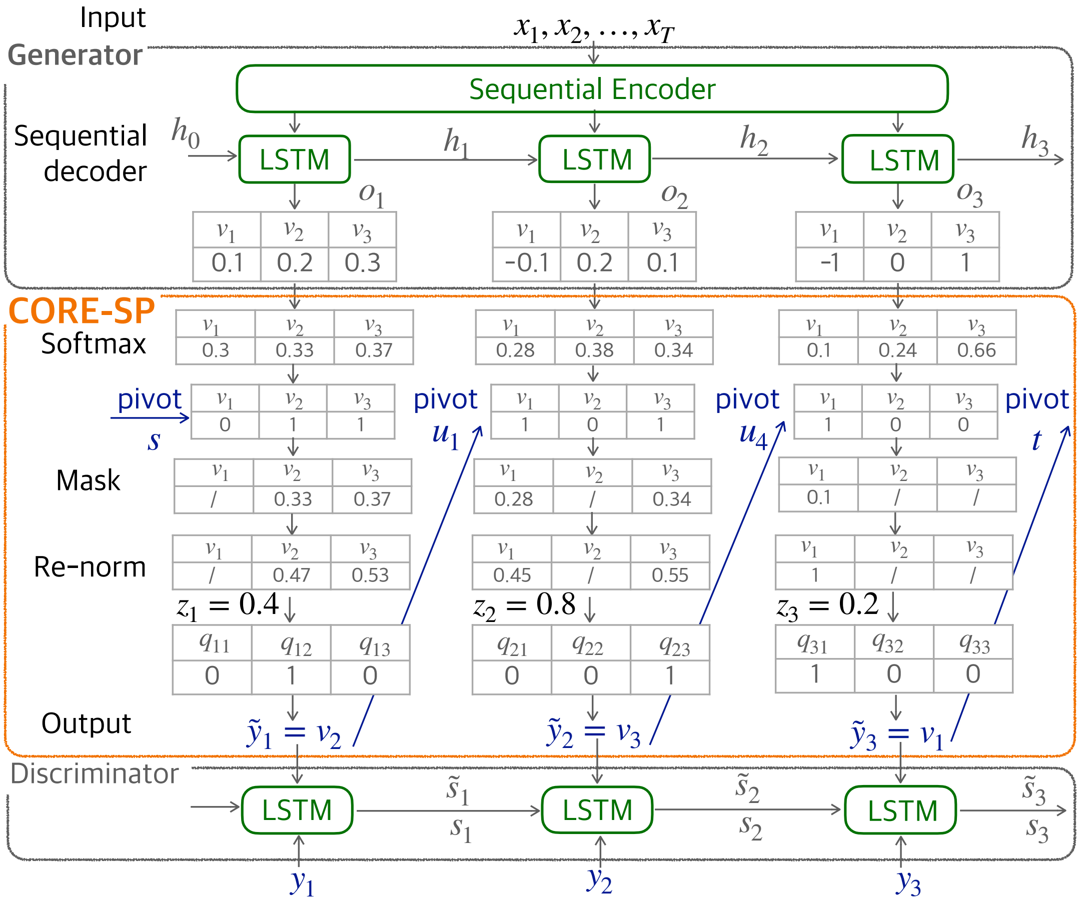


Figure 2.7. The conditional GAN with Core-Sp module for the vehicle dispatching service planning problem. x represents the requested delivery locations in day i. $(\tilde{y}_1, \tilde{y}_2, \ldots)$ represents the generated path from Core-Sp, represented using indicator vectors $(q_1, q_2, \ldots)$. The Generator G takes the set of locations x as input and use a sequential encoder to learn a representational vector. Then it outputs a sequence of score vector $(o_1, o_2, \cdots)$ using a sequential decoder, where o_j denotes the likelihood of picking the next locations at j-th step. The Core-Sp module removes invalid locations. The Discriminator D is used to separate the real path $(y_1, y_2, \ldots)$ from the generated path $(\tilde{y}_1, \tilde{y}_2, \ldots)$.

The overall conditional GAN with Core-Sp architecture is shown in Figure 2.7, which is composed of the generator G and the discriminator D. The generator G take the set of locations x as input and outputs the un-normalized score vectors $(o_1, o_2, \cdots)$, where vector o_j denotes the un-normalized likelihood of visiting each location at the j-th step. Core-Sp takes in these score vectors and the random values $(z_1, z_2, \cdots)$ as inputs, and outputs a valid route $(q_1, q_2, \cdots)$. Here, vector $q_j = (q_{j1}, \ldots, q_{jN})$, and q_{jk} is an indicator variable representing whether location k is visited in the j-th step. Finally, the discriminator function D tries to separate the predicted route $(q_1, q_2, \cdots)$ and the ground-truth route $(y_1, y_2, \cdots)$. Here, each $y_j = (y_{j1}, \ldots, y_{jN})$ is again represented as a vector, where y_{jk} indicates whether to visit location k in the j-th step. The generator G uses a encoder to learn a representation vector for the input and uses a sequential decoder to generate the schedule.. The daily request x is fed as input vectors of each step where the locations in the requested set are marked as 1. G uses the following LSTM structure to generate the schedule:

$$\begin{aligned} h_j &= \texttt{LSTM}(x, h_{j-1}), \\ o_j &= W h_j, \end{aligned}$$

where the score vector o_j represents the likelihood of picking next location in the j-th step. The score vector o_j along with the random variable z_j are fed into the Core-Sp module. The Core-Sp module removes invalid locations and produces q_j according to the random value of z_j. The detailed equations of Core-Sp can be seen in Section 2.3.1. For the discriminator D, it is trained to separate the generated schedule $q = (q_1, q_2, \ldots, q_{T_i})$ from the real schedule $y = (y_1, y_2, \ldots, y_{T_i})$. It uses the following LSTM structure:

$$\begin{aligned} \tilde{s}_j &= \text{LSTM}(q_j, \tilde{s}_{j-1}), \\ s_j &= \text{LSTM}(y_j, s_{j-1}), \end{aligned}$$

where $\tilde{s}_j$ denotes the hidden vector after encoding the first j generated locations and s_j denotes the hidden vector after encoding the first j locations in real data. The output of the discriminator D is $\sigma(Us)$, where U is a linear transformation matrix and s is either s_{T_i} or $\tilde{s}_{T_i}$. $\sigma(s) = 1/(1 + \exp(-s))$ is the sigmoid activation function. Overall, D and G are trained by minimizing the loss function in a competing manner using stochastic gradient descent. The loss function is:

$$\min_G \max_D \mathbb{E}_{x,y} \left[\log D\left(y, x\right)\right] + \mathbb{E}_{z,x,y} \left[\log\left(1 - D\left(G\left(x, z\right), y\right)\right)\right].$$

Input sentence:

Blink light of your Philips Hue when your Amazon Alexa timer hits 0.

Output labels:

trigger-service y_{ts}	trigger-function y_{tf}	action-service y_{as}	action-function y_{af}
amazon-alexa	timer-goes-off	Philips-hue	blink-light

Figure 2.8. An example of if-then program synthesis task. The input is a natural language description of the program. The output are four labels: `trigger-service`, `trigger-function`, `action-service` and `action-function`. The semantics of the synthesized if-then program are: if `trigger-function` happened at `trigger-service`, then take `action-function` at the `action-service`.

2.5.2 If-Then Program Synthesis

Task Definition. Many Internet applications provide automatic services to meet the users' needs, such as the Weather Underground website provides daily weather reports and Youtube provides video streaming services. Connectivity platforms such as IFTTT[1] and Zapier[2] streamline services from different providers by connecting simple services into more complex ones, in the form of if-then programs. For instance, the smart device Philips Hue can automatically blink lights when commands are sent from a cellphone. Amazon Alexa can be programmed as a timer via voices. Given these two services, users can set up an if-then program on the IFTTT platform for more complicated tasks. For example, an if-then program can command the Philips Hue to blink lights when the timer in Amazon Alexa reaches zero. Nevertheless, the interface on the IFTTT website still takes a few hours to learn, especially for beginners without programming experience. It would be great if such if-then programs can be automatically synthesized from the natural language to provide suggestions for those beginners. This helps to reduce the overhead in using such platforms and boost the users' efficiency.

We consider the task of generating if-then programs from the natural language as a structured prediction task. In our setup, an if-then program is made up of four components:

[1] ↑https://ifttt.com/

[2] ↑https://zapier.com/

`trigger-service`, `trigger-function`, `action-service`, and `action-function`. The logic is "if `trigger-function` happens in the `trigger-service`, then take the `action-function` from the `action-service`". Such programs can be represented using this pseudo-code:

```
IF trigger-service.trigger-function THEN
    action-service.action-function
```

Figure 2.8 shows an example of this task. We would like to transform a user's text description: *"Blink light of the Philips Hue when the Amazon Alexa timer hits 0"* into the following if-then program:

```
IF Alexa.timer-go-off  THEN
    Hue.blink-light
```

The challenge in if-then program synthesis is to enforce the constraints between the services and the associated functions. Without enforcing constraints, the output of the structured prediction model may be invalid. For instance, the model can predict "Hue" for `trigger-service`, but assigns "report rain" to the `trigger-function`, which we know before training that the smart device "Hue" does not provide any weather reporting services.

Constraints Definition. We enforce the `Functionality` constraint for the if-then program synthesis. Let s be a service. We define a mapping $F(s)$ to be the set of functions which can be associated with s. For example, if s is "weather service", then the output of $F(s)$ is a set that contains functions of "hourly report", "tomorrow forecast", and "severe weather alarms", etc. The `Functionality` constraints for all the four components are:

$$\texttt{trigger-service} = s \Rightarrow \texttt{trigger-function} \in F(s),$$
$$\texttt{action-service} = s \Rightarrow \texttt{action-function} \in F(s).$$

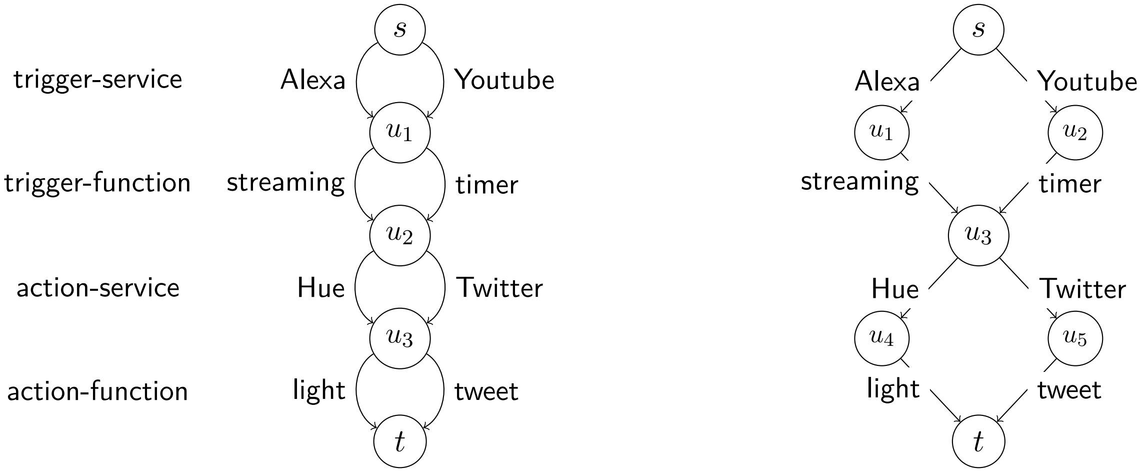
The mapping $F(s)$ are collected from the introduction pages of every internet application and are provided to us as prior information.

MDD Construction. The `Functionality` constraints can be represented using an MDD with five layers, where the first layer has one source node s and last layer has one sink node t. Each arc between the 1st and the 2nd layers corresponds to one value assignment to the variable `trigger-service`. Each arc between the 2nd and 3rd, 3rd and 4th, and 4th and 5th layers corresponds to the value assignment to variable `trigger-function`, `action-service`, and `action-function`, respectively. Figure 2.9(a) and (b) represent a width-1 and width-2 MDD for if-then program synthesis.

In the MDD, multiple arcs from the first layer can be connected to a single node u in the second layer. The node u hence represents the set of if-then programs which have a given subset of trigger services. The set of arcs leaving from u represents the union of all the trigger functions which are associated with the trigger services connecting u. For example, Figure 2.9(a) demonstrates that both Alexa and Youtube can be associated with the streaming and timer services. Notice that this is a relaxed MDD. In practice, only Alexa has the timer service and only Youtube has the streaming service. Similar semantic meaning holds for the arcs between the 3rd and 4th layers, representing action services and action functions.

The width of the MDD can be expanded to enforce constraints more precisely. Figure 2.9(b) shows an example for this. Here, the nodes u_1 in (a) is split into two nodes $\tilde{u}_1$, $\hat{u}_1$ in (b). After arc filtering, $\tilde{u}_1$ is connected to “timer” only while $\hat{u}_1$ is connected to “streaming” only because only “Alexa” has the service “timer” and only “Youtube” has the service “streaming”. Similar node splitting and arc filtering are applied for action services and functions as well. In a nutshell, the relaxed MDD can be expanded into an exact one using repeated node splitting and arc filtering.

Model Structure. We employ Core-Sp on the Latent Attention model proposed by [42], which achieved the state-of-the-art result on if-then program synthesis. The Latent Attention model is a bidirectional LSTM with residual connection, followed by the self-attention mechanism. To be specific, the bidirectional LSTM (Bi-LSTM) encodes a natural sentence input of length T: $x = (x_1, x_2, \dots, x_T)$ into T latent vectors $(h_1, h_2, \dots, h_T)$. Here, x_j is a one-hot vector representing the j-th word in the sentence. Each vector h_j is a concatenation of a forward vector $\overrightarrow{h_j}$ and a backward vector $\overleftarrow{h_j}$. Suppose vector $\overrightarrow{h_j}$ and $\overleftarrow{h_j}$ are of length m, vector h_j will be of length $2m$. The forward vector $\overrightarrow{h_j}$ is the result of encoding input words $x_1, x_2, \dots, x_j$ from the left through an LSTM, and the backward



(a) width-1 relaxed MDD (b) exact MDD

Figure 2.9. Examples of a relaxed and an exact MDD for the if-then program synthesis task. The exact MDD in (b) models constraints that only the timer service is provided by Alexa, and only the streaming service is provided by Youtube. Similarly, only Hue provides the light service, and only Twitter provides the tweet service. (a) is a relaxed MDD, where both trigger services provide both trigger functions, and both action services provide action functions.

vector $\overleftarrow{h_j}$ is the result of encoding input words $x_T, \dots, x_{T-j}$ from the right. More precisely, in mathematical form:

$$\begin{aligned}
\overrightarrow{h_j} &= \texttt{LSTM}(x_j, \overrightarrow{h_{j-1}}),\\
\overleftarrow{h_j} &= \texttt{LSTM}(x_{T-j+1}, \overleftarrow{h_{j+1}}),\\
h_j &= \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right].
\end{aligned}$$

where [;] denotes vector concatentation. The detailed equations for `LSTM` neural network with a residual connection can be seen in [67]. In the second step, we encode the sequence of vectors $(h_1, h_2 \cdots, h_T)$ into a single vector g through an attention mechanism, which is similar to [68]:

$$\begin{aligned}
\alpha_{jk} &= \frac{\exp({h_k}^\top h_j)}{\sum_{k'=1}^{T} \exp({h_{k'}}^\top h_j)}, \qquad \text{for all } j, k \in \{1, \cdots, T\},\\
g &= \sum_{j=1}^{T}\sum_{k=1}^{T} \alpha_{jk} h_k
\end{aligned}$$

This Bi-LSTM with attention neural network structure encodes the entire sentence into one latent vector g of size $2m$. Let U_{ts} be a matrix of size $|\#\text{service}| \times 2m$, where $|\#\text{service}|$ is

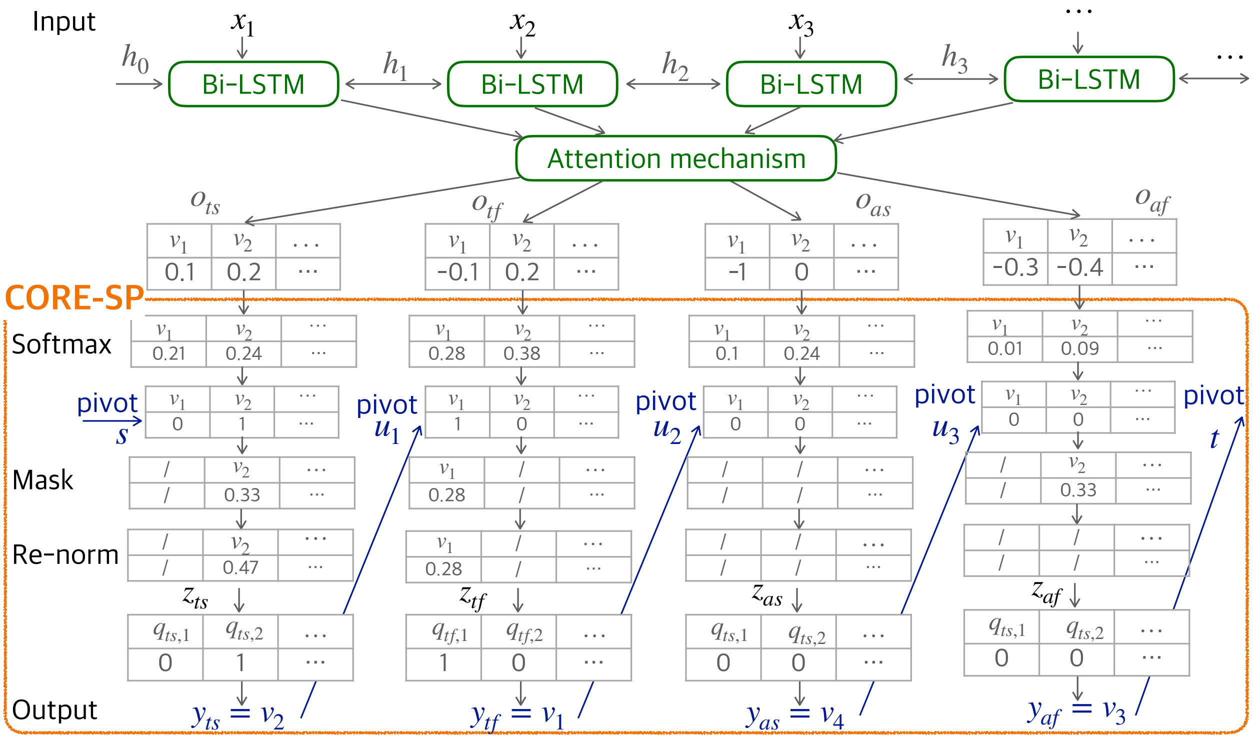


Figure 2.10. Model structure of If-then program synthesis. Text input $x_1, x_2, \cdots, x_T$ is fed into bi-directional LSTM with self attention mechanism. The un-normalized likelihood vectors $o_{ts}, o_{tf}, o_{as}, o_{af}$ are fed into the CORE-SP module for constraint satisfaction.

the number of `trigger-service`. We define matrices U_{tf}, U_{as}, U_{af} for `trigger-function`, `action-service`, `action-function` respectively with their corresponding shapes. The un-normalized likelihoods for `trigger-service` o_{ts}, for `trigger-function` o_{tf}, for `action-service` o_{as}, and for `action-function` o_{af} are defined as

$$o_{ts} = U_{ts}g, \quad o_{tf} = U_{tf}g, \quad o_{as} = U_{as}g, \quad o_{af} = U_{af}g.$$

These vectors $o_{ts}, o_{tf}, o_{as}, o_{af}$ are fed into the CORE-SP module. During training, we minimize a cross entropy loss function between the ground-truth prediction and the probabilities $\tilde{p}_{ts}$, $\tilde{p}_{tf}$, $\tilde{p}_{as}$, $\tilde{p}_{af}$ produced from CORE-SP (definition of these probabilities are in Equation 2.6). This training procedure is similar to the teacher-forcing approach used in [53] to accelerate the learning speed. Variables z are used to sample particular (`trigger-service`, trigger-function, action-service, action-function) quadruples from the probability distribution given by $\tilde{p}_{ts}$, $\tilde{p}_{tf}$, $\tilde{p}_{as}$, $\tilde{p}_{af}$ during testing.

2.5.3 SQL Query Generation from Natural Language

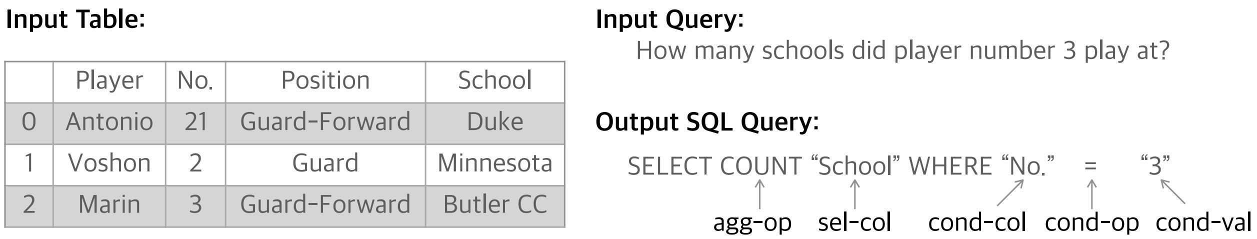
Task Definition. Formatted data such as travel records and stock market transactions are stored in the relational databases. Currently, accessing the database requires a data scientist who masters the SQL query language. Our task is to automatically synthesize SQL queries from natural language sentences using machine learning. Compared with the data expert approach, SQL query generation requires deeper reasoning across the structure of the database, the semantics of the structured query language, and the understanding of natural language. As shown in Figure 2.11, the input of the text2SQL generation is a sentence that describes the query in natural language and the table headers in the relational database. The output is a SQL query with the following structure:

```
SELECT agg-op sel-col
WHERE (cond-col cond-op cond-val) AND ...
```

Here, SELECT and WHERE are the keywords in SQL language. What we need to predict are: (1) the aggregation operator agg-op, which chooses among the set {empty, COUNT, MIN, MAX, SUM, AVG}; (2) the column name in selection sel-col and (3) the column name in condition cond-col, both of which are chosen from the table headers; (4) the conditional operator cond-op, which is in $\{=, <, >\}$; (5) the conditional value cond-val, which are assumed to be a sub-sequence of the given query. Here, one bracket () represents one conditional statement. The SQL query may have multiple conditions, which are denoted by "..." operator. Figure 2.11 displays this SQL query:

```
SELECT COUNT "School"
WHERE "No." = "3"
```

Here agg-op is COUNT; sel-col is "school", which is a column name from the table headers. One cond-col is "No.", which also come from the table headers. The cond-op is "=". The cond-val is "3", which we assume is from the input query. This example has one condition but multiple conditions are allowed.



Input Table:

	Player	No.	Position	School
0	Antonio	21	Guard-Forward	Duke
1	Voshon	2	Guard	Minnesota
2	Marin	3	Guard-Forward	Butler CC

Input Query:
How many schools did player number 3 play at?

Output SQL Query:
SELECT COUNT "School" WHERE "No." = "3"

Figure 2.11. An example for the Text2SQL generation task. The input is the text query "How many schools did player number 3 play at?" and the table header "`Player, No., Position, School`" from the relational database. The output should be the SQL query: `SELECT COUNT "School" WHERE "No." = "3"`.

Constraints Definition. Existing generative neural models for this task are not guaranteed to generate a query that follows the grammar of a SQL query. To avoid these grammar violations, we compile a set of common SQL grammars as constraints into the Core-Sp module. The Core-Sp module will ensure that all the generated SQL queries follow the grammatical constraints. Our constraints are defined on the operators, namely the conditional operator `cond-op` and the aggregation operator `agg-op`. The domains of these operators are dependent upon the data types of the entities (namely, `cond-col` and `sel-col`) they operate on. Taking the previous example. The `agg-op` can only take values between `{empty, COUNT}`, because the `sel-col` is "school", which is of the string type. More precisely, let s be a column header (the value of `sel-col` or `cond-col`) . We define $F_a(s)$ as the set of aggregation operators `agg-op` that can be associated with s, and $F_c(s)$ as the set of condition operators `cond-op` which can be associated with s.

$$F_a(s) = \begin{cases} \texttt{\{empty, COUNT, MIN, MAX, SUM, AVG\}} & \text{If } s \text{ of is numeric type} \\ \texttt{\{empty, COUNT\}} & \text{If } s \text{ of is string type} \end{cases}$$

$$F_c(s) = \begin{cases} \texttt{\{=, >, <\}} & \text{If } s \text{ is of numeric type} \\ \texttt{\{=\}} & \text{If } s \text{ is of string type} \end{cases}$$

The `dataype` constraints are defined as:

$$\texttt{sel-col} = s \Rightarrow \texttt{agg-op} \in F_a(s),$$
$$\texttt{cond-col} = s \Rightarrow \texttt{cond-op} \in F_c(s).$$

Model Structure. We embed the CORE-SP module to SQLova [43], the state-of-the-art neural network for text2SQL generation. SQLova has a sequence-to-sequence architecture. It first encodes a natural language sentence and the table headers into a high-dimensional vector. Then the decoder of SQLova decodes the hidden representation into the predictions of various entities in the SQL query. SQLova first determines the number of conditions in the SQL query then fill in the (`cond-col`, `cond-op`, `cond-val`) for each condition. The operators `agg-op, cond-op` are predicted as a classification task from a fixed set of operators. Column names `cond-col, sel-col` are predicted from the set of table headers in the relational database. The `cond-val` is predicted by a pointer neural network which points at a span of the input natural language sentence. The selected span of the query is used as the `cond-val` [69].

MDD Construction. The associated MDD that encodes the constraints for text2SQL generation is similar to that in if-then program synthesis. The MDD is split into layers and every two layers form a group. One two-layer group is used to enforce constraints on an operator-column name pair. The operator-column name pair can be `agg-op` and `sel-col`, or can be `cond-op` and `cond-col`. Note that there can be only one group of `agg-op` and `sel-col` and more than one group of `cond-op` and `cond-col`. In the first layer of the group, the column name is determined. In the second layer, the invalid operators are ruled out based on the type of the column name selected in the first layer. The two-layer group is copied several times because there can be multiple conditions allowed in the SQL query.

2.6 Results and Analysis

We demonstrate the effectiveness of the CORE-SP module on three applications. We mainly focus on two metrics: (1) the percentage of valid structures generated; and (2) the learning performance. Metric (1) evaluates whether CORE-SP is able to improve constraints satisfaction for the structures generated by neural network models, while metric (2) considers whether CORE-SP improves the overall performance of neural network models in pattern detection from data. For the task of if-then program synthesis and text2SQL generation, accuracy was used as the metric for learning performance. It measures the percentage that the predicted structures match exactly with the ground-truth structures in the testing set. For the vehicle dispatching service planning, we introduce a quantitative metric that measures how close the generated routes resemble those in the training set. The quantitative metric will be discussed later. We also demonstrate the effect of MDD structures, especially the change of the layer width, on the overall performance of the CORE-SP module.

Our experimental results demonstrate the efficiency of CORE-SP in boosting both the percentages of valid structures generated and the learning performance. In terms of constraint satisfaction, the percentage of valid routes generated increases from 1% to 100% for vehicle dispatching service planning with the embedding of CORE-SP on a conditional GAN model. The percentage of valid programs also increases from about 88% to 100% in the task of if-then program synthesis when CORE-SP is added to the Latent Attention model, and from 83% to 100% for text2SQL generation when CORE-SP is added to SQLNova on the hard test set. Both the Latent attention and SQLNova are state-of-the-art models for the corresponding tasks. Furthermore, the CORE-SP module also helps improve learning performance. For the if-then program synthesis, the accuracy is 44% for CORE-SP compared to 42% for the Latent Attention model. The neural network also converges to relatively higher accuracy with fewer training epochs. In the text2SQL task, the execution accuracy improves from 76.1% obtained from the state-of-the-art SQLNova model to 78.0% while the logical accuracy improves from 58.3% to 62.5% with the CORE-SP module embedded. The code for all the experiments is collected at this link[3].

2.6.1 Vehicle Dispatching Service Planning

Our experiments are on a data set consisting 29 cities in Bavaria[4]. We vary the number of maximum locations T in the daily requests from 2 to 29 in generating the training and

[3] ↑code summary: https://jiangnanhugo.github.io/CORE-SP/

[4] ↑TSP data set: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/

testing sets. We generate $N = 10,000$ instances for every given T. The daily requests are randomly sampled from all sets of locations of the specified size. The optimal delivering paths are generated assuming that the delivery agent is maximizing a hidden reward function:

$$R(y) = \sum_{j=1}^{T-1} \texttt{pref}\left(y_j, y_{j+1}\right). \tag{2.13}$$

Here the scalar value $\texttt{pref}\left(y_j, y_{j+1}\right) \in [0, 1]$ is the delivery agent's implicit preference to visit location y_{j+1} after leaving the location y_j. When generating the data set, we enumerate all valid delivery routes, and select the one that maximizes this reward function R. Notice that this reward function R was fixed during the data generation step and was hidden to the machine learning algorithms. During the evaluation, the reward function was used as one quantitative measure for the quality of the routes generated. The higher the reward function values, the better the machine learning algorithm was able to capture the hidden preferences of the delivery agent.

As shown in Figure 2.12 (left), when we relax the MDDs iteratively, the memory usage grows quickly to more than 1GB for moderately large T. The exact MDD is hardly to be loaded into memory when the maximum location is $T = 29$. In this case, we also consider an iterative algorithm which incrementally creates the exact MDD alongside the output of the sequential decoder. At step t, we follow the prediction made by the sequential decoder and only expand the MDD one step starting at the node selected by the sequential decoder. This corresponds to only loading its current outgoing arcs at every predicted step of the exact MDD. Using this idea, we are able to expand the exact MDD even for large number of locations ($T = 29$).

Valid Routes Comparison. We evaluate the performance of Core-Sp in generating valid routes over data sets of different sizes. We also include a post-processing method proposed by the work of [41] as a baseline for comparison. The post-processing step is added to the output of the conditional GAN without Core-Sp. First, we compare the performance of Core-Sp implemented with exact MDDs against baselines. Notice we use the aforementioned iterative algorithm when the number of locations T is too big for the relaxation algorithm. As shown in Figure 2.13 (left), the baseline can only generate around 0.1% of valid routes. From our manual inspection, the generated routes are mostly likely to visit one place more than once. The post-processing method uses a mask vector to enforce that the model can only visit the locations in the daily requested set, and then it removes all the duplicates in the output schedule. Once we apply the post processing method on the output generated by the baseline, the model's performance is improved to 50% for the data set $T = 2$. However, the post-processing method cannot handle the combinatorial complexity of the dispatching problem, as its performance quickly falls close to the baseline when we increase the number of locations in the daily requested set. In contrast, the percentage of valid routes is always 100% using the exact Core-Sp. The percentage of valid routes using relaxed MDDs (width=128) are shown in Figure 2.13 (right). We can see that Core-Sp still produces over 80% valid routes.

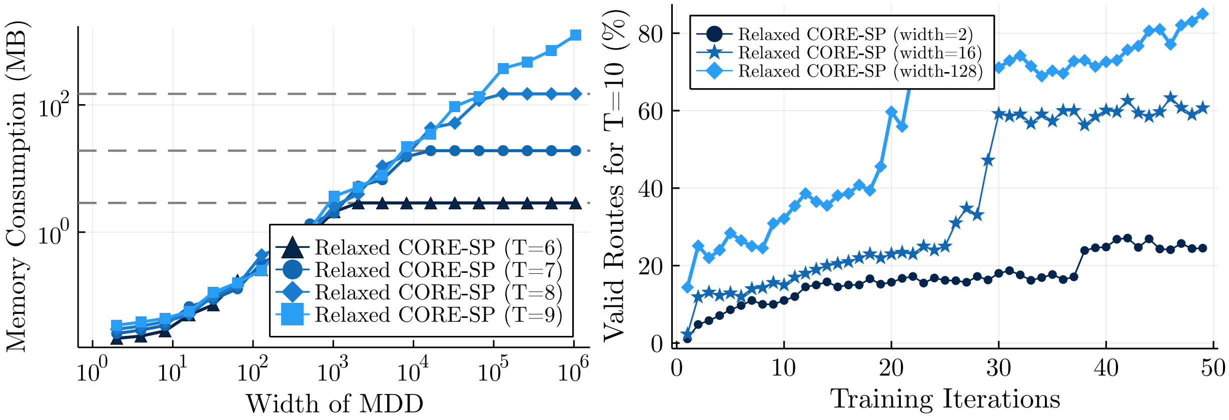


Figure 2.12. (Left) the memory usage of relaxed MDDs. **(Right)** The percentage of valid routes produced by Core-Sp using relaxed MDDs.

Route Reward Comparison. To evaluate the neural model's capability of learning the implicit preferences, we compare the reward function value of the routes generated from the structured prediction algorithms and the ground-truth routes. Notice that the ground truth routes are the ones that maximize the reward function $R(\cdot)$. We define the normalized reward in the following way:

$$\texttt{norm-reward} = \frac{1}{N}\sum_{i=1}^{N}\frac{R(\tilde{y})}{R(y)}, \tag{2.14}$$

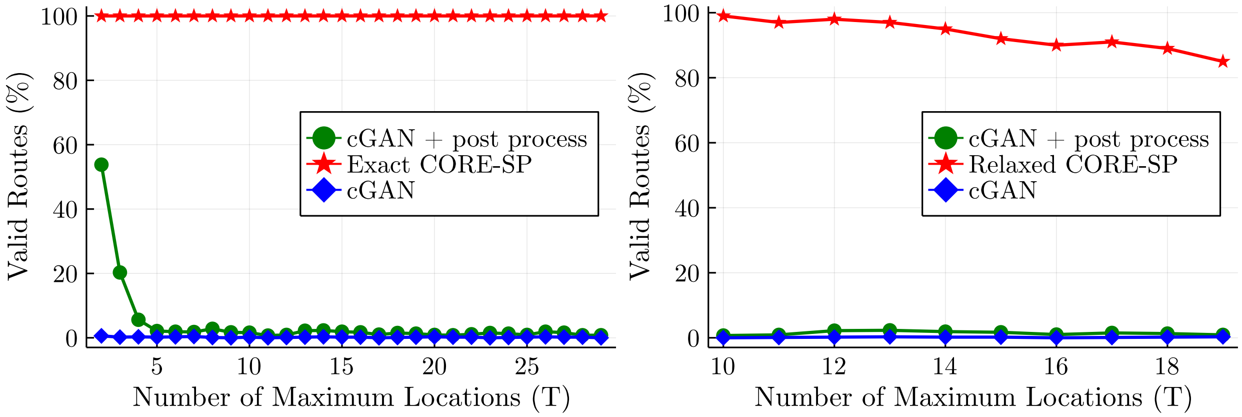


Figure 2.13. Our exact Core-Sp models outputs 100% valid routes in the vehicle service dispatching task, while competing approaches, namely conditional GAN (cGAN) and cGAN with post-processing cannot guarantee valid routes. Experiments are carried out with varying maximum number of locations in the daily requests. **(Left)** Exact MDDs are created by the iterative algorithm described in the main text. **(Right)** Relaxed MDDs are generated with max width 2^{20} to ensure that the memory consumption is less than 1 GB.

where routes $\tilde{y}$ are predicted from the machine learning algorithms and y are the ground-truth routes that maximize the hidden reward function. According to our definition, `norm-reward` cannot be greater than 1. The closer `norm-reward` is to 1, the better the generated routes satisfy the hidden preferences of the delivery man. When computing this metric, we only include valid routes, namely those satisfying the `all-diff` and `full-cover` constraints. N is the number of valid routes generated by one algorithm in the test data.

Figure 2.14 demonstrates the normalized rewards of the valid routes generated by Core-Sp and the baseline cGAN with post-processing. We can see that both methods can generate routes which have a normalized reward score of 0.6 to 1. Compared with the normalized reward value over the testing set, the routes generated by our Core-Sp module have more stable normalized rewards than the model with post-processing.

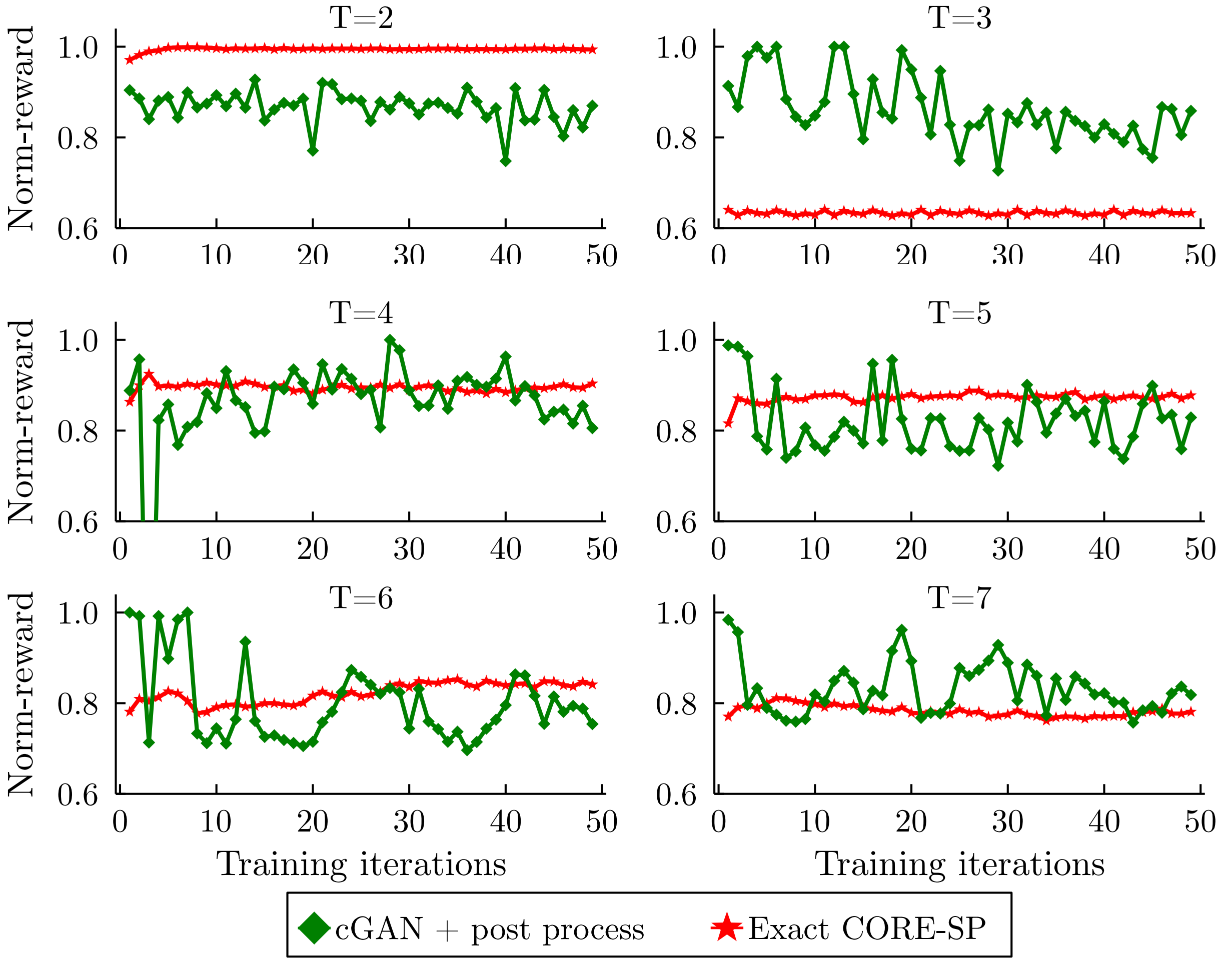


Figure 2.14. Comparing the normalized reward value of the model with exact CORE-SP and with post-processing methods. CORE-SP captures drivers' hidden preferences in the vehicle service dispatching problem. The hidden preferences are reflected by the normalized reward (see the main text for its definition). CORE-SP and cGAN with post-processing both achieve good normalized rewards.

2.6.2 If-Then Program Synthesis

Datasets and Metrics. The data sets of this experiment are crawled from IFTTT[5] and Zapier[6]. The statistics of these two data sets are shown in Table 2.1. The IFTTT data set contains more data samples than the Zapier data set, while the dimensions of the four

[5] ↑https://ifttt.com/

[6] ↑https://zapier.com/

Dataset	#train set	#val set	#test set	#quadruple	#vocabulary
IFTTT	66761	4148	2640	(111, 443, 88, 161)	4000
Zapier	24454	4809	2576	(1353, 1755, 1333, 1466)	3782

Table 2.1. The IFTTT and Zapier dataset statistics.

labels in the Zapier data set are several times larger than those of the IFTTT data set. The sentences in the data set are tokenized by Spacy[7] library.

To evaluate the performance of different models on this data set, we consider two metrics: the percentage of valid if-then programs and accuracy. A program is considered as valid if it satisfies our defined `Functionality` constraint. The accuracy metric is the percentage of predicted programs that match exactly in all four fields with those in the test set. This metric shows the percentage of correctly predicted programs.

Valid Programs Comparison. Core-Sp significantly boosts the percentage of valid programs generated. In this experiment, we start with evaluating the percentage of valid programs generated from the state-of-the-art Latent Attention model without the Core-Sp module. Then we apply the Core-Sp module from Algorithm 1, which iteratively relaxes the MDDs until we arrive at the exact MDD. The results in Figure 2.15 show the performance of all the relaxed and the exact Core-Sp modules when added to the Latent Attention model. Among all programs produced by the Latent Attention model without the Core-Sp layer, around 88% of them are valid on the two data sets. Once we enforce the exact Core-Sp capturing the `Functionality` constraint, all the programs (100%) produced are valid. We also study the effect of restricting the layer width of MDDs used in Core-Sp. We use Algorithm 1 to experiment Core-Sp with MDDs generated from width-2 to the largest width, which is width-111 for IFTTT and width-1353 for Zapier. The percentage of valid programs on a separate testing set is shown in the blue lines. The performance with the relaxed Core-Sp increases gradually with the increase of the MDD's width.

Accuracy Comparison. Figure 2.16 compares the training set and testing set accuracy for the state-of-the-art Latent Attention model and Core-Sp as the training progresses. We also collect the results of the Latent Attention model without Core-Sp, the model

[7] ↑https://spacy.io/api/tokenizer

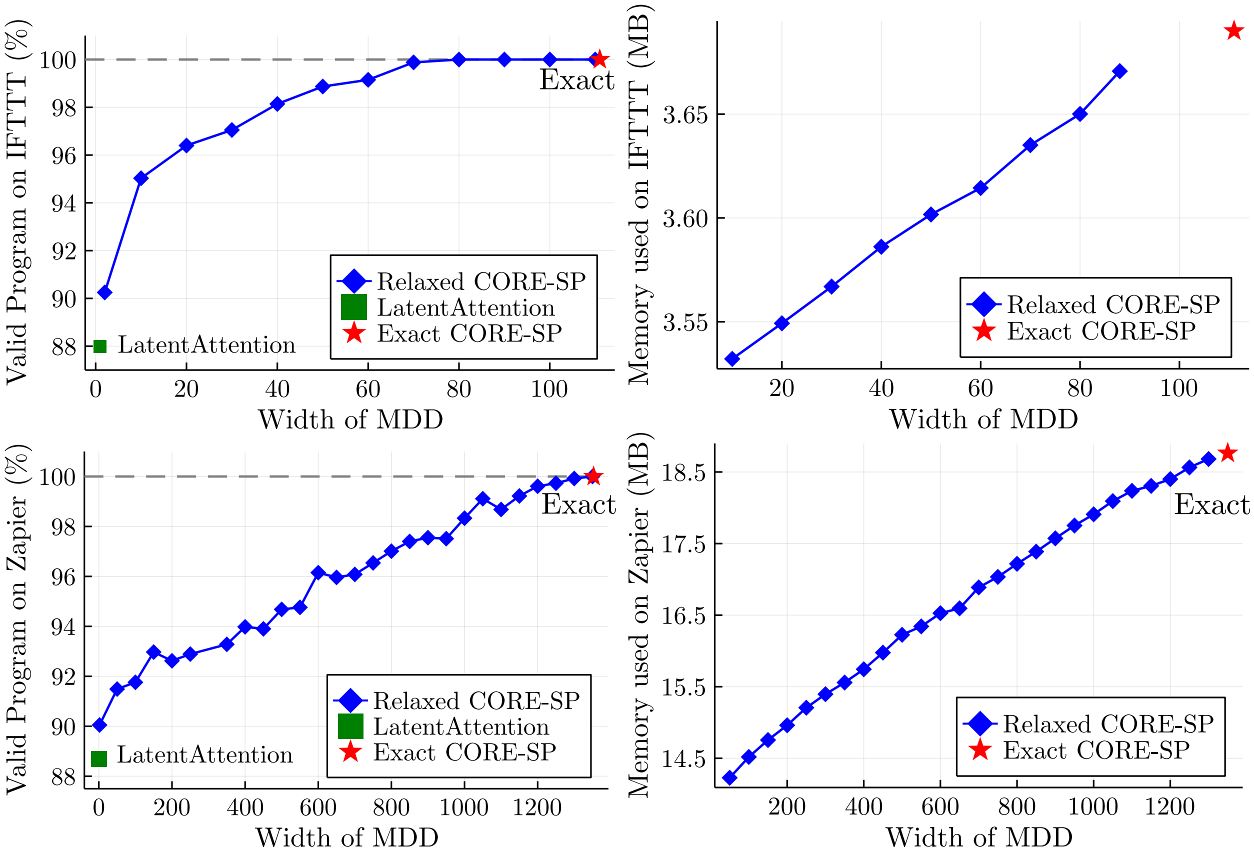


Figure 2.15. Percentage of valid programs and MDD memory consumption on IFTTT and Zapier datasets. Core-Sp outperforms the state-of-the-art approach *LatentAttention* ([42]) in generating valid if-then programs. The percentages of valid programs generated by Core-Sp embedding MDDs with different widths are shown for the IFTTT (top left) and Zapier (bottom left) datasets. Core-Sp model that embeds the exact MDD produces 100% valid programs on the two datasets. The relaxed and exact MDD for the IFTTT dataset takes less than 4 MB and for the Zaiper dataset takes less than 20 MB memory space.

with the best relaxed Core-Sp model (in terms of accuracy), and with the exact Core-Sp model on the two data sets in Table 2.2. The best relaxed Core-Sp model achieves $1-2\%$ higher accuracy than the Latent Attention model and still generates around 11% more valid programs than the Latent Attention model. Similarly, the model with the exact Core-Sp module improves approximately 1% in accuracy but generates 100% valid programs.

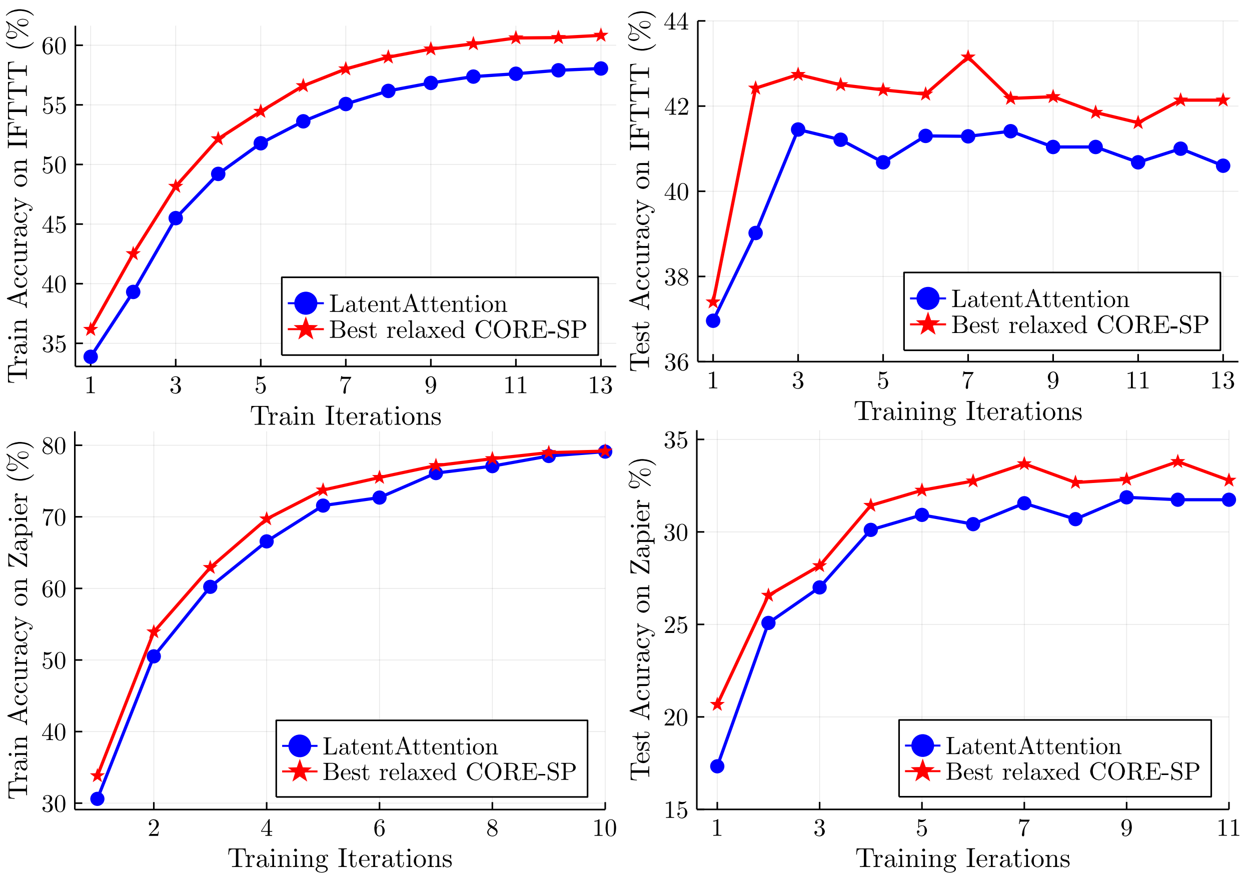


Figure 2.16. The Core-Sp module (red line) brings approximately $1-2\%$ increase in accuracy for the IFTTT data set and 2% increase for the Zapier data set for the if-then program synthesis task. The LatentAttention model (blue) is the previous state-of-the-art, which cannot guarantee the validity of the programs generated.

Table 2.2. The relaxed and exact Core-Sp modules boost the percentage of valid programs generated and the accuracy for the if-then program synthesis task on both the IFTTT and the Zapier data sets. Exact Core-Sp produces 100% valid programs while Core-Sp with the best relaxed MDD produced by the Algorithm 1 leads to the best accuracy in the prediction and close to 100% valid programs.

	IFTTT			Zapier		
Methods	Width	Accuracy	Valid (%)	Width	Accuracy	Valid (%)
LatentAttention	N/A	42.17%	87.51%	N/A	31.74%	88.00%
Best relaxed Core-Sp	80	**44.12%**	99.19%	1200	**34.28%**	99.53%
Exact Core-Sp	111	43.07%	**100%**	1353	32.83%	**100%**

2.6.3 SQL Query Generation from Natural Language

Dataset and Metrics. Experiments are conducted on the large-scale WikiSQL data set [70], which contains $80,654$ examples of questions and SQL queries distributed across $24,241$ tables from Wikipedia. We observe that most of the SQL queries are not complex. In this case, we further select queries within the dataset to form the moderate and the hard test set. The *moderate test set* is formed of those queries containing at least one conditional statement (i.e., "`cond-col cond-op cond-val`"). Similarly, the *hard test set* is composed of those queries that have at least two conditional statements.

The metrics applied for this task are: 1) percentage of valid SQL queries. We evaluate if the generated queries satisfy the `datatype` constraint. 2) execution accuracy. A generated query is considered correct if the returned value of executing the generated SQL query matches the returned value from the ground truth query. 3) logical accuracy, which evaluates the percentage of the generated queries which in every field match exactly the ground truth queries. The implementation is based on SQLNova. We use BERT-base [71] model as the word embedding. The entire model takes at most 3 days to train for 50 epochs. We choose the model that achieves the best execution accuracy on the validation data set for both the baseline and CORE-SP and calculate the corresponding statistics reflected in the table.

Valid SQL Queries Comparison. As shown in Table 2.3, SQLNova with CORE-SP module embedded generates 100% valid SQL programs, demonstrating 0.7% improvement over the original SQLNova model on the full testing set. On the moderate testing set, the improvement increased to 5.7%. On the most difficult hard testing set, the improvement becomes 16.3%. Due to the fact that a majority of the SQL queries in the full test set have `empty` value at `cond-op` and `=` value at `sel-op`, SQLNova has a high probability to predict prevalent labels in the data set and coincidentally satisfies the SQL grammar. This is the main reason that our relative improvement is not significant for the full test set.

Execution and Logical Accuracy. Figure 2.17 compares SQLNova and the **exact** CORE-SP model over execution and logical accuracy metrics as the training progresses. We also collect the accuracy of predicting each field in the SQL queries as shown in the table (top) of Table 2.3. The execution and logical accuracy are shown at the bottom of Table 2.3.

Table 2.3. Core-Sp outperforms the previous state-of-the-art SQLNova on three testing sets in SQL query generation. Core-Sp leads to 100% valid SQL queries generated and increases in both the execution accuracy and the logical accuracy compared with SQLNova for the Text2SQL generation task. The top table shows the accuracy of predicting each field in the SQL queries for both models.

Accuracy	Full test set		Moderate test set		Hard test set	
per component	SQLNova	Core-Sp	SQLNova	Core-Sp	SQLNova	Core-Sp
`sel-col`	96.3%	96.3%	96.4%	**97.0%**	96.6%	**97.7%**
`agg-op`	**89.8%**	89.7%	75.7%	**77.8%**	75.4%	**75.8%**
#`WHERE`	**98.1%**	97.9%	98.5%	**98.6%**	**98.9%**	98.5%
`cond-col`	93.6%	93.6%	**94.0%**	93.8%	93.6%	**93.7%**
`cond-op`	96.7%	**96.9%**	89.8%	**91.6%**	84.8%	**87.9%**
`where-val-idx`	94.5%	**94.8%**	89.4%	**92.3%**	86.7%	**87.5%**
`where-val`	94.7%	**94.9%**	89.3%	**92.2%**	86.4$	**87.1%**
	Full test set		Moderate test set		Hard test set	
Overall Accuracy	SQLNova	Core-Sp	SQLNova	Core-Sp	SQLNova	Core-Sp
Logical Accuracy	79.3%	**79.9%**	61.6%	**65.8%**	58.3%	**62.5%**
Execution Accuracy	85.5%	**86.1%**	75.4%	**79.1%**	76.1%	**78.0%**
Valid SQL	99.3%	**100.0%**	94.3%	**100%**	83.7%	**100%**

Core-Sp gains improvement for predicting `sel-col`, `cond-op`, `where-val-idx` and `where-val` components. For the rest components in the SQL queries, the difference in accuracy between Core-Sp and SQLNova is less than 0.4%. In terms of the execution accuracy, the exact Core-Sp is higher than SQLNova by 0.6%, 3.7%, and 1.9% on the full, moderate, and hard test sets, correspondingly. In terms of logical accuracy, the exact Core-Sp is higher than SQLNova by 0.6%, 4.2%, and 4.2% for the three testing sets. The improvement in the execution and logical accuracy is due to the fact that the Core-Sp module removes invalid operators during SQL generation and as a consequence reduces the modeling space.

2.7 Summary

In this chapter, we propose Core-Sp, an end-to-end neural module that embeds constraint reasoning into machine learning for structured prediction problems. Core-Sp represents the constraints using decision diagrams and filters out invalid solutions. Core-Sp

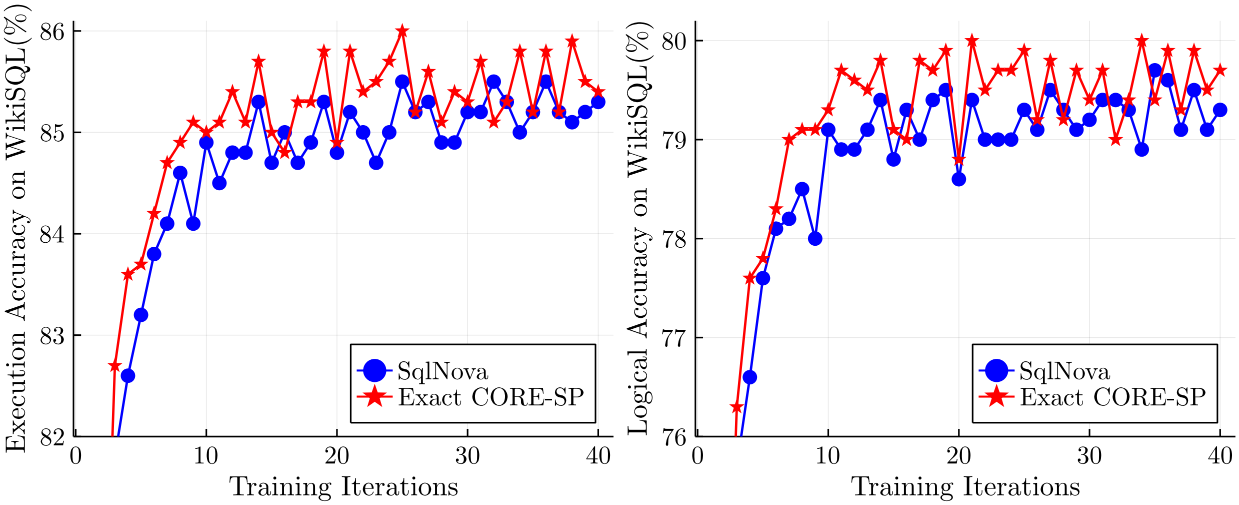


Figure 2.17. The execution accuracy and logical accuracy over training iterations for both Core-Sp and SQLNova. Core-Sp leads to higher execution and logical accuracy throughout training iterations.

is then embedded into a neural network which can be trained in an end-to-end fashion. We demonstrate the effectiveness of Core-Sp on three structured prediction applications including vehicle dispatching service planning, if-then program synthesis, and Text2SQL generation. We also propose an iterative search algorithm to find the optimal decision diagram structure for these applications. We show that the Core-Sp module improves constraint satisfaction in all three applications. In addition, Core-Sp reduces the modeling space. As a consequence, neural networks with Core-Sp embedded learn faster and generalize better than the pure neural network models. For future work, we plan to generalize Core-Sp in continuous domains and in reinforcement learning.

3. Learning Combinatorial Structures via Markov Random Fields with Sampling through Lovász Local Lemma

3.1 Introduction

In recent years, tremendous progress has been made in generative modeling [72–84]. Learning a generative model involves increasing the divergence in likelihood scores between the structures in the training set and those structures sampled from the current generative model distribution. While current approaches have achieved successes in *un-structured* domains such as vision or speech, their performance is degraded in the structured domain, because it is already computationally intractable to search for a valid structure in a combinatorial space subject to constraints, not to mention sampling, which has a higher complexity. In fact, when applied in a constrained domain, existing approaches spend most of their training time manipulating the likelihood of invalid structures, but not learning the difference between valid structures inside and outside of the training set. In the meantime, tremendous progress has been made in automated reasoning [61, 85–89]. Nevertheless, reasoning and learning have been growing independently for a long time. Only recently do ideas emerge exploring the role of reasoning in learning [30–32, 90–92].

The Lovász Local Lemma (LLL) [93] is a classic gem in combinatorics, which at a high level, states that there exists a positive probability that none of a series of *bad* events occur, as long as these events are *mostly* independent from one another and are not too likely individually. Recently, [94] came up with an algorithm, which samples from the probability distribution proven to exist by LLL. [95] proved that the algorithmic-LLL is an unbiased sampler if those bad events satisfy the so-called "extreme" condition. The expected running time of the sampler is also shown to be polynomial to the number of bad events. As one contribution of this chapter, we offer proofs of the two aforementioned results using precise mathematical notations, clarifying a few descriptions not precisely defined in the original proof. While this line of research clearly demonstrates the potential of LLL in generative learning (generating samples that satisfy all hard constraints), it is not clear how to embed LLL-based samplers into learning and no empirical studies have been performed to evaluate LLL-based samplers in machine learning.

In this chapter, we develop NEural Lovász Sampler (Nelson), which implements the LLL-based sampler as a fully differentiable neural network. Our Nelson-CD embeds Nelson into the contrastive divergence learning process of Markov Random Fields (MRFs). Embedding an LLL-based sampler allows the contrastive learning algorithm to focus on learning the difference between the training data and the valid structures drawn from the current model distribution. Baseline approaches, on the other hand, spend most of their training time learning to generate valid structures. In addition, Nelson is fully differentiable, hence allowing for efficient learning harnessing the parallelism of GPUs.

Related to our Nelson are neural-based approaches to solve combinatorial optimization problems [96–98]. Machine learning is also used to discover better heuristics [99, 100]. Reinforcement learning [101, 102] as well as approaches integrating search with neural networks [103] are found to be effective in solving combinatorial optimization problems as well. Regarding probabilistic inference, there are a rich line of research on MCMC-type sampling [104–106] and various versions of belief propagation [82, 107–109]. SampleSearch [110] integrates importance sampling with constraint-driven search. Probabilistic inference based on hashing and randomization obtains probabilistic guarantees for marginal queries and sampling via querying optimization oracles subject to randomized constraints [111–114].

We experiment Nelson-CD on learning preferences towards (i) random K-satisfiability solutions (ii) sink-free orientations of un-directed graphs and (iii) vehicle delivery routes. In all these applications, Nelson-CD (i) has the fastest training time due to seamless integration into the learning framework (shown in Tables 3.1(a), 3.3(a)). (ii) Nelson generates samples 100% satisfying constraints (shown in Tables 3.1(b), 3.3(b)), which facilitates effective contrastive divergence learning. Other baselines either cannot satisfy constraints or time out. (iii) The fast and valid sample generation allows Nelson to obtain the best learning performance (shown in Table 3.1(c), 3.2(a,b), 3.3(c,d)).

Our contributions can be summarized as follows: **(a)** We present Nelson-CD, a contrastive divergence learning algorithm for constrained MRFs driven by sampling through the Lovász Local Lemma (LLL). **(b)** Our LLL-based sampler (Nelson) is implemented as a fully differentiable multi-layer neural network module, allowing for end-to-end training on GPUs. **(c)** We offer a mathematically sound proof of the sample distribution and the expected running time of the Nelson algorithm. **(d)** Experimental results reveal the effectiveness of Nelson in (i) learning models with high likelihoods (ii) generating samples 100% satisfying constraints and (iii) having high time efficiency in training[1].

3.2 Preliminaries

Markov Random Fields (MRF) represent a Boltzmann distribution of the discrete variables $X = \{X_i\}_{i=1}^n$ over a Boolean hypercube $\mathcal{X} = \{0,1\}^n$. For $x \in \mathcal{X}$, we have:

$$P_\theta(X = x) = \frac{\exp\left(\phi_\theta(x)\right)}{Z(\theta)} = \frac{\exp\left(\sum_{j=1}^m \phi_{\theta,j}(x_j)\right)}{Z(\theta)}. \tag{3.1}$$

Here, $Z(\theta) = \sum_{x' \in \mathcal{X}} \exp\left(\phi_\theta(x')\right)$ is the partition function that normalizes the total probability to 1. The potential function is $\phi_\theta(x) = \sum_{j=1}^m \phi_{\theta,j}(x_j)$. Each $\phi_{\theta,j}$ is a *factor potential*, which maps a value assignment over a subset of variables $X_j \subseteq X$ to a real number. We use upper case letters, such as X_j to represent (a set of) random variables, and use lower case letters, such as x_j, to represent its value assignment. We also use $\texttt{var}(\phi_{\theta,j})$ to represent the domain of $\phi_{\theta,j}$, *i.e.*, $\texttt{var}(\phi_{\theta,j}) = X_j$. θ are the parameters to learn.

Constrained MRF is the MRF model subject to a set of hard constraints $\mathcal{C} = \{c_1, c_2, \ldots, c_L\}$. Here, each constraint c_j limits the value assignments of a subset of variables $\texttt{var}(c_j) \subseteq X$. We write $c_j(x) = 1$ if the assignment x satisfies the constraint c_j and 0 otherwise. Note that x is an assignment to all random variables, but c_j only depends on variables $\texttt{var}(c_j)$. We denote $C(x) = \prod_{j=1}^L c_j(x)$ as the indicator function. Clearly, $C(x) = 1$ if all constraints are satisfied and 0 otherwise. The constrained MRF is:

$$P_\theta(X = x | \mathcal{C}) = \frac{\exp\left(\phi_\theta(x)\right) C(x)}{Z_{\mathcal{C}}(\theta)}, \tag{3.2}$$

where the partition function $Z_{\mathcal{C}}(\theta) = \sum_{x' \in \mathcal{X}} \exp\left(\phi_\theta(x)\right) C(x)$ sums over only valid assignments.

Learn Constrained MRF Given a data set $\mathcal{D} = \{x^k\}_{k=1}^N$, where each x^k is a valid assignment that satisfies all constraints, learning can be achieved via maximal likelihood es-

[1] ↑Code is at: https://github.com/jiangnanhugo/nelson-cd

timation. In other words, we find the optimal parameters θ^* by minimizing the negative log-likelihood $\ell_{\mathcal{C}}(\theta)$:

$$\begin{aligned}\ell_{\mathcal{C}}(\theta) &= -\frac{1}{N}\sum_{k=1}^{N}\log P_\theta(X = x^k|\mathcal{C}) \\ &= -\frac{1}{N}\sum_{k=1}^{N}\phi_\theta(x^k) + \log Z_{\mathcal{C}}(\theta).\end{aligned} \tag{3.3}$$

The parameters θ can be trained using gradient-based optimization algorithm: $\theta^{t+1} = \theta^t - \eta\nabla\ell_{\mathcal{C}}(\theta)$, where η is the learning rate. Let $\nabla\ell_{\mathcal{C}}(\theta)$ denotes the gradient of the objective $\ell_{\mathcal{C}}(\theta)$, that is calculated as:

$$\begin{aligned}\nabla\ell_{\mathcal{C}}(\theta) &= -\frac{1}{N}\sum_{k=1}^{N}\nabla\phi_\theta(x^k) + \nabla\log Z_{\mathcal{C}}(\theta) \\ &= -\mathbb{E}_{x\sim\mathcal{D}}\left[\nabla\phi_\theta(x)\right] + \mathbb{E}_{\tilde{x}\sim P_\theta(x|\mathcal{C})}\left[\nabla\phi_\theta(\tilde{x})\right].\end{aligned} \tag{3.4}$$

The first term is the expectation over all data in training set $\mathcal{D}$. During training, this is approximated using a mini-batch of data randomly drawn from the training set $\mathcal{D}$. The second term is the expectation over the current model distribution $P_\theta(X = x|\mathcal{C})$ (detailed in Appendix C.2). Because learning is achieved following the directions given by the divergence of two expectations, this type of learning is commonly known as contrastive divergence (CD) [72]. Estimating the second expectation is the bottleneck of training because it is computationally intractable to sample from this distribution subject to combinatorial constraints. Our approach, Nelson, leverages the sampling through Lovász Local Lemma to approximate the second term.

Factor Potential in Single Variable Form Our method requires each factor potential $\phi_{\theta,j}(x_j)$ in Equation (3.1) to involve only one variable. This is NOT an issue as *all constrained MRF models can be re-written in single variable form* by introducing additional variables and constraints. Our transformation follows the idea in [89]. We illustrate the idea by transforming one-factor potential $\phi_{\theta,j}(x_j)$ into the single variable form. First, notice all functions including $\phi_{\theta,j}(x_j)$ over a Boolean hypercube $\{0,1\}^n$ have a (unique) discrete Fourier expansion:

$$\phi_{\theta,j}(x_j) = \sum_{S\in[\mathtt{var}(\phi_{\theta,j})]}\hat{\phi}_{\theta,j,S}\ \chi_S(x). \tag{3.5}$$

Here $\chi_S(x) = \prod_{X_i \in S} X_i$ is the basis function and $\hat{\phi}_{\theta,j,S}$ are Fourier coefficients. $[\mathtt{var}(\phi_{\theta,j})]$ denotes the power set of $\mathtt{var}(\phi_{\theta,j})$. For example, if $\mathtt{var}(\phi_{\theta,j}) = \{X_1, X_2\}$, then $[\mathtt{var}(\phi_{\theta,j})] = \{\emptyset, \{X_1\}, \{X_2\}, \{X_1, X_2\}\}$. See [115] for details of Fourier transformation. To transform $\phi_{\theta,j}(x_j)$ into single variable form, we introduce a new Boolean variable $\hat{\chi}_S$ for every $\chi_S(x)$. Because $\hat{\chi}_S$ and all X_i's are Boolean, we can use combinatorial constraints to guarantee $\hat{\chi}_S = \prod_{X_i \in S} X_i$. These constraints are incorporated into $\mathcal{C}$. Afterward, $\phi_{\theta,j}(x_j)$ is represented as the sum of several single-variable factors. Notice this transformation is only possible when the MRF is subject to constraints. We offer a detailed example in Appendix A.3.1 for further explanation. Equipped with this transformation, we assume all $\phi_{\theta,j}(x_j)$ are single variable factors for the rest of the chapter.

Extreme Condition The set of constraints $\mathcal{C}$ is called "extremal" if no variable assignment violates two constraints sharing variables, according to [95].

Condition 3.2.1. A set of constraints $\mathcal{C}$ is called extremal if and only if for each pair of constraints $c_i, c_j \in \mathcal{C}$, either of the following two cases should be met:

- their domain variables do not intersect, i.e., $\mathtt{var}(c_i) \cap \mathtt{var}(c_j) = \emptyset$.
- their domain variables intersect and both of them cannot be violated simultaneously for all $x \in \mathcal{X}$.

3.3 Sampling through Lovász Local Lemma

Lovász Local Lemma (LLL) [93] is a fundamental method in combinatorics to show the existence of a valid instance that avoids all the bad events, if the occurrences of these events are "mostly" independent and are not very likely to happen individually. Since the occurrence of a bad event is equivalent to the violation of a constraint, we can use the LLL-based sampler to sample from the space of constrained MRFs. To illustrate the idea of LLL-based sampling, we assume the constrained MRF model is given in the single variable form (as discussed in the previous section):

$$P_\theta(X = x|\mathcal{C}) = \frac{\exp\left(\sum_{i=1}^{n} \theta_i x_i\right) C(x)}{Z_{\mathcal{C}}(\theta)}, \tag{3.6}$$

Algorithm 2: Sampling Through Lovász Local Lemma.

```
Input: Random variables $X = \{X_i\}_{i=1}^n$; Constraints $\mathcal{C} = \{c_j\}_{j=1}^L$; Parameters of the
       constrained MRF $\theta$.
Output: Sample $x$ drawn from $P_\theta(X = x|\mathcal{C})$.
1 $x_i \sim \frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$, for $1 \leq i \leq n$ ;          // initialize
2 while $C(x) = 0$ do
3     Find all violated constraints $S \subseteq \mathcal{C}$ in $x$.
4     $x_k \sim \frac{\exp(\theta_k x_k)}{\sum\limits_{x_k \in \{0,1\}} \exp(\theta_k x_k)}$, for $x_k \in \mathtt{var}(S)$ ;          // resample
5 Return A valid sample $x$ drawn from $P_\theta(X = x|\mathcal{C})$.
```

where $Z_{\mathcal{C}}(\theta) = \sum_{x' \in \mathcal{X}} \exp\left(\sum_{i=1}^n \theta_i x_i\right) C(x)$.

As shown in Algorithm 2, the LLL-based sampler [95] takes the random variables $X = \{X_i\}_{i=1}^n$, the parameters of constrained MRF θ, and constraints $\mathcal{C} = \{c_j\}_{j=1}^L$ that satisfy Condition 3.2.1 as the inputs. In Line 1 of Algorithm 2, the sampler gives an initial random assignment of each variable following its marginal probability: $x_i \sim \frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$, for $1 \leq i \leq n$. Here we mean that x_i is chosen with probability mass $\frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$. Line 2 of Algorithm 2 checks if the current assignment satisfies all constraints in $\mathcal{C}$. If so, the algorithm terminates. Otherwise, the algorithm finds the set of violated constraints $S = \{c_j | c_j(x) = 0, c_j \in \mathcal{C}\}$ and re-samples related variables $X_k \in \mathtt{var}(S)$ using the same marginal probability, *i.e.*, $x_k \sim \frac{\exp(\theta_k x_k)}{\sum_{x_k \in \{0,1\}} \exp(\theta_k x_k)}$. Here $\mathtt{var}(S) = \cup_{c_j \in S} \mathtt{var}(c_j)$ is the set of variables that appeared in all violated clauses. The algorithm repeatedly samples all those random variables violating constraints until all the constraints are satisfied.

Under Condition 3.2.1, Algorithm 2 guarantees each sample is from the constrained MRFs' distribution $P_\theta(X = x|\mathcal{C})$ (in Theorem 3.3.1). In Appendix A.1, we present the detailed proof and clarify the difference to the original descriptive proof [95].

Theorem 3.3.1 (Probability Distribution). *Given random variables $X = \{X_i\}_{i=1}^n$, constraints $\mathcal{C} = \{c_j\}_{j=1}^L$ that satisfy Condition 3.2.1, and the parameters of the constrained MRF in the single variable form θ. Upon termination, Algorithm 2 outputs an assignment x that is randomly drawn from the constrained MRF distribution: $x \sim P_\theta(X = x|\mathcal{C})$.*

Sketch of Proof. We first show that in the last round, the probability of obtaining two possible assignments conditioning on all previous rounds in Algorithm 2 has the same ratio as the probability of those two assignments under distribution $P_\theta(X = x|\mathcal{C})$. Then we show when Algorithm 2 ends, the set of all possible outputs is equal to the domain of non-zero probabilities of $P_\theta(X = x|\mathcal{C})$. Thus we conclude the execution of Algorithm 2 produces a sample from $P_\theta(X = x|\mathcal{C})$ because of the identical domain and the match of probability ratios of any two valid assignments.

The expected running time of Algorithm 2 is determined by the number of rounds of re-sampling. In the uniform case that $\theta_1 = \ldots = \theta_n$, the running time is linear in the size of the constraints $\mathcal{O}(L)$. The running time for the weighted case has a closed form. We leave the details in Appendix A.1.

3.4 Neural Lovász Sampler

We propose Neural Lovász Sampler (Nelson) that implements LLL-based sampling as a neural network that draws multiple samples in parallel with the computing power of GPUs. We then demonstrate how Nelson is embedded in CD-based learning for constrained MRFs.

3.4.1 Neural Lovász Sampler (Nelson)

Represent Constraints as CNF Nelson obtains samples from the constrained MRF model in single variable form, as shown in Equation 3.6. To simplify notations, let $P_\theta(X_i = x_i) = \frac{\exp(\theta_i x_i)}{\sum_{x_i \in \{0,1\}} \exp(\theta_i x_i)}$. Since our constrained MRF model is defined on the Boolean hypercube $\{0,1\}^n$, we assume all constraints $\{c_j\}_{j=1}^L$ are given in the Conjunctive Normal Form (CNF). Note that all propositional logic can be reformulated in CNF format with at most a polynomial-size increase. A formula represented in CNF is a conjunction (noted as $\wedge$) of clauses. A clause is a disjunction (noted as $\vee$) of literals, and a literal is either a variable or its negation ($\neg$). Mathematically, we use c_j to denote a clause and use $l_{j,k}$ to denote a literal. In this case, a CNF formula would be:

$$c_1 \wedge \ldots \wedge c_L, \quad \text{where } c_j = l_{j,1} \vee \ldots \vee l_{j,K} \tag{3.7}$$

A clause is true if and only if at least one of the literals in the clause is true. The whole CNF is true if all clauses are true.

We transform each step of Algorithm 2 into arithmetic operations, hence encoding it as a multi-layer neural network. To do that, we first need to define a few notations:

- Vector of assignment $x^t = (x_1^t, \dots, x_n^t)$, where x_i^t is the assignment of variable X_i in the t-th round of Algorithm 2. $x_i^t = 1$ denotes variable X_i takes value 1 (or true).

- Vector of marginal probabilities $P = (P_1, \dots, P_n)$, where P_i is the probability of variable X_i taking value 0 (false): $P_i = P_\theta(X_i = 0) = \exp(0)/(\exp(0) + \exp(\theta_i))$.

- Tensor $W \in \{-1, 0, 1\}^{L \times K \times n}$ and matrix $b \in \{0, 1\}^{L \times n}$, that are used for checking constraint satisfaction:

$$W_{jki} = \begin{cases} 1 & \text{if } k\text{-th literal of clause } c_j \text{ is } X_i, \\ -1 & \text{if } k\text{-th literal of clause } c_j \text{ is } \neg X_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

$$b_{jk} = \begin{cases} 1 & \text{if } k\text{-th literal of clause } c_j \text{ is negated,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

- Matrix $V \in \{0, 1\}^{L \times n}$, denoting the mapping from clauses to variables in the CNF form for constraints $\mathcal{C}$:

$$V_{ji} = \begin{cases} 1 & \text{if clause } c_j \text{ contains a literal involving } X_i \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

- Vector of resampling indicators A^t, where $A_i^t = 1$ indicates variable X_i needs to be resampled at round t.

Given these defined variables, we represent each step of Algorithm 2 using arithmetic operations as follows:

Initialization To complete line 1 of Algorithm 2, given the marginal probability vector P, the first step is sampling an initial assignment of X, $x^1 = (x_1^1, \dots, x_n^1)$. It is accomplished by:

$$x_i^1 = \begin{cases} 1 & \text{if } u_i > P_i, \\ \\ 0 & \text{otherwise.} \end{cases} \qquad \text{for } 1 \leq i \leq n, \tag{3.11}$$

Here u_i is sampled from the uniform distribution in $[0, 1]$.

Check Constraint Satisfaction To complete line 2 of Algorithm 2, given an assignment x^t at round $t \geq 1$, tensor W and matrix b, we compute Z^t as follows:

$$Z^t = W \circledast x^t + b, \tag{3.12}$$

where $\circledast$ represents a special multiplication between tensor and vector: $(W \circledast x)_{jk} = \sum_{i=1}^n W_{jki} x_i^t$. Note that $Z_{jk}^t = 1$ indicates the k-th literal of j-th clause is true (takes value 1). Hence, we compute S_j^t as:

$$S_j^t = 1 - \max_{1 \leq k \leq K} Z_{jk}, \qquad \text{for } 1 \leq j \leq L. \tag{3.13}$$

Here $S_j^t = 1$ indicates x^t violates j-th clause. We check $\sum_{j=1}^L S_j^t \neq 0$ to see if any clause is violated, which corresponds to $C(x) = 0$ and is the continuation criteria of the while loop.

Extract Variables in Violated Clauses. To complete line 3 of Algorithm 2, we extract all the variables that require resampling based on vector S^t computed from the last step. The vector of resampling indicator A^t can be computed as:

$$A_i^t = \mathbf{1}\left(\sum_{j=1}^L S_j^t V_{ji} \geq 1\right), \qquad \text{for } 1 \leq i \leq n \tag{3.14}$$

where $\sum_{j=1}^L S_j^t V_{ji} \geq 1$ implies X_i requires resampling.

Resample To complete line 4 of Algorithm 2, given the marginal probability vector P, resample indicator vector A^t and assignment x^t, we draw a new random sample x^{t+1}. This can be done using this update rule:

$$x_i^{t+1} = \begin{cases} (1 - A_i^t)x_i^t + A_i^t & \text{if } u_i > P_i, \\ (1 - A_i^t)x_i^t & \text{otherwise.} \end{cases} \qquad \text{for } 1 \leq i \leq n \tag{3.15}$$

Again, u_i is drawn from the uniform distribution in $[0, 1]$. Drawing multiple assignments in parallel is attained by extending x^t with a new dimension (See implementation in Appendix A.4.1). Example 3.4.1 shows the detailed steps of Nelson (See more examples in Appendix A.1.5).

Example 3.4.1. Assume we have random variables X_1, X_2, X_3 with $n = 3$, Constraints $\mathcal{C} = (X_1 \vee X_2) \wedge (\neg X_1 \vee X_3)$ in the CNF form with $L = 2, K = 2$. Tensor W is:

$$W = \begin{bmatrix} w_{11} = [w_{111}, w_{112}, w_{113}], & w_{12} = [w_{121}, w_{122}, w_{123}] \\ w_{21} = [w_{211}, w_{212}, w_{213}], & w_{22} = [w_{221}, w_{222}, w_{223}] \end{bmatrix},$$

$$w_{11} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, w_{12} = \begin{bmatrix} 0, 1, 0 \end{bmatrix}, w_{21} = \begin{bmatrix} -1, 0, 0 \end{bmatrix}, w_{22} = \begin{bmatrix} 0, 0, 1 \end{bmatrix}.$$

Note that $w_{111} = 1$ means X_1 is the 1st literal in the 1st clause and $w_{211} = -1$ means $\neg X_1$ is the 1st literal in the 2nd clause. Matrix b and the mapping matrix V are:

$$b = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},$$

$b_{21} = 1$ indicates the 1st literal in the 2nd clause is negated. For the mapping matrix, $V_{11} = V_{12} = 1$ implies the 1st clause contains X_1 and X_2. For $t = 1$, suppose we have an initialized assignment $x^1 = [0\ 0\ 1]^\top$, meaning $X_1 = X_2 = 0, X_3 = 1$. The intermediate results of Z^1, S^1, A^1 become:

$$Z^1 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \quad S^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad A^1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},$$

where $S_1^1 = 1$ implies the 1st clause is violated. $A_1^1 = A_2^1 = 1$ denotes variables X_1, X_2 require resampling.

Algorithm 3: Learn Constrained MRFs via Nelson-CD.

```
Input: Training set $\mathcal{D}$; constraints $\mathcal{C}$; parameters $\theta$ of constrained MRF; #samples
       $m$; epochs $T$; learning rate $\eta$.
1 Build $(W, b, V)$ from $\mathcal{C}$ as in Equations. (3.8), (3.9), (3.10);
2 for $t = 1$ to $T$ do
3     $\{\tilde{x}^j\}_{j=1}^m \leftarrow$ NELSON$(W, b, V, \theta^t, m)$ ;       // Sample from $P_{\theta^t}(X = x|\mathcal{C})$
4     $\{x^j\}_{j=1}^m \sim \mathcal{D}$ ;                                      // Sample from training data
5     $g^t = \frac{1}{m}\sum_{j=1}^m \nabla\phi(x^j) - \frac{1}{m}\sum_{j=1}^m \nabla\phi(\tilde{x}^j)$;   // Compute divergence
6     $\theta^{t+1} \leftarrow \theta^t - \eta g^t$ ;                                    // Update parameters
7 return The converged model $\theta^T$.
```

3.4.2 Contrastive Divergence-based Learning

The whole learning procedure is shown in Algorithm 3. At every learning iteration, we call Nelson to draw assignments $\{\tilde{x}^j\}_{j=1}^m$ from constrained MRF's distribution $P_\theta(X|\mathcal{C})$. Then we pick m data points at random from the training set $\{x^j\}_{j=1}^m \sim \mathcal{D}$. The divergence g^t in line 5 of Algorithm 3 is an estimation of $\nabla \ell_{\mathcal{C}}(\theta)$ in Equation (3.4). Afterward, the MRFs' parameters are updated, according to line 6 of Algorithm 3. After $T_{\max}$ learning iterations, the algorithm outputs the constrained MRF model with parameters $\theta^{T_{\max}}$.

3.5 Related Work

Results related to the proposed Nelson method for solving combinatorial optimization problems are mainly neural-based approaches [96–98]. Machine learning is also used to discover better heuristics for finding solutions [99, 100]. Reinforcement learning [101, 102] as well as approaches integrating search with neural nets [103] are found to be effective in solving combinatorial optimization problems as well.

Regarding probabilistic inference, there is a rich line of research on MCMC-type sampling [104–106] and various versions of belief propagation [107–109]. SampleSearch [110] integrates importance sampling with constraint-driven search. Probabilistic inference based on hashing and randomization obtains probabilistic guarantees for marginal queries and sampling via querying optimization oracles subject to randomized constraints [111, 112, 114].

3.6 Experiments

We show the efficiency of the proposed Nelson algorithm on learning MRFs defined on the solutions of three combinatorial problems. In all three application domains, we demonstrate that Nelson outperforms baselines on learning performance, i.e., generating structures with high likelihoods and MAP@10 scores (Table 3.2(a,b), 3.3(c,d), 3.1(c)). Nelson also generates samples which 100% satisfy constraints (Tables 3.1(b), 3.3(b)). Finally, Nelson is the most efficient sampler. Baselines either time out or cannot generate valid structures (Tables 3.1(a), 3.3(a)).

3.6.1 Learn Random K-SAT Solutions with Preference

Table 3.1 shows the proposed Nelson is an efficient sampler that generates valid assignments. Equipped with Nelson sampler, Table 3.2 demonstrates that the constrained MRF learns to generate K-SAT solutions that closely resemble those in the training set.

Task Definition. The task is to learn to generate solutions to a K-SAT problem. We are given a training set $\mathcal{D}$ containing solutions to a corresponding CNF formula $c_1 \wedge \ldots \wedge c_L$. Note not all solutions are equally likely presented in $\mathcal{D}$. The *learning* task is to maximize the log-likelihood of the assignments seen in the training set $\mathcal{D}$. Once learning is completed, the *inference* task is to generate valid solutions that closely resemble those in $\mathcal{D}$ [116].

Dataset. We consider several benchmark datasets of different problem sizes characterized by the number of variables generated by CNFGen [117]. To generate the training set $\mathcal{D}$, we use the Glucose4 solver from PySAT to sample solutions.

Baselines We compare Nelson with other contrastive divergence learning algorithms equipped with other sampling approaches. In terms of baseline samplers, we consider:

- Gibbs sampler [118], a special case of MCMC that is widely used in training MRF models.
- Weighted SAT samplers, including WAPS [119], WeightGen [120] and XOR sampler [121, 122].
- Uniform SAT samplers, including CMSGen [123], QuickSampler [124], UniGen [125] and KUS [126]. Notice these samplers cannot sample SAT solutions from a non-uniform distribution. We include them in the learning experiments as a comparison and exclude them in the weighted sampling experiment (Figure 3.2).

Currently, there are only GPU-based SAT solvers [127] and model counter [128], GPU-based SAT sampler is not available by far.

Metrics. In terms of evaluation metrics, we consider

- Training time per epoch: the average time for every learning method to finish one epoch.
- Validness of Variables Assignments: the percentage of generated solutions that satisfy the given SAT formula.
- Mean averaged Precision (MAP@10): the percentage that the solutions in the training set $\mathcal{D}$ reside among the top-10 most likely models. The higher the MAP@10 scores, the better the model generates structures closely resembling those in the training set.
- log-likelihood of the solutions in the training set $\mathcal{D}$: The model that attains the highest log-likelihood learns the closest distribution to the training set.

Table 3.1. Sampling efficiency and accuracy for learning K-SAT solutions with preferences. The proposed Nelson is the most efficient (see Training Time Per Epoch) and always generates valid assignments (see Validness) with a small approximation error (see Approximation Error of Gradient) against all baselines. T.O. means time out. and QS stands for the Quicksampler.

Problem	(a) Training Time Per Epoch (Mins) (↓)								
size	Nelson	XOR	WAPS	WeightGen	CMSGen	KUS	QS	Unigen	Gibbs
10	**0.13**	26.30	1.75	0.64	0.22	0.72	0.40	0.66	0.86
20	**0.15**	134.50	3.04	T.O.	0.26	0.90	0.30	2.12	1.72
30	**0.19**	1102.95	6.62	T.O.	0.28	2.24	0.32	4.72	2.77
40	**0.23**	T.O.	33.70	T.O.	0.31	19.77	0.39	9.38	3.93
50	**0.24**	T.O.	909.18	T.O.	0.33	1532.22	0.37	13.29	5.27
500	**5.99**	T.O.	T.O.	T.O.	34.17	T.O.	T.O.	T.O.	221.83
1000	**34.01**	T.O.	T.O.	T.O.	177.39	T.O.	T.O.	T.O.	854.59
	(b) Validness of Variables Assignments (%) (↑)								
10 − 50	**100**	**100**	**100**	**100**	**100**	**100**	82.65	**100**	90.58
500	**100**	T.O.	T.O.	T.O.	**100**	T.O.	7.42	**100**	54.27
1000	**100**	T.O.	T.O.	T.O.	**100**	T.O.	0.00	**100**	33.91
	(c) Approximation Error of Gradient (↓)								
10	**0.10**	0.21	0.12	3.58	3.96	4.08	3.93	4.16	0.69
12	**0.14**	0.19	0.16	5.58	5.50	5.49	5.55	5.48	0.75
14	**0.15**	0.25	0.19	T.O.	6.55	6.24	7.79	6.34	1.30
16	0.16	0.25	**0.15**	T.O.	9.08	9.05	9.35	9.03	1.67
18	**0.18**	0.30	0.23	T.O.	10.44	10.30	11.73	10.20	1.90

- Approximation Error of Gradient: the L_1 distance between the exact gradient of $\log Z_{\mathcal{C}}(\theta)$ in Equation (3.4) and the approximations given by the sampler.

Please refer to Section A.4 for detailed experiment settings: hyper-parameter settings, implementation of the Nelson algorithm, and detailed definitions of all the metrics.

Efficiency and accuracy benchmark of all samplers. In Table 3.1, we compare all available samplers in terms of sampling efficiency for learning constrained MRF, approximation error for the gradient, and validness of the generated assignments. In Table 3.1(a), Nelson takes much less time for sampling against all the samplers and can train the model with the dataset of problem size 1000 within an hour. In Table 3.1(b), Nelson always generate valid samples. QuickSampler and Gibbs methods' performance decreases when the problem size becomes larger. In Table 3.1(c), Nelson, XOR, and WAPS are the three algorithms that

Table 3.2. The quality of learning outcomes for learning random K-SAT solutions with preferences. Nelson achieves the best likelihood and MAP@10 scores. T.O. is time out.

	(a) Log-likelihood (↑)			
Problem size	Nelson (ours)	Gibbs	CMSGen	Quicksampler, WeightGen, KUS, XOR, WAPS
100	−49.16	**−36.36**	−60.12	T.O.
300	**−52.61**	−53.11	−128.39	
500	**−196.47**	−197.21	−272.49	
700	**−238.60**	−238.75	−389.44	
1000	**−294.22**	−296.33	−532.85	
	(b) MAP@10 (%) (↑)			
100	82.13	83.32	**86.34**	T.O.
300	**66.37**	64.42	64.50	
500	**90.03**	73.14	70.67	
700	**69.74**	**69.74**	48.10	
1000	**91.70**	77.56	78.72	

can effectively estimate the gradient while the other algorithms incur huge estimation errors. Also, the other two methods are much slower than Nelson.

We also evaluated the samplers' efficiency in isolation (not embedded in learning). The sampling cases we considered are uniform and weighted (mainly following the experiment setting in [129]). In weighted sampling, the weights are specified by fixed values to the single factors in Equation (3.6). In the uniform sampling case in Figure 3.1, Nelson and Quicksampler require much less time to draw samples compared to other approaches. However, the solutions generated by Quicksampler rarely satisfy constraints. In the weighted sampling case in Figure 3.2, Nelson scales better than all the competing samplers as the sizes of the K-SAT problems increase.

Learning Quality Comparisons. In Table 3.2, we compare the learned model by measuring the log-likelihood and MAP@10 scores. Note that baselines including Quicksampler, Weightgen, KUS, XOR, and WAPS timed out for the problem sizes we considered. Compared with the remaining algorithms, Nelson attains the best log-likelihood and MAP@10 metric.

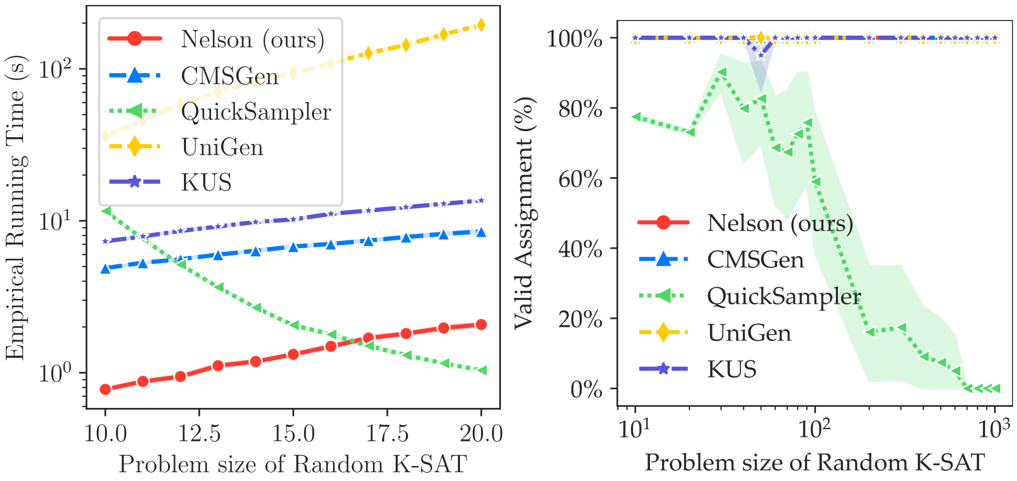


Figure 3.1. Empirical running time and the percentage of valid structures sampled uniformly at random from solutions of K-SAT problems. Nelson always generates valid solutions and is the most efficient sampler.

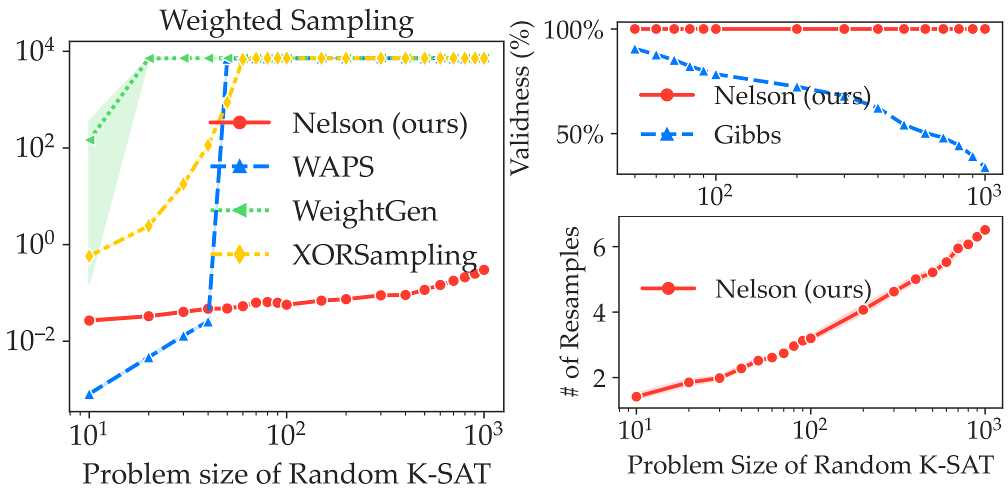


Figure 3.2. Empirical running time, the percentage of valid solutions generated, and the number of resample steps for weighted sample generation of K-SAT solutions. Nelson scales the best among all approaches and always generates valid solutions.

3.6.2 Sink-Free Orientation in Undirected Graphs

Task Definition & Dataset A *sink-free* orientation of an undirected graph is a choice of orientation for each arc such that every vertex has at least one outgoing arc [130]. This task has wide applications in robotics routing and IoT network configuration [131]. Even though finding a sink-free orientation is tractable, sampling a sink-free orientation from the

Table 3.3. Sample efficiency and learning performance of the sink-free orientation task. Nelson is the most efficient (see Training Time Per Epoch) and always generates valid assignments (see Validness), has the smallest error approximating gradients, and has the best learning performance (see MAP@10) among all approaches.

	(a) Training Time Per Epoch (Mins) (↓)		
Problem size	Nelson (ours)	GibbsSampler	CMSGen
10	**0.53**	9.85	0.69
20	**0.53**	80.12	1.93
30	**0.72**	256.38	3.65
40	**0.93**	777.01	5.99
50	**1.17**	T.O.	9.08
60	**1.40**	T.O.	13.19
70	**1.65**	T.O.	17.90
80	**1.93**	T.O.	23.98
90	**2.18**	T.O.	29.49
100	**2.45**	T.O.	38.04
	(b) Validness of Predicted Variables Assignments (%) (↑)		
7	**100**	50.16	**100**
8	**100**	64.63	**100**
9	**100**	47.20	**100**
10	**100**	62.60	**100**
11	**100**	84.95	**100**
12	**100**	85.24	**100**
13	**100**	87.28	**100**
14	**100**	90.51	**100**
15	**100**	93.27	**100**
	(c) Approximation Error of Gradient (↓)		
5	**0.01**	0.09	0.21
6	**0.06**	0.08	3.26
7	**0.05**	0.08	2.37
8	**0.03**	0.11	2.37
9	**0.04**	0.17	8.62
10	**0.05**	0.28	11.27
	(d) MAP@10 (%) (↑)		
10	61.14	60.01	**64.56**
20	**55.26**	55.20	47.79
30	**100.00**	96.29	**100.00**
40	**40.01**	39.88	38.90
50	**46.12**	T.O.	42.11

space of all orientations is still #P-hard. Given a training set of preferred orientations $\mathcal{D}$ for the graph, the *learning* task is to maximize the log-likelihood of the orientations seen in the training set. The *inference* task is to generate valid orientations that resemble those in the training set. To generate the training set, we use the Erdős-Rényi random graph from the NetworkX[2] library. The problem size is characterized by the number of vertices in the graph. The baselines we consider are CD-based learning with Gibbs sampling and CMSGen.
Learning Quality In Table 3.3(a), we show the proposed Nelson method takes much less time to train MRF for one epoch than the competing approaches. Furthermore, in Table 3.3(b), Nelson and CMSGen generate 100% valid orientations of the graph while the Gibbs-based model does not. Note the constraints for this task satisfy Condition 3.2.1, hence Nelson sampler's performance is guaranteed by Theorem 3.3.1. In Table 3.3(c), Nelson attains the smallest approximation error for the gradient (in Equation 3.4) compared to baselines. Finally, Nelson learns a higher MAP@10 than CMSGen. The Gibbs-based approach times out for problem sizes larger than 40. In summary, our Nelson is the best-performing algorithm for this task.

3.6.3 Learn Vehicle Delivery Routes

Task Definition & Dataset Given a set of locations to visit, the task is to generate a sequence to visit these locations in which each location is visited once and only once and the sequence closely resembles the trend presented in the training data. The training data are such routes collected in the past. The dataset is constructed from TSPLIB, which consists of 29 cities in Bavaria, Germany. The constraints for this problem do not satisfy Condition 3.2.1. We still apply the proposed method to evaluate if the Nelson algorithm can handle those general hard constraints.

In Figure 3.3, we see Nelson can obtain samples of this delivery problem efficiently. We measure the number of resamples taken as well as the corresponding time used by the Nelson method. Nelson takes roughly 50 times of resamples with an average time of 0.3 seconds to draw a batch (the batch size is 100) of valid visiting sequences.

[2] ↑https://networkx.org/

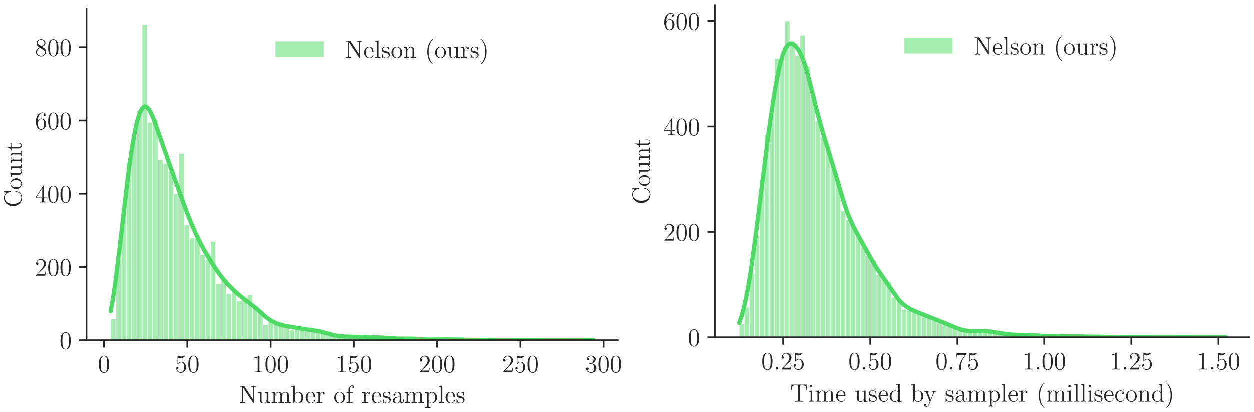


Figure 3.3. Frequency histograms for the number of resample and the total time of NELSON method for uniformly sampling visiting paths for vehicle routing problem.

3.7 Summary

In this chapter, we present NELSON, which embeds a sampler based on Lovász Local Lemma into the contrastive divergence learning of Markov random fields. The embedding is fully differentiable. This approach allows us to learn generative models over constrained domains, which presents significant challenges to other state-of-the-art models. We also give sound proofs of the performance of the LLL-based sampler. Experimental results on several real-world domains reveal that NELSON learns to generate 100% valid structures, while baselines either time out or cannot generate valid structures. NELSON also outperforms other approaches in the running times and in various learning metrics.

4. Controllable Language Generation via Combinatorial Constraint Satisfaction: A Tree Search Enhanced Monte-Carlo Approach

4.1 Introduction

Supervised techniques still dominate in natural language generation tasks. Despite its success, supervised approaches need to be trained with massive datasets of input-output pairs, which is non-trivial to acquire. In addition, it is hard to guarantee that the output sentences satisfy constraints. Recent approaches first pre-train a language model on a general-purpose dataset, then fine-tune the neural net on a task-specific dataset [132, 133]. These approaches partially mitigate data hunger in training large neural networks. Nevertheless, they still require carefully crafted datasets for fine-tuning[1].

We present a combinatorial constraint satisfaction approach for language generation. In particular, we sample sentences that attain high likelihoods from a language model and satisfy task-specific constraints. Sampling sentences that attain high likelihood in the language model ensures the sentence quality. Constraints guarantee that the sentences fit the specific language task. The constraints can be hard ones such as the grammar rules, or soft ones such as attaining positive sentiment scores.

Our method harnesses constraint satisfaction, rather than learning, to guide language generation. In fact, there is no task-specific training in our approach. Our approach is highly flexible since constraints can be switched quickly to be adapted to a different task, even faster than fine-tuning. It also allows us to leverage the latest developments in automated reasoning for language generation. Although the field of language generation is dominated by learning, reasoning should play an equally important role. Human beings can write beautiful words from reasoning over what is needed in the specific writing task, without learning from previous examples.

[1] ↑I collaborate with Maosen Zhang as the second author for this work. This chapter contains sections taken from [2], which Maosen has never used for use of any degree. I've rewritten this chapter and highlighted my individual contributions.

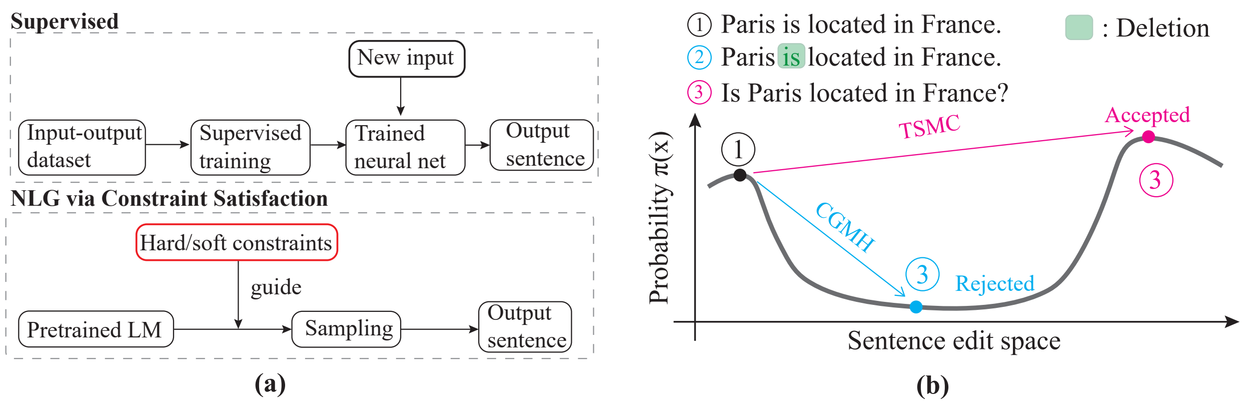


Figure 4.1. **(a)** Language generation via supervised method and constraint satisfaction. **(b)** Our TSMH traverses the probabilistic space of high-quality sentences more effectively than the baseline CGMH. “R, I, D” means replace, insert, and delete operations.

To better handle the combinatorial constraints, a tree search is embedded into the proposal process of the Markov chain Monte Carlo (MCMC) for constrained language generation, which suggests candidate proposals that satisfy more constraints. Our approach is motivated by Sample-Search [110, 134, 135], which integrates backtrack search into importance sampling. Making multiple word-level changes within one proposal step of MCMC allows the direct transition between legitimate sentences, while previous approaches must go through infeasible intermediate states. Such moves are typically rejected by MCMC and therefore result in a slow mixing rate (See Figure 4.1(b)).

In literature, constrained language generation has been attacked in a supervised way in [136–140]. There are also various works which model language rules as decomposed tree structures [141] or sentiment tags [142]. Markov Logic network [143, 144] is also used to formulate grammar rules. The Euclidean distance in semantic space is considered as soft constraints in [48, 145, 146].

To summarize, our contributions are: 1) We define the problem of constraint satisfaction-driven natural language generation, and propose a sampling-based approach to tackle the problem with combinatorial constraints. 2) We propose a **T**ree **S**earch enhanced **M**etropolis **H**astings (TSMH) framework, which mixes faster than standard MCMC in the presence of combinatorial constraints. 3) Experiment results on generating interrogative, imperative sentences with keywords, and sentences with given sentiments demonstrate that our TSMH is able to generate sentences that satisfy more hard and soft constraints as well as retain good fluency.

4.2 Language Generation via Combinatorial Constraint Satisfaction

We provide a general framework for the constrained natural language generation. In this framework, sentences are generated by sampling from a probability distribution that is proportional to the score of a pre-trained language model times the constraint score. Formally, let $\pi(x)$ be the probability that sentence x is sampled, it should be propositional to:

$$\pi(x) \propto P_{\text{LM}}(x) \cdot \text{Constraint}(x). \tag{4.1}$$

Here, $P_{\text{LM}}(x)$ is the score of a language model [133, 147], which measures the quality of sentence x. Higher $P_{\text{LM}}(x)$ means the sentence x is better in quality. Constraint(x) is a task-specific penalty term, which are composed of hard and soft constraint terms:

$$\text{Constraint}(x) = \Phi_{\text{hard}}(x) \cdot \Phi_{\text{soft}}(x). \tag{4.2}$$

Both the hard constraint score $\Phi_{\text{hard}}(x)$ and the soft constraint score $\Phi_{\text{soft}}(x)$ are float values ranging from 0 to 1. The closer to 1, the more satisfied the constraints are.

Unlike supervised methods which require training with massive data, our framework can solve language generation tasks with no task-specific training. $P_{\text{LM}}(x)$ comes from a general language model, only trained on general-purpose language tasks. There is no fine-tuning of $P_{\text{LM}}(x)$ on the specific task. $\Phi_{\text{hard}}(x)$ is based on crafted hard constraints. $\Phi_{\text{soft}}(x)$ comes from either user-defined functions or pre-trained neural networks, which again are not fine-tuned on the specific task. The overall formulation is composed of the language model and the task-specific constraints. It allows us to sample sentences which are close to natural language while satisfying constraints.

4.2.1 Constraint Formulation

Hard Constraints. The hard constraint score of sentence x is computed as: $\Phi_{\text{hard}}(x) = \beta^{M-\sum_i c_i(x)}$, where $\beta \in [0, 1]$. $c_i(x)$ is an indicator variable that takes 1 if the sentence x satisfies the i-th constraint, and M is the total number of hard constraints. We use propositional logic to define $c_i(x)$. Given a sentence x with length m, let $w_i^V \in \{1, 0\}$ be an indicator variable that the i-th word in the sentence is in category V.

For example, given a keyword `K`, we can enforce its existence in the sentence by: $c(x) = w_1^{\texttt{[K]}} \vee w_2^{\texttt{[K]}} \cdots \vee w_m^{\texttt{[K]}}$. Here `[K]` is a set containing the keyword `K`.

Furthermore, we enforce the sentence type to be imperative by: $c(x) = w_1^{\texttt{[VERB]}} \vee (w_1^{\texttt{[ADV]}} \wedge w_2^{\texttt{[VERB]}})$ where the first word in the sentence should be either a verb: $w_1^{\texttt{[VERB]}}$ or an adverb followed by a verb: $w_1^{\texttt{[ADV]}} \wedge w_2^{\texttt{[VERB]}}$. The `[VERB]` and `[ADV]` represent the set of verbs and adverbs accordingly.

After defining every hard constraint, to efficiently evaluate if the sentence preserves all the constraints, we use **template** to represent a set of sentences where each word is either given or specified by a word category. We use the number of hard constraints a sentence satisfies at the template level to reduce the search tree size. For example, a template `[[K],[AUX],[OTH],[OTH]]` represent a series of sentences that the first word is the keyword `K`, the second word is an auxiliary verb and the last two words are the other words.

Soft Constraints. A soft constraint assigns a float value between 0 and 1 to indicate the constraint satisfaction degree. Soft constraint $\Phi_{\text{soft}}(x)$ can be derived quite flexibly, either from a user-defined function or a pre-trained neural network. For example, to ensure two sentences are semantically similar, the soft constraint can be the cosine similarity of their sentence vectors. Furthermore, to ensure the sentences generated with specific sentiment, the soft constraint can be the score of a sentiment analysis neural network, representing whether the sentence has the requested sentiment.

4.3 Tree Search Enhanced MCMC

Markov chain Monte Carlo (MCMC) is a classical approach to sample sentences from probability distribution $\pi(x)$ as defined in Equation (4.1). Starting from one sentence x,

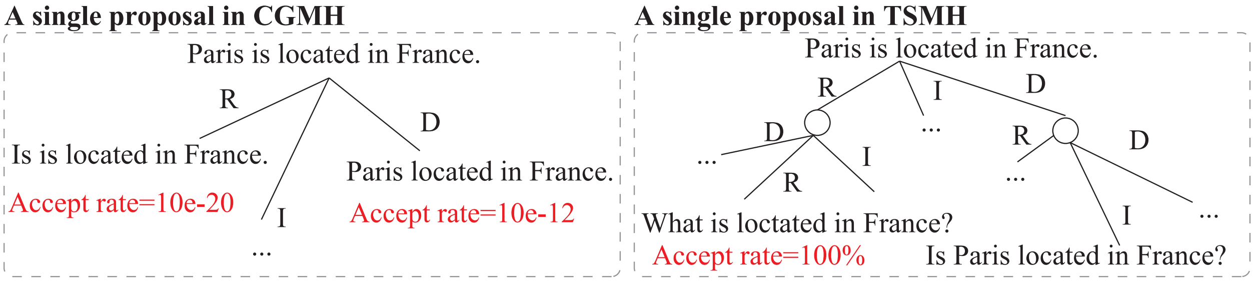

A single proposal in CGMH
Paris is located in France.
R
D
I
Is is located in France.
Accept rate=10e-20
Paris located in France.
Accept rate=10e-12
...
A single proposal in TSMH
Paris is located in France.
R
I
D
...
D
I
R
...
R
D
I
...
...
What is loctated in France?
Accept rate=100%
Is Paris located in France?

Figure 4.2. Our TSMH method significantly outperforms CGMH in terms of acceptance rate, in generating sentences with combinatorial constraints. **(Left)** CGMH must pass intermediate sentence states, which have very low acceptance rate to reach the intermediate sentence "*Is Paris located in France?*" starting from sentence "*Paris is located in France*". This results in the poor performance of CGMH when handling combinatorial constraints. **(Right)** By embedding a tree search into MCMC, TSMH can reach the an intermediate sentence from the starting sentence in one step, and with an acceptance rate of 100%. R, I, D mean *replace, insert, delete.* See Section 4.3.1 for a detailed discussion.

MCMC moves to the next sentence x^* by first generating a sample x^* from the proposal distribution $Q(x^*|x)$ and then accept x^* with the following acceptance rate $A(x^*|x)$:

$$A(x^*|x) = \min\left\{1, \frac{\pi(x^*)Q(x|x^*)}{\pi(x)Q(x^*|x)}\right\}. \tag{4.3}$$

If sentence x^* is rejected, then the sample remains at x. The distribution of samples will converge to the sentence stationary distribution of Markov chain $\pi(x)$ after long steps. Previous work [148] proposes to use MCMC for constrained sentence generation, namely the CGMH algorithm. Their proposal distribution only suggests sentences with one-word modifications. Nevertheless, CGMH cannot handle the combinatorial constraints in our problem definition, because of the *low acceptance ratio problem* caused by the *locality* of the proposal distribution. In other words, the sampling process can only visit a limited number of neighbors, thus the Markov chain will easily be trapped in one infeasible state, resulting in a lot of rejections. We illustrate this problem in detail and hence motivate our tree search embedded MCMC approach using the following example.

4.3.1 Motivation: Breaking the Local "Low Acceptance" Barrier

Suppose we need to generate a question, whose answer comes from an underlined part of a sentence. For example, suppose we underline *France* in the sentence:

①: *Paris is located in France.*

The question we would like to generate is:

②: *Which country is Paris located in?*

Under our constraint satisfaction framework, we define $\mathcal{C}(x)$ so that real interrogative sentences such as question ② would receive high probability in the defined $\pi(x)$. Our constraints are: (i) the whole sentence is in the interrogative form. (ii) "*Paris*" and "*located*" must appear in the sentence. We run MCMC starting from sentence ①.

It is hard for MCMC without tree search to generate question ② in reasonable time-steps starting from ①. Because the edit distance between sentence ① and ② is larger than 2, we cannot generate ② from ① with one step of word insertion, removal, or replacement. In order for CGMH to reach ② from ①, it has to encounter a few intermediate steps. Without loss of generality, suppose CGMH proposes sentence ③ in one MCMC step by removing *is*:

③: *Paris ~~is~~ located in France.*

Notice that ③ is not a legitimate English sentence, so its language model score $P_{\text{LM}}(x)$ becomes much smaller compared to the original sentence ①. In addition, ③ violates more constraints than ①, which decreases its $\mathcal{C}(x)$ score as well. In MCMC, the probability of accepting the move from ① to sentence ③ is given by Equation (4.3), in which the dominating term is

$$\frac{\pi(\textcircled{3})}{\pi(\textcircled{1})} = \frac{P_{\text{LM}}(\textcircled{3})\ \mathcal{C}(\textcircled{3})}{P_{\text{LM}}(\textcircled{1})\ \mathcal{C}(\textcircled{1})}$$

Because both $P_{\text{LM}}(\textcircled{3})$ and $\mathcal{C}(\textcircled{3})$ are smaller, the acceptance ratio becomes really small. In fact, we found the acceptance ratio to be 5.93×10^{-12} in our experiment. This means that it will take CGMH on the order of 10^{12} many steps to move from sentence ① to ③. Figure 4.2 (left) demonstrates that barriers of low acceptance rate exist on every possible sequence of edits from sentence ① to ③.

On the other hand, if we allow the proposal distribution $Q(x^*|x)$ to suggest sentences with multiple word-level changes, one can transit from sentence ① to ② through all legitimate sentences as intermediate steps. Consider the following two-step change:

1. First delete *is* and insert *is* before *Paris*. This changes sentence ① to:

   ④: *Is Paris located in France?*

2. Delete *France* and insert *Which* and *country*. This changes sentence ④ to ②.

Because the intermediate step sentence ④ is a legitimate English sentence and $\mathcal{C}(④) = \mathcal{C}(①)$, $\frac{\pi(④)}{\pi(①)}$ is close to 1, resulting in a 100% acceptance ratio in this step. When changing from ④ to ②, notice that ② is also a legitimate sentence and it satisfies more constraints than ④. In fact, the acceptance ratio is also 100%. Figure 4.2 (right) demonstrates this case.

For tasks with soft constraints, there are also similar rejection problems for CGMH. For example, *"Nothing is impossible"* is a sentence with a positive sentiment. If we insert, replace, or delete one word, it is hard to keep the sentence valid and preserve the positive sentiment.

Motivated by these examples, we propose the embed a tree search into the proposal process of MCMC to solve the Local "low acceptance" barrier problem, which suggests candidate sentences with multiple (rather than one) word-level edits and satisfy more constraints.

4.3.2 Detailed Procedure of TSMH

Our Tree Search enhanced Metropolis-Hastings (TSMH) still follows the classical MCMC procedure. The only difference is a new proposal distribution $Q(x^*|x)$ generated from a tree search process. The tree search defines a probability distribution over *templates* of sentence moves. Each template defines a subset of possible moves. The sentences within the same template satisfy the same hard constraints. The proposal probability distribution induced by the tree search algorithm biases towards templates that have high Constraint(x) scores. The detailed steps are illustrated below:

1. Given a sentence x, our algorithm will randomly select several word positions for editing.

2. For all the selected word positions, we use Tree Search to efficiently enumerate all possible edit operations: to *insert, delete, or replace* the selected positions and what are the word category in the case of *insert* and *replace*. Every leaf branch of the search tree will be our sentence template.

3. We extract all the sentence templates and count the number of constraints satisfied for each template. We randomly sample several templates with respect to a probability distribution that favors templates satisfying more constraints.

4. we fill in the sampled template with words suggested by a language model. According to the language model score times the soft constraint score $P_{\text{LM}}(\hat{x}) \cdot \Phi_{\text{soft}}(\hat{x})$, we select one filled sentence $\hat{x}$ as the proposal.

In summary, our approach alleviates the rejection problem of CGMH by enumerating all possibilities in the space of multiple-word change at the template level. This process enables us to handle combinatorial constraints and the Tree search allows us to prune branches of low-quality sentences.

4.4 Experiments

We evaluate our approach on three applications: interrogative, imperative, and fixed sentiment sentence generation. In each task, we construct the specified type of sentences by sampling starting from keywords and enforcing task-specific constraints.

In general, our method TSMH outperforms baselines and generates sentences that satisfy more constraints, are of good quality ,and are likely to be close to the natural language. Our main results are summarized in Table 4.1, in which Valid% denotes the percentage of generated sentences that satisfy all constraints. $\pi(x)$ is the value of the stationary probability $P_{\text{LM}}(x) \cdot \text{Constraint}(x)$. $P_{\text{GPT2}}(x)$ is language model probability estimated by a pre-trained GPT-2 model, which measures the quality of the sentences. Accept% means the acceptance rate of MCMC.

Table 4.1. Our method TSMH outperforms CGMH by generating sentences that satisfy more constraints, are of good quality and are likely to be natural language. Column Valid% shows the percentage of generated sentences that satisfy all constraints, which TSMH clearly leads baselines. In addition, TSMH has better acceptance rates (Accept%). The language generated by TSMH is also of good quality, because it matches other models in language model scores $P_{\text{GPT2}}(x)$. Multiplying both the language model score and the constraint score, the sentences generated by TSMH tend to attain higher stationary probability $\pi(x)$.

Tasks	Methods	#sample	step	Valid%	$\pi(x)$	$P_{\text{GPT2}}(x)$	Accept%
Interrogative	CGMH	300	1	18.33%	2.60E-04	1.78E-18	5.45
	TSMH (ours)	100	3	**92.67%**	**1.44E-03**	**5.51E-18**	**24.50**
Imperative	CGMH	300	1	91.32%	0.0004	9.86E-16	5.49
	TSMH (ours)	100	3	**97.75%**	**0.0060**	**6.60E-15**	**15.66**
Sentiment	CGMH	300	1	96.33%	4.93E-19	4.57E-22	6.72
	TSMH (ours)	100	3	**96.67%**	**7.94E-04**	**1.82E-18**	**11.09**

4.4.1 Experiment Settings

For each task, we run our TSMH algorithm for 100 steps, with 100 candidate sentences generated. k is set to 3. Since the tree search in TSMH considers changing 3 words at each iteration, we run the baseline CGMH for 300 steps as a comparison. We select the sentence with the highest $\pi(x)$ value among the sentences generated by each algorithm as the output. Detailed experiment settings can be reviewed in Appendix B.1.

4.4.2 Interrogative Sentence Generation

In the interrogative sentence generation, we construct interrogative sentences by sampling starting from the keywords. We enforce that sentences with a high probability of being sampled must satisfy the grammar constraints of being interrogative and contain a few given keywords. The constraint definition for interrogative sentences is in the previous section.

According to the results, in the experiment with keywords, 92.67% of the output sentences of our TSMH algorithm satisfy all the constraints, while merely 18.33% satisfy constraints for the baseline. The numbers are 83.17% and 45.50% for the experiment without keywords,

Table 4.2. Human evaluation on the quality of the generated interrogative sentences from keywords in terms of fluency and grammar. Most human participants (native speakers) agree that the sentences generated by our TSMH are better in quality compared to CGMH.

Methods	Number of Votes	Percentage of Votes
CGMH	196	33.64%
TSMH (Ours)	**384**	**66.36%**

respectively. This demonstrates that our TSMH generates sentences with more constraints satisfied. In addition, our method has a higher $\pi(x)$ (stationary probability value) and acceptance rate, suggesting that the tree search embedded help MCMC to mix faster. Overall, our method TSMH can handle more complicated constraints in language generation tasks.

Human Evaluation We conduct the human evaluation for interrogative sentences generated with keywords. We present human participants from the Amazon Mechanical Turk with a pair of sentences at a time. One sentence is generated by our TSMH model and the other one is from the baseline CGMH. We ask human participants which sentence is better in terms of fluency and grammar. In terms of the experimental setting, we use 100 sentence pairs generated by CGMH and TSMH with the same keyword inputs. We randomly split these 100 test sentence pairs into 5 survey groups, and then deploy them on the Amazon Mechanical Turk. We randomly assign human participants to survey groups. When showing the sentence pairs, we also provide the keywords that the sentences must contain. We ask human participants to vote on which sentence in the pair is better in terms of grammar coherence, keyword coverage, and fluency. We use a gold-standard question to detect if the voter is randomly doing the survey. Every valid survey contains a randomized set of 20 questions. We received all 580 votes. Each question pair receives votes ranging from 5 to 11. As shown in Table 4.2, sentences from our model receive almost twice times of votes than the baseline, which suggests that the sentences generated by our approach is better in human evaluation.

Case Studies As shown in Table 4.3, we compare some output sentences of our method with the baseline using the same inputs and keywords. More examples can be seen in the

Table 4.3. Case study of generating interrogative sentences with keywords. Sentences generated by our method cover all the input keywords. Full case study is in Table B.1.

Keywords	waste heat water
CGMH	what waste is there, it seems now?
TSMH (Ours)	where was the waste - water heater?

Keywords	responses protect lungs
CGMH	how can immune responses also occur by not only infecting pathogens in the central nervous system?
TSMH (Ours)	what responses do your lungs have to protect you from pathogenic bacteria?

Keywords	median temperature winter
CGMH	what do you mean we have median temperature winter and spring, anyways?
TSMH (Ours)	what is the median temperature range in the winter months?

Keywords	catholics concentrated France
CGMH	the catholics are now mainly concentrated there.
TSMH (Ours)	why are the french roman catholics so densely concentrated in southern france?

appendix B.2. From these cases, we can see that our method generates sentences with better quality.

Comparison with Other Methods We compare our TSMH method with UQA [149]. The setting of UQA is different from us: it takes a paragraph as input and generates a corresponding question. Although this comparison is not fair, the baseline is the most similar and the best framework that we can compare with. To run UQA, we use the corresponding original sentences from which the keywords of TSMH are extracted as the input. In other words, for TSMH, the inputs are keywords extracted from the SQuAD 2.0 [150] questions. For UQA, we take the corresponding paragraphs of the selected questions as input. This also gives UQA an additional advantage because it has access to a paragraph, rather than keywords. To make it more comparable, we remove the keyword constraints in this experiment. In Table 4.4, we compare the language model scores $\log P_{\text{LM}}$ of the generated sentences that reflect the naturalness and fluency, and the stationary probability $\pi(x)$ and valid percentage

Table 4.4. Comparison with UQA. Our TSMH outperforms UQA in terms of the percentage of satisfying the interrogative sentence constraints and has a higher score predicted by a language model, despite UQA being trained on specific interrogative sentences while our method is not trained at all.

Methods	$\pi(x)$	Validness %	$\log P_{\text{LM}}$
UQA [149]	0.0024	50%	-92.75
TSMH	**0.0063**	**83.17%**	**-58.27**

Table 4.5. Generate sentences with positive sentiment. Half of the inputs are extracted from positive sentences, and the other half are from negative, which are harder to transform into positive sentences.

Tasks	Method	$\pi(x)$	$P_{\text{GPT-2}}(x)$	Acceptance (%)	Sentiment Score
Positive to Positive	CGMH	9E-19	8E-22	8.16%	0.8647
	TSMH (ours)	**4E-04**	**2E-18**	**12.23%**	**0.8801**
Negative to Positive	CGMH	5E-20	6E-23	5.65%	0.3470
	TSMH (ours)	**1E-03**	**7E-19**	**9.91%**	**0.5254**

Valid% that show how good it satisfies our pre-defined constraints. We pointed out that UQA was trained on the specific interrogative sentences while our method was not trained at all.

4.4.3 Imperative Sentence Generation

We generate imperative sentences via sampling starting from the keywords. We enforce grammar constraints of being an imperative sentence: the starting word should be either a verb $w_1^{\texttt{[VERB]}}$ or an adverb followed by a verb $w_1^{\texttt{[ADV]}} \wedge w_2^{\texttt{[VERB]}}$. We also enforce keyword constraints in this task.

As shown in Table 4.1, our method has a higher valid percentage of 97.75% compared to 91.32% of the baseline, showing that the sentences generated by our method can satisfy more constraints. Our method has a higher $\pi(x)$ (stationary probability value) and acceptance rate, suggesting our approach has a better mixing behavior. Overall, results show that our method using Tree Search Embedded MCMC can handle more complicated combinatorial constraints in language generation.

Table 4.6. Comparison with CtrlGen [138] over the "Negative to Positive" subtask with acceptance rate, language score and sentiment score metrics.

Task	Methods	$\pi(x)$	$P_{\text{GPT-2}}(x)$	Sentiment Score
Negative to Positive	CtrlGen [138]	3.19E-07	4.64E-22	0.4614
	TSMH (ours)	**1.16E-03**	**7.07E-19**	**0.5254**

4.4.4 Sentence Generation with Given Sentiment Score

In this task, we require the sentences to contain the specified keywords and have positive sentiments [151]. We enforce the sentences to attain high scores from a sentiment analysis neural network. We also enforce keyword constraints as hard constraints. We need to emphasize that, our method uses a model pre-trained on a separate dataset for sentiment analysis, which is kept intact in our experiment. No additional fine-tuning to the sentiment analysis model was performed. we consider two sub-tasks in Table 4.5: (i) positive sentiment to positive sentiment (P2P), where the input keywords are extracted from sentences which originally have positive sentiments; (ii) negative sentiment to positive sentiment (N2P), where the keywords are extracted from sentences with negative sentiments. N2P is more difficult as it requires transforming the sentiment.

Our method has a higher sentiment score, suggesting that our method generates sentences with more positive sentiments (better aligned with the target of this experiment). The increase against CGMH is bigger on the more difficult N2P task, which requires flipping the sentiment. Our model also leads in terms of language model scores, suggesting the language quality is better.

Comparison with Other Methods We compare our method with CtrlGen [138]. The setting is a little different from ours: it takes a sentence with a negative sentiment as input and transforms it to positive, without the guarantee of satisfying keyword constraints. Our method takes a set of keywords as input. To make the outputs comparable, we select the same set of negative sentences as the input of CtrlGen and extract the keywords of those sentences as the input of TSMH. Our method requires no additional training besides a pre-trained sentiment analysis model and a pre-trained language model, while CtrlGen requires training the auto-encoder.

The results in Table 4.6 show that our method outperforms CtrlGen in terms of both sentence quality and sentiment, as the sentences generated by our method receive higher language model scores and sentiment scores.

4.5 Summary

We propose a framework for constrained language generation via sampling and combinatorial constraint satisfaction. Our strategy is to sample sentences from the constrained space with probability proportional to the language model scores. To handle the combinatorial constraints, a tree search is embedded into the proposal process of MCMC. Experiments demonstrate that our approach generates sentences that satisfy more constraints and are likely to be close in quality to the natural language.

5. Probabilistic Area Loss Minimization for Protein Sequence Alignment

5.1 Introduction

Protein sequence alignment is a fundamental problem in computational structure biology and has been widely applied to protein sequence, structure and functional study [152], including protein 3D structure prediction [153] and protein homology detection [154][1]. In the past two decades, many computer programs have been developed for automatic pairwise sequence alignment [155–157] and multiple sequence alignment [158, 159]. Here we only focus on pairwise sequence alignment, where we let S and T be two sequences of amino acids. Our goal is to align the two sequences. As shown in Figure 5.1(bottom), each amino acid in one sequence can be aligned to either an amino acid in the homology sequence (called a match) or a gap (called an insertion). Our tasks are that: (1) given a dataset of aligned pairs of amino acid sequences, learn the likelihoods of different alignments of the two sequences; (2) given a new amino acid sequence pair, determine the most likely alignment.

Most probabilistic alignment approaches maximize likelihood functions, which lead to the minimization of the pointwise differences of the two alignments. Nevertheless, this pointwise difference loss function is unrealistic for the biology application. For example, Figure 5.1 presents two predicted alignments against the ground truth. Both predicted alignments are different from the ground truth in four locations. Therefore, they have identical pointwise differences against the ground truth. Nonetheless, the green alignment is considerably better than the orange alignment, because the corresponding amino acids of each pair in the ground-truth are much closer to the green alignment. The missed predictions of the green alignment can be the result of biological measurement noises, while the orange alignment completely misses the ground truth.

In order to introduce the distance between two alignments into the probabilistic model, we propose **<u>P</u>**robabilistic **<u>A</u>**rea **<u>L</u>**oss **<u>M</u>**inimization (PALM) for pairwise protein sequence alignment, which is a two-step approach to model the whole alignment matrix. The key idea

[1] ↑I collaborate with Fan Ding as co-author for this work. This chapter contains sections taken verbatim from [3], which Fan Ding has never used for use of any degree. I will rewrite this chapter in final thesis and mark out individual contributions.

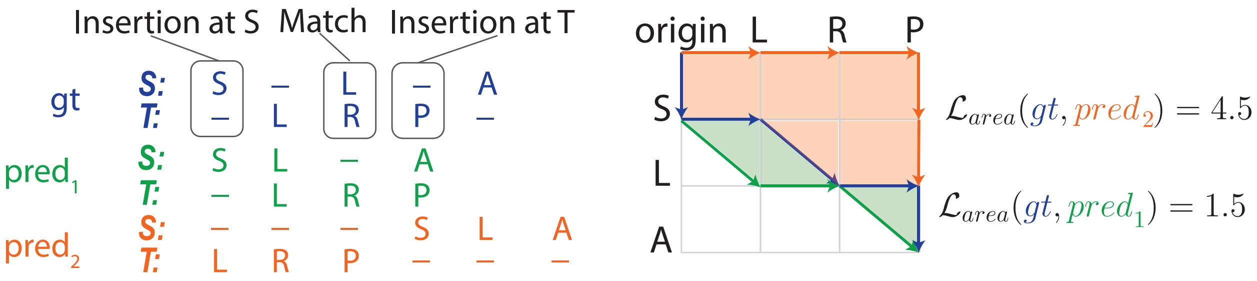


Figure 5.1. Illustration of protein sequence alignment and the area distance. **(Left)** The task is to align two amino acid sequences S and T, where one amino acid from one sequence can be aligned to either one amino acid from the other sequence (match), or to a gap (insertion, marked by -). **(Right)** Such an alignment becomes a path in the alignment matrix, where a diagonal transition represents a match, and a horizontal or a vertical transition represents an insertion. The area between one predicted alignment and ground-truth is viewed as the area distance between them. Both of the two predicted alignments correctly predicted one edge on the ground truth alignment, yet the one with smaller area loss (i.e., "pred$_1$") is much closer to the ground truth.

of PALM is (i) using a generative model $Pr(a|S,T)$ to model the ground-truth alignment a where the matching of each pair of amino acids in the two protein alignment leads to a change in the total likelihood; (ii) the observed alignment a^* is a noisy observation of the ground-truth alignment, the likelihood of observing which $Pr(a^*|a,S,T)$ negatively depends on the area difference of the two alignments. This area difference is capable of greatly penalizing those alignments a that are far from the observed alignment a^*. The learning goal of PALM is to maximize the marginal likelihood of the observed alignment $Pr(a^*|S,T)$, which sums over all the different ground-truth alignment a. However, the closed-form of maximal likelihood estimation of PALM is computationally intractable. We, therefore, propose a novel computing scheme that maximizes a lower bound of $\log Pr(a^*|S,T)$. To optimize the lower bounded objective, we formulate the gradient computation using contrastive divergence framework [160], where dynamic sampling is used to sample alignments to estimate the gradient unbiasedly. We show theoretically that PALM can converge to the global optimum of the objective in a linear number of iterations.

Traditionally, dynamic programming with a deterministic cost function [161, 162] was used to obtain the alignment, including Needleman–Wunsch method for global alignment [163] and Smith-Waterman algorithm for local alignment [164]. However, they are heavily dependent upon the proper design of the cost function between two amino acids. Recently probabilistic learning has been widely used to model sequences [165–168]. [169] employs HMM-HMM comparison for learning the alignment probability which models a protein family using HMM (Hidden Markov Model). MRFalign [165] further uses an MRF-MRF comparison which is more expressive than the HMM model. However, they cannot model long-range residue interaction patterns and thus are not sensitive enough to distant-related homologies. The idea of using area distance to model the gap between different sequences is also found in audio recognition [170, 171], which is under a different setting of ours.

Empirically, we show on the large dataset Protein Data Bank (PDB) [172] that our method has higher precision and recall value, as well as much smaller area distance than the competing methods, especially the lengths of the two sequences are far from each other. We find that the exact/4-offset/10-offset recall of PALM is twice more than that of dynamic programming (DP) when $|S| \in [1, 100], |T| \in [400, +\infty)$ and the exact/4-offset/10-offset precision is also twice more than of DP when $|S| \in [400, +\infty), |T| \in [1, 100]$. In addition, in terms of time efficiency between learning our method using dynamic sampling and the automatic differentiation method, PALM takes only one-fourth time to compute the gradient than the automatic differentiation over six different testing sets.

To summarize, our contributions are as follows:

- We propose PALM, a novel two-step approach for protein sequence alignment, where we first model the ground-truth alignment using a generative model, and then leverage a conditional distribution formed by the area difference of two alignments to denoise the observed alignment from noisy observation.

- We propose a novel computing scheme to maximize a tight lower bound of the computationally intractable log-likelihood, and efficiently compute the gradients by estimating it unbiasedly via dynamically sampling alignments. We show theoretically that PALM is able to converge to the global optimum of the objective in a linear number of iterations.

- Experimentally we test our method on a large PDB dataset [172], and find that PALM outperforms the competing methods in either precision, recall, or area distance, especially on long protein sequences and remote homologies[2].

5.2 Preliminary

In this section, we briefly introduce the Markov Random Field (MRF), which is used as our probabilistic model. We also introduce the problem of pairwise sequence alignment.

5.2.1 Pairwise Sequence Alignment

Given a pair of sequence (S, T), we can formulate an alignment matrix of shape $(|S|, |T|)$ as shown in Figure 5.1. In the matrix, each row represents an amino acid in S and each column represents an amino acid in T. Each alignment for sequences S and T forms a path from the upper-left node to the bottom-right node, where each edge in the path is either horizontal, representing an insertion in T, vertical, representing an insertion in S, or diagonal, representing a match. We use symbols M, I_S, and I_T to represent a match, an insertion in S, and an insertion in S, respectively. Thus an alignment a is a sequential combination of symbols M, I_S, and I_T. Let $Pr_\theta(a|S,T)$ be the probability of alignment a with parameter θ. Our goal is to align the two given sequences. Our task can be divided into the following two parts.

Learning. Given a training set of $\mathcal{D} = \{(S^{(k)}, T^{(k)}, a^{*(k)})\}_{k=1}^{N}$, where $S^{(k)}, T^{(k)}$ is a pair of sequences, and $a^{*(k)}$ is the ground-truth alignment between the two sequences. We want to learn the model by maximizing the log-likelihood, which translates to the following problem:

$$\text{Maximize}_\theta \frac{1}{N}\sum_{k=1}^{N} \log Pr_\theta(a^{*(k)}|S^{(k)}, T^{(k)}). \tag{5.1}$$

where θ is the parameter of the model.

[2] ↑Implementation: https://github.com/jiangnanhugo/PALM

Inference. After learning the model's parameter θ, we can use the model to find the best alignment between two new sequences by solving the following problem:

$$\hat{a} = \arg\max_{a \in \mathcal{A}} Pr(a|S, T) \tag{5.2}$$

where $\mathcal{A} = \{a \mid a \text{ is a valid path}\}$ is the set of all valid paths. The alignment $\hat{a}$ needs to form a consecutive path in the alignment matrix.

Nevertheless, most existing approaches that maximize likelihood functions will lead to the minimization of the pointwise differences between the two alignments, which is unrealistic for biology applications. For instance, Figure 5.1 presents two predicted alignments against the ground truth. Both alignments are different from the ground truth in four locations. Therefore, they have equal pointwise differences against the ground truth. However, the green alignment is considerably better than the orange alignment, because the corresponding amino acids of each pair in the ground-truth are much closer in this alignment. The missed predictions of the green alignment can be the result of biological measurement noises, while the orange alignment completely misses the ground truth. Therefore, the green alignment should have a larger likelihood compared to the orange one given the ground-truth alignment.

5.3 Probabilistic Area Loss Minimization

In this section, we first introduce our two-step model and then explain how it can solve the distant-related gap problem. Followed by the illustration of how to efficiently learn this model via dynamic sampling. Finally, we detail the process of generating alignments in testing.

5.3.1 Two-step Model

Define function π be the mapping from the index on the alignment to the indexes on both sequences, i.e., $\pi_S(a, k)$ is the index on sequence S for the k-th term of alignment a, and

Algorithm 4: Probabilistic Area Loss Minimization for Protein Sequence Alignment

```
   Input: Model parameter $\theta$, Maximum epochs $Epo$, Learning rate $\eta$, Weight $\lambda$,
          Training data $\mathcal{D}$.
   Output: The converged model's parameter $\theta_{Epo}$
1  for $t = 1$ to $Epo$ do
2      Solved by DP for computing matrix $A$
3      Inference $\hat{a} \leftarrow \arg\max_a \{\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k) - \lambda \mathcal{L}_{area}(a^*, a)\}$.
4      Randomly pick data $\{S, T, a^*\}$ from training set $\mathcal{D}$.
5      for $i = |S|$ to $1$, $j = |T|$ to $1$ do
6          Compute $Z(i,j)$.                    // See (5.9) and is used in Step 10
7      for $m = 1$ to $M$ do
8          $i = |S|, j = |T|$
9          while $i \geq 1$ and $j \geq 1$ do
10             Sample an edge $a_k^m$ w.r.t $Pr_\theta(a_k|S_i, T_j)$ and update $i, j$.     // See (5.10)
11             Compute gradient of the neural network $\nabla\phi_\theta(\pi_S(a^m, k), \pi_T(a^m, k), a_k^m)$.
12     Sample $M$ alignments: $[a^1, \dots a^M]$ from probability $Pr_\theta(a|S,T))$.
13     Estimate $\nabla \log Z_\phi \leftarrow \frac{1}{M} \sum_{m=1}^{M} \Big[\sum_{k'=1}^{|a^m|} \nabla\phi_\theta(\pi_S(a,k'), \pi_T(a,k'), a_{k'})\Big]$.
14     Estimate $\nabla\mathcal{L}_{LB} \leftarrow \sum_{k=1}^{|\hat{a}|} \nabla\phi_\theta(\pi_S(\hat{a},k), \pi_T(\hat{a},k), \hat{a}_k) - \nabla \log Z_\phi$.
15     $\theta_{t+1} \leftarrow \theta_t + \eta\nabla\mathcal{L}_{LB}$.                    // Gradient update
16 return $\theta_{Epo}$
```

$\pi_T(a,k)$ is the index on sequence T for that term. $(\pi_S(a,k), \pi_T(a,k))$ are the coordinates of the k-th term of a in the alignment matrix.

Prior Prediction. Let $\mathcal{A} = \{a \mid a \text{ is a valid path}\}$ be the set of all valid paths. The validness ensures that alignment a has to start from the upper-left node and end at the bottom-right node in the matrix. We model the probability of having the alignment a over sequences S and T as:

$$Pr_\theta(a|S,T) = \frac{\exp\left(\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k)\right)}{Z_\phi} \tag{5.3}$$

where $Z_\phi = \sum_{a\in\mathcal{A}} \exp\left(\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k)\right)$ is the partition function. Also the function $\phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k)$ is the feature that takes as input the features extracted from every two amino acids of S and T, and θ represent the model parameters. Notice that function ϕ_θ can be either a linear function, where the model becomes a Markov random field or a neural network of arbitrary architecture.

Denoising via Area Distance. Due to the measurement error of biological tools, the observed alignments a^* are usually noisy [173]. In view of this observation, we introduce a conditional probability distribution over a^* conditioned on the latent variable a to diminish the effect of noise. We leverage the area distance as a probabilistic measure where the predicted alignment a that is similar to the observed one a^* has higher probability value:

$$Pr_{area}(a^*|a, S, T) = \frac{\exp\left(-\lambda \mathcal{L}_{area}(a^*, a)\right)}{Z_{area}} \tag{5.4}$$

where $Z_{area} = \sum_{a' \in \mathcal{A}} \exp\left(-\lambda \mathcal{L}_{area}(a^*, a')\right)$ is the normalization term and λ is the weight. We compute the area distance as the gap of area between the two alignments in the alignment matrix, which penalizes those predicted alignments that is far away from the observed alignment. For instance, as in Figure 5.1, the area distance between the ground-truth and the first predicted alignment (green line) is 1.5, and is 4.5 between the ground-truth and the second one (orange line). Then, we maximize the likelihood of the observed alignment, and the whole model becomes

$$Pr(a^*|S, T) = \sum_{a} Pr_{area}(a^*|a, S, T) Pr_\theta(a|S, T).$$

which sums over the latent variable a and is the marginal distribution of a^*.

5.3.2 Model Learning

To learn the model $Pr(a^*|S, T)$, we would like to maximize the log-likelihood of the observed alignment given sequence S and T:

$$\max_\theta \mathcal{L} = \max_\theta \log Pr(a^*|S, T) \tag{5.5}$$

Combining with our model definition in Equation 5.3 and 5.4, the log likelihood $\mathcal{L}$ can be rewritten as:

$$\begin{aligned}\mathcal{L} &= \log \sum_a \frac{\exp\left(-\lambda \mathcal{L}_{area}(a^*, a)\right)}{Z_{area}} \frac{\exp\left(\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k)\right)}{Z_\phi} \\ &= \log \sum_a \exp\left(\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k) - \lambda \mathcal{L}_{area}(a^*, a)\right) - \log Z_\phi Z_{area}\end{aligned}$$

The evaluation of $\mathcal{L}$ needs to sum over all possible alignments, which is computationally intractable. Since the sum is usually dominated by one alignment that has the maximum likelihood, we optimize the lower bound of $\mathcal{L}$ instead of directly optimizing $\mathcal{L}$. Denote the lower bound as $\mathcal{L}_{LB}$, which is

$$\mathcal{L}_{LB} = \max_a \{\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k) - \lambda \mathcal{L}_{area}(a^*, a)\} - \log Z_{area} - \log Z_\phi \tag{5.6}$$

It is obvious that $\mathcal{L}_{LB} \leq \mathcal{L}$ because of the principle of log-sum-exp function: $\max_x \phi(x) \leq \log \sum_x \exp(\phi(x)) \leq \max_x \phi(x) + \log N$, where N represents the number of all possible x. Then, the learning procedure is separated into two steps, where the first step is to infer $\hat{a}$ that has the maximum likelihood:

$$\hat{a} = \max_a \{\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k) - \lambda \mathcal{L}_{area}(a^*, a)\}$$

One can identify that $\hat{a}$ represents those alignments that are close to the observed alignment a^*. Since the observed alignments are generated by biological tools and contains some noisy observation, our method use a bunch of alignments that are close to the observed alignments to reduce the impact of noise. The second step is to optimize parameter θ in order to maximize $\mathcal{L}_{LB}$ using both a^* and $\hat{a}$. By optimizing the parameter θ, we can keep increasing the likelihood of $\hat{a}$, making the lower bound to approach the true likelihood $\mathcal{L}$. Algorithm 4 shows the learning procedure of PALM. In the next few parts, we will show how to inference $\hat{a}$ via dynamic programming, how to optimize θ via dynamic sampling, and finally give a convergence analysis of our learning algorithm.

Inference via Dynamic Programming Based on the observation that our area loss $\mathcal{L}_{area}$ is decomposable, we propose a dynamic programming approach to inference $\hat{a}$. Specifically, we decompose the area loss into the sum of area-unit distance $\mathcal{L}_{area}(a^*, \hat{a}) = \sum_{k'=1}^{|\hat{a}|} \mathcal{L}_{area}^{k'}(a^*, \hat{a})$ where k' is the index of the predicted alignment $\hat{a}$. Given the observed alignment a^*, we compute $\mathcal{L}_{area}^{k'}$ as follows: we first find it's corresponding coordinates on sequences S and T is $\pi_S(\hat{a}, k'), \pi_T(\hat{a}, k')$ and then find an index k in a such that $\pi_T(a, k) = \pi_T(\hat{a}, k')$. Then $\mathcal{L}_{area}^{k'}$ is defined as:

$$\mathcal{L}_{area}^{k'}(a^*, \hat{a}) = \begin{cases} |\pi_S(a^*, k) - \pi_S(\hat{a}, k') + \frac{1}{2}| & a_k^* = M, \hat{a}_{k'} = I_T \\ |\pi_S(a^*, k) - \pi_S(\hat{a}, k')| & a_k^* = M, \hat{a}_{k'} = M \\ |\pi_S(a^*, k) - \pi_S(\hat{a}, k') - \frac{1}{2}| & a_k^* = I_T, \hat{a}_{k'} = M \\ |\pi_S(a^*, k) - \pi_S(\hat{a}, k')| & a_k^* = I_T, \hat{a}_{k'} = I_T \\ 0 & \text{otherwise} \end{cases}$$

Since we can decompose $\mathcal{L}_{area}$, $\hat{a}$ can be obtained by the following dynamic programming approach. Let $A(i, j)$ represent the maximum likelihood of the path from node (i, j) to the bottom right corner, in the alignment matrix of sequence S and sequence T. Then we have:

$$A(i, j) = \max \begin{cases} A(i+1, j+1) + \phi_\theta(S_i, T_j, M) + \lambda \mathcal{L}_{area}^{k'}(a^*, \hat{a}) \\ A(i+1, j) + \phi_\theta(S_i, T_j, I_S) + \lambda \mathcal{L}_{area}^{k'}(a^*, \hat{a}) \\ A(i, j+1) + \phi_\theta(S_i, T_j, I_T) + \lambda \mathcal{L}_{area}^{k'}(a^*, \hat{a}) \end{cases}$$

where $j = \pi_T(\hat{a}, k')$ and we initialize $A(|S|, |T|) = 0$. The computation is line by line from the bottom right corner to the up left corner in the alignment matrix. The corresponding alignment $\hat{a}$ can be extracted from the matrix A by following the path that gives the largest likelihood.

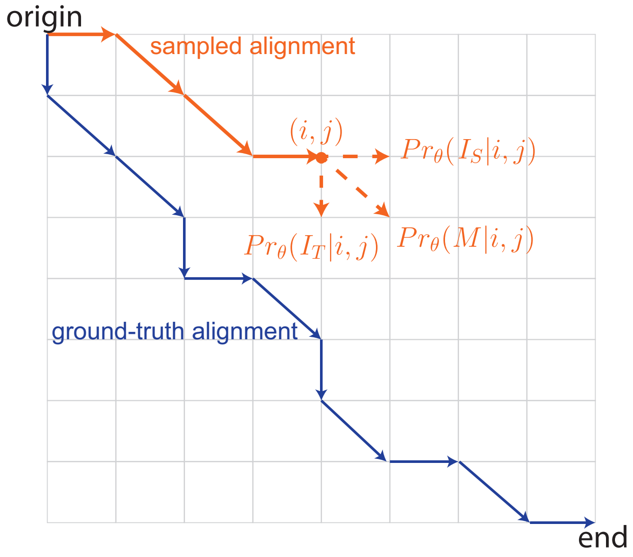


Figure 5.2. Sampling a path from the original until reaching the bottom-right corner in the alignment matrix. At point (i, j), the sampling approach first calculates the probability of taking the options M, I_S, I_T at this point, and then samples one option according to the probability value.

Optimization via Dynamic Sampling Once $\hat{a}$ is obtained according to the area distance, we optimize the lower bound $\mathcal{L}_{LB}$ using stochastic gradient descent. The gradient of $\mathcal{L}_{LB}$ can be written as:

$$\nabla\mathcal{L}_{LB} = \sum_{k=1}^{|\hat{a}|} \nabla\phi_\theta(\pi_S(\hat{a},k), \pi_T(\hat{a},k), \hat{a}_k) - \nabla \log Z_\phi \tag{5.7}$$

The term $\nabla\phi_\theta(\pi_S(\hat{a},k), \pi_T(\hat{a},k), \hat{a}_k)$ is the gradient of function ϕ_θ, which can be directly computed. Overall, there are $|\hat{a}|$ number of gradient terms with respect to this function. $\log Z_{area}$ term does not contain parameter to optimize, so it does not has the corresponding gradient term in $\nabla\mathcal{L}_{LB}$. For computing $\nabla \log Z_\phi$, we follow the idea of contrastive divergence [160] that formulate it as an expectation over probability $P(a|S,T)$:

$$\begin{aligned}
\nabla \log Z_\phi &= \frac{1}{Z_\phi}\sum_{a\in\mathcal{A}} \nabla \exp\left(\sum_{k'=1}^{|a|} \phi_\theta(\pi_S(a,k'), \pi_T(a,k'), a_{k'})\right) \\
&= \mathbb{E}_{a\sim Pr_\theta(a|S,T)}\left[\sum_{k'=1}^{|a|} \nabla\phi_\theta(\pi_S(a,k'), \pi_T(a,k'), a_{k'})\right] \\
&\approx \frac{1}{M}\sum_{a^m\sim Pr_\theta(a|S,T)}\left[\sum_{k'=1}^{|a^m|} \nabla\phi_\theta(\pi_S(a^m,k'), \pi_T(a^m,k'), a^m_{k'})\right]
\end{aligned}$$

Therefore, we can approximate the exact gradient $\nabla \log Z_\phi$ unbiasedly by first sampling M paths from distribution $Pr_\theta(a|S,T)$ and then sum the gradients $\nabla\phi_\theta$ of all sampled path $\{a^m\}_{m=1}^M$. Below we propose a backward-forward approach to sample one alignment under the probability distribution $Pr_\theta(a|S,T)$.

Backward Computing $Z(i,j)$. We denote $Z(i,j)$ as the sum of all the unnormalized energy values of every path starting from point (i,j) to the end $(|S|,|T|)$.

$$Z_\phi(i,j) = \sum_{a\in\mathcal{A}(i,j)} \exp\left(\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k)\right) \tag{5.8}$$

where $\mathcal{A}(i,j)$ is the set of all possible valid paths starting from node (i,j) to the end $(|S|,|T|)$. Then, we can backward compute $Z(i,j)$ via dynamic programming:

$$\begin{aligned}
Z_\phi(i,j) =& Z_\phi(i+1,j+1)\exp\left(\phi_\theta(i,j,M)\right) \\
&+ Z_\phi(i+1,j)\exp\left(\phi_\theta(i,j,I_S)\right) \\
&+ Z_\phi(i,j+1)\exp\left(\phi_\theta(i,j,I_T)\right)
\end{aligned} \tag{5.9}$$

Forward Sampling Alignment. To sample a path a^m starting from the original point in the alignment matrix to the bottom right corner, we recursively sample the k-th edge of the alignment from $\{M, I_S, I_T\}$ at point (i,j) based on the probability distribution $Pr_\theta(a_k^m|S_i, T_j)$ correspondingly.

$$Pr_\theta(a_k^m|S_i,T_j) = \begin{cases} \frac{Z(i+1,j+1)}{Z(i,j)} \exp\left(\phi_\theta(i,j,M)\right) & a_k^m = M \\ \frac{Z(i+1,j)}{Z(i,j)} \exp\left(\phi_\theta(i,j,I_S)\right) & a_k^m = I_S \\ \frac{Z(i,j+1)}{Z(i,j)} \exp\left(\phi_\theta(i,j,I_T)\right) & a_k^m = I_T \end{cases} \tag{5.10}$$

where $1 \leq i \leq |S|$ and $1 \leq j \leq |T|$. See Figure 5.2 as an example. We start from the top-left corner and iteratively compute and sample with respect to the probability until we arrive at the bottom-right corner. After we have obtained all the samples $\{a^m\}_{m=1}^M$, we estimate the gradient $\nabla\mathcal{L}_{LB}$ and update parameter θ.

Remark 5.3.1. The time complexity of computing gradient via dynamic sampling is $\mathcal{O}(|S||T|+(|S|+|T|)M)$. The complexity of computing Z is $\mathcal{O}(|S||T|)$. The complexity of computing the probability distribution for sampling is $\mathcal{O}((|S|+|T|)M)$. Thus, the whole optimization process of each iteration is $\mathcal{O}(|S||T|+(|S|+|T|)M)$.

Convergence Analysis In view of the convexity of $\mathcal{L}_{LB}$, we analyze our algorithm in terms of convergence rate towards the global optimal. We show in Theorem 5.3.2 that PALM is guaranteed to converge in a linear number of iterations to the global optimum.

Theorem 5.3.2. *Let $\mathcal{L}_{LB}$ be the lower bound of the log-likelihood function in Equation 5.6. Denote $\theta = \arg\max_\theta \mathcal{L}_{LB}$ and the total variation $Var_{Pr_\theta(a)}(\phi_\theta(a)) \leq L$. In iteration t of PALM in Algorithm 4, $\theta_{t+1} = \theta_t + \eta g_t$ where g_t is an unbiased estimation of the exact gradient $\nabla\mathcal{L}_{LB}(\theta_t)$. $Var(g_t) \leq \frac{\sigma^2}{M}$ where M is sample size. Then, for any number of epochs $T > 1$, step size $\eta \leq \frac{2}{L}$, and $\overline{\theta_T} = \frac{1}{T}\sum_{t=1}^T \theta_t$,we have:*

$$\mathbb{E}[\mathcal{L}_{LB}(\overline{\theta_T})] - \mathcal{L}_{LB}(\theta^*) \leq \frac{||\theta_0 - \theta^*||_2^2}{2\eta T} + \frac{\eta\sigma^2}{M}. \tag{5.11}$$

Theorem 5.3.2 states that PALM converges to the global optimal of $\mathcal{L}_{LB}$ with $O(T)$ iterations. This is mainly because the objective function $\mathcal{L}_{LB}$ is convex w.r.t parameter θ. In addition, since the total variation is bounded $Var_{Pr_\theta(a)}(\phi_\theta(a)) \leq L$, we can prove that $\mathcal{L}_{LB}$ is also

$L-$smooth. Therefore, by unbiasedly estimating the gradient, we can prove the convergence rate based on classic results in the literature of stochastic gradient descent. The complete proof of Theorem 5.3.2 is left to supplementary materials. In practice, we can increase either the number of epochs T or the sample size M to approach a better result.

5.3.3 Inference in Testing

Inference in testing is to predict an alignment that attains the most likelihood. It is different from Section 5.3.2, in which we use inference to generate alignment to estimate lower bounded loss function $\mathcal{L}_{LB}$. Given sequence S, T and the learned model, the inference case can be computed as finding an alignment that has the highest likelihood without area distance:

$$\hat{a} = \arg\max_{a \in \mathcal{A}} \exp\left(\sum_{k=1}^{|a|} \phi_\theta(\pi_S(a,k), \pi_T(a,k), a_k)\right) \tag{5.12}$$

Instead of enumerating all the possible valid paths in the matrix, we can still use dynamic programming to infer an alignment with $O(|S||T|)$ time complexity. Let $A'(i,j)$ represent the path from node (i,j) to the right corner with the maximum energy value in the alignment matrix of sequence S and sequence T. Then we have

$$A'(i,j) = \begin{cases} A'(i+1,j) + \phi_\theta(S_i, T_j, I_S), & \text{if } 1 \le i \le |S|, j = |T|{+}1; \\ A'(i,j+1) + \phi_\theta(S_i, T_j, I_T), & \text{if } i = |S|{+}1, 1 \le j \le |T|; \\ \max \begin{cases} A'(i+1,j+1) + \phi_\theta(S_i, T_j, M) \\ A'(i+1,j) + \phi_\theta(S_i, T_j, I_S) \\ A'(i,j+1) + \phi_\theta(S_i, T_j, I_T) \end{cases} & \text{otherwise} \end{cases}$$

where the initial condition is $A'(|S|,|T|) = 0$ and the computation is line by line from the bottom right corner to the up left corner in the alignment matrix. The corresponding alignment $\hat{a}$ can be extracted from the matrix A' by following the edge that gives the largest energy value.

Table 5.1. Comparison of precision and recall between our method and dynamic programming (DP) over different lengths of protein sequences on PDB [172] dataset. 4-off/10-off are the relaxed measures. PALM gets better results especially on longer sequences and remote homologies than the competing approach.

	$\lvert S\rvert \in [1, 100], \lvert T\rvert \in [100, 200]$		$\lvert S\rvert \in [100, 200], \lvert T\rvert \in [1, 100]$	
	Precision (%) exact/4-off/10-off	Recall (%) exact/4-offset /10-offset	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off
DP	7.8/**31.3**/**51.2**	20.4/39.0/56.3	20.2/40.4/59.4	6.1/26.3/**45.1**
PALM	**9.9**/29.8/48.7	**23.5**/**43.1**/**62.3**	**26.8**/**44.6**/**63.2**	**6.4**/**26.6**/43.1
	$\lvert S\rvert \in [1, 100], \lvert T\rvert \in [200, 400]$		$\lvert S\rvert \in [200, 400], \lvert T\rvert \in [1, 100]$	
	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off
DP	5.2/**27.6**/**46.1**	32.0/39.8/46.7	30.0/37.5/44.7	**3.8**/**19.8**/**34.4**
PALM	**6.5**/26.9/43.3	**51.4**/**62.5**/**73.3**	**52.7**/**63.5**/**73.8**	3.3/18.7/31.0
	$\lvert S\rvert \in [1, 100], \lvert T\rvert \in [400, +\infty)$		$\lvert S\rvert \in [400, \infty), \lvert T\rvert \in [1, 100]$	
	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off
DP	4.9/**26.4**/**45.2**	31.5/34.0/36.2	32.5/34.5/36.4	2.5/15.6/27.0
PALM	**5.1**/21.4/35.3	**75.3**/**81.1**/**86.3**	**76.0**/**81.1**/**86.1**	**3.1**/**18.1**/**29.1**
	$\lvert S\rvert \in [100, 200], \lvert T\rvert \in [200, 400]$		$\lvert S\rvert \in [200, 400], \lvert T\rvert \in [100, 200]$	
	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off
DP	6.5/27.0/45.2	26.1/38.6/50.5	25.4/38.9/51.2	**5.9**/**22.2**/**37.0**
PALM	**10.4**/**30.0**/**47.0**	**34.8**/**49.4**/**62.9**	**36.2**/**50.4**/**63.4**	4.6/18.5/31.0
	$\lvert S\rvert \in [100, 200], \lvert T\rvert \in [400, +\infty)$		$\lvert S\rvert \in [400, +\infty), \lvert T\rvert \in [100, 200]$	
	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off	Precision (%) exact/4-off/10-off	Recall (%) exact/4-off/10-off
DP	4.9/**24.1**/**41.0**	33.4/38.1/42.6	34.9/39.9/44.6	2.8/**14.4**/**24.8**
PALM	**6.1**/23.4/38.3	**61.1**/**69.0**/**76.5**	**62.5**/**71.0**/**78.8**	**3.2**/14.1/23.6

5.4 Experiments

In this section, we first illustrate the experiment setups and then compare PALM with competing methods in terms of testing performance and learning efficiency, followed by an ablation study on the hyper-parameter of the area distance.

Dataset. The full dataset for the protein alignment task is from [165], which contains 10567 distinct sequences and 210477 pairwise aligned sequences. The ground-truth alignments are generated from DeepAlign tool [174]. The feature for every amino acid describes the geometric similarity, evolutionary relationship, and hydrogen-bonding similarity of proteins. The feature dimension is 41. We use the full dataset for the training step. For testing, we first partition the full dataset by the length of two sequences and then randomly pick 200 pairwise aligned sequences from each group. The length of sequences are divided into 5 groups: $[1, 100], [100, 200], [200, 400], [1, 200], [400, \infty)$.

Baselines. We consider a dynamic programming algorithm (DP) with a deterministic cost function [163] to find the global alignment given two sequences. We use the cost for matching (i.e., M) as the summation over the feature vectors of two amino acids. The cost for insertion on sequence S (i.e., I_S) is the summation over the feature vector for the corresponding amino acid on sequence S. The cost for insertion on sequence T (i.e., I_T) is defined similarly. For algorithms that use a deep neural network with a richer set of features for the protein alignment problem, such as CNF [175] and DRNF [172], we do not compare with them for the sake of fairness. Because the parameter size of the implemented PALM model is much smaller and the feature used is also limited. However, PALM can be easily extended to a deep neural network by changing the feature function ϕ_θ. We leave the comparison with these deep neural methods as future work.

Evaluation Metrics. We use Precision and Recall to measure the quality of the predicted alignments. In the "exact" scenario, only an exactly matched result is used for computing the true positive rate. The "4-offset" scenario is a relaxed measure that a 4-position off the exact match is allowed. The "10-offset" case is defined similarly. These two relaxed metrics are applied to depict the model's performance for longer sequences. We also include the averaged computing time for estimating $\nabla \log Z_\theta$ over 100 epochs to reveal the time efficiency of our sampling approaches.

Implementation. The feature function ϕ_θ is a neural network with one single layer and the dimension of parameter θ is 82. ϕ_θ is adaptable to deeper neural networks with more parameters. For the "match" case, we use the concatenation of feature vectors. For insertion on sequence S case, we use the concatenation of feature vector for S_i and a zero vector. For

Table 5.2. Ablation study on hyper-parameter λ. When λ approaches infinity, area distance becomes more important in the inference of $\hat{a}$ during training, which leads to $\hat{a}$ more similar to the ground-truth alignment a^*. It can be seen that when we select a suitable λ that strikes a balance between the area distance and the score function, we can learn a better model than pure maximum likelihood learning (when $\lambda \to +\infty$).

	$\lvert S\rvert \in [1, 100], \lvert T\rvert \in [400, +\infty)$		$\lvert S\rvert \in [400, +\infty), \lvert T\rvert \in [1, 100]$	
	Precision (%)	Recall (%)	Precision (%)	Recall (%)
λ	exact/4-off/10-off	exact/4-off/10-off	exact/4-off/10-off	exact/4-off/10-off
50	**5.1/22.6/36.4**	75.3/81.1/86.3	75.9/**81.1**/86.0	2.6/17.0/27.2
100	4.6/21.3/35.2	75.3/81.1/86.3	**76.0/81.1/86.1**	3.1/**18.1/29.1**
500	4.5/20.9/34.0	**75.4/81.2/86.4**	75.9/81.0/85.9	3.1/17.4/28.3
∞	4.2/20.8/35.7	75.1/80.9/85.0	75.0/80.7/85.0	**3.5**/16.8/27.8

insertion on sequence T, we use the concatenation of a zero vector and feature vector for T_j. For the hyper-parameters, we set the number of sampled paths M to be 100 and the relative weight of area loss $\lambda = 50$. The learning rate η is initialized as 1 and decays by factor 0.9 for every 50 epochs. The maximum training epochs is set as 10^6. It takes a day to converge on the training set.

5.4.1 Learning Effectiveness for PALM

We compare with DP over 8 different testing sets for sequence alignment tasks. Results are collected in Table 5.1, where we observe that PALM has a higher Recall over exact/4-off/10-off settings when sequence T is much longer than sequence S and PALM has higher Precision over the exact/4-off/10-off settings when sequence S is longer. For $|S| \in [1, 100], |T| \in [400, +\infty)$, where the length difference of two sequences is large, Rthe ecall of PALM is twice as high as DP. Similarly, for $|S| \in [400, +\infty), |T| \in [1, 100]$, Precision of PALM is roughly twice as high as DP. When the difference of lengths of the two sequences becomes close, e.g., for $|S| \in [1, 100], |T| \in [100, 200]$, PALM has a relatively 3-6% higher Recall value than DP. Similarly, for $|S| \in [100, 200], |T| \in [1, 100]$, PALM has a relatively 4-6% higher Precision than DP.

5.4.2 Ablation Study on Weight Hyper-parameter

Here we analyze how the hyper-parameter λ balances the area distance and the score function for inference during training. When λ approaches infinity, area distance becomes more important in the inference of $\hat{a}$ during training, which leads to $\hat{a}$ more similar to the ground-truth alignment a^*. It can be seen from Table 5.2 that when we select a suitable λ that strikes a balance between the area distance and the score function, we can learn a better model than pure maximum likelihood learning (when $\lambda \to +\infty$).

5.4.3 Time Efficiency for Gradient Computation

We compare the time efficiency of computing the gradient with Automatic Differentiation (Autograd) [176], which computes the exact gradient by automatically back-propagation, and our approach in Table 5.3. We implement both of the methods using PyTorch framework and select 100 sequence pairs in every testing set at random to measure the average time. Since the computational time is approximately proportional to the sequence length, the standard deviation of computation time is included to show the impact of sequence length. We find that PALM is much more time efficient than Autograd among all the sequence length settings, where PALM only needs one-fourth of the time than the competing approach to approximate the gradient. In the extreme case where $|S| \in [400, +\infty)$ and $|T| \in [400, +\infty)$, the Autograd method takes roughly 5 minutes to compute the gradient for just one sequence pair while PALM only takes 1 minutes. Additionally, the standard deviation of the computing time for the competing approach is much higher than PALM, which means PALM is more stable to the variation of sequence lengths.

5.5 Summary

In this chapter, we present a novel method PALM for the generative pairwise protein sequence alignment problem, which is a two-step generative model based on the area distance between two alignments in the alignment matrix. We propose a novel lower bound of the log-likelihood as the objective and efficiently estimate the gradient during optimization by

Table 5.3. Time efficiency of computing the gradient among different testing sets. PALM is much time efficient than the competing method Autograd, which computes the exact gradient by automatically back-propagation, among all length intervals of two protein sequences.

Length of sequences		Averaged Run Time (sec)	
$\|S\|$	$\|T\|$	PALM	Autograd
$[1, 100]$	$[1, 100]$	$\mathbf{0.7 \pm 0.2}$	2.5 ± 0.8
$[100, 200]$	$[100, 200]$	$\mathbf{2.7 \pm 0.9}$	9.4 ± 3.3
$[100, 200]$	$[200, 400]$	$\mathbf{6.6 \pm 2.3}$	25.4 ± 9.4
$[200, 400]$	$[100, 200]$	$\mathbf{6.2 \pm 2.0}$	23.2 ± 8.1
$[200, 400]$	$[200, 400]$	$\mathbf{12.5 \pm 2.3}$	51.7 ± 11.2
$[400, +\infty)$	$[400, +\infty)$	$\mathbf{63.4 \pm 32.0}$	297.6 ± 282

dynamically sampling the alignments. We showed theoretically that PALM converges to the global optimum of the lower bound in a linear number of iterations. In experiments, PALM can generate sequence alignments with higher precision and recall than competing methods especially when proteins under consideration are remote homologies. We also show that the optimization of PALM is much more computationally efficient by dynamically sampling alignments than the automatic gradient differentiable algorithm. For future work, we plan to model the feature function with deep neural nets, while feeding more informative features. We are also active in finding more meaningful distance functions to model the difference between two alignments.

6. Symbolic Regression via Control Variable Genetic Programming

6.1 Introduction

Discovering scientific laws automatically from experiment data has been a grand goal of Artificial Intelligence (AI). Its success will greatly accelerate the pace of scientific discovery. Symbolic regression, *i.e.*, learning symbolic expressions from data, consists of a vital step in realizing this grand goal. Recently, exciting progress [177–184] has been made in this domain, especially with the aid of deep neural networks. Despite great achievements, state-of-the-art approaches are limited to learning relatively simple expressions, often involving a few independent variables. Regressing symbolic expressions involving multiple independent variables still remains out of reach of current approaches. The difficulty mainly lies in the exponentially large search space of symbolic expressions.

Our work attacks this major gap of symbolic regression, leveraging control variable experimentation – a classic procedure widely implemented in the science community [185, 186]. In the analysis of complex scientific phenomena involving many contributing factors, control variable experiments are conducted where a set of factors are held constant (*i.e.*, controlled variables), and the dependence between the output variable and the remaining input variables is studied [187, 188]. The result is a reduced-form expression that models the relationship only between the output and the non-controlled variables. Once the reduced-form equation is validated, scientists introduce more variables into play by freeing a few controlled variables in previous experiments. The new goal is to extend the previous equation to a general one including the newly introduced variables. This process continues until all independent variables are introduced.

Our proposed **C**ontrol **V**ariable **G**enetic **P**rogramming (CVGP) approach implements the aforementioned scientific discovery process using Genetic Programming (GP) for symbolic regression over many independent variables. The key insight of CVGP is to learn from *a customized set of control variable experiments*; in other words, the experiment data collection adapts to the learning process. This is in contrast to the current learning paradigm of most symbolic regression approaches, where they learn from a fixed dataset collected a priori.

In CVGP, first, we hold all independent variables except for one as constants and learn a symbolic expression that maps the single variable to the dependent variable using GP. GP maintains a pool of candidate equations and improves the fitness of these equations via mating, mutating, and selection over several generations. Mapping the dependence of one independent variable is easy. Hence GP can usually recover the ground-truth reduced-form equation. Then, CVGP frees one independent variable at a time. In each iteration, GP is used to modify the equations learned in previous generations to incorporate the new independent variable. This step is again conducted via mating, mutating, and selection. Such a procedure repeats until all the independent variables have been incorporated into the symbolic expression.

After discovering CVGP independently, the authors learned in private communications a line of research work [189–194] that also implemented the human scientific discovery process using AI, pioneered by the BACON systems developed by Langley, P. in 1978-1981 [189–191]. While BACON's discovery was driven by rule-based engines and our CVGP uses modern machine learning approaches such as genetic programming, indeed both approaches share a common vision – the *integration of experiment design and model learning* can further expedite scientific discovery.

Theoretically, we show CVGP as an incremental builder can reduce the exponential-sized search space for candidate expressions into a polynomial one when fitting a class of symbolic expressions. Experimentally, we show CVGP outperforms a number of state-of-the-art approaches on symbolic regression over multiple independent variables. Our contributions can be summarized as:

1. We propose CVGP, an incremental builder for symbolic regression over many independent variables. CVGP fits increasingly more complex equations via conducting control variable experiments with fewer and fewer controlled variables[1].

2. Theoretically, we show such an incremental builder as CVGP can reduce exponential-sized search spaces for symbolic regression to polynomial ones when searching for a class of symbolic expressions.

[1] ↑The code is at: https://github.com/jiangnanhugo/cvgp/.

3. Empirically, we demonstrate CVGP outperforms state-of-the-art symbolic regression approaches in discovering multi-variable equations from data.

6.2 Preliminaries

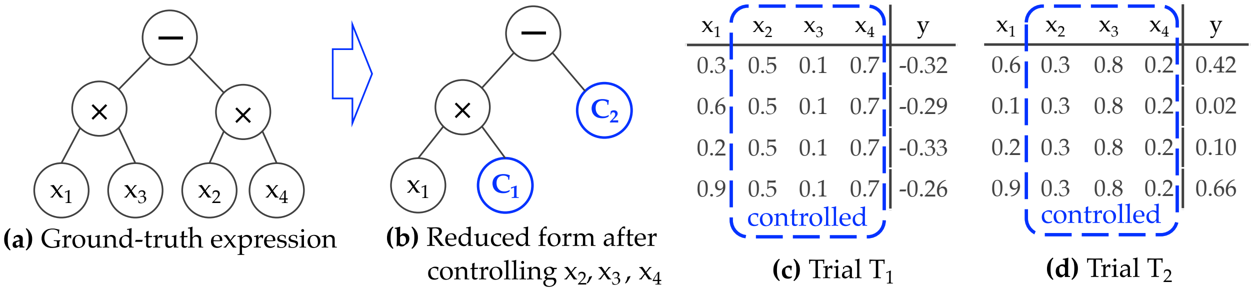
Symbolic Expression. A symbolic expression ϕ is expressed as variables and constants connected by a set of operators. Variables are allowed to vary while constants remain the same. Each operand of an operator is either a variable, a constant, or a self-contained symbolic expression. A symbolic expression can also be drawn as a tree, where variables and constants reside in leaves, and operators reside in inner nodes. See Figure 6.1(a) for an example. in this chapter, we deal with expressions involving real numbers. The semantic meaning of a symbolic expression follows its standard definition in arithmetics.

Symbolic Regression. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and a loss function $\ell(\cdot, \cdot)$, where $\mathbf{x}_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$, the objective of symbolic regression (SR) is to search for the optimal symbolic expression ϕ^* within the space of all candidate expressions Π that minimizes the average loss:

$$\phi^* \leftarrow \arg\min_{\phi \in \Pi} \frac{1}{n} \sum_{i=1}^{n} \ell(\phi(\mathbf{x}_i), y_i), \tag{6.1}$$

in addition to regularization terms. Symbolic regression is challenging and is shown to be NP-hard [195], due to the exponentially large space of candidate symbolic expressions.

Genetic Programming for Symbolic Regression. Genetic Programming (GP) has been a popular method to solve symbolic regression. Recently, a few other approaches based on neural networks surpassed the performance of GP in symbolic regression. We leave the discussions of these methods to the related work section. The high-level idea of GP is to maintain a pool of candidate symbolic expressions. In each generation, candidate expressions are *mutated* with probability P_{mu} and *mated* with probability P_{ma}. Then in the *selection* step, those with the highest fitness scores, measured by how each expression predicts the output from the input, are selected as the candidates for the next generation, together with a few randomly chosen ones to maintain diversity. After several generations, expressions with high fitness scores, *i.e.*, those fit data well survive in the pool of candidate solutions. The best expressions found in all generations are recorded as *hall-of-fame* solutions.



(a) Ground-truth expression **(b)** Reduced form after controlling x_2, x_3, x_4

x_1	x_2	x_3	x_4	y
0.3	0.5	0.1	0.7	-0.32
0.6	0.5	0.1	0.7	-0.29
0.2	0.5	0.1	0.7	-0.33
0.9	0.5	0.1	0.7	-0.26

(c) Trial T_1

x_1	x_2	x_3	x_4	y
0.6	0.3	0.8	0.2	0.42
0.1	0.3	0.8	0.2	0.02
0.2	0.3	0.8	0.2	0.10
0.9	0.3	0.8	0.2	0.66

(d) Trial T_2

Figure 6.1. An example of two trials of a control variable experiment. **(a)** The data of the experiment is generated by the ground-truth expression $\phi = x_1x_3 - x_2x_4$. **(b)** If we control $\mathbf{v}_c = \{x_2, x_3, x_4\}$ and only allow $\mathbf{v}_f = \{x_1\}$ to vary, it *looks like* the data are generated from the reduced-form equation $\phi' = C_1x_1 - C_2$. **(c, d)** The generated data in two trials of the control variable experiments. The controlled variables are fixed within each trial but vary across trials.

6.3 Control Variable Genetic Programming

In this section, we present our control variable genetic programming algorithm. Before we dive into the algorithm description, we first need to study what are the outcomes of a control variable experiment and what conclusions we can draw on the symbolic regression expression by observing such outcomes.

6.3.1 Control Variable Experiment

A control variable experiment $\texttt{CVExp}(\phi, \mathbf{v}_c, \mathbf{v}_f, \{T_k\}_{k=1}^K)$ consists of the trial symbolic expression ϕ, a set of controlled variables $\mathbf{v}_c$, a set of free variables $\mathbf{v}_f$, and K trial experiments $T_1, \ldots, T_K$. The expression ϕ may have zero or multiple *open constants*. The values of open constants are determined by fitting the equation to the training data.

One Trial in a Control Variable Experiment. A single trial of a control variable experiment T_k fits the symbolic expression ϕ with a batch of data. To avoid abusing notations, we also use T_k to denote the batch of data. In the generated data T_k, every controlled variable is fixed to the same value while the free variables are set randomly. We assume the values of the dependent variables in a batch are (noisy observations) of the ground-truth expressions with the values of independent variables set in the batch. In science, this step is achieved by conducting real-world experiments, *i.e.*, controlling independent variables, and performing measurements on the dependent variable.

For example, Figure 6.1(c,d) demonstrates two trials ($K = 2$) of a control variable experiment in which variable x_2, x_3, x_4 are controlled, *i.e.*, $\mathbf{v}_c = \{x_2, x_3, x_4\}$. They are fixed to one value in trial T_1 (in Figure 6.1(c)) and another value in trial T_2 (in Figure 6.1(d)). x_1 is the only free variable, *i.e.*, $\mathbf{v}_f = \{x_1\}$.

Reduced-form Expression in a Control Variable Setting. We assume there is a ground-truth symbolic expression that produces the experiment data. In other words, the observed output is the execution of the ground-truth expression from the input, possibly in addition to some noise. In control variable experiments, because the values of controlled variables are fixed in each trial, what we observe is the ground-truth expression in its *reduced form*, where sub-expressions involving only controlled variables are replaced with constants.

Figure 6.1(b) provides an example of the reduced form expression. Assume the data is generated from the ground-truth expression in Figure (a): $\phi = x_1 x_3 - x_2 x_4$. When we control the values of variable in $\mathbf{v}_c = \{x_2, x_3, x_4\}$, the data *looks like* they are generated from the *reduced* expression: $\phi' = C_1 x_1 - C_2$. We can see both C_1 and C_2 hold constant values in each trial. However, their values vary across trials because the values of controlled variables change. In trial T_1, when x_2, x_3, and x_4 are fixed to 0.5, 0.1, 0.7, C_1 takes the value of x_3, *i.e.*, 0.1. C_2 takes the value of $x_2 x_4$, *i.e.*, 0.35. In trial T_2, $C_1 = 0.8$ and $C_2 = 0.06$.

We call constants which represent sub-expressions involving controlled variables in the ground-truth expression *summary constants*, and refer to constants in the ground-truth expression *stand-alone constants*. For example, C_1 and C_2 in Figure 6.1(b) are both summary constants, because C_1 replaces the controlled variable x_3 and C_2 replaces a sub-expression $x_2 x_4$ in the ground-truth expression. Notice the types of constants are *unknown* in the process of fitting an expression to control variable experiment data. However, the best-fitted values of these constants across several trials reveal important information: a constant is probably a summary constant if its fitted values vary greatly across trials, while a constant that remains the almost same value across trials is probably stand-alone.

Outcome of a Single Trial. The outcomes of the k-th trial are two-fold: (1) the values of the constants that best fit the given batch of data. We denote these values as vector $\mathbf{c}_k$.

(2) the fitness score measuring the goodness-of-fit, denoted as o_k. One typical fitness score is the mean squared error (MSE). See Appendix F.3.1 for the exact definition of MSE. For the example in Figure 6.1, if we fit the reduced expression in (b) to data in trial T_1, the best-fitted values are $\mathbf{c}_1 = (C_1 = 0.1, C_2 = 0.35)$. For trial T_2, the best-fitted values are $\mathbf{c}_2 = (C_1 = 0.8, C_2 = 0.06)$. In both trials, the fitness scores (i.e., the MSE value) are 0, indicating no errors.

Outcome of Multiple trials. We let the values of control variables vary across different trials. This corresponds to changing experimental conditions in real science experiments. The outcomes of an experiment with K trials are: (1) $\phi.\mathbf{o} = (o_1, \ldots, o_K)$, where each o_k is the fitness score of trial k and (2) $\phi.\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_K)$, the best-fitted values to open constants across trials.

Critical information is obtained by examining the outcomes of multi-trial control variable experiments: (1) consistent close-to-zero fitness scores $\{o_1, \ldots, o_K\}$ suggest the fitted expression is close to the ground-truth equation in the reduced form. (2) given the equation is close to the ground truth, an open constant having similar best-fitted values across K trials $\{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$ suggests the open constants are stand-alone.

6.3.2 Control Variable Genetic Programming

The high-level idea of the CVGP algorithm is to build more complex symbolic expressions involving more and more variables based on control variable experiments with fewer and fewer controlled variables.

Algorithm 5: Control Variable Genetic Programming (CVGP)

```
Input: GP pool size $M$; #generations #Gen; #trials $K$; #expressions in
       hall-of-fame set #Hof; mutate probability $P_{mu}$; mate probability $P_{ma}$;
       operand set $O_p$
. $\mathbf{v}_c \leftarrow \{x_1, \ldots, x_m\}$;          $\mathbf{v}_f \leftarrow \emptyset$;
$\mathcal{P}_{gp} \leftarrow$ CreateInitGPPool($M$);
for $x_i \in \{x_1, \ldots, x_m\}$ do
    $\mathbf{v}_c \leftarrow \mathbf{v}_c \setminus \{x_i\}$;   $\mathbf{v}_f \leftarrow \mathbf{v}_f \cup \{x_i\}$ ;            // Set $x_i$ to be free variable
    $\mathcal{D}^o_i \leftarrow$ DataOracle($\mathbf{v}_c, \mathbf{v}_f$);
    for $\phi \in \mathcal{P}_{gp}$ do
        $\{T_k\}_{k=1}^K \leftarrow$ GenData($\mathcal{D}^o_i$) ;         // Query Oracle for the trial data
        $\phi.\mathbf{o}, \phi.\mathbf{c} \leftarrow$ CVExp($\phi, \mathbf{v}_c, \mathbf{v}_f, \{T_k\}_{k=1}^K$) ;   // Control variable experiments
    $\mathcal{P}_{gp}, \mathcal{H} \leftarrow$ GP($\mathcal{P}_{gp}, \mathcal{D}^o_i, K, M,$ #Gen, #Hof$, P_{mu}, P_{ma}, O_p \cup \{\text{const}, x_i\}$);
    for $\phi \in \mathcal{P}_{gp}$ do
        FreezeEquation($\phi, \phi.\mathbf{o}, \phi.\mathbf{c}$);
return The set of hall-of-fame equations $\mathcal{H}$.
```

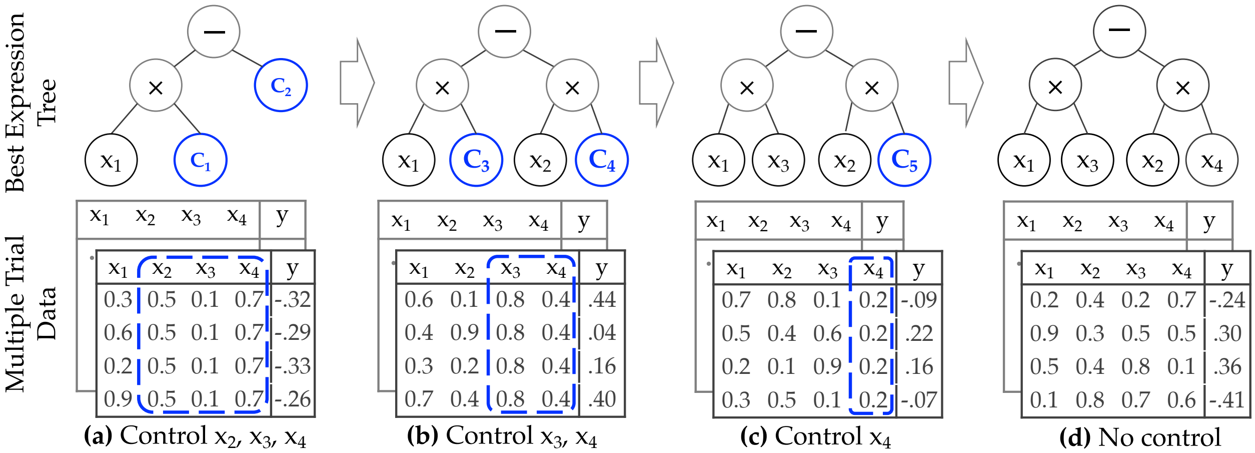
To fit an expression of m variables, initially, we control the values of all $m-1$ variables and allow only one variable to vary. Using Genetic Programming (GP), we find a pool of expressions $\{\phi_{1,1}, \ldots, \phi_{1,M}\}$ which best fit the data from this controlled experiment. Notice $\{\phi_{1,1}, \ldots, \phi_{1,M}\}$ are restricted to contain the only one free variable. This fact renders fitting them a lot easier than fitting the expressions involving all m variables. Next, for each $\phi_{1,l}$, we examine (1) if the errors of the fitting are consistently small across all trials. A small error implies $\phi_{1,l}$ is close to the ground-truth formula reduced to the one free variable. We hence freeze all operands of $\phi_{1,l}$ in this case. Freezing means GP in later steps cannot change these operands. (2) In the case of a small fitting error, we also inspect the best-fitted values of each open constant in $\phi_{1,l}$ across different trials. The constant is probably a summary constant if its values vary across trials. In other words, these constants represent sub-expressions involving the controlled variables. We thus mark these constants as *expandable* for later steps. The remaining constants are probably stand-alone. Therefore we also freeze them.

After the first step, CVGP adds a second free variable and starts fitting $\{\phi_{2,1}, \ldots, \phi_{2,M}\}$ using the data from control variable experiments involving the two free variables. Similar to the previous step, all $\phi_{2,l}$ are restricted to only contain the two free variables. Moreover, they can only be mated or mutated by GP from the first generation $\{\phi_{1,1}, \ldots, \phi_{1,M}\}$. The mutation can only happen on non-frozen nodes. After GP, a similar inspection is conducted for every equation in the GP pool, and corresponding variables and/or operands are frozen. This process continues to involve more and more variables. Eventually, the expressions in the GP pool consider all m variables.

The whole procedure of CVGP is shown in Algorithm 5. Here, $x_1, \ldots, x_m$ are moved from the controlled to free variables in numerical order. We agree other orders may boost its performance even further. However, we leave the exploration of this direction as future work. When a new variable becomes free, the control variable experiment `CVExp` needs to be repeated for every equation ϕ in the GP pool $\mathcal{P}_{gp}$ (Line 5-9 in Algorithm 5). This is because the fitness scores and the fitted open constant values will both change when the set of controlled variables is updated. Then function `GP` is called. `GP` is a minimally modified genetic programming algorithm for symbolic regression whose pseudo-code is in Algorithm 6. The only differences are that it uses data from control variable experiments and the mutation operation at step i only allows to use all operands, the constant node, and variable x_i at non-frozen nodes. Finally, in Lines 12-14 of Algorithm 5, `FreezeEquation` is called for every equation in the GP pool. The high-level idea of freezing is discussed above. $\mathcal{H}$ is returned as the set of "hall of fame" expressions.

Figure 6.2 shows the high-level idea of fitting an equation using CVGP. Here the process has four stages, each stage with a decreased number of controlled variables. The trial data in each stage is shown at the bottom and the best expression found is shown at the top. The expandable constants are bold and blue. The readers can see how the fitted equations grow into the final ground-truth equation, with one free variable added at a time.

The Availability of a Data Oracle. A crucial assumption behind the success of CVGP is the availability of a data oracle $\mathcal{D}^o$ that returns a (noisy) observation of the dependent output variable with input variables in $\mathbf{v}_c$ controlled and $\mathbf{v}_f$ free. This differs from the classical setting of symbolic regression, where a dataset is obtained before learning [196, 197]. Such a



x_1	x_2	x_3	x_4	y
0.3	0.5	0.1	0.7	-.32
0.6	0.5	0.1	0.7	-.29
0.2	0.5	0.1	0.7	-.33
0.9	0.5	0.1	0.7	-.26

(a) Control x_2, x_3, x_4

x_1	x_2	x_3	x_4	y
0.6	0.1	0.8	0.4	.44
0.4	0.9	0.8	0.4	.04
0.3	0.2	0.8	0.4	.16
0.7	0.4	0.8	0.4	.40

(b) Control x_3, x_4

x_1	x_2	x_3	x_4	y
0.7	0.8	0.1	0.2	-.09
0.5	0.4	0.6	0.2	.22
0.2	0.1	0.9	0.2	.16
0.3	0.5	0.1	0.2	-.07

(c) Control x_4

x_1	x_2	x_3	x_4	y
0.2	0.4	0.2	0.7	-.24
0.9	0.3	0.5	0.5	.30
0.5	0.4	0.8	0.1	.36
0.1	0.8	0.7	0.6	-.41

(d) No control

Figure 6.2. Running example of Algorithm 5. **(a)** Initially, a reduced-form equation $\phi' = C_1 x_1 - C_2$ is found via fitting control variable data in which x_2, x_3, x_4 are held as constants and only x_1 is allowed to vary. Two leaves nodes C_1, C_2 are as summary constants (colored blue). **(b)** This equation is expanded to $C_3 x_1 - C_4 x_2$ in the second stage via fitting the data in which only x_3, x_4 are held as constants. **(c,d)** This process continues until the ground-truth equation $\phi = x_1 x_3 - x_2 x_4$ is found. The data generated for control variable experiment trials in each stage are shown at the bottom.

data oracle represents conducting control variable experiments in the real world, which can be expensive.

However, we argue that the integration of experiment design in the discovery of scientific knowledge is indeed the main driver of the successes of CVGP. This idea has received tremendous success in early works [189–191] but unfortunately has been largely forgotten in today's symbolic regression community. Our work does not intend to show the superiority of one approach. Instead, we would like to point out that carefully designed experiments can improve any method, and GP is used as an example. We acknowledge that fully controlled experiments may be difficult in some scenarios. In cases where it is difficult to obtain such a data oracle, We leave such effort as future work.

6.3.3 Theoretical Analysis

We show in this section that the idea of control variable experiments may bring an exponential reduction in the search space for particular classes of symbolic expressions. To

Algorithm 6: $\texttt{GP}(\mathcal{P}_{gp}, \mathcal{D}^o, K, M, \texttt{\#Gen}, \texttt{\#Hof}, P_{mu}, P_{ma}, O_p)$

```
Input: Initial GP Pool $\mathcal{P}_{gp}$; data Oracle $\mathcal{D}^o$; #trials $K$; GP pool size $M$;
       #generations #Gen; #expressions in hall-of-fame set #Hof; mutate
       probability $P_{mu}$; mate probability $P_{ma}$; mutation node library $O_p$
for $j \leftarrow 1$ to #Gen do
    $\mathcal{P}_{new} \leftarrow \emptyset$;
    for $\phi \in \mathcal{P}_{gp}$ do
        if with probability $P_{mu}$ then
            $\phi \leftarrow \texttt{Mutate}(\phi, O_p)$; ;                                  // Mutation
            $\{T_k\}_{k=1}^{K} \leftarrow \texttt{GenData}(\mathcal{D}^o)$;
            $\phi.\mathbf{o}$, $\phi.\mathbf{c} \leftarrow \texttt{CVExp}(\phi, \mathbf{v}_c, \mathbf{v}_f, \{T_k\}_{k=1}^{K})$;
        $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\phi\}$;
    $\mathcal{P}_{gp} \leftarrow \mathcal{P}_{new}$;
    $\mathcal{P}_{new} \leftarrow \emptyset$;
    for $\phi_l, \phi_{l+1} \in \mathcal{P}_{gp}$ do
        if with probability $P_{ma}$ then
            $\phi_l, \phi_{l+1} \leftarrow \texttt{Mate}(\phi_l, \phi_{l+1})$; ;                          // Mating
            $\{T_k\}_{k=1}^{K} \leftarrow \texttt{genData}(\mathcal{D}^o)$;
            $\phi_l.\mathbf{o}, \phi_l.\mathbf{c} \leftarrow \texttt{CVExp}(\phi_l, \mathbf{v}_c, \mathbf{v}_f, \{T_k\}_{k=1}^{K})$;
            $\phi_{l+1}.\mathbf{o}, \phi_{l+1}.\mathbf{c} \leftarrow \texttt{CVExp}(\phi_{l+1}, \mathbf{v}_c, \mathbf{v}_f, \{T_k\}_{k=1}^{K})$;
            $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\phi_l, \phi_{l+1}\}$;
    $\mathcal{H} \leftarrow \texttt{TopK}(\mathcal{P}_{new} \cup \mathcal{H}, K = \texttt{\#Hof})$ ;        // Update the hall of fame set.
    $\mathcal{P}_{gp} \leftarrow \texttt{selection}(\mathcal{P}_{new}, M)$;
Return GP pool and hall-of-fame $\mathcal{P}_{gp}, \mathcal{H}$
```

see this, we assume the learning algorithm follows a search order from simple to complex symbolic expressions and the data is noiseless.

Definition 6.3.1. *The search space of symbolic expression trees of l nodes $S(l)$* is the set of all symbolic expression trees involving at most l nodes.

Lemma 6.3.2. *For simplicity, assume all operands are binary, and let o be the number of operands and m be the number of input variables. The size of the search space of symbolic expression trees of l nodes scales exponentially; more precisely at $\mathcal{O}((4(m+1)o)^{\frac{l-1}{2}})$ and $\Omega((4(m+1)o)^{\frac{l-1}{4}})$.*

The proof of Lemma 6.3.2 mainly involves counting binary trees. We leave its detailed proof in Appendix D.1. For our purposes, it is sufficient to know the size is exponential in the size of expression tree l.

Definition 6.3.3 (Simple to complex search order). A symbolic regression algorithm follows a simple to complex search order if it expands its search space from short to long symbolic expressions; *i.e.*, first search for the best symbolic expressions in $S(1)$, then in $S(2) \setminus S(1)$, etc.

In general, it is difficult to quantify the search order of any symbolic regression algorithms. However, we believe the simple to complex order reflects the search procedures of a large class of symbolic regression algorithms, including our CVGP. In fact, [198] explicitly use regularizers to promote the search of simple and short expressions. Our CVGP follows the simple to complex search order approximately. Indeed, it is possible that genetic programming encounters more complex equations before their simpler counterparts. However, in general, the expressions are built from simple to complex equations by mating and mutating operations in genetic programming algorithms.

Proposition 6.3.4 (Exponential Reduction in the Search Space). *There exists a symbolic expression ϕ of $(4m-1)$ nodes, a normal symbolic regression algorithm following the simple to complex search order has to explore a search space whose size is exponential in m to find the expression, while CVGP following the simple to complex order only expands $\mathcal{O}(m)$ constant-sized search spaces.*

Proof. Consider a dataset generated by the ground-truth symbolic expression made up of 2 operands $(+, \times)$, $2m$ input variables and $(4m-1)$ nodes:

$$(x_1 + x_2)(x_3 + x_4) \ldots (x_{2m-1} + x_{2m}). \tag{6.2}$$

To search for this symbolic regression, a normal algorithm following the simple to complex order needs to consider all expression trees up to $(4m-1)$ nodes. According to Lemma 6.3.2, the normal algorithm has a search space of at least $\Omega((16m+8)^{m-1/2})$, which is exponential in m.

On the other hand, in the first step of CVGP, $x_2, \ldots, x_{2m}$ are controlled and only x_1 is free. In this case, the ground-truth equation in the reduced form is

$$(x_1 + C_1)D_1, \tag{6.3}$$

in which both C_1 and D_1 are summary constants. Here C_1 represents x_2 and D_1 represents $(x_3+x_4)\ldots(x_{2m-1}+x_{2m})$ in the control variable experiments. The reduced equation is quite simple under the controlled environment. CVGP should be able to find the ground-truth expression exploring search space $S(5)$.

Proving using induction. In step $2i$ ($1 \leq i \leq m$), variables $x_{2i+1}, x_{2i+2}, \ldots, x_{2m}$ are held as constants, $x_1, \ldots, x_{2i}$ are allowed to vary. The ground-truth expression in the reduced form found in the previous $(2i-1)$-th step is:

$$(x_1 + x_2)\ldots(x_{2i-1} + C_{2i-1})D_{2i-1}. \tag{6.4}$$

CVGP needs to extend this equation to be the ground-truth expression in the reduced form for the $2i$-th step, which is:

$$(x_1 + x_2)\ldots(x_{2i-1} + x_{2i})D_{2i}. \tag{6.5}$$

We can see the change is to replace the summary constant C_{2i-1} to x_{2i}. Assume the data is noiseless and CVGP can confirm expression (6.4) is the ground-truth reduced-form expression for the previous step. This means all the operands and variables will be frozen by CVGP, and only C_{2i-1} and D_{2i-1} are allowed to be replaced by new expressions. Assume CVGP follows the simple to complex search order, it should find the ground-truth expression (6.5) by searching replacement expressions of lengths up to 1.

Similarly, in step $2i+1$, assume CVGP confirms the ground-truth expression in the reduced form in step $2i$, CVGP also only needs to search in constant-sized spaces to find the new ground-truth expression. Overall, we can see only $\mathcal{O}(m)$ searches in constant-sized spaces are required for CVGP to find the final ground-truth expression.

6.4 Related Work

Symbolic Regression. Symbolic Regression is proven to be NP-hard [195], due to the search space of all possible symbolic expressions being exponential in the number of input variables. Early works in this domain are based on heuristic search [199, 200]. Genetic programming turns out to be effective in searching for good candidates of symbolic expressions [178, 182, 184, 201]. Reinforcement learning-based methods propose a risk-seeking policy gradient to find the expressions [180–182]. Other works use RL to adjust the probabilities of genetic operations [202]. Also, there are works that reduced the combinatorial search space by considering the composition of base functions, *e.g.* Fast function extraction [203] and elite bases regression [204]. In terms of the families of expressions, research efforts have been devoted to searching for polynomials with single or two variables [205], time series equations [206], and also equations in physics [201]. Multi-variable symbolic regression is more challenging because the search space increases exponentially with respect to the number of independent variables. Existing works for multi-variable regression are mainly based on pre-trained encoder-decoder methods with a massive training dataset (e.g., millions of datasets [207]), and even larger generative models (e.g., about 100 million parameters [208]). Our CVGP is a tailored algorithm to solve multi-variable symbolic regression problems.

AI-driven Scientific Discovery. Recently AI has been highlighted to enable scientific discoveries in diverse domains [209, 210]. Early work in this domain focuses on learning logic (symbolic) representations [211, 212]. Recently, learning Partial Differential Equations (PDEs) from data has also been studied extensively [213–223]. In this domain, a line of works develops robots that automatically refine the hypothesis space, some with human interactions [192, 193, 224]. These works are quite related to ours because they also actively probe the hypothesis spaces, albeit they are in biology and chemistry.

Active Learning and Reasoning. Active learning considers querying data points actively to maximize the learning performance [225, 226]. Our approach is related to active learning because control variable experiments can be viewed as a way to actively collect data. However, besides active data collection, our CVGP builds simple to complex models, which is not in active learning.

Meta-reasoning – Thinking Fast and Slow. The co-existence of fast and slow cognition systems marks an interesting side of human intelligence [227–229]. Our CVGP is motivated by this dual cognition process. In essence, we argue instead of entirely relying on the brute-force way of learning using big data and heavy computation (fast thinking), incrementally expanding from reduced-form equations to the full equation may result in better outcomes (slow thinking).

Causality. Control variable experiments are closely related to the idea of intervention, which is commonly used to discover causal relationships [230–234]. However, we mainly use control variable experiments to accelerate symbolic regression, which still identifies correlations.

6.5 Experiments

In this section, we demonstrate CVGP finds the symbolic expressions with the smallest Normalized Mean-Square Errors (NMSE) among all 7 competing approaches on 21 noiseless benchmark datasets (in Table 6.1) and 20 noisy benchmark datasets (in Table 6.2). In the ablation studies, we show our CVGP is consistently better than the baselines when evaluated in different evaluation metrics, evaluating different quantiles of the NMSE metric, with different amounts of Gaussian noise added to the data (Figure 6.3, more complete results in Figure D.1 and D.2 in the appendix). In Table 6.3, we show our CVGP has a higher rate of recovering the ground-truth expressions than baselines.

6.5.1 Experimental Settings

Datasets. To highlight the performance of CVGP in regressing multi-variable expressions, we consider synthesized datasets, involving randomly generated expressions with multiple variables. A dataset is labeled by the ground-truth equation that generates it. The ground-truth equations we consider are multi-variable polynomials characterized by their operands and a tuple (a, b, c). Here a is the number of independent variables. b is the number of singular terms. A singular term can be an independent variable, such as x_1, or a unary operand on a variable, such as $\sin(x_1)$. c is the number of cross terms. They look like $C_1x_3x_4$ or $C_2\sin(x_1)\texttt{inv}(x_5)$, etc. Here C_1, C_2 are randomly generated constants. The tuples and operands listed in different tables and charts indicate how the ground-truth expressions are generated. For each dataset configuration, we repeat our experiments 10 times, each time with a randomly generated symbolic expression of the given configuration. For noiseless datasets, the output is exactly the evaluation of the ground-truth expression. For noisy datasets, the output is further perturbed by Gaussian noise of zero means and a given standard deviation.

Remarks on Public Available Datasets. Most public datasets are black-box [235], containing randomly generated input and output pairs of an unknown symbolic equation. The point of our work is to show that customized collected control variable experiment data improves symbolic regression, and hence we cannot use these randomly generated data. In addition, most datasets are on equations of a small number of independent variables. We intentionally test on benchmark sets involving many variables to highlight our approach.

Evaluation. In terms of the evaluation metric, the median (50%) and 75%-percentile of the NMSE across these 10 experiments are reported. We choose to report median values instead of mean due to outliers (see box plots in Figure 6.3(a-d)). This is a common practice for combinatorial optimization problems. The mathematical definition of NMSE and other NRMSE, MSE, RMSE metrics are presented in Appendix D.2.2.

Baselines. We consider the following baselines based on evolutionary algorithms: 1) Genetic Programming (GP) [236]. 2) Eureqa [237]. We also consider a series of baselines using reinforcement learning: 3) Priority queue training (PQT) [238]. 4) Vanilla Policy Gradient (VPG) that uses the REINFORCE algorithm [239] to train the model. 5) Deep Symbolic Regression (DSR) [180]. 6) Neural-Guided Genetic Programming Population Seeding (GP-Meld) [181].

We leave detailed descriptions of the configurations of our CVGP and baseline algorithms in the appendix and only mention a few implementation notes here. We implemented GP and CVGP. They use a data oracle, which returns (noisy) observations of the ground-truth equation when queried with inputs. We cannot implement the same oracle for other baselines because of code complexity and/or no available code. To ensure fairness, the sizes of the

Table 6.1. Median (50%) and 75%-quantile NMSE values of the symbolic expressions found by all the algorithms on several *noiseless* benchmark datasets. Our CVGP finds symbolic expressions with the smallest NMSEs.

Dataset	CVGP (ours)		GP		DSR		PQT		VPG		GPMeld		Eureqa	
configs	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
(2,1,1)	**< 1e-6**	**<1e-6**	2.19*e*-3	7.91*e*-2	3.507	4.787	0.262	1.16	0.359	19.16	0.273	1.356	**<1e-6**	**<1e-6**
(3,2,2)	0.001	0.004	0.015	0.135	1.53	43.09	0.58	1.13	0.83	1.32	1.06	2.18	**<1e-6**	**<1e-6**
(4,4,6)	**0.008**	0.059	0.012	**0.054**	1.006	1.249	1.006	2.459	1.221	2.322	1.127	2.286	1.191	6.001
(5,5,5)	**0.011**	**0.019**	0.025	0.177	1.038	8.805	1.048	4.736	1.401	38.26	1.008	1.969	0.996	6.340
(5,5,8)	**0.007**	**0.013**	0.010	0.017	1.403	5.161	1.530	41.27	4.133	27.42	1.386	8.092	1.002	1.495
(6,6,8)	**0.044**	**0.074**	0.058	0.200	1.963	90.53	4.212	8.194	4.425	22.91	15.58	269.6	1.005	1.150
(6,6,10)	**0.012**	**0.027**	0.381	0.820	1.021	1.036	1.006	1.048	1.003	1.020	1.022	1.689	1.764	49.041

(a) Datasets containing operands $\{\texttt{inv}, +, -, \times\}$

Dataset	CVGP (ours)		GP		DSR		PQT		VPG		GPMeld		Eureqa	
configs	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
(2,1,1)	1.06*e*-4	6.69*e*-2	7.56*e*-4	7.72*e*-2	1.87	8.16	0.20	0.22	2.38	6.14	**<1e-6**	**<1e-6**	**<1e-6**	**<1e-6**
(3,2,2)	0.005	0.123	0.023	0.374	0.087	0.392	0.161	0.469	0.277	0.493	0.112	0.183	**<1e-6**	**<1e-6**
(4,4,6)	**0.028**	0.132	0.044	**0.106**	2.815	9.958	2.381	13.844	2.990	11.316	1.670	2.697	0.024	0.122
(5,5,5)	0.086	0.402	**0.063**	**0.232**	2.558	3.313	2.168	2.679	1.903	2.780	1.501	2.295	0.158	0.377
(5,5,8)	**0.014**	**0.066**	0.102	0.683	2.535	2.933	2.482	2.773	2.440	3.062	2.422	3.853	0.284	0.514
(6,6,8)	**0.066**	**0.166**	0.127	0.591	0.936	1.079	0.983	1.053	0.900	1.018	0.964	1.428	0.433	1.564
(6,6,10)	**0.104**	**0.177**	0.159	0.230	6.121	16.32	5.750	16.29	3.857	19.82	7.393	21.709	0.910	1.927

(b) Datasets containing operands $\{\sin, \cos, +, -, \times\}$.

Dataset	CVGP (ours)		GP		DSR		PQT		VPG		GPMeld		Eureqa	
configs	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
(2,1,1)	**<1e-6**	0.004	**<1e-6**	0.76	0.032	4.778	0.038	4.782	0.115	4.095	0.008	5.859	**<1e-6**	**<1e-6**
(3,2,2)	0.039	0.083	0.043	0.551	0.227	7.856	0.855	2.885	0.233	0.400	0.944	1.263	**<1e-6**	**<1e-6**
(4,4,6)	**0.015**	**0.121**	0.042	0.347	1.040	1.155	1.039	1.055	1.049	1.068	1.886	4.104	0.984	1.196
(5,5,5)	**0.038**	**0.097**	0.197	0.514	3.892	69.98	4.311	23.66	5.542	8.839	9.553	16.92	0.901	1.007
(5,5,8)	**0.050**	**0.102**	0.111	0.177	2.379	2.526	1.205	2.336	1.824	2.481	1.142	1.874	1.002	2.445
(6,6,8)	**0.029**	**0.038**	0.091	0.151	1.605	8.005	1.718	7.783	4.691	39.03	1.398	16.60	1.001	1.008
(6,6,10)	**0.018**	**0.113**	0.087	0.194	2.083	23.57	1.797	4.521	1.888	35.45	2.590	8.784	1.001	1.008

(c) Datasets containing operands $\{\sin, \cos, \texttt{inv}, +, -, \times\}$.

training datasets we use for those baselines are larger than the total number of data points accessed in the full execution of those algorithms. In other words, their access to data would have no difference if the same oracle has been implemented for them because it does not affect the executions whether the data is generated ahead of the execution or on the fly. The reported NMSE scores in all charts and tables are based on separately generated data that have never been used in training. The threshold to freeze operands in CVGP is if the MSE to fit a data batch is below 0.01. The threshold to freeze the value of a constant in CVGP is if the variance of best-fitted values of the constant across trials drops below 0.001.

6.5.2 Experimental Analysis

Learning Result. Our CVGP attains the smallest median (50%) and 75%-quantile NMSE values among all the baselines mentioned in Section 6.5.1, when evaluated on noiseless datasets (Table 6.1) and noisy datasets (Table 6.2). This shows our method can better

Table 6.2. Median (50%) and 75%-quantile NMSE values of the symbolic expressions found by all the algorithms on several *noisy* benchmark datasets (Gaussian noise with zero mean and standard deviation 0.1 is added). Our CVGP finds symbolic expressions with the smallest NMSEs.

Dataset	CVGP (ours)		GP		DSR		PQT		VPG		GPMeld	
configs	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
(2,1,1)	0.198	0.490	**0.024**	**0.053**	0.032	3.048	0.029	0.953	0.041	0.678	0.387	22.806
(4,4,6)	**0.036**	**0.088**	0.038	0.108	1.163	3.714	1.016	1.122	1.087	1.275	1.058	1.374
(5,5,5)	0.076	0.126	**0.075**	**0.102**	1.028	2.270	1.983	4.637	1.075	2.811	1.479	2.855
(5,5,8)	**0.061**	**0.118**	0.121	0.186	1.004	1.013	1.005	1.006	1.002	1.009	1.108	2.399
(6,6,8)	**0.098**	**0.144**	0.104	0.167	1.006	1.027	1.006	1.020	1.009	1.066	1.035	2.671
(6,6,10)	**0.055**	**0.097**	0.074	0.132	1.003	1.009	1.005	1.008	1.004	1.015	1.021	1.126

(a) Datasets containing operands $\{\sin, \cos, \texttt{inv}, +, -, \times\}$.

Dataset configs	CVGP 50%	CVGP 75%	GP 50%	GP 75%	DSR 50%	DSR 75%	PQT 50%	PQT 75%	VPG 50%	VPG 75%	GPMeld 50%	GPMeld 75%
(2,1,1)	**0.049**	0.812	0.103	0.263	0.069	0.144	0.066	**0.093**	0.094	0.416	0.066	0.118
(3,2,2)	**0.098**	**0.165**	0.108	0.425	0.350	0.713	0.351	1.831	0.439	0.581	0.102	0.597
(4,4,6)	**0.078**	**0.121**	0.120	0.305	7.056	16.321	5.093	19.429	2.458	13.762	2.225	3.754
(5,5,5)	**0.067**	**0.230**	0.091	0.313	32.45	234.31	36.797	229.529	14.435	46.191	28.440	421.63
(5,5,8)	**0.113**	**0.207**	0.119	0.388	195.22	573.33	449.83	565.69	206.06	629.41	363.79	666.57
(6,6,8)	**0.170**	**0.481**	0.186	0.727	1.752	3.824	4.887	15.248	2.396	7.051	1.478	6.271
(6,6,10)	**0.161**	**0.251**	0.312	0.342	11.678	26.941	5.667	24.042	7.398	25.156	11.513	28.439

(b) Datasets containing operands $\{\sin, \cos, +, -, \times\}$.

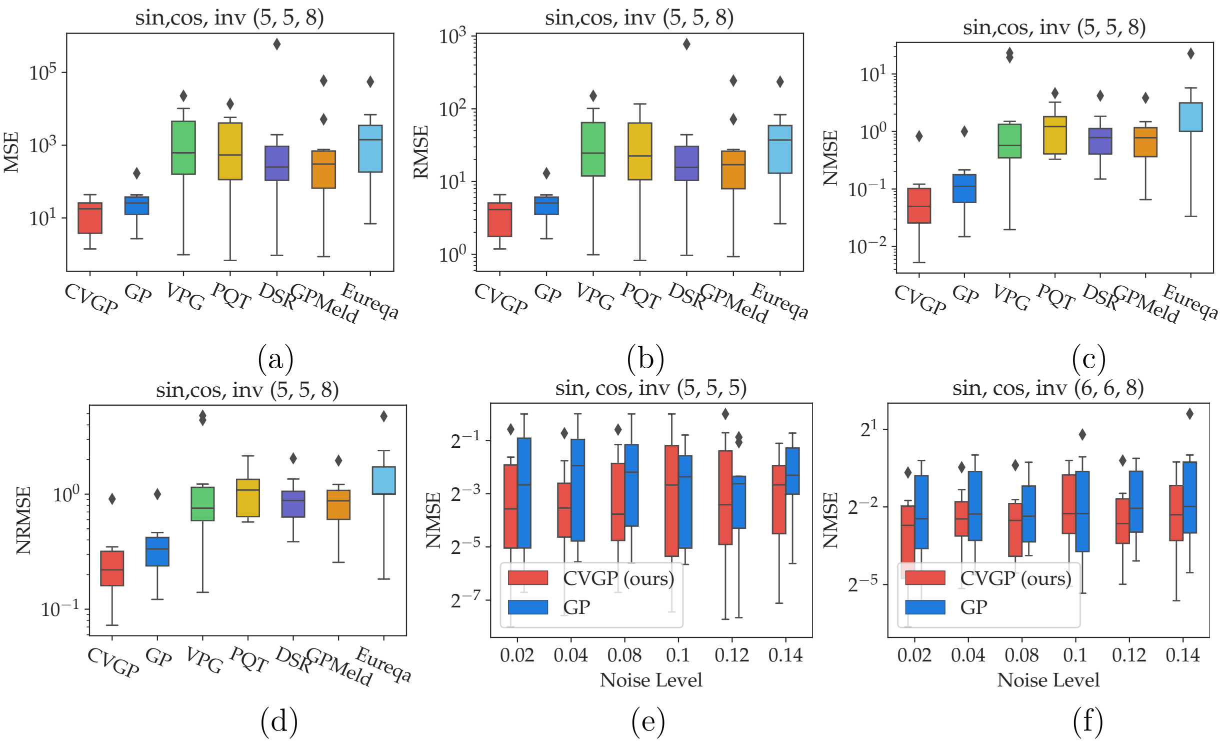
Dataset configs	CVGP 50%	CVGP 75%	GP 50%	GP 75%	DSR 50%	DSR 75%	PQT 50%	PQT 75%	VPG 50%	VPG 75%	GPMeld 50%	GPMeld 75%
(2,1,1)	0.241	0.873	0.102	1.0018	0.440	1.648	0.757	9.401	0.2142	3.349	**0.0002**	**0.0007**
(3,2,2)	0.049	**0.113**	**0.023**	0.166	0.663	2.773	1.002	1.992	0.969	1.310	0.413	2.510
(4,4,6)	**0.141**	**0.220**	0.238	0.662	1.031	1.051	1.297	1.463	1.051	1.774	1.093	1.769
(5,5,5)	**0.157**	0.438	0.195	**0.337**	1.098	3.617	1.018	5.296	1.012	1.27	1.036	3.617
(5,5,8)	**0.122**	**0.153**	0.166	0.186	1.009	1.103	1.017	1.429	1.007	1.132	1.07	2.904
(6,6,8)	**0.209**	**0.590**	**0.209**	0.646	1.003	1.153	1.047	1.134	1.059	1.302	1.029	3.365
(6,6,10)	0.139	0.232	**0.073**	**0.159**	1.654	3.408	1.027	1.069	1.009	1.654	1.445	2.106

(c) Datasets containing operands $\{\sin, \cos, \texttt{inv}, +, -, \times\}$.

handle multiple variables symbolic regression problems than the current best algorithms in this area.

Ablation Studies. We use box plots in Figure 6.3(a-d) to show that the superiority of our CVGP generalizes to other quantiles beyond the 50% and 75%-quantile. We also show the performance is consistent under the variations of evaluation metrics in Figure 6.3(a-d), and noise levels in Figure 6.3(e-f).

Recovering Ground-truth Equations. For relatively less challenging noiseless datasets (*i.e.*, $(2, 1, 1)$ with various operand sets), our CVGP sometimes recovers ground-truth expressions. We evaluate the percentage that each algorithm successfully detects the ground-truth expressions on 50 randomly generated benchmark datasets. Table 6.3 shows that our CVGP algorithm has a higher chance to recover ground-truth expressions than the GP method.



(a) (b) (c)

(d) (e) (f)

Figure 6.3. (a-d) Box plots of evaluation metrics for the expressions found by different algorithms on the noiseless dataset. **(e-f)** Box plots in NMSE values for the expressions found by CVGP and GP over benchmark datasets with different noise levels. Our CVGP is consistently the best regardless of the evaluation metrics and noise levels.

Table 6.3. Ground-truth recovery rate comparison. Our CVGP has a higher rate of recovering the ground-truth expressions compared to GP on 3 simple datasets.

Set of allowed operations	Dataset configuration	CVGP (ours)	GP
$\{\texttt{inv}, +, -, \times\}$	(2,1,1)	**64%**	44%
$\{\sin, \cos, +, -, \times\}$		**46%**	22%
$\{\sin, \cos, \texttt{inv}, +, -, \times\}$		**44%**	32%

6.6 Summary

In this chapter, we propose Control Variable Genetic Programming (CVGP) for symbolic regression with many independent variables. This is beyond current state-of-the-art approaches mostly tested on equations with one or two variables. CVGP builds equations involving more and more independent variables via control variable experimentation. Theoretically, we show CVGP as an incremental building approach can bring an exponential reduction in the search spaces when learning a class of expressions. In experiments, CVGP finds the best-fitted expressions among 7 competing approaches and on dozens of benchmarks.

7. Racing Control Variable Genetic Programming for Symbolic Regression

7.1 Introduction

Automatically discovering scientific laws from experimental data has been a long-standing aspiration of Artificial Intelligence. Its success holds the promise of significantly accelerating scientific discovery. A crucial step towards achieving this ambitious goal is symbolic regression, which involves learning explicit expressions from the experimental data. Recent advancements in this field have shown exciting progress, including works on genetic programming, Monte Carlo tree search, deep reinforcement learning and their combinations [177–181, 183, 184, 240, 241].

Despite remarkable achievements, the current state-of-the-art approaches are still limited to learning relatively simple expressions, typically involving only a few independent variables. The real challenge lies in symbolic regression involving multiple independent variables. The aforementioned approaches learn symbolic equations from a fixed dataset. As a result, these methods require massive datasets and extensive training time to discover complex equations.

Recently, a novel approach called Control Variable Genetic Programming (CVGP) [242] is introduced to accelerate symbolic regression. Instead of learning from fixed datasets collected a-priori, CVGP carries out symbolic regression using customized control variable experiments. As a motivating example, to learn the ideal gas law $pV = nRT$, one can hold n (gas amount) and T (temperature) as constants. It is relatively easy to learn p (pressure) is inversely proportional to V (volume). Indeed, CVGP discovers a chain of simple-to-complex symbolic expressions; e.g., first an expression involving only p and V, then involving p, V, T, etc. In each step, learning is carried out on specially collected datasets where a set of variables held constant. The major difference between CVGP and previous approaches is that CVGP *actively explores* the space of all expressions via control variable experiments, instead of learning passively from a pre-collected dataset.

However, the set of experiments is fixed a-priori in CVGP. It first learns an equation involving only the first variable, then involving the first two variables, etc. In particular, CVGP works with a fixed *experiment schedule* (noted as π), that is the sequences of con-

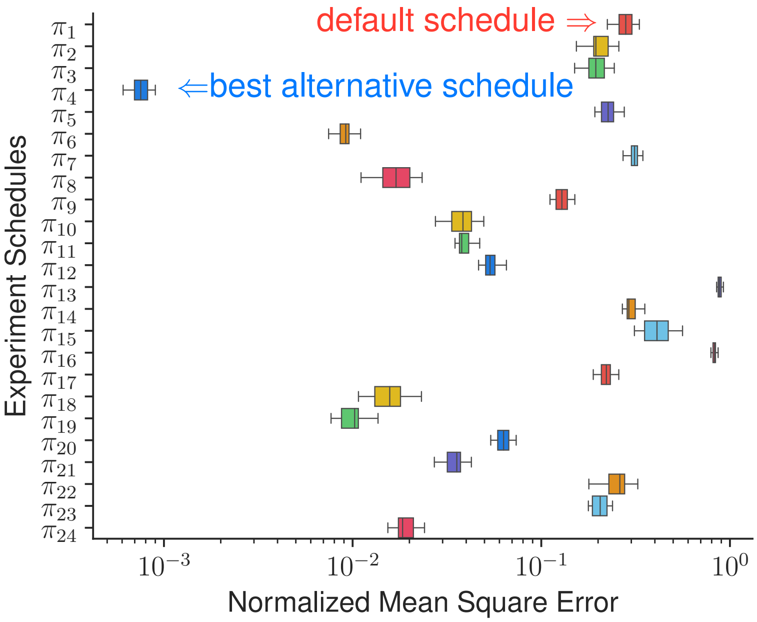


Figure 7.1. Impact of experiment schedules (noted as π) on learning performance of control variable genetic programming. For the discovery of expression with 4 variables, there exists a better experiment schedule (*i.e.*, π_4) among all schedules than the default one (*i.e.*, π_1), in terms of normalized mean square error (more examples in Appendix E.4.1).

trolled variables. We observe that the sub-optimal selection of experiment schedules delays the discovery process significantly. In Figure 7.1, we run CVGP with all 24 possible experiment schedules and report the quartiles of normalized mean squared errors (NMSE) of the discovered top 20 expressions. We see that certain experiment schedules (such as π_4) are significantly better than others including the default schedule π_1.

To overcome this limitation, we propose Racing-CVGP, which automatically discovers good experiment schedules that lead to accurate symbolic regression. A selection scheme over the experiment schedules is implemented, similar to that used in selecting good symbolic equations in genetic programming, to ensure that promising experiment schedules eventually win over the average schedules. The unfavorable schedules are terminated early to save time for promising schedules. Racing-CVGP allows flexible control variables experiments to be performed during the discovery process. If a specific set of controlled variable experiments fails to discover a good expression, it is ranked at the bottom and is eventually removed

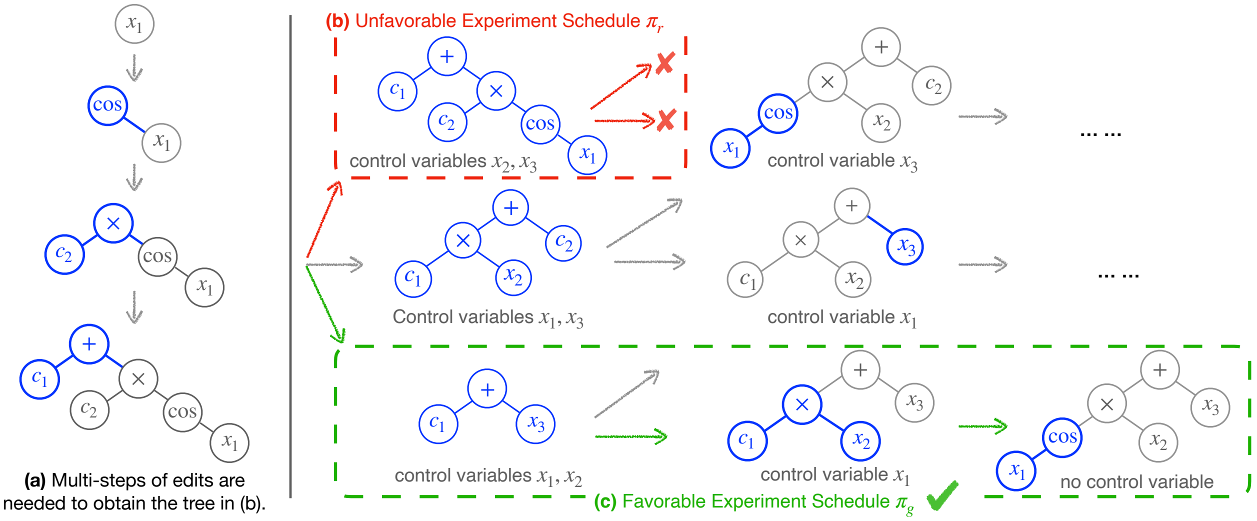

(a) Multi-steps of edits are needed to obtain the tree in (b).
(b) Unfavorable Experiment Schedule π_r
control variables x_2, x_3
control variable x_3
... ...
Control variables x_1, x_3
control variable x_1
... ...
control variables x_1, x_2
control variable x_1
no control variable
(c) Favorable Experiment Schedule π_g

Figure 7.2. The favorable experiment schedule π_g is survived while the unfavorable schedule π_r is early stopped under our racing experiment schedule scheme. (a) Multiple steps of edits are needed to transform from a randomly initialized expression "x_1" to a complex expression "$c_1+c_2\cos(x_1)$". The newly inserted parts (by genetic programming algorithm) are highlighted in blue. (b) The red experiment schedule π_r is unfavorable because it requires many edits to reach the expression tree in the red box (shown in (a)). The red schedule is thus stopped early. (c) The green experiment schedule π_g is promising since it is relatively easy to discover, and every change in the expression tree is reasonable. Section 7.3.1 provides a detailed explanation.

by the selection scheme. Our idea allows the algorithm to avoid spending excessive time on unfavorable experiment schedules and to focus on exploring promising experiment schedules.

In experiments, we compare Racing-CVGP against several popular symbolic regression baselines using challenging datasets with multiple variables. On several datasets, we observe that Racing-CVGP discovers higher quality expressions in terms of the NMSE metric against several baselines. Our Racing-CVGP also takes less computational time than all the baselines. Our Racing-CVGP stops those unfavorable schedules early, which commonly leads to a longer training time. Notably, our method scales well to expressions with 8 variables while the GP, CVGP, and GPMeld methods take more than 48 hours and thus are time-consuming. Our contributions can be summarized as follows:

- We identify that a sub-optimal selection of the experiment schedule greatly delays the discovery process of symbolic regression. We propose Racing-CVGP to accelerate scientific discovery by maintaining good experiment schedules during learning challenging symbolic regression tasks.

- Under our racing schedule, a favorable schedule is survived while unfavorable schedules are stopped early. We show the time complexity of our Racing-CVGP is approximately close to that of the CVGP, under mild assumptions.

- In experiments, we showcase that our Racing-CVGP leads to faster discovery of symbolic expressions with smaller NMSE metrics, compared to current popular baselines over several challenging datasets[1].

7.2 Preliminaries

7.2.1 Symbolic Regression for Scientific Discovery

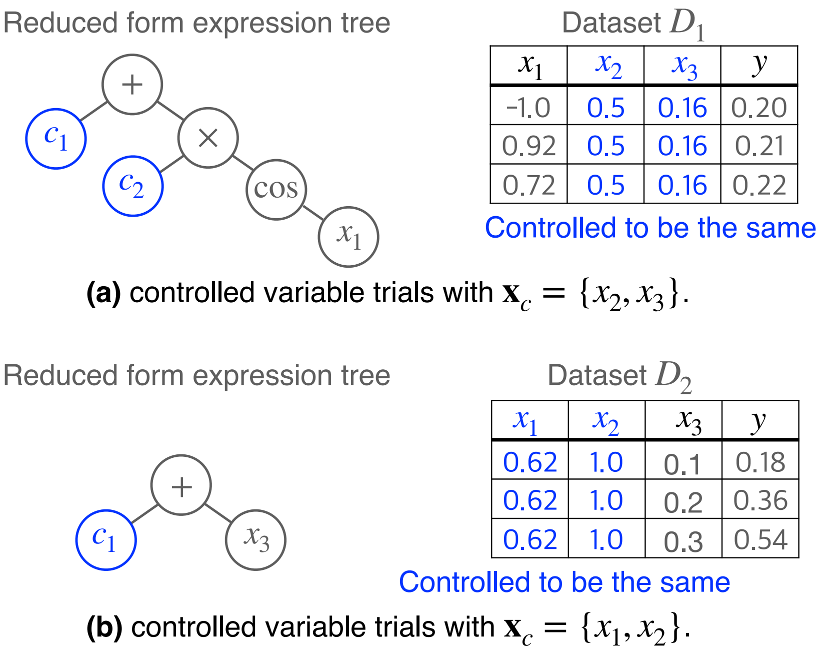
A *symbolic expression* ϕ is expressed as variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and constants $\mathbf{c} = \{c_1, \ldots, c_m\}$, connected by a set of binary operators (like $\{+, -, \times, \div\}$) and/or unary operators (like $\{\sin, \cos, \log, \exp\}$). The operator set is noted as O_p. Each operand of an operator is either a variable, a constant, or a self-contained sub-expression. For example, "x_1+x_2" is a expression with 2 variables (x_1 and x_2) and one binary operator ($+$). A symbolic expression can be equivalently represented as a *binary expression tree*, where the leaf nodes correspond to variables and constants and the inner nodes correspond to those operators. Figure 7.3 presents two example expression trees.

Given a dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and a loss function $\ell(\cdot, \cdot)$, the task of *symbolic regression* is to find the optimal symbolic expression ϕ^* with minimum loss over dataset D, among the set of all candidate expressions (noted as Π):

$$\phi^* \leftarrow \arg\min_{\phi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} \ell(\phi(\mathbf{x}_i, \mathbf{c}), y_i), \tag{7.1}$$

where the values of the open constants $\mathbf{c}$ in ϕ are determined by fitting the expression to the dataset D. The loss function $\ell(\cdot, \cdot)$ measures the distance between the output from the candidate expression $\phi(\mathbf{x}_i, \mathbf{c}) \in \mathbb{R}$ and the ground truth $y_i \in \mathbb{R}$. A common choice of the

[1] ↑The code is at https://bitbucket.org/xlnxyx/racing_cvgp.


Reduced form expression tree
+
c_1
×
c_2
cos
x_1
Dataset D_1
Controlled to be the same
Reduced form expression tree
+
c_1
x_3
Dataset D_2
Controlled to be the same

x_1	x_2	x_3	y
-1.0	0.5	0.16	0.20
0.92	0.5	0.16	0.21
0.72	0.5	0.16	0.22

(a) controlled variable trials with $\mathbf{x}_c = \{x_2, x_3\}$.

x_1	x_2	x_3	y
0.62	1.0	0.1	0.18
0.62	1.0	0.2	0.36
0.62	1.0	0.3	0.54

(b) controlled variable trials with $\mathbf{x}_c = \{x_1, x_2\}$.

Figure 7.3. (a) When controlling variables x_2 and x_3, the ground-truth expression $\phi = x_2 \cos(x_1) + x_3$ reduces to $c_1 \cos(x_1) + c_2$. (b) Controlling variables x_1 and x_2 reduces the ground-truth to $c_1 x_3$.

loss function is Normalized Mean Squared Error (NMSE). Symbolic regression is shown to be NP-hard [195], due to the exponentially large size of all the candidate expressions Π.

Genetic Programming for Symbolic Regression. Genetic Programming (GP) has been a popular method for solving symbolic regression. The core idea of GP involves managing a pool of candidate expressions, noted as $\mathcal{P}$. In each generation, these candidates undergo *mutation* and *mating* steps with certain probabilities. The mutation operations randomly replace, insert a node in the expression tree, or delete a sub-tree. The mating operations pick a pair of parent expression trees and exchange their two random sub-trees. In the *selection* step, expressions with the highest fitness scores, are chosen as candidates for the next generation. Here the fitness scores (noted as $\mathbf{o} \in \mathbb{R}^N$) indicate the closeness of the predicted outputs to the ground-truth outputs, like the negative NMSE. Over several generations, the expressions that fit the data well, exhibiting high fitness scores, survive in the pool of candidate solutions. The best expressions discovered throughout all generations are recorded as *hall-of-fame* solutions, noted as $\mathcal{H}$.

7.2.2 Control Variable Trials

In a regression problem, control variable trials study the relationship between a few input variables and the output with the remaining input variables fixed to be the same [185]. The control variable idea was historically proposed to discover natural physical law, known as the BACON system [189–191]. Recently, this idea has been explored for solving multivariable symbolic regression problems [242], *i.e.*, CVGP.

Let $\mathbf{x}_c \subseteq \mathbf{x}$ denote those control variables, and the rest are free variables. The values of controlled variables are fixed in each trial, which behaves exactly the same as constants for the learning method. In the controlled setting, the ground-truth expression behaves the same after setting those controlled variables as constants, which is noted as the *reduced form expression*. See Figure 7.3 for two reduced form expressions with different control variable settings.

For a single control variable trial in symbolic regression, the corresponding dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ is first generated, where the controlled variables are fixed to one value and the remaining variables are randomly assigned. That is $\mathbf{x}_{i,k} = \mathbf{x}_{j,k}$ for the control variable x_k ($x_k \in \mathbf{x}_c$) and $1 \leq i, j \leq N$. Figure 7.3 gives two example datasets generated from different control variable trials. Given a reduced form expression and corresponding dataset, the values of open constants in the expression are determined by gradient-based optimizers, like the BFGS algorithm. In Figure 7.3(a), the optimal values of open constants are $c_1 = 0.5, c_2 = 0.16$. Similarly in Figure 7.3(b), we have $c_1 = 1.8$. The loss values (defined in Equation (7.1)) of these two controlled variable trails over the dataset D_1 and dataset D_2 are equal to 0, indicating the optimal fitness scores.

The CVGP is built on top of the above control variable trials and the GP algorithm. To fit an expression of n variables, CVGP initially only allows variable x_1 to vary and controls the values of all $n-1$ variables (*i.e.*, $\mathbf{x}_c = \mathbf{x}\backslash\{x_1\}$). Using GP as a subroutine, CVGP finds a pool of expressions $\{\phi_1, \ldots, \phi_{N_p}\}$ which best fit the data from this controlled experiment. Notice $\{\phi_1, \ldots, \phi_{N_p}\}$ are restricted to contain only one free variable x_1 and N_p is the pool size. This fact renders fitting them a lot easier than directly fitting the expressions involving all n variables. A small error implies that ϕ_i is close to the ground truth reduced to the one free variable. In the 2nd round, CVGP adds a second free variable x_2 and starts fitting $\{\phi'_1, \ldots, \phi'_{N_p}\}$ using the data from control variable experiments involving the two free variables x_1, x_2. After n rounds, the expressions in the CVGP pool consider all n variables. Note that CVGP assumes the existence of a `DataOracle` that allows for query a batch data with specified control variables.

7.3 Methodology

We first brief the issue with a fixed experiment schedule for the existing CVGP method in discovering symbolic regression. Then we present our racing experiment schedule for control variable genetic programming (Racing-CVGP).

7.3.1 Motivation

We define an *experiment schedule*, noted as π, as a sequence of variables controlled over all the rounds in CVGP. We use Figure 7.2 to demonstrate different experiment schedules for the discovery of the ground-truth expression $\phi = \cos(x_1)x_2+x_3$. In Figure 7.2(c), CVGP runs an experiment schedule with control variables $\{x_1, x_2\}$ in the first round and runs with control variables $\{x_1\}$ in the second round and with no variable control $\emptyset$ in the last round. The corresponding experiment schedule is $\pi = (\{x_1, x_2\}, \{x_1\}, \emptyset)$. Similarly, Figure 7.2(b) shows the default experiment schedule of CVGP that control variables $\{x_2, x_3\}$ initially and then control variable $\{x_3\}$, finally control no variable $\emptyset$, which is denoted as $\pi = (\{x_2, x_3\}, \{x_3\}, \emptyset)$.

Our key observations are as follows: (1) The experiment schedule plays a vital impact on the performance of CVGP than other components in the algorithm. (2) Some expressions are much easier to detect for specific experiment schedules. The existing CVGP method only considers a fixed experiment schedule $\pi = (\{x_2, \ldots, x_n\}, \{x_3, \ldots, x_n\}, \ldots, \{x_n\}, \emptyset)$ for discovering expression involving n variables. This fixed experiment schedule leads to sub-optimal performance of CVGP over some expressions, requiring more training data and computational time than other alternative schedules. See Figure 7.1 for an empirical evaluation of different experiment schedules over the final identified expressions by the same CVGP method. See more examples in Appendix E.4.1.

In Figure 7.2, we use the discovery of an expression $\phi = \cos(x_1)x_2 + x_3$ from the Feynman dataset as an example. The alternative (green) experiment schedule π_g in Figure 7.2(c) is favorable while the default (red) schedule π_r in Figure 7.2(b) is not. In Figure 7.2(a), we visualize 3 necessary steps to reach from randomly initialized expression tree "x_1" to the final tree "$c_1 + c_2 \cos(x_1)$" in Figure 7.2(b). Every step of editing is conducted by the GP and requires drawing batches of training data to fit every intermediate expression. The edited subtrees are highlighted in blue. In comparison, it takes 1 step of edits in the tree to reach the first expression "c_1+x_3" in the green experiment schedule, which leads to faster discovery using less training data. Following the green experiment schedule π_g, it takes 1 step of edits to reach the expression at the second round "$c_1x_2 + x_3$" and the last round "$\cos(x_1)x_2 + x_3$". Therefore, CVGP needs much more data and time in the 1st round following the default (red) experiment schedule π_r. The alternative (green) experiment schedule π_g is easier for the GP algorithm to discover the ground-truth expression using less data and time.

Directly evoking CVGP as a subroutine with multiple experiment schedules will not solve the problem. The expression in Figure 7.1 has 24 different experiment schedules. The total running time is summarized in Figure 7.6. In general, for an expression involving n variables, there are $n!$ many experiment schedules. It is time-intractable to run CVGP with all the experiment schedules for real-world scale problems.

To tackle the above issue, we propose a racing scheme over the experiment schedules. Our main principles are (1) maintaining multiple experiment schedules rather than one, and (2) allowing promising experiment schedules to survive while letting unfavorable schedules early stop. Our Racing-CVGP has a much higher chance of detecting high-quality expression using less training data and computational time than the existing CVGP.

Specifically, we implement a schedule selection procedure. Every expression in the population pool $\phi \in \mathcal{P}$ is attached with its own experiment schedule. In each round, we execute GP over all the expressions in the population pool for several generations. At the end of every round, the racing selection scheme removes (*resp.* preserves) those expressions with bad (*resp.* good) experiment schedules, based on their fitness scores. So those schedules that lead to higher fitness scores have a higher probability of survival.

We use Figure 7.2 to visualize the process of our Racing-CVGP. We first initialize the population pool $\mathcal{P}$ in GP with several expressions for each control variable setting. We randomly generate simple expressions involving only x_1 with the control variables being $\{x_2, x_3\}$, where every expression is attached with a (partial) experiment schedule $\pi = (\{x_2, x_3\})$. We repeat this random expression generation for all the rest $n-1$ control variable settings. For the 1st round, the GP algorithm is evoked over the population pool for several generations. Then we rank the expressions in the pool by the fitness score of the expression, where those expressions with higher fitness scores rank at the top of the pool. We only preserve top N_p expressions in population pool $\mathcal{P}$. Since it is much easier to detect $c_1 + x_3$ under control variable $\{x_1, x_2\}$ setting, the preserved majority expressions are attached with the experiment schedule $\pi_1 = \{x_1, x_2\}$. This ensures that we early stop the unfavorable experiment schedule $\pi = \{x_2, x_3\}$ in Figure 7.2(b). Prior to the 2nd round, we randomly set free one variable from π_1. Figure 7.2(c) set the free variable x_2 and only variable x_1 is controlled in the 2nd round. In the 3rd round, the majority of the expressions in the population is attached to the experiment schedule $\pi_g = (\{x_1, x_2\}, \{x_1\}, \emptyset)$, since every change over the expression tree is reasonable. The total computational resources are saved from spending time searching for the expression tree in Figure 7.2(b) to explore expressions with experiment schedule $\pi = (\{x_1, x_2\}, \{x_1\})$ in Figure 7.2(c).

7.3.2 Racing Control Variable Genetic Programming

The high-level idea of Racing-CVGP is building simple to complex symbolic expressions involving increasingly more variables following those promising experiment schedules.

Notations. Denote K multiple control variable trials as a tuple $\langle \phi, \mathbf{o}, \mathbf{c}, \mathbf{x}_c, \pi, \{\mathcal{D}_k\}_{k=1}^{K} \rangle$. Here ϕ stands for the symbolic expression; the fitness scores $\mathbf{o} \in \mathbb{R}^K$ for expression ϕ indicates the closeness of predicted outputs to the ground-truth outputs; $\mathbf{c} \in \mathbb{R}^{K \times L}$ are the best-fitted values (by gradient-based optimizers) to open constants. Here L is the number of open constants in the expression ϕ; $\mathbf{x}_c \subseteq \mathbf{x}$ is the set of control variables; π is the (partial) experiment schedule that leads to the current expression ϕ. $\mathcal{D}_k = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ $(1 \leq k \leq K)$

is a randomly sampled batch of data from `DataOracle` with control variables $\mathbf{x}_c$. m denotes the batch size of the data.

Initialization. For single variable $x_i \in \mathbf{x}$, we create a set of candidate expressions that only contain variable x_i and save them into the population pool $\mathcal{P}$. Then we apply a GP-based algorithm to find the best-fitted expressions, which is referred to as the `BuildGPPool` function. The initialization step corresponds to Lines 2-6 in Algorithm 7.

Execution Pipeline. Given the current control variables $\mathbf{x}_c$, we first evoke the `DataOracle` to generate data batches $\{\mathcal{D}_k\}_{k=1}^{K}$. This corresponds to changing experimental conditions in real science experiments. We then fit open constants in the candidate expression ϕ_{new} with the data batches by gradient-based optimizers like BFGS [243]. This step is noted as the `Optimize` function. Then we obtain the fitness score vector $\mathbf{o}$ and solutions to open constants $\mathbf{c}$. We save the tuple $\langle \phi, \mathbf{o}, \mathbf{c}, \pi, \mathbf{x}_c \rangle$ into new population pool $\mathcal{P}_{new}$. This step corresponds to Lines 8-11 in Algorithm 7.

Then GP algorithm is applied for #`Gen` generations to search for optimal structures of the expression trees in the population pool P_{new}. The function `GP` is a minimally modified genetic programming algorithm for symbolic regression, which is detailed in Appendix E.1. The key differences between classic GP and our Racing-CVGP are

1. During mutation, our Racing-CVGP only alters the *mutable* nodes of the candidate expression trees. In classic GP, all the tree nodes are mutable, while in Racing-CVGP, the mutable nodes of the expression trees and set of operators O_p are preset by the `FreezeEquation`.

2. Mating is only applied over a pair of expressions with the same set of controlled variables in our Racing-CVGP. Classic GP, a random pair of expressions is selected for the mating operation.

3. `Optimize` operation in Racing-CVGP dynamically samples data with oracle $\mathcal{D}_o$ under control variable setup, whereas classic GP uses data with no variable controlled.

We preserve N_p best equations in the population $\mathcal{P}$. Every expression is evaluated with the different data from *its own control variables*. An unfavorable (partial) experiment schedule will be removed at this step when the corresponding expression ϕ has a low fitness score. The schedules in the pruned population pool $\mathcal{P}$ indicate that they are favorable.

Key information is obtained by examining the outcomes of K-trials control variable experiments: (1) Consistent close-to-zero fitness value, implies that the fitted expression is close to the ground-truth equation in the reduced form. That is $\sum_{k=1}^{K} \mathbb{I}(o_k \leq \varepsilon)$ should equal to K, where $\mathbb{I}(\cdot)$ is an indicator function and ε is the threshold for the fitness scores. (2) Given that the equation is close to the ground truth, an open constant having similar best-fitted values across K trials suggests that the open constants are stand-alone. Otherwise, that open constant is a *summary* constant, that corresponds to a sub-expression involving those control variables $\mathbf{x}_c$. The j-th open constant is a standalone constant when the empirical variance of its fitted values across K trials is less than a threshold ε'. The above steps are noted as `FreezeEquation` function. This freezing operation reduces the search space and accelerates the discovery.

Finally, we randomly drop a control variable in $\mathbf{x}_c$ and update the schedule π for each equation ϕ in the population pool $\mathcal{P}$. After n rounds, we return the equations in hall-of-fame $\mathcal{H}$ with best fitness values over all the schedules. Equations in $\mathcal{H}$ are evaluated on data with no variable controlled.

Running Time Analysis. The major hyper-parameters that impact the running time of Racing-CVGP are 1) the number of genetic operations per round M; 2) total rounds n; 3) the maximum size of population pool N_p. A rough estimation of the time complexity of the proposed Racing-CVGP is $\mathcal{O}(nMN_p)$, which is the same as the CVGP algorithm. Another implicit factor of running time is the number of open constants $|\mathbf{c}|$ for every expression $\phi(\mathbf{x}, \mathbf{c})$. An expression with more open constants needs more time for optimizers (like BFGS and CG) or more advanced optimizers (like Basin Hopping [244]) to find the solutions. We leave it to the empirical time evaluation in Figure 7.6.

Connection to Existing Methods. Our work is relevant to a line of work [189–194] that implemented human scientific discovery using AI, pioneered by the BACON systems [189–191]. While BACON's discovery was driven by rule-based engines, our Racing-CVGP uses modern learning approaches such as genetic programming.

Algorithm 7: Racing Control Variable Genetic Programming

```
Input: #input variables $n$; operator set $O_p$; DataOracle.
Output: #genetic operations per rounds #Gen; Size of population pool $N_p$;
        #experiment trials $K$.
$\mathcal{P} = \{\}$;
$\mathcal{H} = \{\}$;
for $i \leftarrow 1$ to $n$ do
    $\mathbf{x}_c = \{x_1, \ldots, x_n\} \setminus \{x_i\}$ ;                         // initialize
    $\mathcal{P} \leftarrow \mathcal{P} \cup$ BuildGPPool($\mathbf{x}_c, O_p \cup \{\text{const}, x_i\}$));
for $i \leftarrow 1$ to $n$ do
    for $\langle \phi_{new}, \pi, \mathbf{x}_c \rangle \in \mathcal{P}$ do
        $\{\mathcal{D}_k\}_{k=1}^K \leftarrow$ DataOracle($\mathbf{x}_c, K$) ;         // control variable trials
        $\mathbf{o}, \mathbf{c} \leftarrow$ Optimize($\phi_{new}, \{\mathcal{D}_k\}_{k=1}^K$);
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{\langle \phi, \mathbf{o}, \mathbf{c}, \pi, \mathbf{x}_c \rangle\}$;
    $\mathcal{P}, \mathcal{H} \leftarrow$ GP($\mathcal{P}, \mathcal{H}$, DataOracle, $O_p \cup \{\text{const}, x_i\}$);
    for $\langle \phi, \pi, \mathbf{x}_c \rangle \in \mathcal{P}$ do
        $\phi \leftarrow$FreezeEquation($\phi$) ;                                  // racing schedule
        randomly drop a variable in $\mathbf{x}_c$;
        save $\mathbf{x}_c$ into $\pi$;
return the set of hall-of-fame equations $\mathcal{H}$.
```

7.4 Related Work

Early works in symbolic regression [199, 200] use heuristic search. Genetic programming is effective in searching for good candidates [178, 184, 201]. Reinforcement learning-based methods use a risk-seeking policy gradient to find the expressions [180, 181]. Other works use RL to adjust the probabilities of genetic operations [202]. Some works reduce the search space by considering the composition of base functions [203, 204].

Current research efforts are devoted to searching for polynomials with a few variables [205], time series equations [206], and equations in physics [201]. Multivariable symbolic regression is challenging since the search space increases exponentially w.r.t. the number of input variables. Existing works for multi-variable regression are based on pre-trained encoder-decoder methods with a massive training dataset (e.g., millions of data points [207]), and even larger generative models (e.g., millions of parameters [208]). Our Racing-CVGP is a tailored algorithm to solve multi-variable symbolic regression.

The choice of variables is an important topic in AI, including variable ordering in decision diagrams [245], variable selection in tree search [246], variable elimination in probabilistic inference [247, 248] and backtracking search in constraint satisfaction problems [249–251]. Our method is one variant of variable ordering in symbolic regression.

Our work is also relevant to experiment design, which considers drawing a minimum amount of data for determining coefficients in linear regression models [252, 253]. Our work considers reducing the amount of total data needed to uncover the ground truth expression.

7.5 Experiments

This section demonstrates that Racing-CVGP finds the expressions with the smallest Normalized Mean-Square Errors (NMSE) (in Table 7.1 and Table 7.2) and takes less computational time (in Figure 7.4), among all competing approaches on several noiseless datasets. In the ablation studies, we show our Racing-CVGP is consistently better than the baselines when evaluated in different metrics (in Figure 7.5). Also, our Racing-CVGP methods save a great portion of time than evoke CVGP with all the possible schedules.

7.5.1 Experimental Settings

Datasets. We consider several publicly available and multivariable datasets, including 1) Trigonometric datasets [242], 2) Livermore2 datasets [180], and 3) Feynamn datasets [254].

Evaluation Metrics. We consider two evaluation criteria for the learning algorithms: 1) The goodness-of-fit measure (NMSE), indicates how well the learning algorithms perform in discovering symbolic expressions. The medians (50%) and 75%-percentiles of the NMSE are reported. We report median values instead of means due to outliers (see Ablation Studies). This is a common practice for combinatorial optimization problems. 2) The total running time of each learning algorithm.

Baselines. We consider the following baselines based on evolutionary algorithms: 1) Genetic Programming (GP) [236]. 2) Eureqa [237]. We also consider a series of baselines using reinforcement learning: 3) Priority queue training (PQT) [238]. 4) Vanilla Policy Gradient (VPG) [239]. 5) Deep Symbolic Regression (DSR) [180]. 6) Neural-Guided Genetic Pro-

	(3, 2, 2)		(4, 4, 6)		(5, 5, 5)		(6, 6, 10)		(8, 8, 12)	
	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
Racing-CVGP (ours)	< **1E-6**	< **1E-6**	**0.016**	**0.021**	**0.043**	**0.098**	**0.069**	**0.104**	**0.095**	**0.286**
CVGP	0.039	0.083	0.028	0.132	0.086	0.402	0.104	0.177	*T.O.*	*T.O.*
GP	0.043	0.551	0.044	0.106	0.063	0.232	0.159	0.230	*T.O.*	*T.O.*
Eureqa	< **1E-6**	< **1E-6**	0.024	0.122	0.158	0.377	0.910	1.927	0.162	2.223
DSR	0.227	7.856	2.815	9.958	2.558	3.313	6.121	16.32	0.335	0.410
PQT	0.855	2.885	2.381	13.84	2.168	2.679	5.750	16.29	0.232	0.313
VPG	0.233	0.400	2.990	11.32	1.903	2.780	3.857	19.82	0.451	0.529
GPMeld	0.944	1.263	1.670	2.697	1.501	2.295	7.393	21.71	*T.O.*	*T.O.*
SPL	0.010	0.011	0.144	0.231	0.147	0.280	0.472	0.627	0.599	0.746

Table 7.1. On Trigonometric datasets, median (50%) and 75%-quantile NMSE values of the expressions found by all the algorithms. Our Racing-CVGP finds symbolic expressions with the smallest NMSEs. "*T.O.*" implies the algorithm is timed out for 48 hours. The 3-tuples at the top $(\cdot, \cdot, \cdot)$ indicate the number of input variables, singular terms, and cross terms in the expression.

gramming Population Seeding (GPMeld) [181]. 7) Symbolic Physics Learner (SPL) [240]. The remaining details are provided in Appendix E.3.

7.5.2 Experimental Result Analysis

Goodness-of-fit Benchmark. Our Racing-CVGP attains the smallest median (50%) and 75%-quantile NMSE values among all the baselines when evaluated on selected Trigonometric, Livermore2, and Feynman datasets (Table 7.1). This shows that our method can better handle multivariable symbolic regression problems than the current best algorithms in this area. For the Trigonometric dataset with $n = 8$ variables, both the GP and CVGP take more than 2 days to find the optimal expression. The reason is that there are too many open constants in the expressions in the population pool, making the optimization problem itself time-consuming to find the solution. This behavior is another indication that CVGP is stuck at some unfavorable experiment schedule.

Empirical Running Time Analysis. We summarize the running time analysis in Figure 7.4. Our Racing-CVGP method takes less time than CVGP as well as the rest baselines. The main reason is early stop those unfavorable experiment schedules. See Appendix E.4 for more figures.

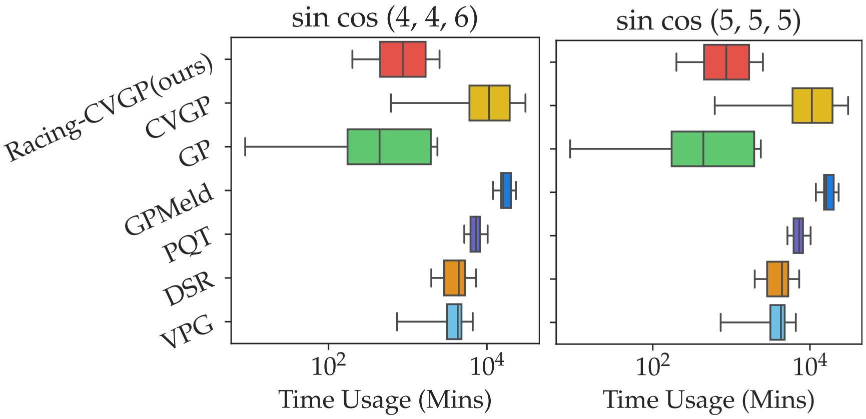


Figure 7.4. On selected Trigonometric datasets, quartiles of the total running time of all the methods. Our Racing-CVGP method takes less time than CVGP by early stopping those unfavorable experiment schedules.

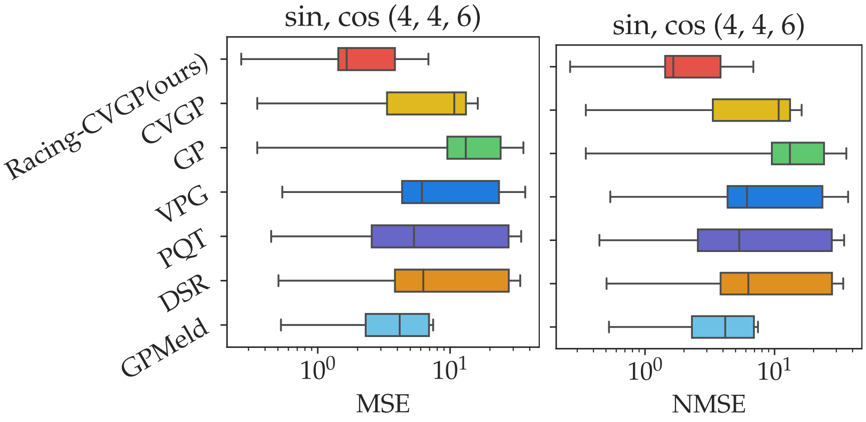


Figure 7.5. On selected Trigonometric datasets, MSE and NMSE evaluation metrics of the expressions found by different algorithms.

Ablation Studies We collect the benchmark of different evaluation metrics in Figure 7.5, *i.e.,* MSE and NMSE, during testing over the selected Trigonometric datasets. The RMSE and NRMSE evaluation metrics are available in Appendix E.4.

We further collect the time comparison between our Racing-CVGP and the CVGP with all the experiment schedules in Figure 7.6. The quartiles of time distribution over 10 random expressions with 4 variables show that Our Racing-CVGP saves a great portion of the time compared with CVGP with all the schedules.

	Livermore2 dataset						Feynman dataset			
	$(n = 4)$		$(n = 5)$		$(n = 6)$		$(n = 4)$		$(n = 5)$	
	50%	75%	50%	75%	50%	75%	50%	75%	50%	75%
Racing-CVGP (ours)	**< 1E-6**	**2.03E-3**	**0.004**	**0.047**	**0.001**	**0.073**	**0.015**	**0.195**	0.577	**0.790**
CVGP	0.052	0.810	0.275	1.007	0.328	1.012	1.002	1.010	1.001	1.002
GP	0.059	0.962	0.331	1.003	1.001	1.026	1.003	1.010	1.002	1.011
Eureqa	0.508	0.980	0.083	0.249	0.026	0.302	0.026	0.397	**0.434**	0.943
DSR	0.030	0.048	0.050	0.284	0.230	0.486	0.216	0.920	0.976	1.001
PQT	0.042	0.063	0.074	0.227	0.170	0.410	0.172	0.765	1.003	1.027
VPG	0.037	0.074	0.093	0.322	0.206	0.535	0.188	0.971	1.006	1.025
GPMeld	0.029	0.061	0.049	0.259	0.144	0.504	0.177	0.708	0.940	1.002
SPL	0.035	0.463	0.181	0.201	0.229	1.005	0.143	0.542	0.632	1.002

Table 7.2. On Livermore2 and Feynman datasets, median (50%) and 75%-quantile NMSE values of the symbolic expressions found by all the algorithms. Our Racing-CVGP finds symbolic expressions with the smallest NMSEs. n is the number of independent variables in the expression.

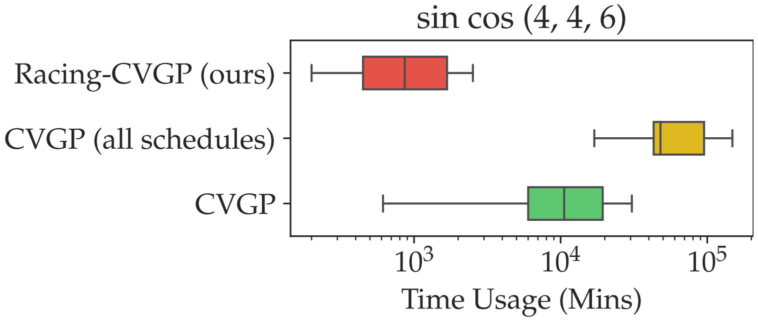


Figure 7.6. On a selected Trigonometric dataset, quartiles of the total running time of Racing-CVGP, CVGP, and CVGP with all the experiment schedules. Our Racing-CVGP saves a great portion of time compared with CVGP with all the schedules for expressions with $n = 4$ variables.

7.6 Summary

In this chapter, we propose Racing Control Variable Genetic Programming (Racing-CVGP) for symbolic regression with many independent variables. Our Racing-CVGP can accelerate the regression process by discovering equations from promising experiment schedules and early stop those unfavorable experiment schedules. We evaluate Racing-CVGP on several synthetic and real-world datasets corresponding to true physics laws. We demonstrate that Racing-CVGP outperforms CVGP and a series of symbolic regressors that discover equations from fixed datasets.

8. Vertical Symbolic Regression via Deep Policy Gradient

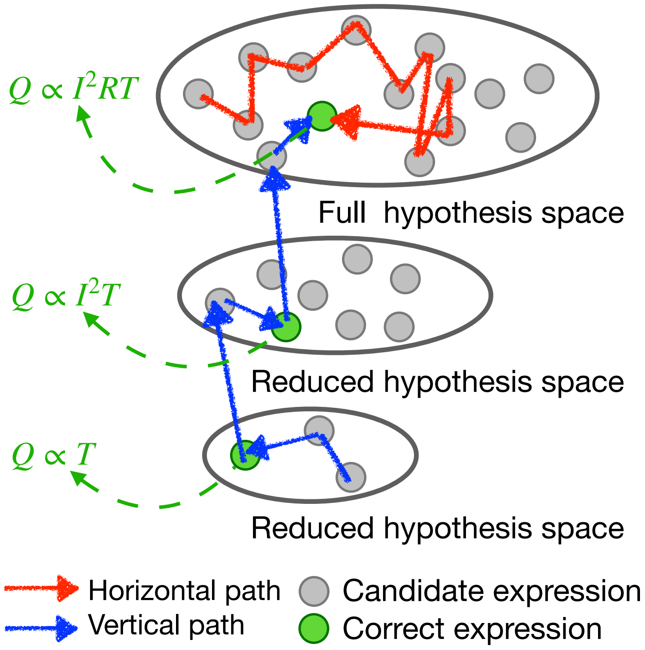
8.1 Introduction

Exciting progress has been made to accelerate scientific discovery using Artificial Intelligence (AI) [209, 255, 256]. Symbolic regression, as an important task in AI-driven scientific discovery, distills physics models in the form of symbolic equations from experiment data [177]. Notable progress in symbolic regression encompasses diverse methodologies, includes search-based methods [200], genetic programming [177, 178], Monte Carlo tree search [240, 257, 258], and deep reinforcement learning [180, 181].

Recently, Vertical Symbolic Regression (VSR) [242, 259] has been proposed to expedite the discovery of symbolic equations with many independent variables. Unlike previous approaches, VSR reduces the search spaces following a vertical path – it extends from reduced-form equations involving a subset of independent variables to full-fledged ones, adding one variable into the equation at a time. Figure 8.1 gives an example discovery of Joule's law $Q \propto I^2RT$ [260], where Q is heat, I is current, R is resistance, and T is time. VSR first holds I and R as constants and finds $Q \propto T$. Next, I is introduced with targeted experiments that study the effect of I on Q. Such rounds repeat until all factors are considered. Compared with the horizontal paths, which model all independent variables simultaneously, vertical discovery can be significantly cheaper because the search spaces of the first few steps are exponentially smaller than the full search space.

Meanwhile, deep learning, especially deep policy gradient [180], has greatly boosted the performance of symbolic regression approaches. However, VSR was implemented using genetic programming. Although we hypothesize deep neural nets should boost VSR, directly integrating deep neural nets to predict the symbolic equation tree in each vertical expansion step encounters difficulties. This will result in (1) difficulty passing gradients from trees to deep neural nets and (2) complications concatenating deep networks for predictions in each vertical expansion step. We provide a detailed analysis of the difficulty in Appendix F.1.

In this work, we propose **V**ertical **S**ymbolic **R**egression using **D**eep **P**olicy **G**radient (VSR-DPG). We demonstrate that VSR-DPG can recover ground-truth equations involving 50 variables, which is beyond both deep reinforcement learning-based approaches and the



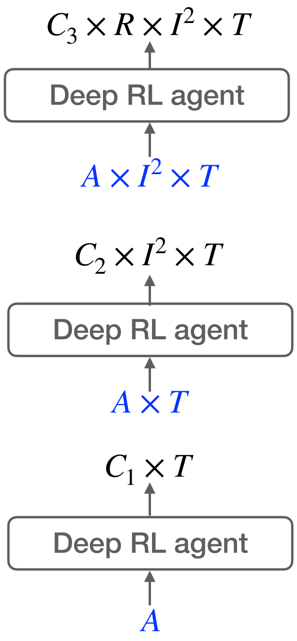


Figure 8.1. Our Vsr-Dpg follows a vertical path (colored blue) better than the horizontal path (colored red), in the scientific discovery of Joule's first law. **(Left)** The vertical discovery starts by finding the relationship between two factors (Q, T) in a reduced hypothesis space with other factors held constant. It then finds models in extended hypothesis space with three factors (Q, I, T), and finally in the full hypothesis space. Searching following the vertical paths is way cheaper since the sizes of the reduced hypothesis spaces in the first few steps are exponentially smaller than the full hypothesis space. **(Right)** Our Vsr-Dpg extends the equation in each step. The placeholder A indicates a sub-expression.

previous VSR variant (i.e., VSR-GP). Our Vsr-Dpg solves the above difficulty based on the following key idea: each symbolic expression can be treated as repeated applications of grammar rules. Hence, discovering the best symbolic equations in the space of all candidate expressions is viewed as the sequential decision-making of predicting the optimal sequence of grammar rules.

In Figure 8.1(right), the expansion from C_1T to C_2I^2T is to replace constant C_1 with a sub-expression C_2I^2. We define a context-free grammar on symbolic expression, denoting the rules that certain constants can be replaced with other variables, constants, or sub-expressions. All candidate expressions that are compatible with C_1T can be generated by repetitively applying the defined grammar rules in different order. Vsr-Dpg generates many sub-expressions (including C_2I^2) by sequentially sampling rules from the Recurrent Neural Networks (RNN). A vertical discovery path is built on top of this sequential decision-making process of reduced-form symbolic expressions. The RNN is trained to generate expansions that lead to high fitness scores on the dataset. In this regard, we train the RNN to maximize a policy gradient objective, similar to that proposed in Petersen *et al.* The main difference is the RNN in our VSR-DPG predicts the next rules in the vertical discovery space, while the model from Petersen *et al.* predicts the next math token in the expression tree in the horizontal discovery space.

In experiments, we consider several challenging multi-variable datasets of algebraic equations and of ordinary differential equations in material science and biology. (1) In Table 8.1, our VSR-DPG attains the smallest median NMSE values in 7 out of 8 datasets, against current popular baselines including VSR-GP. The main reason is deep networks offer more parameters than GP, which can better adapt to different datasets and sample higher-quality expressions from the networks. (2) Further analysis on the best-discovered equation (in Table 8.3) shows that VSR-DPG uncovers up to 50% of the exact governing equations with 5 input variables, where the baselines only attain 0%. (3) In Table 8.2, our VSR-DPG can find high-quality expressions on datasets with up to 50 variables, because of the vertical discovery idea. (4) On discovery of ordinary differential equations in Table 8.4, our VSR-DPG also improves over current baselines.[1]

8.2 Preliminaries

Symbolic Regression aims to discover governing equations from the experimental data. An example of such mathematical expression is Joule's first law: $Q = I^2RT$, which quantifies the amount of heat Q generated when electric current I flows through a conductor with resistance R for time T. Formally, a mathematical expression ϕ connects a set of input variables $\mathbf{x}$ and a set of constant coefficients $\mathbf{c}$ by mathematical operators. The possible mathematical operators include addition, subtraction, multiplication, division, trigonometric functions, etc. The meaning of these mathematical expressions follows their standard arithmetic definition.

[1] ↑The code is at https://github.com/jiangnanhugo/VSR-DPG.

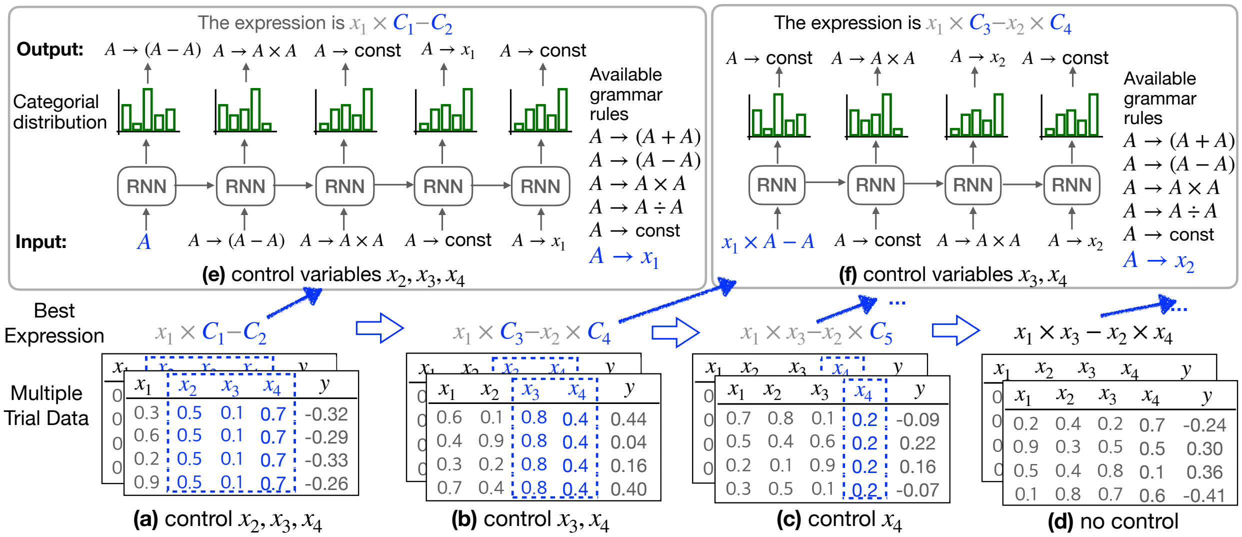


Figure 8.2. The proposed VSR-DPG for the discovery of expression $\phi = x_1 \times x_3 - x_2 \times x_4$. **(a)** Initially, a reduced-form equation $\phi = x_1 \times C_1 - C_2$ is found, in which variables x_2, x_3, x_4 are held constant and only variable x_1 is allowed to vary. C_1 and C_2 (colored blue) are summary constants, which are sub-expressions containing the controlled variables. The open constants in the expression are fitted by the corresponding controlled variable data. **(b)** In the second stage, this equation is expanded to $x_1 \times C_3 - x_2 \times C_4$. **(c, d)** This process continues until the ground-truth equation $\phi = x_1 x_3 - x_2 x_4$ is found. **(e, f)** Under those controlled variables, the deep recurrent neural network predicts a categorical distribution over the available grammar rules. The controlled variables are excited in the grammar rules. The best-predicted expression in (e) is reformulated as the start symbol in (f), that is $x_1 \times A - A$.

Given a dataset $D = \{(\mathbf{x}_i, y_i)_{i=1}^N | \mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}\}$ with N samples, symbolic regression searches for the optimal expression ϕ^*, such that $\phi^*(\mathbf{x}_i, \mathbf{c}) \approx y_i$. From an optimization perspective, ϕ^* minimizes the averaged loss on the dataset:

$$\phi^* \leftarrow \arg\min_{\phi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} \ell(\phi(\mathbf{x}_i, \mathbf{c}), y_i). \tag{8.1}$$

Here Π is the set of all candidate mathematical expressions; $\mathbf{c}$ denotes the constant coefficients in the expression; $\ell(\cdot, \cdot)$ denotes a loss function that measures the difference between the output of the candidate expression $\phi(\mathbf{x}_i, \mathbf{c})$ and the ground truth y_i. The set of all possible expressions, i.e., the hypothesis space Π, can be exponentially large. As a result, finding the optimal expression is challenging and is shown to be NP-hard [195].

Deep Policy Gradient for Symbolic Regression. Recently, a line of work proposes the use of deep reinforcement learning (RL) for searching the governing equations [180, 181, 238]. They represent expressions as binary trees, where the interior nodes of the tree correspond to mathematical operators and leaf nodes of the tree correspond to variables or constants. The key idea is to model the search of different expressions, as a sequential decision-making process for the preorder traversal sequence of the expression trees, using an RL algorithm. A reward function is defined to measure how well a predicted expression can fit the dataset. The deep RNN is used as the RL agent for predicting the next possible node in the expression tree at every step of decision-making. The parameters of the RNN are trained using a policy gradient.

Control Variable Experiment. studies the relationship between a few input variables and the output in the regression problem, with the remaining input variables fixed to be the same [185]. In the controlled setting, the ground-truth equation behaves the same after setting those controlled variables as constants, which is noted as the *reduced-form equation*. For example, the ground-truth equation $\phi = x_1 \times x_3 - x_2 \times x_4$ in Figure 8.2(a) is reduced to $x_1 \times C_1 - C_2$ when controlling x_2, x_3, x_4. Figure 8.2(b,c) presents other reduced-form equations when the control variables are changed. For the corresponding dataset D, the controlled variables are fixed to one value and the remaining variables are randomly assigned. See Figure 8.2(a,b,c) for example datasets generated under different controlling variables.

Vertical Symbolic Regression. starts by finding a symbolic equation involving a small subset of the input variables and iteratively expands the discovered expression by introducing more variables. VSR relies on the control variable experiments introduced above. VSR-GP was the first implementation of vertical symbolic regression using genetic programming (GP) [242]. To fit an expression of n variables, VSR-GP initially only allows variable x_1 to vary and controls the values of all the rest of the variables. Using GP as a subroutine, VSR-GP finds a pool of expressions $\{\phi_1, \ldots, \phi_m\}$ which best fit the data from this controlled experiment. Notice $\{\phi_1, \ldots, \phi_m\}$ are restricted to contain only one free variable x_1 and m is the pool size. A small error indicates ϕ_i is close to the ground truth reduced to the one free variable and thus is marked unmutable by the genetic operations in the following rounds. In the 2nd round, VSR-GP adds a second free variable x_2 and starts fitting $\{\phi'_1, \ldots, \phi'_m\}$ using the data from control variable experiments involving the two free variables x_1, x_2. After n rounds, the expressions in the VSR-GP pool consider all n variables. Overall, VSR expedites the discovery process because the first few rounds of VSR are significantly cheaper than the traditional *horizontal* discovery process, which searches for optimal expression involving all input variables.

8.3 Methodology

Motivation. The prior work of VSR-GP uses genetic programming to edit the expression tree. GP is not allowed to edit internal nodes in the best-discovered expression trees from the previous vertical discovery step, to ensure later genetic operations do not delete the prior knowledge on the governing equation. However, this idea cannot be easily integrated with deep RL-based symbolic regressors.

We explored using deep neural networks in vertical symbolic regression to predict the symbolic equation tree in each vertical expansion step, as detailed in Appendix F.1. However, we encountered two main issues: (1) difficulty in passing gradients from trees to deep neural nets. We need different constraints to embed into the output of RNN to ensure the whole model follows the vertical discovery idea. (2) complexity in concatenating deep networks for predictions at each vertical expansion step. The underlying issue is the expression tree representation.

Our solution is to adopt a new representation for expressions. We extend the context-free grammar definition for symbolic expressions, where a sequence of grammar rules uniquely corresponds to an expression. We view the prediction of symbolic expressions as a sequential decision-making process, selecting grammar rules step-by-step. The RNN predicts grammar rules instead of nodes in the expression tree. The best-discovered reduced-form equation is converted into the start symbol in the grammar, ensuring that the predicted expression from our RNN is always compatible with the prior knowledge of the governing equation. This reduces the hypothesis space and accelerates scientific discovery, as other non-reducible expressions are never sampled from the RNN.

Main Pipeline. Figure 8.1 shows our deep vertical symbolic regression (Vsr-Dpg) pipeline. The core idea is to construct increasingly complex symbolic expressions involving more input variables, based on control variable experiments with fewer controlled variables.

To fit an expression of n variables, we first hold $n-1$ variables as constant and allow only one variable to vary. We aim to find the best expression ϕ_1, that fits the data in this controlled experiment. We use the RNN as the RL agent to search for the best possible expression, which is achieved by using the RNN to sample sequences of grammar rules defined for symbolic equations. Every sequence of rules is converted into an expression, where the constants in the expression are fitted with the dataset. The parameters of the RNN model will be trained through the policy gradient objective. The expression with the best fitness score is returned as the prediction of the RNN. A visualized process is in Figure 8.1(a, e).

Following the idea in VSR, the next step is to classify each constant as either a summary constant or a standalone constant. (1) A constant not relevant to any controlled variables is considered standalone and is preserved in subsequent rounds. (2) A constant that is a sub-expression involving controlled variables is labeled as a summary type and will be expanded in subsequent rounds. In our implementation, a constant with high variance in fitted values across multiple control variable experiments is classified as a summary type; otherwise, it is classified as standalone.

Assuming we find the correct reduced-from equation ϕ_1 after several learning epochs. To ensure Vsr-Dpg does not forget this discovered knowledge of the first round, we want all the expressions to be discovered in the following rounds can be reduced to ϕ_1. Therefore, we construct ϕ_1 as the start symbol for the following round by replacing every summary constant in ϕ_1 as a non-terminal symbol in the grammar (noted as A), indicating a sub-expression. For example, in Figure 8.2(a), both of them are summary constants so the 1st round best-predicted expression $x_1 \times C_1 - C_2$ is converted to the start symbol $x_1 \times A - A$ for the second round.

In the second round, Vsr-Dpg adds one more free variable and starts fitting ϕ_2 using the data from control variable experiments involving two free variables. Similar to the first round, we restrict our search to sub-expressions with the second variable. It is achieved by limiting the output vocabulary of the RNN model. In Figure 8.2(f), the RNN model finds an expression $\phi_2 = x_1 \times C_3 - (x_2 \times C_4)$. This means that the RL agent learns to expand C_1 with another constant C_3 and C_2 with sub-expression $(x_2 \times C_4)$, based on the best-discovered result of the 1st round $\phi_1 = x_1 \times C_1 - C_2$.

VSR-DPG introduces one free variable at a time and expands the equation learned in the previous round to include this newly added variable. This process continues until all the variables are considered. After n rounds, we return the equations with the best fitness scores. The predicted equation will be evaluated on data with no variable controlled. See the steps in Figure 8.1(b,c,d) for a visual demonstration. We summarize the whole process of VSR-DPG in Algorithm 9 in Appendix F.2. The major difference of this approach from most state-of-the-art approaches is that those baselines learn to find the expressions in the full hypothesis space with all input variables, from a fixed dataset collected before training. Our VSR-DPG accelerates the discovery process, because of the small size of the reduced hypothesis space, i.e., the set of candidate expressions involving only a few variables. The task is much easier than fitting the expression in the full hypothesis space involving all input variables.

8.3.1 Expression Represented as Grammar Rules

We propose to represent symbolic expression by extending the existing context-free grammar [257]. A context-free grammar is represented by a tuple (V, Σ, R, S), where V is a set of non-terminal symbols, Σ is a set of terminal symbols, R is a set of production rules and S is a start symbol. In our formulation, (1) Σ is the set of input variables and constants $\{x_1, \ldots, x_n, \texttt{const}\}$. (2) The set of non-terminal symbols V represents sub-expressions, like $\{A\}$. Here A is a placeholder symbol. (3) The set of production rules R represents mathematical operations such as addition, subtraction, multiplication, and division. That is $\{A \to (A + A), A \to (A - A), A \to A \times A, A \to A \div A\}$, where → indicates that the left-hand side is replaced with the right-hand side. (4) The start symbol S is extended to be A, $x_1 \times A - A$, or other symbols constructed from the best-predicted expression under the controlled variables.

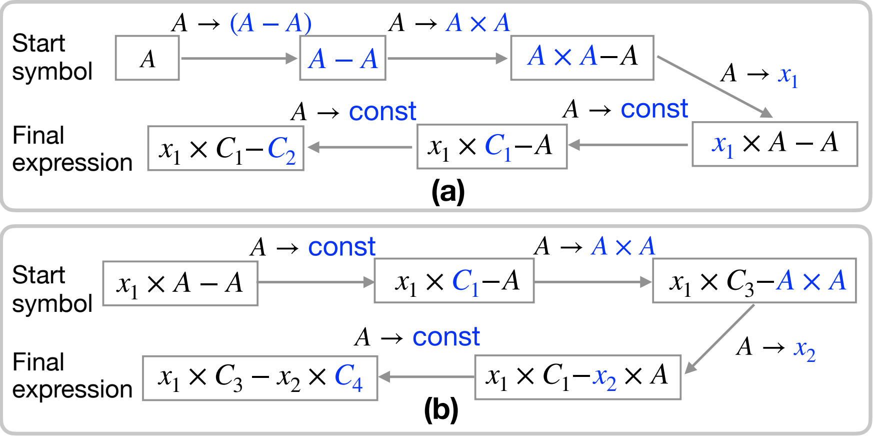


Figure 8.3. Convert a sequence of grammar rules into a valid expression. Each rule expands the first non-terminal symbol in the squared box. The parts that get expanded are color-highlighted.

Starting with the start symbol, successively applying the grammar rules in different orders results in different expressions. Each rule expands the *first* non-terminal symbol in the start symbol. An expression with only terminal symbols is a valid mathematical expression, whereas an expression with a mixture of non-terminal and terminal symbols is invalid. The expression can also be represented as a binary tree. We chose the grammar representation over the binary tree representation because it simplifies the vertical symbolic regression process, avoiding gradient passing and heavy engineering issues.

Figure 8.3 presents two examples of constructing the expression ϕ from the start symbol using a given sequence of grammar rules. In Figure 8.3(a), we first use the subtraction rule $A \to (A - A)$, indicating that the symbol A in expression $\phi = A$ is expanded to $\phi = (A - A)$. By repeatedly using grammar rules to expand the non-terminal symbols, we eventually arrive at our desired expression $\phi = x_1 \times C_1 - C_2$. In Figure 8.3(b), the start symbol is $x_1 \times A - A$. The rule $A \to$ `const` replaces the first non-terminal symbol with a constant C_1, resulting in $x_1 \times C_1 - A$. Finally, we obtain a valid expression $x_1 \times C_1 - C_2$. Figure 8.3 provides another example conversion with the start symbol $x_1 \times A - A$.

8.3.2 Expression Sampling from Recurrent Network

In our vertical symbolic regression setting, the input and output are in the set of grammar rules that cover each input variable, constants, and math operations. We create an embedding layer for the input, noted as $\texttt{Embd}$. For each input rule $r \in R$, its d-dimensional embedding vector is $\texttt{Embd}(r) \in \mathbb{R}^d$.

Sampling Procedure. The RNN module samples an expression by sampling the sequence of grammar rules in a sequential decision-making process. Denote the sampled sequence of rules as $\tau = (\tau_1, \tau_2, \ldots)$. Initially, the RNN takes in the start symbol $\tau_1 = S$ and computes the first step hidden state vector $\mathbf{h}_1$. At t-th time step, RNN uses the predicted output from the previous step as the input of current step τ_t. RNN computes its hidden state vector $\mathbf{h}_t$ using the embedding vector of input token τ_t and the previous time-step hidden state vector $\mathbf{h}_{t-1}$. The linear layer and softmax function are applied to emit a categorical distribution $p(\tau_{t+1}|\tau_t, \mathbf{h}_t)$ over every token in the output vocabulary, which represents the probability of the next possible rule in the half-completed expression $p_\theta(\tau_{t+1}|\tau_t, \ldots, \tau_1)$. The RNN samples one token from the categorical distribution $\tau_{t+1} \sim p(\tau_{t+1}|\tau_t, \mathbf{h}_t)$ as the prediction of the next possible rule. To conclude, the computational pipeline at the t-th step is shown below:

$$\begin{aligned}
\tau_t &= \texttt{Embd}(\tau_t), \\
\mathbf{h}_t &= \texttt{RNN}(\tau_t, \mathbf{h}_{t-1}), \\
\mathbf{s}_t &= W\mathbf{h}_t + \mathbf{b}, \\
p(\tau_{t+1} = r_i|\tau_t, \mathbf{h}_t) &= \frac{\exp(\mathbf{s}_{t,i})}{\sum_{r_j \in R} \exp(\mathbf{s}_{t,j})}, \quad \text{for } r_i \in R
\end{aligned} \tag{8.2}$$

The weight matrix $W \in \mathbb{R}^{d\times|R|}$ and bias vector $\mathbf{b} \in \mathbb{R}^d$ are the parameters of the linear layer. The last row in Equation 8.2 is the softmax layer. The sampled rule r_{t+1} will be the input for the $t+1$-th step. We denote θ as the total parameters of the embedding layer, the RNN, and the linear layer.

After L steps, we obtain the sequence $\tau = (\tau_1, \ldots, \tau_L)$ with probability $p_\theta(\tau) = \prod_{t=1}^{L-1} p_\theta(\tau_{t+1}|\tau_1, \ldots, \tau_t)$. We convert this sequence into an expression by following the procedure described in Section 8.3.1. If we reach the end of the sequence while there are still non-terminal symbols in the converted expression, we randomly add rules containing only terminal symbols to complete the expression. Conversely, if we obtain a valid expression before the sequence ends, we disregard the remaining elements of the sequence and return the valid expression.

Policy Gradient-based Training. We follow the reinforcement learning formulation to train the parameters of the RNN module [261]. The sampled rules before the current step t, i.e., $(\tau_1, \ldots, \tau_t)$, is viewed as the *state* of the t-th step for the RL agent. Those rules in the output vocabulary are the available *actions* for the RL agent. In the formulated decision-making process, the RNN takes in the current state and outputs a distribution over next-step possible actions. The objective of the RL agent is to learn to pick the optimal sequences of grammar rules to maximize the expected rewards. Denote the converted expression from τ as ϕ. A typical reward function is defined from the fitness score of the expression $\texttt{reward}(\tau) = 1/(1 + \texttt{NMSE}(\phi))$. The objective that maximizes the expected reward from the RNN model is defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\texttt{reward}(\tau)\right],$$

where $p_\theta(\tau)$ is the probability of sampling sequence τ from the RNN.

In the next step, the gradient with respect to the objective $\nabla_\theta J(\theta)$ needs to be estimated. We follow the classic REINFORCE policy gradient algorithm [239]. We first sample several times from the RNN module and obtain N sequences $(\tau^1, \ldots, \tau^N)$, an unbiased estimation of the gradient of the objective is computed as $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \texttt{reward}(\tau^i)\nabla_\theta \log p_\theta(\tau^i)$. The parameters of the deep network are updated by the gradient descent algorithm with the estimated policy gradient value. In the literature, several practical tricks are proposed to reduce the estimation variance of the policy gradient. A common choice is to subtract a baseline function b from the reward, as long as the baseline is not a function of the sample batch of expressions. Our implementation adopts this trick and the detailed derivation is presented in Appendix E.1.

Throughout the whole training process, the expression with optimal fitness score from all the sampled expressions is used as the prediction of VSR-DPG at the current round.

Start Symbol Construction in Vertical Discovery. Given the best-predicted equation ϕ and controlled variables $\mathbf{x}_c$, the following step is to construct the start symbol of the next rounds. This operation ensures all the future expressions can be reduced to any previously discovered equation thus all the discovered knowledge is remembered. It expedites the discovery of symbolic expression since other expressions that cannot be reduced to ϕ will never be sampled from the RNN. It requires first classifying the type of every constant in the expression into stand-alone or summary type, through multi-trail control variable experiments. Then we replace each summary constant with a placeholder symbol (i.e., "A") indicating a sub-expression containing controlled variables.

Following the procedure proposed in [242], we first query K data batches $(D_1, \ldots, D_K)$ with the same controlled variables $\mathbf{x}_c$. The controlled variables take the same value within each batch while taking different values across data batches. We fit open constants in the candidate expression ϕ with each data batch by the gradient-based optimizer, like BFGS [243]. We obtain multiple fitness scores $(o_1, \ldots, o_K)$ and multiple solutions to open constants $(\mathbf{c}_1, \ldots, \mathbf{c}_K)$. By examining the outcomes of K-trials control variable experiments, we have: (1) Consistent close-to-zero fitness scores imply the fitted expression is close to the ground-truth equation in the reduced form. That is $o_k \leq \varepsilon$ for all $1 \leq k \leq K$, where ε is the threshold for the fitness scores. (2) Conditioning on the result in case (1), the j-th open constant is a standalone constant when the empirical variance of its fitted values across K trials is less than a threshold ε'. In practice, if the best-predicted expression by the RNN module is not consistently close to zero, then all the constants in the expression are summary constants. Finally, the start symbol is obtained by replacing every summary constant with the symbol "A" according to our grammar.

8.4 Related Work

Recently AI has been highlighted to enable scientific discoveries in diverse domains [256, 262, 263]. Early work in this domain focuses on learning logic (symbolic) representations [211]. Recently, there has been extensive research on learning algebraic equations from data [180, 181] and learning differential equations from data [213–223]. In this domain, a line of works develops robots that automatically refine the hypothesis space, some with human interactions [192, 193, 209, 224]. These works are related to ours because they also actively explore the hypothesis spaces, while they are in biology and chemistry.

Existing works on multivariate regression predominantly rely on pre-trained encoder-decoder structures with extensive training datasets [207], and larger-scale deep models [208]. Our VSR-DPG method is tailored to solve multivariate symbolic regression.

The idea of using a control variable experiment tightly connects to the BACON system [189–194]. While their methods are rule-based systems due to historical limitations, our approach leverages contemporary deep recurrent neural networks.

Our method is also related to symbolic regression methods using probabilistic context-free grammar [257, 264, 265]. While they use a fixed probability to sample rules, we use a deep neural network to learn the probability distribution.

Our VSR-DPG is also tightly connected to deep symbolic regression [180, 181]. We both use deep recurrent networks to predict a sequence of tokens that can be composed into a symbolic expression. However, their method predicts the preorder traversal sequence for the expression tree while our method predicts the sequence of production rules for the expression.

8.5 Experiment

8.5.1 Regression on Algebraic Equations

Experiment Settings. For the dataset on algebraic expressions, we consider the 8 groups of expressions from the Trigonometric dataset [242], where each group contains 10 randomly sampled expressions. In terms of baselines, we consider (1) evolutionary algorithms: Genetic Programming (GP), Eureqa [237] and Control Variable Genetic Programming (VSR-GP) [242]. (2) deep reinforcement learning: Priority queue training (PQT) [238], Vanilla Policy Gradient (VPG) [239], Deep Symbolic Regression (DSR) [180], and Neural-Guided Genetic Programming Population Seeding (GPMeld) [181]. (3) Monte Carlo Tree Search: Symbolic Physics Learner (SPL) [240], (4) pretrained Transformer: end-to-end Transformer (E2ETransformer) [208]. In terms of evaluation metrics, we use the normalized mean-squared error (NMSE) of the best-predicted expression by each algorithm, on a separately-generated testing dataset. We report the median instead of the mean value. Symbolic regression be-

Methods	$(2,1,1)$	$(3,2,2)$	$(4,4,6)$	$(5,5,5)$	$(5,5,8)$	$(6,6,8)$	$(6,6,10)$	$(8,8,12)$
VSR-GP	0.005	0.028	0.086	0.014	0.066	0.066	**0.104**	T.O.
GP	$7E{-}4$	0.023	0.044	0.063	0.102	0.127	0.159	0.872
Eureqa	**<1E-6**	**<1E-6**	0.024	0.158	0.284	0.433	0.910	0.162
SPL	0.006	0.033	0.144	0.147	0.307	0.391	0.472	0.599
E2ETransformer	0.018	0.0015	0.030	0.121	0.072	0.194	0.142	0.112
DSR	**< 1E-6**	0.008	2.815	2.558	2.535	0.936	6.121	0.335
PQT	0.020	0.161	2.381	2.168	2.482	0.983	5.750	0.232
VPG	0.030	0.277	2.990	1.903	2.440	0.900	3.857	0.451
GPMeld	$< 1E{-}6$	0.112	1.670	1.501	2.422	0.964	7.393	T.O.
Vsr-Dpg (ours)	**< 1E-6**	**< 1E-6**	**< 1E-6**	**< 1E-6**	**0.026**	**0.063**	0.114	**0.101**

Table 8.1. On selected algebraic equation datasets, median (50%-quartile) of NMSE values of the best-predicted expressions found by all the algorithms. The set of mathematical operator is $O_p = \{+, -, \times, \sin, \cos, \texttt{const}\}$. The 3-tuples at the top $(\cdot, \cdot, \cdot)$ indicate the number of free variables, singular terms, and cross terms in the ground-truth expressions generating the dataset. O_p stands for the set of allowed operators. "T.O." implies the algorithm is timed out for 48 hours.

longs to combinatorial optimization problems, which have no mean values and are sensitive to outliers. The detailed settings are in Appendix F.3.

Goodness-of-fit Comparison. We consider our Vsr-Dpg against several challenging datasets involving multiple variables. In Table 8.1, we report the median NMSE on the selected algebraic datasets. Our Vsr-Dpg attains the smallest median NMSE values in 7 out of 8 datasets, against a line of current popular baselines including the original VSR-GP. The main reason is deep networks offer many more parameters than the GP algorithm, which can better adapt to different datasets and sample higher-quality expressions from the deep networks.

Extended Large-scale Comparison. In the real world, scientists may collect all available variables that are more than needed into symbolic regression, where only part of the inputs will be included in the ground-truth expression. We randomly pick 5 variables from all the n variables and replace the appeared variable in expressions confined as $(5, 5, 5)$ in Table 8.1. In Table 8.2, we collect the median NMSE values on this large-scale dataset setting. Our Vsr-Dpg scales well because it first detects all the contributing inputs using the control

	Total input variables n in the data				
Methods	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
SPL	0.386	0.554	0.554	0.714	0.815
GP	0.159	0.172	0.218	0.229	0.517
DSR	0.284	0.521	0.522	0.660	0.719
VPG	0.415	0.695	0.726	0.726	0.779
PQT	0.384	0.488	0.615	0.620	0.594
VSR-DPG	**< 1E-6**	**< 1E-6**	**< 1E-6**	**0.002**	**0.021**

Table 8.2. On large-scale algebraic equation dataset, with reported Median NMSE values, our VSR-DPG scales better to more variable settings than baselines due to the control variable experiment.

Methods	$(2, 1, 1)$	$(3, 2, 2)$	$(4, 4, 6)$	$(5, 5, 5)$
SPL	20%	10%	0%	0%
E2ETransformer	0%	0%	0%	0%
VSR-GP	60%	50%	0%	0%
VSR-DPG (ours)	**100%**	**70%**	**60%**	**40%**

Table 8.3. On selected algebraic equations, the exact recovery rate over the best-predicted found by all the algorithms. Our VSR-DPG has a higher rate of recovering the ground-truth expressions compared to baselines.

variable experiments. Notice that those baselines that are easily timeout in this setting are excluded for comparison.

Exact Recovery Comparison. We compare if each learning algorithm finds the exact equation, the result of which is collected in Table 8.3. The discovered equation by each algorithm is further collected in Appendix F.4. We can observe that our VSR-DPG has a higher rate of recovering the ground-truth expressions compared to baselines. This is because our method first use a control variable experiment to pick what are the contributing variables to the data and what are not.

8.5.2 Regression on Ordinary Differential Equations

Task Definition. The temporal evolution of the dynamic system is modeled by the time derivatives of the state variables. Let $\mathbf{x}$ be the n-dimensional vector of state variables, and $\dot{\mathbf{x}}$ is the vector of their time derivatives. The ordinary differential equation (ODE) is of the form $\dot{\mathbf{x}} = \phi(\mathbf{x}, \mathbf{c})$, where constant vector $\mathbf{c} \in \mathbb{R}^m$ are parameters of the dynamic system.

Following the definition of symbolic regression on ODE [240, 265], given a trajectory dataset of state variable and its time derivatives $\{(\mathbf{x}(t_i), \dot{\mathbf{x}}(t_i))\}_{i=1}^{N}$, the symbolic regression task is to predict the best expression $\phi(\mathbf{x}, \mathbf{c})$ that minimizes the average loss on trajectory data. Other formulations of this problem [266] assume we have no access to its time derivatives, i.e., $\{(t_i, \mathbf{x}(t_i))\}_{i=1}^{N}$.

Experiment Setting. We consider recent popular baselines for differential equations, including (1) SINDy [214], (2) ODEFormer [266], (3) Symbolic Physics Learner (SPL) [240]. (4) Probabilistic grammar for equation discovery (ProGED) [264]. In terms of the dataset, we consider the Lorenz Attractor with $n = 3$ variables, Magnetohydrodynamic (MHD) turbulence with $n = 6$ variables, and Glycolysis Oscillation with $n = 7$ variables. All of them are collected from [214]. To evaluate whether the algorithm identifies the ground-truth expression, we use the Accuracy metric based on the coefficient of determination (R^2). The detailed experiment configurations are in Appendix F.3.

Result Analysis. The results are summarized in Table 8.4. Our proposed Vsr-Dpg discovers a set of differential expressions with much higher quality than the considered baselines. We further provide a visual understanding of the proposed Vsr-Dpg method in Figure 8.4. The data of our Vsr-Dpg are drawn from the intersection of the mesh plane and the curve on the Lorenz attractor. In comparison, the current baselines draw data by picking a random trajectory or many random points on the curve. We notice the ODEFormer is pre-trained on differential equations up to two variables, and thus does not scale well with more variable settings. The predicted differential equations by each algorithm are in Appendix F.4.

8.6 Summary

In this chapter, we propose Vsr-Dpg to accelerate the discovery of governing equations involving many variables, which is beyond the capabilities of current state-of-the-art approaches. Vsr-Dpg follows a vertical discovery path, building equations involving more and more input variables using control variable experiments. Vsr-Dpg has the potential to supercharge current popular approaches because the first few steps following the vertical discovery route are much cheaper than discovering the equation in the full hypothesis space.

	Lorenz Attractor (3 variables)	MHD Turbulence (5 variables)	Glycolysis Oscillations (7 variables)
SPL	**100%**	50%	14.2%
SINDy	**100%**	0%	0%
ProGED	0%	0%	0%
ODEFormer	0%	0%	NA
VSR-DPG (ours)	**100%**	**100%**	**87%**

Table 8.4. On the differential equation dataset, ($R^2 \geq 0.9999$)-based accuracy is reported over the best-predicted expression found by all the algorithms. Our VSR-DPG method can discover the governing expressions with a much higher accuracy rate than baselines.

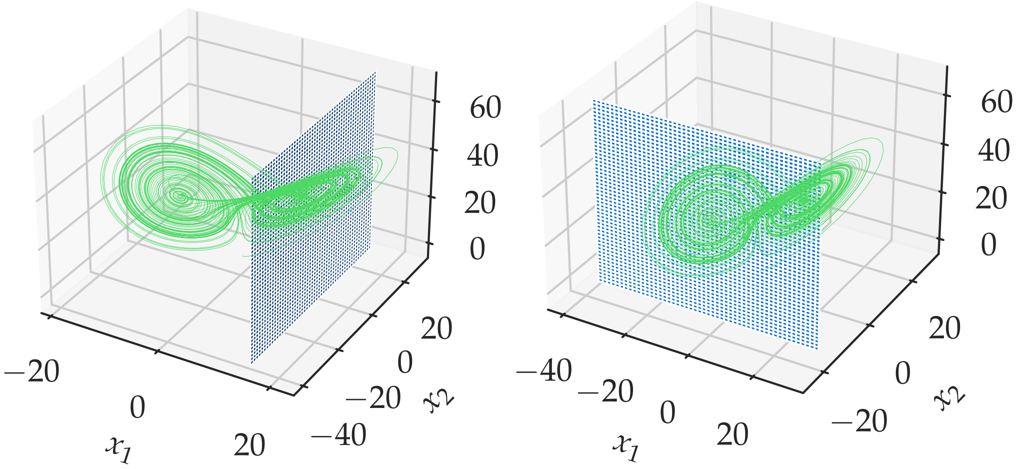


Figure 8.4. Visualization of VSR-DPG controlling variables x_1 (Left) and x_2 (Right) for the Lorenz attractor. The data of our VSR-DPG are drawn from the intersection of the mesh plane and the curve on the Lorenz attractor. In comparison, the ODEFormer draws data by picking a consecutive sequence $\{(t_i, \mathbf{x}(t_i))\}_{t=0}^{N}$ without knowing its time derivative on the curve.

Experimental results show VSR-DPG can uncover complex equations with more variables than what current approaches can handle.

9. Active Symbolic Discovery of Ordinary Differential Equations via Phase Portrait Sketching

9.1 Introduction

Uncovering the governing principles of physical systems from experimental data is a crucial task in AI-driven scientific discovery [177, 215, 216]. Recent advancements have introduced various methods for uncovering knowledge of dynamical systems in symbolic Ordinary Differential Equation (ODE) form, leveraging techniques such as genetic programming [184], sparse regression [214, 267], Monte Carlo tree search [240], pre-trained Transformers [268], and deep reinforcement learning [269].

State-of-the-art approaches discover the symbolic ODEs using a fixed, pre-collected training dataset. However, their performance is often heavily influenced by the quality of the collected data. As illustrated in Figure 9.1, we find that the best-discovered ODEs from the most recent baseline, that is ODEFormer [266], may fit some test trajectories well, but fit other test trajectories poorly. This observation highlights the need for new methods that actively query informative trajectory data to improve ODE discovery.

Suppose trajectory data can be obtained from a data oracle by specifying the initial conditions. To minimize excessively querying the oracle, a key challenge emerges: given a set of candidate ODEs predicted by a learning method, how can initial conditions within the variable intervals be strategically selected to obtain informative data?

Previous work in the active learning literature typically maintains a large set of data, evaluates their informativeness, and then queries the most informative data points [226, 270]. However, the chaotic nature of dynamical systems complicates the direct application of such methods. The Butterfly effect states that small variations in initial conditions can lead to vastly different outcomes. For instance, as illustrated in Figure 9.2(c), selecting initial conditions near $(3, 0)$ for ϕ_1 can result in trajectories that diverge in opposite directions. Effectively addressing this variability requires densely sampling initial conditions to thoroughly explore the space. Existing active learning-based approaches will be computationally prohibitive and demand significant memory resources, particularly in high-dimensional dynamical systems.

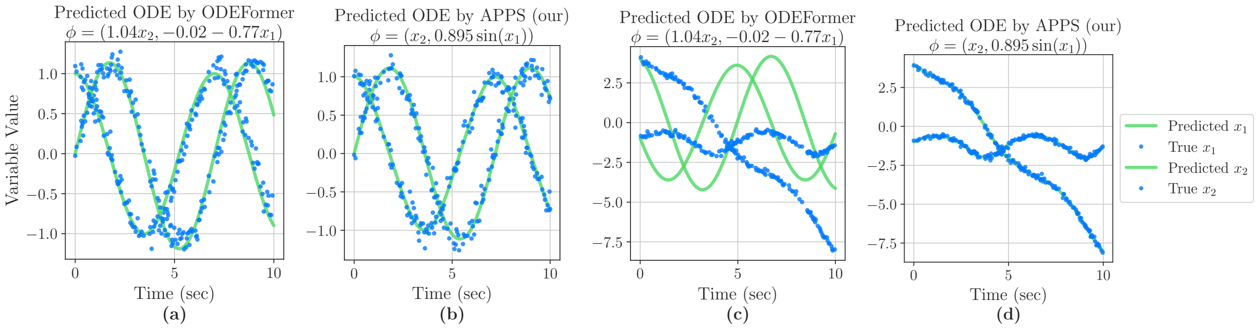


Figure 9.1. The performance of predicted ODE from passively-learned baseline is heavily influenced by the collected training data while our APPS method is not. The dots represent noisy ground-truth trajectory data, and the lines show predicted values of state variables under identical initial conditions. **(a, b)** Our APPS and the baseline predict accurately for the trajectory starting at $\mathbf{x}_0 = (0, 1)$. **(c, d)** For the trajectory starting at $\mathbf{x}_0 = (4, -1)$, the baseline performs poorly while APPS maintains accuracy.

To address these challenges, we propose a novel approach to data querying. We consider selecting a batch of close-neighbor initial conditions instead of individual initial conditions. This process begins by sketching the dynamics in smaller regions, identifying an *informative region* in the phase space, and then sampling a batch of initial conditions from this region. Figure 9.2(c) illustrates this idea using phase portraits for three candidate ODEs. Region u_2 is chosen because the trajectories generated by the candidate ODEs exhibit greater divergence in this region than region u_1. Section 9.3 provides detailed region selection criteria.

Thus, we introduce **A**ctive Symbolic Discovery of Ordinary Differential Equations via **P**hase **P**ortrait **S**ketching (APPS), which consists of two key components: (1) a deep sequential decoder, which guides the search for candidate ODEs by sampling from the defined grammar rules. (2) a data query and evaluation module that actively queries the data using sketched phase portraits and evaluates the candidate ODE. In experiments, we evaluate APPS against several popular baselines on two large-scale ODE datasets. 1) APPS achieves the lowest median NMSE (in Table 9.1 and Table 9.2) across multiple datasets under noiseless and noisy settings. 2) Compared to other active learning strategies, APPS is more time efficient in benchmark datasets (in Table 9.3). [1].

[1] ↑The code is at https://github.com/jiangnanhugo/APPS-ODE.

9.2 Preliminaries

Ordinary Differential Equations (ODEs) describe the evolution of dynamical systems in continuous time. Let vector $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t)) \in \mathbb{R}^n$ be the state variables of the system of time t. The temporal evolution of the system is governed by the time derivatives of the state variables, denoted as $\frac{dx_i}{dt}$. The general form of the ODE is written as:

$$\frac{dx_i}{dt} = f_i(\mathbf{x}(t), \mathbf{c}), \quad \text{for } i = 1, \ldots, n,$$

where f_i can be a linear or nonlinear function of the state variables $\mathbf{x}$ and coefficients $\mathbf{c}$. The ODE is noted as a tuple $(f_1, f_2, \ldots, f_n)$ for simplicity in this chapter. Function f_i is symbolically expressed using a subset of input variables in $\mathbf{x}$ and coefficients in $\mathbf{c}$, connected by mathematical operators such as addition and cosine functions. For example, we use $(10\sin(x_2), 4\cos(x_1 + 2))$ to represent the ODE:

$$\frac{dx_1}{dt} = 10\sin(x_2), \frac{dx_2}{dt} = 4\cos(x_1 + 2).$$

Given an initial condition $\mathbf{x}_0$, the solution to the ODE is a *trajectory* of state variables $(\mathbf{x}_0, \mathbf{x}(t_1), \ldots, \mathbf{x}(t_k))$ observed at discrete time points $(t_1, \ldots, t_k)$, possibly with noise. The trajectory is noted as τ for simplicity.

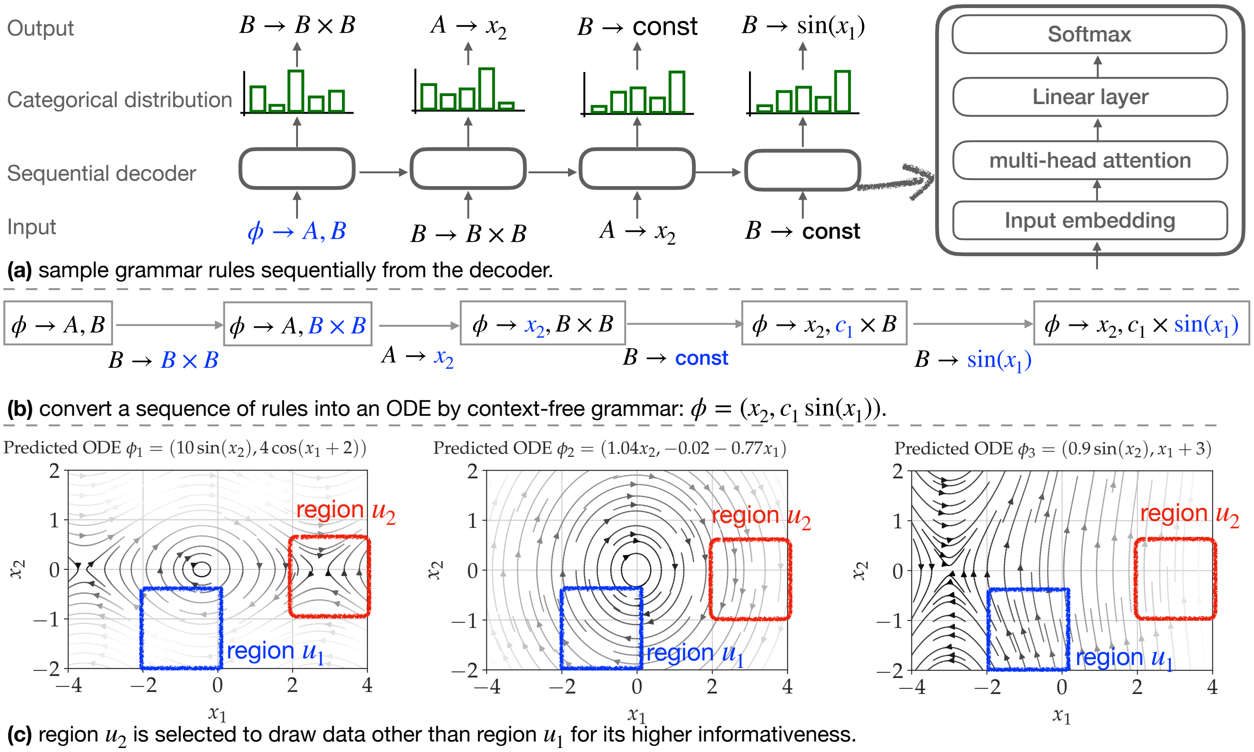
Phase Portrait is a qualitative analysis tool for studying the behavior of dynamical systems [271]. Phase portraits are plotted using the state variables $\mathbf{x}$ and their time derivatives $(f_1, \ldots, f_n)$. A curve in the phase portrait is a short trajectory of the system over time from a given initial condition. The arrow on the curve indicates the direction of change. By examining these curves, we can infer key properties of the system, such as stability, equilibrium points, and periodic behavior. Figure 9.2(c) shows phase portraits for three different ODEs. These portraits are generated by sampling random initial conditions within the variable intervals and evolving the system for a short time.

Symbolic Discovery of Ordinary Differential Equations seeks to uncover the symbolic form of an ODE that best fits a dataset of observed trajectories. According to [265] and [240], we are given a dataset of collected trajectories $D = \{\tau_1, \ldots, \tau_N\}$ and a set of mathematical operators $\{+,-,\times,\div,\sin\ldots\}$. Denote $\phi(\mathbf{x}(t),\mathbf{c})$ as a candidate ODE, where $\mathbf{c}$ indicates the coefficients. The objective is to predict the symbolic form of the ODE that minimizes the distance between the predicted and observed trajectories, which is formalized as an optimization problem:

$$\arg\min_{\phi\in\Pi}\frac{1}{|D|}\sum_{\tau\in D}\sum_{i=1}^{k}\ell(\mathbf{x}(t_i),\hat{\mathbf{x}}(t_i)),\qquad \text{where } \hat{\mathbf{x}}(t_i)=\mathbf{x}_0+\int_0^{t_i}\phi(\mathbf{x}(t),\mathbf{c})\,dt. \tag{9.1}$$

Π is the set of all possible ODEs, trajectory $\tau := (\mathbf{x}_0, \mathbf{x}(t_1),\,\ldots,\mathbf{x}(t_k))$, $\mathbf{x}(t)$ is the ground-truth observations of the state variable. Trajectory $(\mathbf{x}_0,\hat{\mathbf{x}}(t_1),\ldots,\hat{\mathbf{x}}(t_k))$ is the predicted state variables according to the candidate ODE ϕ. The predicted trajectory $(\mathbf{x}_0,\hat{\mathbf{x}}(t_1),\ldots,\hat{\mathbf{x}}(t_k))$ is obtained by numerically integrating the ODE from the given initial state $\mathbf{x}_0$ to the final time t_k. The loss function ℓ computes the summarized distance between the predicted and ground-truth trajectories at each time step. A typical loss is the Normalized Mean Squared Error (NMSE, defined in Appendix F.3). Except for the above formulation, prior works in symbolic regression use the approximated time derivative as the *label* to discover each expression f_i separately, which is known as gradient matching. We leave the discussion to Related Work.

Recent research explored deep reinforcement learning to discover the governing equations from data [180, 181, 238]. In these approaches, each expression is represented as a binary tree, with interior nodes corresponding to mathematical operators and leaf nodes to variables or constants. An ODE with n variables is represented by n trees. The key idea is to frame the search for different ODEs as a sequential decision-making process based on the preorder traversal sequence of expression trees. A high reward is assigned to candidates which fit the data well. The search is guided by a deep sequential decoder, often based on RNN, LSTM, or decoder-only Transformer, that learns the optimal probability distribution for selecting the next node in the expression tree at each step. The parameters of the decoder are trained with the policy gradient algorithm.



Output
$B \to B \times B$
$A \to x_2$
$B \to$ const
$B \to \sin(x_1)$
Categorical distribution
Sequential decoder
Input
$\phi \to A, B$
$B \to B \times B$
$A \to x_2$
$B \to$ const
Softmax
Linear layer
multi-head attention
Input embedding
(a) sample grammar rules sequentially from the decoder.
$\phi \to A, B$
$\phi \to A, B \times B$
$\phi \to x_2, B \times B$
$\phi \to x_2, c_1 \times B$
$\phi \to x_2, c_1 \times \sin(x_1)$
$B \to B \times B$
$A \to x_2$
$B \to$ const
$B \to \sin(x_1)$
(b) convert a sequence of rules into an ODE by context-free grammar: $\phi = (x_2, c_1 \sin(x_1))$.
Predicted ODE $\phi_1 = (10\sin(x_2), 4\cos(x_1+2))$
Predicted ODE $\phi_2 = (1.04x_2, -0.02 - 0.77x_1)$
Predicted ODE $\phi_3 = (0.9\sin(x_2), x_1 + 3)$
region u_2
region u_1
x_1
x_2
(c) region u_2 is selected to draw data other than region u_1 for its higher informativeness.

Figure 9.2. The pipeline of APPS for symbolic discovery of ODEs consists of 3 steps: **(a)** ODEs are sampled from the sequential decoder by iteratively sampling grammar rules. The predicted rule at each step serves as input for the decoder in the subsequent step. **(b)** The sampled sequence of grammar rules is converted into a valid ODE with $n = 2$ variables. Each rule expands the first non-terminal symbol, with the expanded parts highlighted in blue colors for clarity. **(c)** The phase portrait for the predicted ODEs (e.g., ϕ_1, ϕ_2, ϕ_3) is sketched, and regions with high informativeness, such as u_2, are identified to query the new trajectory data. In region u_2, ϕ_1 exhibits a saddle point, ϕ_2 moves downward, and ϕ_3 moves upward. In contrast, in region u_1, all trajectories move from right to left. Differentiating the predicted expressions is easier in region u_2 than in region u_1.

9.3 Methodology

9.3.1 Motivation

For the task of symbolic discovery of ODEs, we observe that existing methods frequently overfit the training data. This issue is illustrated in Figure 9.1 using ODEFormer [266], a recent baseline designed to learn ODEs from a fixed training dataset. In the example, the best-predicted ODE is given by $\phi = (1.04x_2, -0.02 - 0.77x_1)$. We evaluate ϕ on noisy test trajectories (depicted as blue dots) with two distinct initial conditions. While ϕ closely aligns with the trajectory originating at $\mathbf{x}_0 = (0, 1)$, as shown by the green curve, it produces substantial errors for a trajectory starting at $\mathbf{x}_0 = (4, -1)$, where the predicted curve deviates significantly from the ground truth.

This observation motivates us to actively identify informative trajectory data to better differentiate candidate expressions during the learning process. Each trajectory is generated by querying the data oracle with a specified initial condition $\mathbf{x}_0$. An initial condition is deemed *informative* if the resulting trajectory for different candidate ODEs diverges significantly. The key challenge lies in selecting such informative initial conditions from the variable intervals for a given set of candidate ODEs.

For addressing this issue, a common approach in active learning [226] is to maintain a large set of potential initial conditions, evaluate their informativeness, and query the most informative points. However, the butterfly effect in chaos theory [272] suggests existing works in active learning are not directly applicable. The chaotic behavior states small changes in initial conditions can lead to drastically different outcomes in dynamical systems. For example, as shown in Figure 9.2(c), selecting points near $(3, 0)$ (inside the red region u_2) for ϕ_1 can lead to trajectories diverging either towards the top right or the bottom left. Such chaotic behavior necessitates the existing active learning methods to maintain a large set of initial conditions to adequately cover the domain, which becomes infeasible for high-dimensional dynamical systems.

To mitigate this issue, we consider selecting a beam of near-neighbor points rather than individual points. We propose first to select a highly informative region and sample a batch of initial conditions within that region. In this research, the region is represented as an n-dimensional cube of fixed width. A region is regarded as *informative* if the majority of sampled initial conditions within it yield informative trajectories for the given candidate ODEs.

Figure 9.2(c) illustrates our region-based approach using the phase portraits of three candidate ODEs: ϕ_1, ϕ_2, and ϕ_3. Each curve in the phase portrait represents a short trajectory, with its starting point and direction indicating the initial conditions and the direction of evolution over time. A closer look reveals significant differences in dynamics within region u_2 across the ODEs. While the curves in region $u_1 = [-2, 0] \times [-2, 0]$ consistently move from the bottom right to the top left in all phase portraits, the trajectories in region $u_2 = [2, 4] \times [-1, 1]$ exhibit drastically different behaviors. This indicates that trajectories originating from region u_2 are more divergent and thus more informative.

Main Procedure. The proposed Apps, illustrated in Figure 9.2, comprises two key components: (1) Deep Sequential Decoder. This module predicts candidate ODEs by sampling sequences of grammar rules defined for symbolic ODE representation. (2) Data Sampling Module. Using the proposed phase portrait sketching, this module selects a batch of informative ground-truth data points.

Throughout the training process, the reward for the predicted ODEs is computed using the queried data, and the decoder parameters are updated via policy gradient estimation. Among all sampled candidates, Apps selects the ODE with the smallest loss value (as defined in Equation 9.1) as the final prediction.

Connection to Existing Approaches. Like [266], Apps employs a Transformer-based decoder. However, unlike [266], which learns from fixed data, Apps actively queries new data. The learning objective of Apps is inspired by [180], where both approaches guide the search for the optimal equation as a decision-making process over a sequence of tokens.

Existing active learning methods, particularly in symbolic regression, have largely overlooked the chaotic behaviors inherent in dynamical systems. For instance, [273] proposed a separate generative model for sampling informative data, assuming that input data within a small region should exhibit minimal output divergence. However, this assumption fails to hold in the context of dynamical systems. Additionally, [274] formulated an optimization problem based on the Query-By-Committee (QBC) method in active learning, to find those informative initial conditions. But the optimization needs to maintain a large set of data points, to account for the chaotic behaviors. The rest of the discussion is provided in the Related Work.

9.3.2 The Learning Pipeline

Data Assumption. Our method relies on the assumption that we can query a data oracle $\mathcal{O}$ by specifying the initial conditions $\mathbf{x}_0$ and discrete times $T = (t_1, \ldots, t_k)$. The oracle executes $\mathcal{O}(\mathbf{x}_0, T)$ and returns a (noisy) observation of the trajectory at the specified discrete times T. In science, this data query process is achieved by conducting real-world experiments with specified configurations. Recent work [275–277] also highlight the importance of having the oracle that can actively query data points, rather than learning from a fixed dataset.

Expression Representation. To enable the sequential decoder to predict an ODE by generating a sequence step-by-step, we extend the context-free grammar to represent an ODE as a sequence of grammar rules [240, 257, 265]. The grammar is defined by the tuple $\langle V, \Sigma, R, S\rangle$, where V is a set of non-terminal symbols, Σ is a set of terminal symbols, R is a set of production rules and $S \in V$ is the start symbol.

More specifically, each component of the grammar is: 1) For the non-terminal symbols, we use A to represent a sub-expression for $\frac{dx_1}{dt}$ and B to represent a sub-expression for $\frac{dx_2}{/}dt$. For dynamical systems with n variables, we use n distinct non-terminal symbols. 2) The terminal symbols include the input variables and constants $\{x_1, \ldots, x_n, \texttt{const}\}$. 3) The production rules correspond to mathematical operations. For example, the addition operation is represented as $A \to (A + A)$, where the rule replaces the left-hand symbol with the right-hand side. 4) The start symbol is redefined as "$\phi \to A, B$", where the comma notation indicates that A and B represent two separate equations in a two-variable dynamical system. Similarly, there will be n non-terminal symbols connected by $n-1$ comma for n-dimensional dynamical system.

Starting from the start symbol, different symbolic ODEs are constructed by applying the grammar rules in various sequences. An ODE is valid if it only consists of terminal symbols; otherwise, it is invalid. Figure 9.2(b) provides an example of constructing the ODE $\frac{dx_1}{dt} = x_2$, $\frac{dx_2}{dt} = -0.9\sin(x_1)$ from the start symbol $\phi \to A, B$ using a sequence of grammar rules. The replaced parts are color highlighted. Initially, the multiplication rule $B \to B \times B$ is applied, replacing the symbol B in $f_2 = B$ with $B \times B$, resulting in $\phi \to A, B \times B$. Next, the rule $A \to x_2$ is applied, yielding $\phi \to x_2, B \times B$. Iteratively applying the rules, we eventually obtain $\phi \rightarrow x_2, c_1 \times \sin(x_1)$, which corresponds to one candidate ODE $\phi = (x_2, c_1 \sin(x_1))$. The coefficient $c_1 = -0.9$ is obtained when fitting to the trajectory data. The procedure of coefficient fitting is described in Appendix G.2 "Implementation of APPS" section.

Sampling ODEs from Decoder. The proposed APPS is built on top of a sequential decoder, which generates different ODEs as a sequential decision-making process. The decoder can be RNN, LSTM, or the decoder-only Transformer. The input and output vocabulary is the set of allowed rules covering input variables, coefficients, and mathematical operators. Predicting ODEs involves using the decoder to sample a sequence of grammar rules, where each sequence corresponds to a candidate ODE using previously defined grammar. The objective of APPS is to maximize the probability of sampling those ODEs that fit the data well. This is achieved through the REINFORCE objective, where the objective computes the expected reward of ODE to the data. In our formulation, the reward is evaluated on selected data by the phase portrait sketching module.

As shown in Figure 9.2(a), the decoder receives the start symbol $s_0 =$ "$\phi \rightarrow A, B$" and outputs a categorical distribution $p_\theta(s_1|s_0)$ over rules in the output vocabulary. This distribution represents the probabilities of possible next rules in the partially completed expression. One token is drawn from this distribution, $s_1 \sim p(s_1|s_0)$, which serves as the prediction for the second rule and is used as the input for the next step. At t-th step, the predicted output from the previous step s_t is used as the input for the current step. The decoder draws rule s_{t+1} from the probability distribution $s_{t+1} \sim p_\theta(s_{t+1}|s_0, \ldots, s_t)$. This process iterates until maximum steps are reached, with a probability of $p_\theta(s) = \prod_{i=1}^{m-1} p_\theta(s_i|s_1, \ldots, s_{i-1})$. The sampled sequence is converted into an expression following the definition previously described in "Expression Representation".

Active Query Data with Phase Portrait Sketching. To evaluate the goodness-of-fit of generated ODEs from the decoder and differentiate which one is better, we propose comparing the phase portrait of predicted ODEs. We sketch the phase portrait using collections of short trajectories, all starting from the same initial conditions and sharing the same time sequence.

Following our discussion in the "Motivation" section, a region is considered informative for distinguishing between two candidate ODEs if their sketched phase portraits differ. To identify such regions, we randomly sample several and sketch the phase portraits for all candidate ODEs within each. The most informative region is then selected, and we query the data oracle (noted as $\mathcal{O}$) for the ground-truth trajectory in that region.

Formally, assume we are given M ODEs, $\{\phi_1, \ldots, \phi_M\}$, and K randomly selected regions, $\{u_1, \ldots, u_K\}$. Each region u_k is a Cartesian product of n intervals, expressed as $u_k = [a_1, b_1] \times \cdots \times [a_n, b_n]$. To sketch the dynamics of candidates in the region u_k, we uniformly sample L points in u_k, $\mathbf{x}^1, \ldots, \mathbf{x}^L$, as initial conditions. For region u_k, the trajectory $\tau_{m,k,l} = (\mathbf{x}^l, \hat{\mathbf{x}}(t_1), \ldots, \hat{\mathbf{x}}(t_k))$ is generated by the expression ϕ_m, starting from the l-th initial condition $\mathbf{x}^l$ and evolving over time according to the numerical integration $\hat{\mathbf{x}}(t_i) = \mathbf{x}^l + \int_0^{t_i} \phi_m(\mathbf{x}(t), \mathbf{c})\, \mathrm{d}t$ for $t_i \in \{t_1, \ldots, t_k\}$. The resulting L short trajectories form a sketched phase portrait for ODE ϕ_m in the region u_k.

Two expressions, ϕ_m and $\phi_{m'}$, have similar sketches in region u_k if their corresponding trajectories, starting from the same initial condition, are close. Specifically, this occurs when $\sum_{l=1}^{L} \|\tau_{m,k,l} - \tau_{m',k,l}\| \approx 0$. We define the pairwise informative score between ϕ_m and $\phi_{m'}$ in region u_k as:

$$\texttt{IF}(\phi_m, \phi_{m'}, u_k) = \frac{1}{L} \sum_{l=1}^{L} \|\tau_{m,k,l} - \tau_{m',k,l}\|_2^2 \tag{9.2}$$

The total informative score for a region (denoted as $\texttt{IF}(u_k)$) is the sum of the pairwise informative scores for every pair of candidate ODEs. The informative score for region u_k is:

$$\texttt{IF}(u_k) = \frac{1}{M} \sum_{m=1}^{M} \sum_{m'=m+1}^{M} \texttt{IF}(\phi_m, \phi_{m'}, u_k) \tag{9.3}$$

We select the region with the highest informative score, denoted $u^* \leftarrow \arg\max_{k=1}^{K} \texttt{IF}(u_k)$. A batch of m initial conditions, $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, is then sampled from region u^*, and the data oracle $\mathcal{O}(\mathbf{x}_i, T)$ is queried with the given initial conditions. The obtained ground-truth trajectories are used to compute the reward function for the objective, which in turn updates the model's parameters. In practice, the relative size of the regions and the number of sampled regions are set as hyper-parameters in the experiments.

Policy Gradient-based Training. The REINFORCE objective that maximizes the expected reward is

$$J(\theta) := \mathbb{E}_{s\sim p_\theta(s)}[\texttt{reward}(s)]$$

where $p_\theta(s)$ is the probability of sampling sequence s and θ represents the parameters of the decoder. Following the REINFORCE policy gradient algorithm [239], the gradient *w.r.t.* the objective $\nabla_\theta J(\theta)$ is estimated by the empirical average over the samples from the probability distribution $p_\theta(s)$. We first sample N sequences $(s^1, \ldots, s^N)$, and an unbiased estimation of the gradient of the objective is computed as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}(\texttt{reward}(s^i) - b)\nabla_\theta \log p_\theta(s^i)$$

where the baseline b is the empirical mean reward of current sampled sequences. The baseline is used to reduce the variance of the learning. The parameters of the decoder are updated using the estimated policy gradient value. This update process increases the probability of generating high goodness-of-fit ODEs. Detailed derivations are presented in Appendix G.2.

9.4 Related Work

AI-driven Scientific Discovery. Artificial intelligence has increasingly been employed to accelerate discoveries in learning ordinary and partial differential equations directly from data [214–219, 221–223, 278].

Symbolic Regression for ODEs. Symbolic regression, traditionally used to identify algebraic equations between input variables and output labels, has been extended to discover ODEs. A key ingredient is gradient matching, which approximates labels for symbolic regression by using finite differences of consecutive states along a trajectory [240, 264, 265, 268]. Recent methods, such as SINDy and its extensions [214, 279], leverage sparse regression techniques to directly learn the structure of ODEs and PDEs from data. They perform particularly well with trajectory data sampled at small, regular time intervals, where the approximations closely align with true derivatives.

Neural Networks Learns Implicit ODEs. This research direction involves learning ODE implicitly. Early work employed Gaussian Processes to model ODEs [280]. Neural ODEs further advanced the field by parameterizing ODEs with neural networks, enabling training through backpropagation via ODE solvers [223]. Physics-informed neural networks integrate physical knowledge, such as conservation laws, into the modeling process [278]. Meanwhile, Fourier neural operators use neural networks to learn the functional representation [281].

Active Learning aims to query informative unlabeled data to accelerate convergence with fewer samples [282–285]. In symbolic regression, query-by-committee strategies have been explored to actively query data for discovering algebraic equations [277, 286]. For example, [273] proposed a method that learns uncertainty distributions using neural networks and queries data with high uncertainty. However, all these methods largely overlooked the chaotic behaviors inherent in dynamical systems.

9.5 Experiments

This section shows our Apps can find ODEs with the smallest errors (Normalized MSE) among all competing approaches, under noiseless, noisy, and irregular time settings (see Table 9.1 and Table 9.2). Compared to the baselines, our Apps data query strategy requires fewer data and attains a better ranking of the TopK candidate ODEs (see Table 9.3).

9.5.1 Experimental Settings

Datasets. We consider 2 datasets of multivariate variables, including (1) Strogatz dataset [266] of 80 instances, collected from the Strogatz textbook [271]. It is formalized as a benchmark dataset by [266]. (2) ODEBase dataset [287] of 114 instances, containing equations from chemistry and biology. Each dataset is further partitioned by the number of variables contained in the ODE.

We consider 3 different conditions: (1) regular time noiseless condition, (2) regular time noisy condition, and (3) irregular time condition. In the noiseless setting, the obtained data is exactly the evaluation of the ground-truth expression. In the noisy setting, the obtained data is further perturbed by Gaussian noise. We add multiplicative noise by replacing each $\mathbf{x}(t_i)$

	Strogatz dataset				ODEbase dataset			
	($\sigma^2 = 0, \alpha = 0$)				($\sigma^2 = 0, \alpha = 0$)			
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
SPL	0.787	0.892	1.921	2.865	0.867	2.17	4.75	13.16
E2ETransformer	$6.47E{-}4$	1.620	$T.O.$	$T.O.$	0.757	$T.O.$	$T.O.$	$T.O.$
ProGED	0.129	0.666	2.68	3.856	0.317	2.134	$T.O.$	$T.O.$
SINDy	$1.90E{-}4$	**0.217**	1.539	4.810	0.521	2.112	8.334	52.12
ODEFormer	0.0303	0.9261	1.033	1.010	0.213	0.245	1.213	3.148
Apps (ours)	**2.06E−6**	0.2912	**1.011**	**0.521**	**0.1318**	**0.1306**	**1.046**	**3.054**

Table 9.1. On the *noiseless* datasets with regular time sequence ($\sigma^2 = 0, \alpha = 0$), Median NMSE is reported over the best-predicted expression found by all the algorithms. Our Apps method can discover the governing expressions with smaller NMSE values than baselines, under the noiseless setting. T.O. means termination with a 24-hour limit.

	Noisy Strogatz datasets				Irregular Strogatz dataset			
	($\sigma^2 = 0.01, \alpha = 0$)				($\sigma^2 = 0, \alpha = 0.1$)			
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
SPL	0.938	1.019	2.915	3.068	0.127	0.526	3.196	4.193
SINDy	$6.4E{-}3$	4.152	2.498	5.21	$6.66E{-}4$	0.472	0.827	4.163
ProGED	0.121	0.658	3.673	3.856	0.134	0.769	2.766	4.181
ODEFormer	0.139	0.621	2.392	0.812	0.031	1.036	1.51	1.011
Apps (ours)	**7.75E-4**	**0.369**	**1.381**	**0.657**	**1.06E-6**	**0.215**	**1.012**	**0.947**

Table 9.2. On the Strogatz dataset, the Median NMSE is reported over the best-predicted expression found by all the algorithms under noisy or irregular time sequence settings.

with $(1+\varepsilon)\mathbf{x}(t_i)$, and ε is sampled from a zero mean multivariate Gaussian distribution with diagonal variances $\texttt{diag}(\sigma^2, \dots, \sigma^2)$. The noise rate is determined by σ^2. For both noiseless and noisy settings, the data points are sampled at regular time intervals. In the irregular time setting, we first generate the regular time sequence and drop a fraction with probability α. The rate of time irregularity is determined by α.

Baselines. We consider a line of recent works for symbolic equation discovery as our baselines. The methods using passive data query strategy are as follows: (1) SINDy [214], (2) ODEFormer [266], (3) Symbolic Physics Learner (SPL) [240], (4) Probabilistic grammar for equation discovery (ProGED) [265], (5) end-to-end Transformer (E2ETransformer) [208].

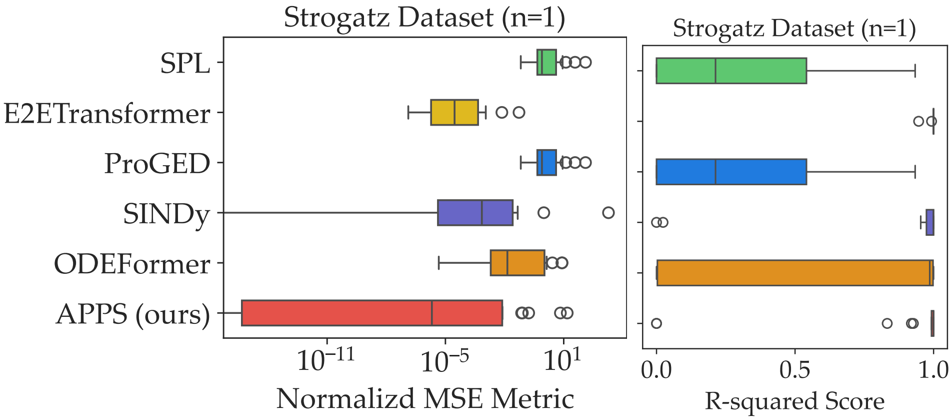


Figure 9.3. On the selected data (Strogatz dataset with $n = 1$), quartiles of NMSE and R^2 scores of the learning algorithms.

Evaluation. For evaluating all the methods, we considered 3 different metrics: (1) goodness-of-fit using NMSE, (2) empirical running time of data querying step, and (3) ranking-based distance. The goodness-of-fit using the NMSE indicates how well the learning algorithms perform in discovering symbolic expressions. Given the best-predicted expression by each algorithm, we evaluate the goodness-of-fit on a larger testing set with longer time steps and a larger batch size of data. The median (50%) of the NMSE is reported in the benchmark table. The full quantiles $(25\%, 50\%, 75\%)$ of the NMSE are further provided. The remaining details of the experiment settings are in Appendix G.3.

9.5.2 Experimental Analysis

Goodness-of-fit Benchmark. We summarize our APPS on several challenging multivariate datasets with noiseless data in Table 9.1. It shows our APPS attains the smallest median NMSE values on all datasets, against a line of current popular baselines. The performance of SPL and E2Etransformer drops greatly on irregular time sequences because the approximated time derivative becomes inaccurate when missing the intermediate sequence. Our APPS does not suffer from that because it outputs the predicted trajectory and does not need to approximate the time derivative. Another reason is the decoder with massive parameters can better adapt to actively collected datasets.

Noisy and Irregular Time Settings. We examine the performance of predicting trajectories in the presence of noise and irregular time sequences in Table 9.2. The ground-truth trajectory is subject to Gaussian noise with zero mean and $\sigma^2 = 0.05$, and an irregularly sampled sequence where 50% of evenly spaced points are uniformly dropped. The predicted trajectory by each algorithm is compared against the ground truth, utilizing identical initial conditions. Our Apps still attains a relatively smaller NMSE against baselines under the two settings.

Quantiles of Evaluation Metrics. We further report the quantiles of the NMSE metric in Figure 9.3 to assist the result in Table 9.1(a). Note that we cut off the negative values as zero when demonstrating R^2 score. The two box plots in Figure 9.3 show the proposed Apps is consistently better than the baselines in terms of the full quantiles $(25\%, 50\%, 75\%)$ of the NMSE metric.

Benchmark with other Active Strategies. Two baseline methods using active learning strategy are: (1) query-by-committee (QbC) proposed in [277, 286]. (2) Core-Set [284] proposes to sample diverse data. These methods were originally proposed with different neural networks, thus we evaluate these different active learning methods using the same decoder in our Apps. Current active learning methods are not directly available for evaluation in our problem setting (in Equation 9.1), so we re-implement these query strategies with the new problem setting.

The running time of the data querying step measures the efficiency of every active learning algorithm for this task. The ranking-based distance indicates if the ranking of many candidate expressions is exactly the same as evaluated on full data. If the predicted ODEs are ranked in the same order as the full data, then the ranking-based distance (Kendall tau score) will be close to zero.

In Table 9.3, given a set of 20 predicted ODEs, we compare the TopK ranking (i.e., top 3) of predicted ODEs by each active learning strategy is the same as using full data. We find both our phase portrait and QbC rank those predicted ODEs in proper ranking order. Our Apps takes the least memory to locate the most informative region and is also time efficient because we only pick one region among all the available regions. The QbC takes much more time because it finds every initial condition as an optimization problem over the

	Ranking-based distance (↓)	Running Time (↓)	Peak Memory (↓)
Apps (ours)	**0.08**	5.2 sec	**3.76** MB
QbC	0.13	13.4 sec	51 MB
CoreSet	0.22	**4.3** sec	2.74 GB

Table 9.3. Ranking comparison with different active learning strategies. Apps shows a smaller ranking-based distance than other strategies, which is better for ranking those best-predicted expressions. Also Apps takes less memory consumption and less computational time because the sketching step itself is lightweight.

input variables, which is solved by a separate gradient-based optimizer. CoreSet first runs a clustering algorithm over the ground-truth data and then samples a diverse set of initial conditions from each cluster. So the memory usage of Coreset is mainly determined by the first clustering step.

9.6 Summary

In this chapter, we introduced Apps, a novel approach for discovering ODEs from trajectory data. By actively reasoning about the most informative regions within the phase portrait of candidate ODEs, Apps overcomes the limitations of passively learned methods that rely on pre-collected datasets. Our approach also reduces the need for extensive data collection while still yielding highly accurate and generalizable ODE models. The experimental results demonstrate that Apps consistently outperforms baseline methods, achieving the lowest median NMSE across various datasets under both noiseless and noisy conditions.

10. Future Work

10.1 Automatic Discovery of New Knowledge for Novel Materials

Designing new materials and understanding their behaviors is fundamental at the intersection of materials science and computer science[1]. My goal is to build an automatic theory discovery system to answer critical questions from material scientists, such as predicting the long-term stability of materials under extreme conditions. Building on my previous work in discovering equations from experimental data, I aim to develop a framework that: (1) automated material knowledge discovery, by automatically predicting systems of algebraic or differential equations to explain material defects and the dynamic evolution of material phases over time., and (2) Incorporates Scientific Hypotheses by collaborating with domain experts to integrate diverse formats of scientific knowledge, effectively reducing the search space and accelerating the discovery process.

10.2 Combining AR and ML to accelerate Automatic Theorem Proving

The AI4Math community has recently witnessed a great breakthrough: an integrated system, combining the large language model and symbolic engine, won the silver award in the IMO 2024. It opens a new field for artificial intelligence over mathematics, requiring neural networks to understand concepts over the vast math literature and use symbolic engines to critically evaluate the derivation steps. The developed tools can assist mathematicians in proving those new challenging theorems from months to hours. Recent works primarily based on large-language models are very sensitive to the symbols in the math problem description, where the generated output is not mathematically sound. My research aims to: 1) Design Powerful Symbolic Engines that ensure the mathematical soundness of generated outputs. 2) Enhance ML Models for Mathematical Understanding by enabling them to interpret mathematical concepts and communicate solutions effectively in natural language. 3) Assist Mathematicians by developing tools that can tackle challenging theorems more efficiently, reducing problem-solving times from months to hours.

[1] ↑Materials Genome Initiative (https://www.mgi.gov) by the federal government offers millions of funding opportunities for discovering new materials.

10.3 Providing Safety Guarantees on high-stake AI-driven system

While prior work demonstrated the effectiveness of integrating constraint reasoning into auto-regressive models, the evolving trend is towards more general and large-scale models like Large Language Models (LLMs), which offer greater flexibility across domains. However, these models often fall short in high-stakes applications requiring strict adherence to physical constraints. My research focuses on: 1) Integrating Constraint Reasoning into Large-Scale Models to ensure feasible and safe outputs in critical applications such as autonomous driving. 2) Developing Adaptive AI Systems that can actively acquire and incorporate emerging, unseen constraints, moving beyond the assumption of known constraints in current models. 3) Building an AI Ecosystem with Robustness Guarantees by creating models that can adapt to new domains and safety requirements, enhancing their reliability in practical, real-world scenarios.

REFERENCES

[1] N. Jiang, M. Zhang, W.-J. van Hoeve, and Y. Xue, “Constraint reasoning embedded structured prediction,” *J. Mach. Learn. Res.*, vol. 31, pp. 1–40, 2022.

[2] M. Zhang, N. Jiang, L. Li, and Y. Xue, “Constraint satisfaction driven natural language generation: A tree search embedded MCMC approach,” in *EMNLP (Findings)*, Association for Computational Linguistics, 2020, pp. 1286–1298.

[3] F. Ding, N. Jiang, J. Ma, J. Peng, J. Xu, and Y. Xue, “PALM: probabilistic area loss minimization for protein sequence alignment,” in *UAI*, ser. Proceedings of Machine Learning Research, vol. 161, AUAI Press, 2021, pp. 1100–1109.

[4] A. Choi, Y. Xue, and A. Darwiche, “Same-decision probability: A confidence measure for threshold-based decisions,” *Int. J. Approx. Reason.*, vol. 53, no. 9, pp. 1415–1428, 2012.

[5] R. S. Michalski and J. R. Anderson, *Machine learning - an artificial intelligence approach* (Symbolic computation). Springer, 1984.

[6] C. M. Bishop, *Pattern recognition and machine learning, 5th Edition* (Information science and statistics). Springer, 2007.

[7] R. Socher *et al.*, “Recursive deep models for semantic compositionality over a sentiment treebank,” in *EMNLP*, ACL, 2013, pp. 1631–1642.

[8] R. Xiang and J. Neville, “Collective inference for network data with copula latent markov networks,” in *WSDM*, ACM, 2013, pp. 647–656.

[9] L. Tang, Y. Xue, D. Chen, and C. P. Gomes, “Multi-entity dependence learning with rich context via conditional variational auto-encoder,” in *AAAI*, AAAI Press, 2018, pp. 824–832.

[10] D. Chen, Y. Xue, and C. P. Gomes, “End-to-end learning for the deep multivariate probit model,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 931–940.

[11] S. B. Akers, “Binary decision diagrams,” *IEEE Trans. Computers*, vol. 27, no. 6, pp. 509–516, 1978.

[12] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, 1986.

[13] R. Coletta, C. Bessiere, B. O’Sullivan, E. C. Freuder, S. O’Connell, and J. Quinqueton, “Constraint acquisition as semi-automatic modeling,” in *SGAI Conf*, Springer, 2003, pp. 111–124.

[14] A. Lallouet, M. Lopez, L. Martin, and C. Vrain, "On learning constraint problems," in *ICTAI*, IEEE Computer Society, 2010, pp. 45–52.

[15] N. Beldiceanu and H. Simonis, "A model seeker: Extracting global constraint models from positive examples," in *CP*, ser. Lecture Notes in Computer Science, vol. 7514, Springer, 2012, pp. 141–157.

[16] C. Bessiere, F. Koriche, N. Lazaar, and B. O'Sullivan, "Constraint acquisition," *Artificial Intelligence*, vol. 244, pp. 315–342, 2017.

[17] H. A. Addi, C. Bessiere, R. Ezzahir, and N. Lazaar, "Time-Bounded Query Generator for Constraint Acquisition," in *Proceedings of CPAIOR*, ser. LNCS, vol. 10848, Springer, 2018, pp. 1–17.

[18] V. Punyakanok, D. Roth, W. Yih, and D. Zimak, "Semantic role labeling via integer linear programming inference," in *COLING*, Association for Computational Linguistics, 2004, pp. 1346–1353.

[19] D. Roth and W. Yih, "Integer linear programming inference for conditional random fields," in *ICML*, ser. ACM International Conference Proceeding Series, vol. 119, ACM, 2005, pp. 736–743.

[20] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 136–145.

[21] A. M. Ferber, B. Wilder, B. Dilkina, and M. Tambe, "Mipaal: Mixed integer program as a layer," in *AAAI*, AAAI Press, 2020, pp. 1504–1511.

[22] D. Deutsch, S. Upadhyay, and D. Roth, "A general-purpose algorithm for constrained sequential inference," in *CoNLL*, Association for Computational Linguistics, 2019, pp. 482–492.

[23] B. Peters, V. Niculae, and A. F. T. Martins, "Sparse sequence-to-sequence models," in *ACL*, Association for Computational Linguistics, 2019, pp. 1504–1519.

[24] H. Wu, Z. Chen, W. Sun, B. Zheng, and W. Wang, "Modeling trajectories with recurrent neural networks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017, pp. 3083–3090.

[25] Z. Yang, J.-L. Wu, and H. Xiao, "Enforcing deterministic constraints on generative adversarial networks for emulating physical systems," *arXiv preprint arXiv:1911.06671*, 2019.

[26] E. Heim, "Constrained generative adversarial networks for interactive image generation," in *CVPR*, Computer Vision Foundation / IEEE, 2019, pp. 10 753–10 761.

[27] A. Lallouet and A. Legtchenko, "Building consistencies for partially defined constraints with decision trees and neural networks," *Int. J. Artif. Intell. Tools*, vol. 16, no. 4, pp. 683–706, 2007.

[28] M. Lombardi, M. Milano, and A. Bartolini, "Empirical decision model learning," *Artif. Intell.*, vol. 244, pp. 343–367, 2017.

[29] M. Lombardi and S. Gualandi, "A lagrangian propagator for artificial neural networks in constraint programming," *Constraints*, vol. 21, no. 4, pp. 435–462, 2016.

[30] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 1945–1954.

[31] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for structured data," in *ICLR (Poster)*, OpenReview.net, 2018.

[32] W. Jin, R. Barzilay, and T. S. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 2328–2337.

[33] A. Galassi, M. Lombardi, P. Mello, and M. Milano, "Model Agnostic Solution of CSPs via Deep Learning: A Preliminary Study," in *Proceedings of CPAIOR*, ser. LNCS, vol. 10848, Springer, 2018, pp. 254–262.

[34] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NIPS*, 2015, pp. 2692–2700.

[35] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, 2017.

[36] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT solver from single-bit supervision," in *ICLR (Poster)*, OpenReview.net, 2019.

[37] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[38] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *NIPS*, 2017, pp. 6348–6358.

[39] K. Guu, P. Pasupat, E. Z. Liu, and P. Liang, "From language to programs: Bridging reinforcement learning and maximum marginal likelihood," in *ACL*, Association for Computational Linguistics, 2017, pp. 1051–1062.

[40] K. Shi, J. Steinhardt, and P. Liang, “Frangel: Component-based synthesis with control structures,” *Proc. ACM Program. Lang.*, vol. 3, no. POPL, 73:1–73:29, 2019.

[41] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L. Rousseau, “Learning heuristics for the TSP by policy gradient,” in *CPAIOR*, ser. Lecture Notes in Computer Science, vol. 10848, Springer, 2018, pp. 170–181.

[42] C. Liu, X. Chen, E. C. R. Shin, M. Chen, and D. X. Song, “Latent attention for if-then program synthesis,” in *NIPS*, 2016, pp. 4574–4582.

[43] W. Hwang, J. Yim, S. Park, and M. Seo, “A comprehensive exploration on wikisql with table-aware word contextualization,” *NeurIPS Workshop on Knowledge Representation & Reasoning Meets Machine Learning*, 2019.

[44] J. Read, L. Martino, P. M. Olmos, and D. Luengo, “Scalable multi-output label prediction: From classifier chains to classifier trellises,” *Pattern Recognit.*, vol. 48, pp. 2096–2109, 2015.

[45] K. Dembczynski, W. Cheng, and E. Hüllermeier, “Bayes optimal multilabel classification via probabilistic classifier chains,” in *ICML*, Omnipress, 2010, pp. 279–286.

[46] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” in *ICML*, Morgan Kaufmann, 2001, pp. 282–289.

[47] S. Geman and D. Geman, “Stochastic relaxation, gibbs distributions, and the bayesian restoration of images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 6, pp. 721–741, 1984.

[48] D. Belanger and A. McCallum, “Structured prediction energy networks,” in *ICML*, vol. 48, JMLR.org, 2016, pp. 983–992.

[49] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, “Large margin methods for structured and interdependent output variables,” *J. Mach. Learn. Res.*, vol. 6, pp. 1453–1484, 2005.

[50] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *CoRR*, vol. abs/1411.1784, 2014.

[51] I. J. Goodfellow *et al.*, “Generative adversarial nets,” in *NIPS*, 2014, pp. 2672–2680.

[52] D. J. Rezende and S. Mohamed, “Variational inference with normalizing flows,” in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, 2015, pp. 1530–1538.

[53] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *NIPS*, 2014, pp. 3104–3112.

[54] X. Pan and V. Srikumar, “Learning to speed up structured output prediction,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 3993–4002.

[55] K. Bello, A. Ghoshal, and J. Honorio, “Minimax bounds for structured prediction based on factor graphs,” in *AISTATS*, ser. Proceedings of Machine Learning Research, vol. 108, PMLR, 2020, pp. 213–222.

[56] V. Niculae, A. F. T. Martins, M. Blondel, and C. Cardie, “Sparsemap: Differentiable sparse structured inference,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 3796–3805.

[57] I. Wegener, *Branching Programs and Binary Decision Diagrams*. Society for Industrial and Applied Mathematics, 2000. DOI: 10.1137/1.9780898719789.

[58] D. Bergman, A. A. Ciré, W. van Hoeve, and J. N. Hooker, “Discrete optimization with decision diagrams,” *INFORMS J. Comput.*, vol. 28, no. 1, pp. 47–66, 2016.

[59] W. van Hoeve, “Graph coloring with decision diagrams,” *Math. Program.*, vol. 192, no. 1, pp. 631–674, 2022.

[60] A. A. Ciré and W. J. van Hoeve, “Multivalued decision diagrams for sequencing problems,” *Oper. Res.*, vol. 61, no. 6, pp. 1411–1428, 2013.

[61] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann, “A Constraint Store Based on Multivalued Decision Diagrams,” in *Proceedings of CP*, ser. LNCS, vol. 4741, Springer, 2007, pp. 118–132.

[62] S. J. Friedman and K. J. Supowit, “Finding the optimal variable ordering for binary decision diagrams,” *IEEE Trans. Computers*, vol. 39, no. 5, pp. 710–713, 1990. DOI: 10.1109/12.53586.

[63] D. Bergman, A. A. Ciré, W. van Hoeve, and J. N. Hooker, *Decision Diagrams for Optimization* (Artificial Intelligence: Foundations, Theory, and Algorithms). Springer, 2016.

[64] V. Niculae and A. F. T. Martins, “Lp-sparsemap: Differentiable relaxed optimization for sparse structured prediction,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 7348–7359.

[65] J. J. M. Bront, I. Mendez-Diaz, and P. Zabala, “An integer programming approach for the time-dependent TSP,” *Electron. Notes Discret. Math.*, vol. 36, pp. 351–358, 2010.

[66] U. Junior Mele, L. Maria Gambardella, and R. Montemanni, “Machine learning approaches for the traveling salesman problem: A survey,” in *2021 The 8th International Conference on Industrial Engineering and Applications (Europe)*, 2021, pp. 182–186.

[67] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, 2017.

[68] P. Shaw, J. Uszkoreit, and A. Vaswani, “Self-attention with relative position representations,” in *NAACL-HLT*, Association for Computational Linguistics, 2018, pp. 464–468.

[69] L. Dong and M. Lapata, “Coarse-to-fine decoding for neural semantic parsing,” in *ACL*, Association for Computational Linguistics, 2018, pp. 731–742.

[70] V. Zhong, C. Xiong, and R. Socher, “Seq2sql: Generating structured queries from natural language using reinforcement learning,” *CoRR*, vol. abs/1709.00103, 2017.

[71] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” in *NAACL-HLT*, Association for Computational Linguistics, 2019, pp. 4171–4186.

[72] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.

[73] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, “Large margin methods for structured and interdependent output variables,” *J. Mach. Learn. Res.*, vol. 6, pp. 1453–1484, Dec. 2005.

[74] I. J. Goodfellow *et al.*, “Generative adversarial nets,” in *NIPS*, 2014, pp. 2672–2680.

[75] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *ICLR*, 2014.

[76] M. Germain, K. Gregor, I. Murray, and H. Larochelle, “MADE: masked autoencoder for distribution estimation,” in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, 2015, pp. 881–889.

[77] H. Larochelle and I. Murray, “The neural autoregressive distribution estimator,” in *AISTATS*, ser. JMLR Proceedings, vol. 15, JMLR.org, 2011, pp. 29–37.

[78] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48, JMLR.org, 2016, pp. 1747–1756.

[79] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 214–223.

[80] Y. Song and S. Ermon, “Generative modeling by estimating gradients of the data distribution,” in *NeurIPS*, 2019, pp. 11 895–11 907.

[81] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-based generative modeling through stochastic differential equations,” in *ICLR*, OpenReview.net, 2021.

[82] K. P. Murphy, Y. Weiss, and M. I. Jordan, “Loopy belief propagation for approximate inference: An empirical study,” in *UAI*, Morgan Kaufmann, 1999, pp. 467–475.

[83] J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Generalized belief propagation,” in *NIPS*, MIT Press, 2000, pp. 689–695.

[84] M. J. Wainwright and M. I. Jordan, “Log-determinant relaxation for approximate inference in discrete markov random fields,” *IEEE Trans. Signal Process.*, vol. 54, no. 6-1, pp. 2099–2109, 2006.

[85] A. Braunstein, M. Mézard, and R. Zecchina, “Survey propagation: An algorithm for satisfiability,” *Random Struct. Algorithms*, vol. 27, pp. 201–226, 2005.

[86] M. Chavira, A. Darwiche, and M. Jaeger, “Compiling relational bayesian networks for exact inference,” *Int. J. Approx. Reason.*, vol. 42, no. 1-2, pp. 4–20, 2006.

[87] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming.* Cambridge, MA, USA: MIT Press, 1989, ISBN: 0-262-08181-4.

[88] V. Gogate and R. Dechter, “Importance sampling-based estimation over and/or search spaces for graphical models,” *Artificial Intelligence*, vol. 184-185, pp. 38–77, 2012.

[89] T. Sang, P. Bearne, and H. Kautz, “Performing bayesian inference by weighted model counting,” in *AAAI*, ser. AAAI’05, 2005, pp. 475–481.

[90] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, “Toward controlled generation of text,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 1587–1596.

[91] D. Lowd and P. M. Domingos, “Learning arithmetic circuits,” in *UAI*, AUAI Press, 2008, pp. 383–392.

[92] F. Ding, J. Ma, J. Xu, and Y. Xue, “XOR-CD: linearly convergent constrained structure generation,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 2728–2738.

[93] P. Erdős and L. Lovász, “Problems and results on 3-chromatic hypergraphs and some related questions,” in *Colloquia Mathematica Societatis Janos Bolyai 10. Infinite and Finite Sets, Keszthely (Hungary)*, Citeseer, 1973.

[94] R. A. Moser and G. Tardos, “A constructive proof of the general lovász local lemma,” *J. ACM*, vol. 57, no. 2, 11:1–11:15, 2010.

[95] H. Guo, M. Jerrum, and J. Liu, “Uniform sampling through the lovász local lemma,” *J. ACM*, vol. 66, no. 3, 18:1–18:31, 2019.

[96] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, “Learning a SAT solver from single-bit supervision,” in *ICLR (Poster)*, OpenReview.net, 2019.

[97] H. Duan, P. Vaezipoor, M. B. Paulus, Y. Ruan, and C. J. Maddison, “Augment with care: Contrastive learning for combinatorial problems,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 5627–5642.

[98] Z. Li, Q. Chen, and V. Koltun, “Combinatorial optimization with graph convolutional networks and guided tree search,” in *NeurIPS*, 2018, pp. 537–546.

[99] E. Yolcu and B. Póczos, “Learning local search heuristics for boolean satisfiability,” in *NeurIPS*, 2019, pp. 7990–8001.

[100] X. Chen and Y. Tian, “Learning to perform local rewriting for combinatorial optimization,” in *NeurIPS*, 2019, pp. 6278–6289.

[101] N. Karalias and A. Loukas, “Erdos goes neural: An unsupervised learning framework for combinatorial optimization on graphs,” in *NeurIPS*, 2020, pp. 6659–6672.

[102] Y. Bengio, A. Lodi, and A. Prouvost, “Machine learning for combinatorial optimization: A methodological tour d’horizon,” *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.

[103] J. Mandi, E. Demirovic, P. J. Stuckey, and T. Guns, “Smart predict-and-optimize for hard combinatorial optimization problems,” in *AAAI*, AAAI Press, 2020, pp. 1603–1610.

[104] R. M. Neal, *Probabilistic inference using Markov chain Monte Carlo methods.* Department of Computer Science, University of Toronto Toronto, ON, Canada, 1993.

[105] P. Dagum and R. M. Chavez, “Approximating probabilistic inference in bayesian belief networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 3, pp. 246–255, 1993.

[106] H. Ge, K. Xu, and Z. Ghahramani, “Turing: A language for flexible probabilistic inference,” in *AISTATS*, vol. 84, PMLR, Apr. 2018, pp. 1682–1690.

[107] A. T. Ihler, J. W. Fisher III, and A. S. Willsky, "Loopy belief propagation: Convergence and effects of message errors," *J. Mach. Learn. Res.*, vol. 6, pp. 905–936, 2005.

[108] A. Coja-Oghlan, N. Müller, and J. B. Ravelomanana, *Belief propagation on the random k-sat model*, 2020. arXiv: 2011.02303.

[109] F. Ding and Y. Xue, "Contrastive divergence learning with chained belief propagation," in *PGM*, ser. Proceedings of Machine Learning Research, vol. 138, PMLR, 2020, pp. 161–172.

[110] V. Gogate and R. Dechter, "Samplesearch: Importance sampling in presence of determinism," *Artif. Intell.*, vol. 175, no. 2, pp. 694–729, 2011.

[111] C. P. Gomes, A. Sabharwal, and B. Selman, "Near-uniform sampling of combinatorial spaces using XOR constraints," in *NIPS*, MIT Press, 2006, pp. 481–488.

[112] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, "Taming the curse of dimensionality: Discrete integration by hashing and optimization," in *ICML (2)*, ser. JMLR Workshop and Conference Proceedings, vol. 28, JMLR.org, 2013, pp. 334–342.

[113] D. Achlioptas and P. Theodoropoulos, "Probabilistic model counting with short xors," in *SAT*, ser. Lecture Notes in Computer Science, vol. 10491, Springer, 2017, pp. 3–19.

[114] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable approximate model counter," *CoRR*, vol. abs/1306.5726, 2013.

[115] Y. Mansour, "Learning boolean functions via the fourier transform," in *Theoretical advances in neural computation and learning*, Springer, 1994, pp. 391–424.

[116] C. Dodaro and A. Previti, "Minipref: A tool for preferences in SAT (short paper)," in *RCRA + RiCeRcA*, ser. CEUR Workshop Proceedings, vol. 2538, CEUR-WS.org, 2019.

[117] M. Lauria, J. Elffers, J. Nordström, and M. Vinyals, "Cnfgen: A generator of crafted benchmarks," in *SAT*, vol. 10491, Springer, 2017, pp. 464–473.

[118] C. K. Carter and R. Kohn, "On gibbs sampling for state space models," *Biometrika*, vol. 81, no. 3, pp. 541–553, 1994.

[119] R. Gupta, S. Sharma, S. Roy, and K. S. Meel, "WAPS: weighted and projected sampling," in *TACAS*, vol. 11427, 2019, pp. 59–76.

[120] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, "Distribution-aware sampling and weighted model counting for SAT," in *AAAI*, AAAI Press, 2014, pp. 1722–1730.

[121] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, “Embed and project: Discrete sampling with universal hashing,” in *NIPS*, 2013, pp. 2085–2093.

[122] F. Ding and Y. Xue, “XOR-SGD: provable convex stochastic optimization for decision-making under uncertainty,” in *UAI*, ser. Proceedings of Machine Learning Research, vol. 161, AUAI Press, 2021, pp. 151–160.

[123] P. Golia, M. Soos, S. Chakraborty, and K. S. Meel, “Designing samplers is easy: The boon of testers,” in *FMCAD*, IEEE, 2021, pp. 222–230.

[124] R. Dutra, K. Laeufer, J. Bachrach, and K. Sen, “Efficient sampling of SAT solutions for testing,” in *ICSE*, ACM, 2018, pp. 549–559.

[125] M. Soos, S. Gocht, and K. S. Meel, “Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling,” in *CAV (1)*, ser. Lecture Notes in Computer Science, vol. 12224, Springer, 2020, pp. 463–484.

[126] S. Sharma, R. Gupta, S. Roy, and K. S. Meel, “Knowledge compilation meets uniform sampling,” in *LPAR*, ser. EPiC Series in Computing, vol. 57, 2018, pp. 620–636.

[127] M. Mahmoud, “Gpu enabled automated reasoning,” Ph.D. dissertation, Mathematics and Computer Science, Mar. 2022.

[128] J. K. Fichte, M. Hecher, and M. Zisser, “An improved gpu-based SAT model counter,” in *CP*, 2019, pp. 491–509.

[129] S. Chakraborty and K. S. Meel, “On testing of uniform samplers,” in *AAAI*, 2019, pp. 7777–7784.

[130] H. Cohn, R. Pemantle, and J. G. Propp, “Generating a random sink-free orientation in quadratic time,” *Electron. J. Comb.*, vol. 9, no. 1, 2002.

[131] J. Takahashi, T. Yamaguchi, K. Sekiyama, and T. Fukuda, “Communication timing control and topology reconfiguration of a sink-free meshed sensor network with mobile robots,” *IEEE/ASME transactions on mechatronics*, vol. 14, no. 2, pp. 187–197, 2009.

[132] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” in *NAACL-HLT*, Association for Computational Linguistics, 2019, pp. 4171–4186.

[133] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[134] V. Gogate and R. Dechter, “Approximate counting by sampling the backtrack-free search space,” in *AAAI*, 2007, pp. 198–203.

[135] V. Gogate and R. Dechter, “Samplesearch: A scheme that searches for consistent samples,” in *AISTATS*, 2007, pp. 147–154.

[136] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *NIPS*, 2014, pp. 3104–3112.

[137] M. Berglund, T. Raiko, M. Honkala, L. Kärkkäinen, A. Vetek, and J. Karhunen, “Bidirectional recurrent neural networks as generative models,” in *Advances in Neural Information Processing Systems*, 2015, pp. 856–864.

[138] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, “Toward controlled generation of text,” in *ICML*, 2017, pp. 1587–1596.

[139] H. Zhang, H. Zhou, N. Miao, and L. Li, “Generating fluent adversarial examples for natural languages,” in *ACL (1)*, Association for Computational Linguistics, 2019, pp. 5564–5569.

[140] N. Miao, Y. Song, H. Zhou, and L. Li, “Do you have the right scissors? tailoring pre-trained language models via monte-carlo methods,” in *ACL*, Association for Computational Linguistics, 2020, pp. 3436–3441.

[141] J. Y. Lee, S. V. Mehta, M. Wick, J. Tristan, and J. G. Carbonell, “Gradient-based inference for networks with output constraints,” in *AAAI*, 2019, pp. 4147–4154.

[142] J. Su, J. Xu, X. Qiu, and X. Huang, “Incorporating discriminator in sentence generation: A gibbs sampling method,” in *AAAI*, 2018, pp. 5496–5503.

[143] M. Richardson and P. M. Domingos, “Markov logic networks,” *Mach. Learn.*, vol. 62, no. 1-2, pp. 107–136, 2006.

[144] T. Khot, N. Balasubramanian, E. Gribkoff, A. Sabharwal, P. Clark, and O. Etzioni, “Exploring markov logic networks for question answering,” in *EMNLP*, 2015, pp. 685–694.

[145] S. Prabhumoye, Y. Tsvetkov, R. Salakhutdinov, and A. W. Black, “Style transfer through back-translation,” in *ACL (1)*, Association for Computational Linguistics, 2018, pp. 866–876.

[146] M. S. Amato and M. C. MacDonald, “Sentence processing in an artificial language: Learning and using combinatorial constraints,” *Cognition*, vol. 116, no. 1, pp. 143–148, 2010.

[147] M. Sundermeyer, R. Schlüter, and H. Ney, “LSTM neural networks for language modeling,” in *INTERSPEECH*, ISCA, 2012, pp. 194–197.

[148] N. Miao, H. Zhou, L. Mou, R. Yan, and L. Li, “CGMH: constrained sentence generation by metropolis-hastings sampling,” in *AAAI*, 2019, pp. 6834–6842.

[149] P. S. H. Lewis, L. Denoyer, and S. Riedel, “Unsupervised question answering by cloze translation,” in *ACL (1)*, Association for Computational Linguistics, 2019, pp. 4896–4910.

[150] P. Rajpurkar, R. Jia, and P. Liang, “Know what you don’t know: Unanswerable questions for squad,” in *ACL*, 2018, pp. 784–789.

[151] Y. Fu, H. Zhou, J. Chen, and L. Li, “Rethinking text attribute transfer: A lexical analysis,” in *INLG*, Association for Computational Linguistics, 2019, pp. 24–33.

[152] C. B. Do, S. S. Gross, and S. Batzoglou, “Contralign: Discriminative training for protein sequence alignment,” in *Research in Computational Molecular Biology*, vol. 3909, Springer, 2006, pp. 160–174.

[153] D. S. Marks *et al.*, “Protein 3d structure computed from evolutionary sequence variation,” *PLOS ONE*, vol. 6, no. 12, pp. 1–20, Dec. 2011.

[154] J. Söding, A. Biegert, and A. N. Lupas, “The hhpred interactive server for protein homology detection and structure prediction,” *Nucleic acids research*, vol. 33, no. suppl_2, W244–W248, 2005.

[155] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[156] S. Kumar, K. Tamura, and M. Nei, “Mega3: Integrated software for molecular evolutionary genetics analysis and sequence alignment,” *Briefings in bioinformatics*, vol. 5, no. 2, pp. 150–163, 2004.

[157] K. Hou, H. Wang, and W.-c. Feng, “Aalign: A simd framework for pairwise sequence alignment on x86-based multi-and many-core processors,” in *2016 IEEE International Parallel and Distributed Processing Symposium*, IEEE, 2016, pp. 780–789.

[158] F. Armougom *et al.*, “Expresso: Automatic incorporation of structural information in multiple sequence alignments using 3d-coffee,” *Nucleic Acids Res.*, vol. 34, no. Web-Server-Issue, pp. 604–608, 2006.

[159] K. Katoh and H. Toh, “Recent developments in the MAFFT multiple sequence alignment program,” *Briefings in bioinformatics*, vol. 9, no. 4, pp. 286–298, 2008.

[160] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[161] M. Tompa, “Lecture notes on biological sequence analysis,” *University of Washington, Seattle, Technical report*, 2000.

[162] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[163] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970, ISSN: 0022-2836.

[164] T. Smith and M. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981, ISSN: 0022-2836.

[165] J. Ma, S. Wang, Z. Wang, and J. Xu, “Mrfalign: Protein homology detection through alignment of markov random fields,” *PLoS computational biology*, vol. 10, no. 3, 2014.

[166] C.-S. Jeong and D. Kim, “Structure-based markov random field model for representing evolutionary constraints on functional sites,” *BMC bioinformatics*, vol. 17, no. 1, pp. 1–11, 2016.

[167] N. M. Daniels, A. Gallant, N. Ramsey, and L. J. Cowen, “Mrfy: Remote homology detection for beta-structural proteins using markov random fields and stochastic search,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, no. 1, pp. 4–16, 2015.

[168] S. Balakrishnan, H. Kamisetty, J. G. Carbonell, S.-I. Lee, and C. J. Langmead, “Learning generative models for protein fold families,” *Proteins: Structure, Function, and Bioinformatics*, vol. 79, no. 4, pp. 1061–1078, 2011.

[169] J. Söding, “Protein homology detection by hmm–hmm comparison,” *Bioinformatics*, vol. 21, no. 7, pp. 951–960, 2005.

[170] R. Lajugie, D. Garreau, F. R. Bach, and S. Arlot, “Metric learning for temporal sequence alignment,” in *Advances in Neural Information Processing Systems*, vol. 27, 2014, pp. 1817–1825.

[171] C. Gupta, E. Yilmaz, and H. Li, “Acoustic modeling for automatic lyrics-to-audio alignment,” in *20th Annual Conference of the International Speech Communication Association*, ISCA, 2019, pp. 2040–2044.

[172] F. Wu and J. Xu, “Deep template-based protein structure prediction,” *PLOS Computational Biology*, vol. 17, no. 5, pp. 1–18, 2021.

[173] S. F. Altschul and D. J. Lipman, “Protein database searches for multiple alignments.,” *Proceedings of the National Academy of Sciences*, vol. 87, no. 14, pp. 5509–5513, 1990.

[174] S. Wang, J. Ma, J. Peng, and J. Xu, "Protein structure alignment beyond spatial proximity," *Scientific reports*, vol. 3, p. 1448, 2013.

[175] J. Ma, J. Peng, S. Wang, and J. Xu, "A conditional neural fields model for protein threading," *Bioinformatics*, vol. 28, no. 12, pp. 59–66, 2012.

[176] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *NIPS 2017 Workshop Autodiff*, 2017.

[177] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[178] M. Virgolin, T. Alderliesten, and P. A. N. Bosman, "Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression," in *GECCO*, ACM, 2019, pp. 1084–1092.

[179] R. Guimerà *et al.*, "A bayesian machine scientist to aid in the solution of challenging scientific problems," *Science advances*, vol. 6, no. 5, eaav6971, 2020.

[180] B. K. Petersen, M. Landajuela, T. N. Mundhenk, C. P. Santiago, S. Kim, and J. T. Kim, "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," in *ICLR*, OpenReview.net, 2021.

[181] T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen, "Symbolic regression via deep reinforcement learning enhanced genetic programming seeding," in *NeurIPS*, 2021, pp. 24 912–24 923.

[182] L. Scavuzzo *et al.*, "Learning to branch with tree mdps," in *NeurIPS*, 2022.

[183] S. Razavi and E. R. Gamazon, "Neural-network-directed genetic programmer for discovery of governing equations," *CoRR*, vol. abs/2203.08808, 2022.

[184] B. He, Q. Lu, Q. Yang, J. Luo, and Z. Wang, "Taylor genetic programming for symbolic regression," in *GECCO*, ACM, 2022, pp. 946–954.

[185] J. S. Lehman, T. J. Santner, and W. I. Notz, "Designing computer experiments to determine robust control variables," *Statistica Sinica*, pp. 571–590, 2004.

[186] T. J. Santner, B. J. Williams, and W. I. Notz, *The Design and Analysis of Computer Experiments* (Springer series in statistics). Springer, 2003.

[187] P. Langley, "Machine learning as an experimental science," *Mach. Learn.*, vol. 3, pp. 5–8, 1988.

[188] D. F. Kibler and P. Langley, "The experimental study of machine learning," 1991.

[189] P. Langley, “BACON: A production system that discovers empirical laws,” in *IJCAI*, William Kaufmann, 1977, p. 344.

[190] P. Langley, “Rediscovering physics with BACON.3,” in *IJCAI*, William Kaufmann, 1979, pp. 505–507.

[191] P. Langley, G. L. Bradshaw, and H. A. Simon, “BACON.5: the discovery of conservation laws,” in *IJCAI*, William Kaufmann, 1981, pp. 121–126.

[192] R. D. King *et al.*, “Functional genomic hypothesis generation and experimentation by a robot scientist,” *Nature*, vol. 427, no. 6971, pp. 247–252, 2004.

[193] R. D. King *et al.*, “The automation of science,” *Science*, vol. 324, no. 5923, pp. 85–89, 2009.

[194] M. Cerrato, J. Brugger, N. Schmitt, and S. Kramer, “Reinforcement learning for automated scientific discovery,” in *AAAI Spring Symposium on Computational Approaches to Scientific Discovery*, 2023.

[195] M. Virgolin and S. P. Pissis, “Symbolic regression is NP-hard,” *Transactions on Machine Learning Research*, 2022.

[196] Y. Matsubara, N. Chiba, R. Igarashi, T. Taniai, and Y. Ushiku, “Rethinking symbolic regression datasets and benchmarks for scientific discovery,” *arXiv preprint arXiv:2206.10540*, 2022.

[197] T. P. Ryan and J. P. Morgan, “Modern experimental design,” *Journal of Statistical Theory and Practice*, vol. 1, no. 3-4, pp. 501–506, 2007.

[198] Q. Chen, B. Xue, and M. Zhang, “Rademacher complexity for enhancing the generalization of genetic programming for symbolic regression,” *IEEE Trans. Cybern.*, vol. 52, no. 4, pp. 2382–2395, 2022.

[199] P. Langley, “Data-driven discovery of physical laws,” *Cognitive Science*, vol. 5, no. 1, pp. 31–54, 1981.

[200] D. B. Lenat, “The ubiquity of discovery,” *Artificial Intelligence*, vol. 9, no. 3, pp. 257–285, 1977, ISSN: 0004-3702.

[201] S.-M. Udrescu and M. Tegmark, “Ai feynman: A physics-inspired method for symbolic regression,” *Science Advances*, vol. 6, no. 16, 2020.

[202] D. Chen, Y. Wang, and W. Gao, “Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning,” *Appl. Intell.*, vol. 50, no. 10, pp. 3301–3317, 2020.

[203] T. McConaghy, "Ffx: Fast, scalable, deterministic symbolic regression technology," in *Genetic Programming Theory and Practice IX*, Springer, 2011, pp. 235–260.

[204] C. Chen, C. Luo, and Z. Jiang, "Elite bases regression: A real-time algorithm for symbolic regression," in *ICNC-FSKD*, IEEE, 2017, pp. 529–535.

[205] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. G. López, "Semantically-based crossover in genetic programming: Application to real-valued symbolic regression," *Genet. Program. Evolvable Mach.*, vol. 12, no. 2, pp. 91–119, 2011.

[206] M. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 353–362.

[207] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo, "Neural symbolic regression that scales," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 936–945.

[208] P. Kamienny, S. d'Ascoli, G. Lample, and F. Charton, "End-to-end symbolic regression with transformers," in *NeurIPS*, 2022.

[209] P. W. Langley, H. A. Simon, G. Bradshaw, and J. M. Zytkow, *Scientific Discovery: Computational Explorations of the Creative Process*. The MIT Press, Feb. 1987, ISBN: 9780262316002.

[210] H. Wang *et al.*, "Enabling scientific discovery with artificial intelligence," *Nature*, 2022.

[211] E. Bradley, M. Easley, and R. Stolle, "Reasoning about nonlinear system identification," *Artificial Intelligence*, vol. 133, no. 1, pp. 139–188, 2001.

[212] W. Bridewell, P. Langley, L. Todorovski, and S. Džeroski, "Inductive process modeling," *Machine Learning*, vol. 71, pp. 1–32, 2008.

[213] S. Dzeroski and L. Todorovski, "Discovering dynamics: From inductive logic programming to machine discovery," *J. Intell. Inf. Syst.*, vol. 4, no. 1, pp. 89–108, 1995.

[214] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[215] T. Wu and M. Tegmark, "Toward an artificial intelligence physicist for unsupervised learning," *Phys. Rev. E*, vol. 100, p. 033 311, 3 Sep. 2019.

[216] S. Zhang and G. Lin, "Robust data-driven discovery of governing physical laws with error bars," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 474, no. 2217, p. 20 180 305, 2018.

[217] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner, “Discovering physical concepts with neural networks,” *Physical review letters*, vol. 124, no. 1, p. 010 508, 2020.

[218] M. D. Cranmer *et al.*, “Discovering symbolic models from deep learning with inductive biases,” in *NeurIPS*, 2020.

[219] M. Raissi, A. Yazdani, and G. E. Karniadakis, “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations,” *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[220] M. Raissi, P. Perdikaris, and G. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[221] Z. Liu and M. Tegmark, “Machine learning conservation laws from trajectories,” *Phys. Rev. Lett.*, vol. 126, p. 180 604, 18 May 2021.

[222] Y. Xue, M. Nasim, M. Zhang, C. Fan, X. Zhang, and A. El-Azab, “Physics knowledge discovery via neural differential equation embedding,” in *ECML/PKDD (5)*, ser. Lecture Notes in Computer Science, vol. 12979, Springer, 2021, pp. 118–134.

[223] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural ordinary differential equations,” *Advances in neural information processing systems*, vol. 31, 2018.

[224] R. Valdés-Pérez, “Human/computer interactive elucidation of reaction mechanisms: Application to catalyzed hydrogenolysis of ethane,” *Catalysis Letters*, vol. 28, pp. 79–87, 1994.

[225] S. Hanneke, “Theory of disagreement-based active learning,” *Found. Trends Mach. Learn.*, vol. 7, no. 2-3, pp. 131–309, 2014.

[226] D. Golovin, A. Krause, and D. Ray, “Near-optimal bayesian active learning with noisy observations,” *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[227] D. Kahneman, *Thinking, fast and slow.* Macmillan, 2011.

[228] T. Anthony, Z. Tian, and D. Barber, “Thinking fast and slow with deep learning and tree search,” in *NIPS*, 2017, pp. 5360–5370.

[229] G. Booch *et al.*, “Thinking fast and slow in AI,” in *AAAI*, AAAI Press, 2021, pp. 15 042–15 046.

[230] H. A. Simon, “Spurious correlation: A causal interpretation,” *Journal of the American statistical Association*, vol. 49, no. 267, pp. 467–479, 1954.

[231] P. Langley, “Scientific discovery, causal explanation, and process model induction,” *Mind & Society*, vol. 18, no. 1, pp. 43–56, 2019.

[232] C. Glymour, R. Scheines, and P. Spirtes, *Discovering causal structure: Artificial intelligence, philosophy of science, and statistical modeling.* Academic Press, 2014.

[233] A. Jaber, A. Ribeiro, J. Zhang, and E. Bareinboim, “Causal identification under markov equivalence: Calculus, algorithm, and completeness,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 3679–3690, 2022.

[234] J. Pearl, *Causality.* Cambridge university press, 2009.

[235] W. La Cava *et al.*, “Contemporary symbolic regression methods and their relative performance,” *arXiv preprint arXiv:2107.14351*, 2021.

[236] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, “DEAP: Evolutionary algorithms made easy,” *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jun. 2012.

[237] R. Dubcáková, “Eureqa: Software review,” *Genet. Program. Evolvable Mach.*, vol. 12, no. 2, pp. 173–178, 2011.

[238] D. A. Abolafia, M. Norouzi, and Q. V. Le, “Neural program synthesis with priority queue training,” *CoRR*, vol. abs/1801.03526, 2018.

[239] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Mach. Learn.*, vol. 8, pp. 229–256, 1992.

[240] F. Sun, Y. Liu, J. Wang, and H. Sun, “Symbolic physics learner: Discovering governing equations via monte carlo tree search,” in *ICLR*, 2023.

[241] T. Tohme, D. Liu, and K. Youcef-Toumi, “GSR: A generalized symbolic regression approach,” *Trans. Mach. Learn. Res.*, vol. 2023, 2023.

[242] N. Jiang and Y. Xue, “Symbolic regression via control variable genetic programming,” in *ECML/PKDD*, Springer, 2023, pp. 178–195.

[243] R. Fletcher, *Practical methods of optimization.* John Wiley & Sons, 2000.

[244] D. J. Wales and J. P. Doye, “Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms,” *The Journal of Physical Chemistry A*, vol. 101, no. 28, pp. 5111–5116, 1997.

[245] Q. Cappart, D. Bergman, L. Rousseau, I. Prémont-Schwarz, and A. Parjadis, "Improving variable orderings of approximate decision diagrams using reinforcement learning," *INFORMS J. Comput.*, vol. 34, no. 5, pp. 2552–2570, 2022.

[246] L. Song, K. Xue, X. Huang, and C. Qian, "Monte carlo tree search based variable selection for high dimensional bayesian optimization," in *NeurIPS*, 2022.

[247] R. Dechter, *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms, Second Edition* (Synthesis Lectures on Artificial Intelligence and Machine Learning). Morgan & Claypool Publishers, 2019.

[248] V. Derkinderen, E. Heylen, P. Z. D. Martires, S. Kolb, and L. D. Raedt, "Ordering variables for weighted model integration," in *UAI*, ser. Proceedings of Machine Learning Research, vol. 124, AUAI Press, 2020, pp. 879–888.

[249] J. C. Ortiz-Bayliss, I. Amaya, S. E. Conant-Pablos, and H. Terashima-Marín, "Exploring the impact of early decisions in variable ordering for constraint satisfaction problems," *Comput. Intell. Neurosci.*, vol. 2018, 6103726:1–6103726:14, 2018.

[250] H. Li, G. Feng, and M. Yin, "On combining variable ordering heuristics for constraint satisfaction problems," *J. Heuristics*, vol. 26, no. 4, pp. 453–474, 2020.

[251] W. Song, Z. Cao, J. Zhang, C. Xu, and A. Lim, "Learning variable ordering heuristics for solving constraint satisfaction problems," *Eng. Appl. Artif. Intell.*, vol. 109, p. 104 603, 2022.

[252] H. Dette and I. Röder, "Optimal discrimination designs for multifactor experiments," *The Annals of Statistics*, vol. 25, no. 3, pp. 1161–1175, 1997. DOI: 10.1214/aos/1069362742. [Online]. Available: https://doi.org/10.1214/aos/1069362742.

[253] M. Yang and J. Stufken, "Identifying locally optimal designs for nonlinear models: A simple extension with profound consequences," 2012.

[254] Y. Matsubara, N. Chiba, R. Igarashi, and Y. Ushiku, "SRSD: Rethinking datasets of symbolic regression for scientific discovery," in *NeurIPS 2022 AI for Science: Progress and Promises*, 2022. [Online]. Available: https://openreview.net/forum?id=oKwyEqClqkb.

[255] D. Kulkarni and H. A. Simon, "The processes of scientific discovery: The strategy of experimentation," *Cogn. Sci.*, vol. 12, no. 2, pp. 139–175, 1988.

[256] H. Wang *et al.*, "Scientific discovery in the age of artificial intelligence," *Nature*, vol. 620, no. 7972, pp. 47–60, 2023.

[257] L. Todorovski and S. Dzeroski, "Declarative bias in equation discovery," in *ICML*, Morgan Kaufmann, 1997, pp. 376–384.

[258] P. Kamienny, G. Lample, S. Lamprier, and M. Virgolin, “Deep generative symbolic regression with monte-carlo-tree-search,” in *ICML*, vol. 202, PMLR, 2023.

[259] N. Jiang, M. Nasim, and Y. Xue, “Vertical symbolic regression,” *arXiv:2312.11955*, 2023.

[260] J. P. Joule, “On the production of heat by voltaic electricity,” in *Abstracts of the Papers Printed in the Philosophical Transactions of the Royal Society of London*, 1843, pp. 280–282.

[261] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, “Recurrent policy gradients,” *Log. J. IGPL*, vol. 18, no. 5, pp. 620–634, 2010.

[262] J. Kirkpatrick *et al.*, “Pushing the frontiers of density functionals by solving the fractional electron problem,” *Science*, vol. 374, no. 6573, pp. 1385–1389, 2021.

[263] J. Jumper *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[264] J. Brence, L. Todorovski, and S. Dzeroski, “Probabilistic grammars for equation discovery,” *Knowl. Based Syst.*, vol. 224, p. 107 077, 2021.

[265] B. Gec, N. Omejc, J. Brence, S. Dzeroski, and L. Todorovski, “Discovery of differential equations using probabilistic grammars,” in *DS*, vol. 13601, Springer, 2022, pp. 22–31.

[266] S. d’Ascoli, S. Becker, A. Mathis, P. Schwaller, and N. Kilbertus, “Odeformer: Symbolic regression of dynamical systems with transformers,” in *ICLR*, OpenReview.net, 2024.

[267] U. Fasel, J. N. Kutz, B. W. Brunton, and S. L. Brunton, “Ensemble-sindy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control,” *Proceedings of the Royal Society A*, vol. 478, no. 2260, p. 20 210 904, 2022.

[268] Z. Qian, K. Kacprzyk, and M. van der Schaar, “D-CODE: discovering closed-form odes from observed trajectories,” in *ICLR*, OpenReview.net, 2022.

[269] N. Jiang, M. Nasim, and Y. Xue, “Vertical symbolic regression via deep policy gradient,” in *IJCAI*, ijcai.org, 2024, pp. 5891–5899.

[270] J. Medina and A. D. White, “Active learning in symbolic regression performance with physical constraints,” *arXiv preprint arXiv:2305.10379*, 2023.

[271] S. H. Strogatz, *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering.* CRC press, 2018.

[272] E. Lorenz, "Deterministic nonperiodic flow in journal of the atmospheric science," 1963.

[273] P. Jin *et al.*, "Online symbolic regression with informative query," in *AAAI*, AAAI Press, 2023, pp. 5122–5130.

[274] N. Haut, W. Banzhaf, and B. Punch, "Active learning in genetic programming: Guiding efficient data collection for symbolic regression," *IEEE Transactions on Evolutionary Computation*, pp. 1–13, 2024. DOI: 10.1109/TEVC.2024.3471341.

[275] Q. Chen and B. Xue, "Generalisation in genetic programming for symbolic regression: Challenges and future directions," in *Women in Computational Intelligence: Key Advances and Perspectives on Emerging Topics*, Springer, 2022, pp. 281–302.

[276] L. S. Keren, A. Liberzon, and T. Lazebnik, "A computational framework for physics-informed symbolic regression with straightforward integration of domain knowledge," *Scientific Reports*, vol. 13, no. 1, p. 1249, 2023.

[277] N. Haut, B. Punch, and W. Banzhaf, "Active learning informs symbolic regression model development in genetic programming," in *GECCO Companion*, 2023, pp. 587–590.

[278] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, 2019.

[279] K. Egan, W. Li, and R. Carvalho, "Automatically discovering ordinary differential equations from data with sparse regression," *Communications Physics*, vol. 7, no. 1, p. 20, 2024.

[280] M. Heinonen, Ç. Yildiz, H. Mannerström, J. Intosalmi, and H. Lähdesmäki, "Learning unknown ODE models with gaussian processes," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1964–1973.

[281] Z. Li *et al.*, "Fourier neural operator for parametric partial differential equations," in *ICLR*, OpenReview.net, 2021.

[282] A. Wagenmaker and K. G. Jamieson, "Active learning for identification of linear dynamical systems," in *COLT*, ser. Proceedings of Machine Learning Research, vol. 125, PMLR, 2020, pp. 3487–3582.

[283] H. Mania, M. I. Jordan, and B. Recht, "Active learning for nonlinear system identification with guarantees," *J. Mach. Learn. Res.*, vol. 23, 32:1–32:30, 2022.

[284] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," in *ICLR*, OpenReview.net, 2018.

[285] J. T. Ash, C. Zhang, A. Krishnamurthy, J. Langford, and A. Agarwal, “Deep batch active learning by diverse, uncertain gradient lower bounds,” in *ICLR*, OpenReview.net, 2020.

[286] N. Haut, W. Banzhaf, and B. Punch, “Active learning improves performance on symbolic regression tasks in stackgp,” in *GECCO Companion*, 2022, pp. 550–553.

[287] C. Lüders, T. Sturm, and O. Radulescu, “Odebase: A repository of ode systems for systems biology,” *Bioinformatics Advances*, vol. 2, no. 1, vbac027, 2022.

[288] M. Jerrum, *Fundamentals of partial rejection sampling*, 2021. arXiv: 2106.07744.

[289] E. D. Rosa, E. Giunchiglia, and B. O’Sullivan, “Optimal stopping methods for finding high quality solutions to satisfiability problems with preferences,” in *SAC*, ACM, 2011, pp. 901–906.

[290] A. Ignatiev, A. Morgado, and J. Marques-Silva, “Pysat: A python toolkit for prototyping with SAT oracles,” in *SAT*, ser. Lecture Notes in Computer Science, vol. 10929, Springer, 2018, pp. 428–437.

[291] N. Prevot, M. Soos, and K. S. Meel, “Leveraging gpus for effective clause sharing in parallel SAT solving,” in *SAT*, 2021, pp. 471–487.

[292] T. Wolf *et al.*, “Huggingface’s transformers: State-of-the-art natural language processing,” *CoRR*, vol. abs/1910.03771, 2019. arXiv: 1910.03771. [Online]. Available: http://arxiv.org/abs/1910.03771.

[293] R. Wolfinger and M. O’connell, “Generalized linear mixed models a pseudo-likelihood approach,” *Journal of statistical Computation and Simulation*, vol. 48, no. 3-4, pp. 233–243, 1993.

[294] S. Rose, D. Engel, N. Cramer, and W. Cowley, “Automatic keyword extraction from individual documents,” *Text mining: applications and theory*, vol. 1, pp. 1–20, 2010.

[295] G. A. Miller, “Wordnet: A lexical database for english,” *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[296] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, “The stanford corenlp natural language processing toolkit,” in *ACL*, The Association for Computer Linguistics, 2014, pp. 55–60.

[297] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification,” in *EACL (2)*, Association for Computational Linguistics, 2017, pp. 427–431.

[298] X. Zhang, J. J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *NIPS*, 2015, pp. 649–657.

[299] N. J. Nagelkerke *et al.*, "A note on a general definition of the coefficient of determination," *Biometrika*, vol. 78, no. 3, pp. 691–692, 1991.

[300] W. G. L. Cava *et al.*, "Contemporary symbolic regression methods and their relative performance," in *NeurIPS Datasets and Benchmarks*, 2021.

[301] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[302] F. Gao and L. Han, "Implementing the nelder-mead simplex algorithm with adaptive parameters," *Comput. Optim. Appl.*, vol. 51, no. 1, pp. 259–277, 2012.

[303] S. C. Endres, C. Sandrock, and W. W. Focke, "A simplicial homology algorithm for lipschitz optimisation," *J. Glob. Optim.*, vol. 72, no. 2, pp. 181–217, 2018.

[304] C. Tsallis and D. A. Stariolo, "Generalized simulated annealing," *Physica A: Statistical Mechanics and its Applications*, vol. 233, no. 1-2, pp. 395–406, 1996.

[305] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, pp. 87–112, 1994.

[306] P. Brechmann and A. D. Rendall, "Unbounded solutions of models for glycolysis," *Journal of mathematical biology*, vol. 82, pp. 1–23, 2021.

[307] R. De Bartolo and V. Carbone, "The role of the basic three-modes interaction during the free decay of magnetohydrodynamic turbulence," *Europhysics Letters*, vol. 73, no. 4, p. 547, 2006.

[308] A. Coddington and N. Levinson, *Theory of Ordinary Differential Equations* (International series in pure and applied mathematics). McGraw-Hill, 1955, ISBN: 9780070992566.

[309] M. Buisson-Fenet, F. Solowjow, and S. Trimpe, "Actively learning gaussian process dynamics," in *L4DC*, ser. Proceedings of Machine Learning Research, vol. 120, PMLR, 2020, pp. 5–15.

[310] S. Becker, M. Klein, A. Neitz, G. Parascandolo, and N. Kilbertus, "Predicting ordinary differential equations with transformers," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 202, PMLR, 2023, pp. 1978–2002.

A. Appendix for Chapter 3

A.1 Probability Distribution of Algorithm 2

A.1.1 Definitions and Notations

This section is for the proofs related to the probability distribution in the proposed Algorithm 2. For convenience, commonly used notations are listed in Table A.1. We make some slight changes to some notations that appear in the main paper to make sure they are consistent and well-defined in this proof.

Similar to the previous analysis [95, 288], we begin by introducing the concept "dependency graph" (in Definition A.1.1) for the constraints $\mathcal{C}$.

Definition A.1.1 (Dependency Graph). The dependency graph $G = (\mathcal{C}, E)$, where the vertex set is the set of constraints $\mathcal{C}$. Two vertices c_i and c_j are connected with an edge $(c_i, c_j) \in E$ if and only if they are defined on at least one common random variable, i.e., $\texttt{var}(c_i) \cap \texttt{var}(c_j) \neq \emptyset$.

For keeping track of the whole sampling procedure, we need the concept "sampling record" (in Definition A.1.2) [95], which are those violated constraints at every round in Algorithm 2. It is also known as the witness tree in [94]. This allows us to check the constraint satisfaction of the assignment at every round.

Under Condition 3.2.1, for any edge in the dependency graph $(c_i, c_j) \in E$, either $\mathbf{1}(x, c_i) = 0$ or $\mathbf{1}(x, c_j) = 0$ for all $x \in \mathcal{X}$. In other words, two constraints with shared related variables, representing two *adjacent vertices* in the dependency graph G, are not broken simultaneously. Thus, the constraints in record S_t form an *independent set*[1] over the dependency graph under Condition 3.2.1.

Definition A.1.2 (Sampling Record). Given dependency graph $G(\mathcal{C}, E)$, let $X_t = x$ be one possible assignment obtained at round t of Algorithm 2. Let $S_t \subseteq \mathcal{C}$ be the set of vertices in graph G (subset of constraints) that x violates, that is:

$$S_t = \{c_i | c_i \in \mathcal{C} \text{ and } \mathbf{1}(x, c_i) = 0\}, \tag{A.1}$$

[1] ↑A set of vertices with no two adjacent vertices in the graph.

Table A.1. Summary of all the notations used in the theoretical analysis of Algorithm 2.

Notation	Definition
$X = \{X_i\}_{i=1}^n$	set of discrete random variables
$x \in \mathcal{X}$	possible assignments for variables X
$x_i \in \mathcal{X}_i$	variable X_i can take all values in $\mathcal{X}_i$
$\mathcal{C} = \{c_j\}_{j=1}^m$	given constraints
$S_t \subseteq \mathcal{C}$	subset of constraints violated at round t of Algorithm 2
$\texttt{var}(c_j)$	the indices of domain variables that are related to constraint c_i
$\texttt{var}(S_t)$	the indices for domain variables that are related to constraints S_t
$G(\mathcal{C}, E)$	the dependency graph (in Definition A.1.1)
$\Gamma(c_j)$	c_j and its direct neighbors in the dependency graph
$\Gamma(S_t)$	S_t and direct neighbors of S_t in the dependency graph
$\mathcal{C}\backslash\Gamma(S_t)$	all constraints in $\mathcal{C}$ but not in $\Gamma(S_t)$
$S_1, \ldots, S_T, \emptyset$	a sampling record of Algorithm 2 (in Definition A.1.2)
$\mathbf{1}(x, c_i)$	indicator function that evaluates if assignment x satisfies constraint c_i
$\mathbf{1}(x, S_t)$	indicator function that evaluates if assignment x satisfies constraints in S_t
$\mathbf{1}(x, \mathcal{C})$	indicator function that evaluates if assignment x satisfies all constraints $\mathcal{C}$
$P_\theta(X\|\mathcal{C}\backslash\Gamma(S_t))$	see Definition A.1.4
$\mathbb{P}(X\|S_t)$	see Definition A.1.5

where indicator function $\mathbf{1}(x, c_i) = 0$ implies x violates constraint c_i at round t. Define the sampling record as the sequence of violated constraints $S_1, \ldots, S_t$ throughout the execution.

At round t ($t \geq 1$) of Algorithm 2, suppose the violated constraints is $S_t \subseteq \mathcal{C}$. The constraints that are not adjacent to S_t in the dependency graph are still satisfied after re-sample. The only possible constraints that might be broken after the re-sample operation are among S_t itself, or those constraints directly connected to S_t in the dependency graph. Therefore,

$$S_{t+1} \subset \Gamma(S_t), \qquad \text{for all } t \geq 1.$$

where $\Gamma(S_t)$ is the set of vertices of S_t and its adjacent neighbors in the dependency graph G (see Table A.1). When Algorithm 2 terminates at round $T + 1$, no constraints are violated anymore, i.e., $s_{T+1} = \emptyset$. To summarize the above discussion on a sampling record by Algorithm 2, we have the following Claim A.1.3.

Claim A.1.3. Under Condition 3.2.1, a potential sampling record of length $T + 1$ by the Algorithm 2 is a sequence of independent sets: $S_1, S_2, \ldots, S_T, \emptyset$ with

1. $S_{t+1} \subseteq \Gamma(S_t)$ and $S_t \neq \emptyset$, for $1 \leq t \leq T$;
2. $s_{T+1} = \emptyset$.

Extra Notations related to Constrained MRF. The constrained MRF model over constraints set $\mathcal{C}$ is defined as:

$$P_\theta(X = x|\mathcal{C}) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C})}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}(x', \mathcal{C})}$$

where the partition function only sums over valid assignments in $\mathcal{X}$. Note that $C(x)$ in Equation (3.6) is the same as $\mathbf{1}(x', \mathcal{C})$ in the above equation. We slightly change the notations for consistency in this proof. Also notice that the output distribution can no longer be factorized after constraints are enforced, since the partition function cannot be factorized. Our task is to draw samples from this distribution.

To analyze the intermediate steps in Algorithm 2, we further need to define the following notations.

Definition A.1.4. The constrained MRF distribution for constraints $\mathcal{C}\backslash\Gamma(S_t)$ is

$$P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t)) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C}\backslash\Gamma(S_t))}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}(x', \mathcal{C}\backslash\Gamma(S_t))}$$

where $\mathcal{C}\backslash\Gamma(S_t)$ denotes all constraints in $\mathcal{C}$ but not in $\Gamma(S_t)$.

Definition A.1.5. At round t of Algorithm 2, assume $S_t \subseteq \mathcal{C}$ are the set of broken constraints, Define $\mathbb{P}(X_{t+1} = x|S_1, \ldots, S_t)$ to be the probability of obtaining a new assignment x after we re-sample random variables indexed by $\mathtt{var}(S_t)$.

A.1.2 Ratio Property Lemma

Lemma A.1.6 (Ratio Property). *Under Condition 3.2.1, assume Algorithm 2 is at round t. Conditioning on observing one possible sampling record $S_1, \ldots, S_t$, Algorithm 2 step 4 will re-sample variables in $\mathtt{var}(S_t)$ at round $t + 1$. Let $x, x' \in \mathcal{X}$ be two possible assignments* *after this re-sample. The probability ratio of obtaining these two results equals that under constrained MRF $P_\theta(x|\mathcal{C}\backslash\Gamma(S_t))$:*

$$\frac{\mathbb{P}(X_{t+1}=x|S_1,\dots,S_t)}{\mathbb{P}(X_{t+1}=x'|S_1,\dots,S_t)}=\frac{P_\theta(X=x|\mathcal{C}\backslash\Gamma(S_t))}{P_\theta(X=x'|\mathcal{C}\backslash\Gamma(S_t))}, \tag{A.2}$$

where $\mathbb{P}(X_{t+1}=x|S_1,\dots,S_t)$ is the probability of Algorithm 2 step 4 produces assignment x at round $t+1$, conditioning on the observed record $S_1,\dots,S_t$ and re-sample variables indexed by $\texttt{var}(S_t)$. *$P_\theta(X=x|\mathcal{C}\backslash\Gamma(S_t))$ is the constrained MRF (for the constraints $\mathcal{C}\backslash\Gamma(S_t)$) probability on assignment x.*

Proof. During the intermediate step of the algorithm, assume the set of constraints S_t are violated. We want to re-sample variables indexed by $\texttt{var}(S_t)$, so variables indexed by $\texttt{var}(\mathcal{C}\backslash\Gamma(S_t))$ won't change assignments. Also, because $\Gamma(S_t)$ is the largest possible set of constraints that can be infected by the re-sample, constraints $\mathcal{C}\backslash\Gamma(S_t)$ are still satisfied after the re-sample.

At round t, we re-sample variables in $\texttt{var}(S_t)$ according to step 4 in Algorithm 2, we thus have:

$$\mathbb{P}(X^{t+1}_{\texttt{var}(S_t)}=x_{\texttt{var}(S_t)}|S_1\dots S_t)=\prod_{i\in\texttt{var}(S_t)}\frac{\exp(\theta_i x_i)}{\sum_{x'_i\in\mathcal{X}_i}\exp(\theta_i x'_i)}.$$

Here the notation $X^{t+1}_{\texttt{var}(S_t)}=x_{\texttt{var}(S_t)}$ means $X_i=x_i$ for $i\in\texttt{var}(S_t)$ at round t. For any two possible assignments x,x' after the re-sample,

$$x_i=x'_i,\qquad \text{for } i\in\{1,\dots,n\}\backslash\texttt{var}(S_t)$$

since the rest variable's assignments are kept the same after re-sample.

Based on the above derivations, we can have the ratio:

$$\frac{\mathbb{P}(X_{t+1}=x|S_1,\dots,S_t)}{\mathbb{P}(X_{t+1}=x'|S_1,\dots,S_t)}=\frac{\exp(\sum_{i\in\texttt{var}(S_t)}\theta_i x_i)}{\exp(\sum_{i\in\texttt{var}(S_t)}\theta_i x'_i)}=\frac{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t))}\theta_i x_i)}{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t))}\theta_i x'_i)}. \tag{A.3}$$

The last equality holds because every assignment outside $\texttt{var}(S_t)$ is not changed, we can enlarge the index set of summation to $\Gamma(S_t)$ by multiplying

$$1 = \frac{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t)\backslash S_t)}\theta_i x_i)}{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t)\backslash S_t)}\theta_i x_i')}.$$

After re-sample, we knows that x must satisfy the constraints $\mathcal{C}\backslash\Gamma(S_t)$. Thus, the probability of this x conditioned on constraints $\mathcal{C}\backslash\Gamma(S_t)$ holding in the constrained MRF model is:

$$P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t)) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C}\backslash\Gamma(S_t))}{\sum_{x'\in\mathcal{X}}\exp(\sum_{i=1}^n \theta_i x_i')\mathbf{1}\left(x', \mathcal{C}\backslash\Gamma(S_t)\right)} = \frac{\exp(\sum_{i=1}^n \theta_i x_i)}{\sum_{x'\in\mathcal{X}}\exp(\sum_{i=1}^n \theta_i x_i')\mathbf{1}\left(x', \mathcal{C}\backslash\Gamma(S_t)\right)}.$$

In the constrained MRF model (for constraints $\mathcal{C}\backslash\Gamma(S_t)$), the ratio of these two probabilistic assignments x, x' is:

$$\frac{P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t))}{P_\theta(X = x'|\mathcal{C}\backslash\Gamma(S_t))} = \frac{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t))}\theta_i x_i)}{\exp(\sum_{i\in\texttt{var}(\Gamma(S_t))}\theta_i x_i')}, \tag{A.4}$$

because the x_i outside $\texttt{var}(\Gamma(S_t))$ remains the same.

Note that x, x' are two possible assignments produced according to to step 4 in Algorithm 2 at round t. Combining Equation (A.3) and Equation (A.4), we conclude that:

$$\frac{\mathbb{P}(X_{t+1} = x|S_1, \ldots, S_t)}{\mathbb{P}(X_{t+1} = x'|S_1, \ldots, S_t)} = \frac{P_\theta(X = x|\mathcal{C}\backslash\Gamma(S_t))}{P_\theta(X = x'|\mathcal{C}\backslash\Gamma(S_t))}.$$

The proof is finished.

A.1.3 Proof of Theorem 3.3.1

Suppose the re-sampling process terminates at round $T+1$ and we obtain a valid sample x. Upon the termination of Algorithm 2, all the constraints are satisfied. So we have: $S_{T+1} = \emptyset$. In other words, $\mathbf{1}(x, \mathcal{C}) = 1$.

Let x, x' be two possible valid assignments produced at round $T+1$ by the Algorithm 2. Using the analysis in Lemma A.1.6, we can still have:

$$\frac{\mathbb{P}(X_{T+1} = x|S_1, \ldots, S_T)}{\mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)} = \frac{\exp(\sum_{i \in \texttt{var}(S_T)} \theta_i x_i)}{\exp(\sum_{i \in \texttt{var}(S_T)} \theta_i x'_i)}.$$

The probability of this x in the constrained MRF model (for constraints $\mathcal{C}$) is:

$$P_\theta(X = x|\mathcal{C}) = \frac{\exp(\sum_{i=1}^n \theta_i x_i)\mathbf{1}(x, \mathcal{C})}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}\,(x', \mathcal{C})} = \frac{\exp(\sum_{i=1}^n \theta_i x_i)}{\sum_{x' \in \mathcal{X}} \exp(\sum_{i=1}^n \theta_i x'_i)\mathbf{1}\,(x', \mathcal{C})}.$$

Then we conclude that:

$$\frac{\mathbb{P}(X_{T+1} = x|S_1, \ldots, S_T)}{\mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)} = \frac{P_\theta(X = x|\mathcal{C})}{P_\theta(X = x'|\mathcal{C})}.$$

Note that this ratio property holds for all the possible sampling records $S_1, \ldots, S_T, \emptyset$ generated from the algorithm 2.

Summation of All Possible Sampling Records. Define $\mathbb{P}(S_1, \ldots, S_T)$ to be the probability of observing record $S_1, \ldots, S_T$ by Algorithm 2. For any possible sampling record $S_1, \ldots, S_T, \emptyset$, the ratio property still holds:

$$\frac{\mathbb{P}(X_{T+1} = x|S_1, \ldots S_T)\mathbb{P}(S_1, \ldots, S_T)}{\mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)\mathbb{P}(S_1, \ldots, S_T)} = \frac{P_\theta(X = x|\mathcal{C})}{P_\theta(X = x'|\mathcal{C})}$$

where the term $\mathbb{P}(S_1, \ldots, S_T)$ on the left-hand-side (LHS) is actually the same. After we summarize over all possible sampling records $S_1, \ldots, S_T, \emptyset$, the ratio property still holds. Let $\mathbb{P}(X_{T+1} = x)$ be the probability of obtaining one valid assignment x by the execution of Algorithm 2.

$$\frac{\mathbb{P}(X_{T+1} = x)}{\mathbb{P}(X_{T+1} = x')} = \frac{\sum_{S_1, \ldots, S_T} \mathbb{P}(X_{T+1} = x|S_1, \ldots, S_T)\mathbb{P}(S_1, \ldots, S_T)}{\sum_{S_1, \ldots, S_T} \mathbb{P}(X_{T+1} = x'|S_1, \ldots, S_T)\mathbb{P}(S_1, \ldots, S_T)} = \frac{P_\theta(X = x|\mathcal{C})}{P_\theta(X = x'|\mathcal{C})} \tag{A.5}$$

Sample Space Analysis At Termination We need one more statement to show Theorem 3.3.1 holds. Let $\mathcal{X}_{\text{LLL}}$ be the set of all possible assignments x that can be generated by Algorithm 2:

$$\mathcal{X}_{\text{LLL}} = \bigcup_{S_1 \dots S_T} \{x | \mathbb{P}(X_{T+1} = x | S_1, \dots, S_T) \neq 0 \text{ and } \mathbb{P}(S_1, \dots, S_T) \neq 0\}.$$

where $\mathbb{P}(S_1, \dots, S_T) \neq 0$ means $S_1, \dots, S_T$ is a possible record. $\mathbb{P}(X_{T+1} = x | S_1, \dots, S_T) \neq 0$ means it is possible to obtain x given the record $S_1, \dots, S_T$.

Let $\mathcal{X}_{\mathcal{C}}$ be the set of assignments x that satisfy all the constraints in the constrained MRF (for constraints $\mathcal{C}$):

$$\mathcal{X}_{\mathcal{C}} = \{x | P_\theta(X = x | \mathcal{C}) \neq 0, \text{ for all } x \in \mathcal{X}\}.$$

Lemma A.1.7. *$\mathcal{X}_{LLL} \subseteq \mathcal{X}_{\mathcal{C}}$ and $\mathcal{X}_{\mathcal{C}} \subseteq \mathcal{X}_{LLL}$, thus $\mathcal{X}_{LLL} = \mathcal{X}_{\mathcal{C}}$.*

Proof. When Algorithm 2 terminates, it only produces valid assignments; thus, we must have: $\mathcal{X}_{\text{LLL}} \subseteq \mathcal{X}_{\mathcal{C}}$. On the other hand, there is always a non-zero probability that Algorithm 2 will generate every valid assignment $x \in \mathcal{X}_{\mathcal{C}}$, which implies that $\mathcal{X}_{\mathcal{C}} \subseteq \mathcal{X}_{\text{LLL}}$. Therefore we can conclude that $\mathcal{X}_{\text{LLL}} = \mathcal{X}_{\mathcal{C}}$.

Lemma A.1.7 shows that the two distributions have the same sample space when Algorithm 2 terminates. What's more, Equation (A.5) shows they have the same probability ratio for any possible valid assignments x, x'. This shows that the execution of the Algorithm 2 is a random draw from the constrained MRF distribution $P_\theta(X = x | \mathcal{C})$. The proof of Theorem 3.3.1 is finished.

A.1.4 Difference to the Original Proof

The main difference in the above proof to the existing proof in [95, Lemma 7] is that: We show Lemma A.1.6 that characterizes the proportional ratio of getting different assignments of variables, which is more general than the descriptive proof for Guo *et al.*, Lemma 7.

A.1.5 A Running Example from the Markov Chain Monte Carlo Perspective

We dedicate this section to demonstrate the execution of Algorithm 2 with Example 3.4.1. Algorithm 2 can be viewed as a Markov chain, so we will show the probability of obtaining valid samples is unbiased by running thousands of steps of this Markov chain. The constraints are $\mathcal{C} = \{c_1 = (X_1 \vee X_2), c_2 = (\neg X_1 \vee X_3)\}$. We use each assignment of all variables as the states $s_1, \ldots, s_8$ in the rounds of Algorithm 2.

$$\begin{aligned}
s_1 &= (X_0 = 0, X_1 = 0, X_2 = 0)\\
s_2 &= (X_0 = 0, X_1 = 0, X_2 = 1)\\
s_3 &= (X_0 = 0, X_1 = 1, X_2 = 0)\\
s_4 &= (X_0 = 0, X_1 = 1, X_2 = 1)\\
s_5 &= (X_0 = 1, X_1 = 0, X_2 = 0)\\
s_6 &= (X_0 = 1, X_1 = 0, X_2 = 1)\\
s_7 &= (X_0 = 1, X_1 = 1, X_2 = 0)\\
s_8 &= (X_0 = 1, X_1 = 1, X_2 = 1)
\end{aligned} \tag{A.6}$$

Here s_1, s_2, s_3, s_4 correspond to *valid* variables' assignments with respect to the constraints $\mathcal{C}$ and s_5, s_6, s_7, s_8 correspond to *invalid* assignments of variables, that requires resampling.

For simplicity, we consider the uniform setting where $\theta_1 = \theta_2 = \theta_3$. The goal is to sample every valid assignment with equal probability. Therefore, the probability for every variable can be simplified as:

$$P(X_i) = \begin{cases} \frac{1}{2} & \text{for variable } X_i \text{ taking value } 1 \\ \frac{1}{2} & \text{for variable } X_i \text{ taking value } 0 \end{cases} \qquad \text{for } i = 1, 2, 3.$$

Based on Algorithm 2, we know the probability of transferring from s_i to s_j $(1 \leq i, j \leq 8)$. Thus we can construct the transition matrix between every pair of states:

$$T = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix} \end{array} \tag{A.7}$$

where $T_{ij} = T(s_i, s_j)$ denotes the transition probability from state s_i to state s_j.

Taking state s_5 as an example, it violates constraint C_2 thus X_2, X_3 will be resampled. There are 4 possible assignments of X_2, X_3, which corresponds to states $\{s_1, s_2, s_3, s_5\}$. Since each variable is resampled uniformly at random, the probability of transition from state s_5 to the states $\{s_1, s_2, s_3, s_5\}$ are $1/4$. The Algorithm 2 will terminate once it reaches states $\{s_1, s_2, s_3, s_4\}$, which corresponds to those valid states. The valid states only transit to itself with probability 1. Thus we have $T(s_i, s_i) = 1$ for $i = 1, 2, 3, 4$.

For a randomly initialized assignment:

$$x = \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \end{bmatrix} \end{array} \tag{A.8}$$

that has an equal probability of being any state. After executing Algorithm 2 for 2000 steps, we have:

$$T^{2000} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \!\!\begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 & \frac{1}{4} \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{2} & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}, xT^{2000} = \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{A.9}$$

This implies Algorithm 2 outputs every valid assignment with the same probability in the uniform setting, which follows the result in Theorem 3.3.1.

A.2 Running Time Analysis of Algorithm 2

We dedicate this section to showing the running time of Algorithm 2 on a general weighted case. The expected running time of Algorithm 2 is determined by the number of rounds of re-sampling. Algorithm 2 re-sample all the related random variables simultaneously in every single round. However, it is hard to get an estimation of the exact total running time over the *random variables*. Instead, we can only have a loose upper bound of the expected running time over the *sequence of sampling record* (the sequence of violated constraints).

The overall structure of the proof is similar to the proof in Guo *et al.*, Theorem 13. We show the difference in our proof at the end of this section.

A.2.1 Definitions and Notations

We define the following terms to simplify our notations.

Definition A.2.1. Let S_t be a subset of vertices in a dependency graph. 1) Define p_{S_t} as the probability of constraints in S_t being violated:

$$p_{S_t} = \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i\right) \tag{A.10}$$

where we use $\neg c_i$ to indicate the constraint c_i is violated. 2) Define q_{S_t} as the probability that only the constraints in S_t are violated and nothing else.

$$q_{S_t} = \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j\right) \tag{A.11}$$

where $\bigwedge_{c_i \in S_t} \neg c_i$ corresponds to only the constraints in S_t are violated and $\bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j$ corresponds to all the rest constraints are satisfied. So $q_{\{c_i\}}$ is the probability that only constraint c_i is broken and all the rest still hold. Similarly, $q_\emptyset$ denotes the probability that all the constraints are satisfied.

Lemma A.2.2. *Given Definition A.2.1, we can further expand q_{S_t} under Condition 3.2.1:*

$$q_{S_t} = p_{S_t} \mathbb{P}\left(\wedge_{c_j \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right)$$

Proof. We can split q_{S_t} into the probability of two independent events:

$$\begin{aligned}
q_{S_t} &= \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \setminus S_t} c_j\right) && \text{By definition of } q_{S_t} \text{ in Equation (A.11)} \\
&= \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i \wedge \bigwedge_{c_j \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right) \\
&= \mathbb{P}\left(\bigwedge_{c_i \in S_t} \neg c_i\right) \mathbb{P}\left(\bigwedge_{c_n \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right) \\
&= p_{S_t} \mathbb{P}\left(\wedge_{c_j \in \mathcal{C} \setminus \Gamma(S_t)} c_j\right). && \text{By definition of } p_{S_t} \text{ in Equation (A.10)}
\end{aligned}$$

The second equality holds because under Condition 3.2.1, adjacent vertices have zero probability. In other words, when we observe that constraints in S_t are violated, constraints in $\Gamma(S_t)\backslash S_t$ cannot be violated. The third equality holds because the random variables in $\texttt{var}(S_t)$ are *independent to* those variables in $\texttt{var}(\mathcal{C}\backslash\Gamma(S_t))$. So we can apply $P(AB) = P(A)P(B)$ when the events A, B are independent to each other.

Remark A.2.3 (Equivalence of Record). At round t of Algorithm 2, it finds all the constraints S_t that are broken $(\bigwedge_{c_i\in S_t} \neg c_i)$, which implies the rest of the constraints $\mathcal{C}\backslash\Gamma(S_t)$ are satisfied $(\bigwedge_{c_j\in\mathcal{C}\backslash S_t} c_j)$. Thus the probability of observing S_t in the record is equivalent to the following:

$$\mathbb{P}(S_t) = \mathbb{P}\left(\bigwedge_{c_i\in S_t} \neg c_i \wedge \bigwedge_{c_j\in\mathcal{C}\backslash S_t} c_j\right) \tag{A.12}$$

Lemma A.2.4. *Given a possible sampling record $S_1 \dots S_{t-1}$,S_t by Algorithm 2, the following equality holds for the pair (S_{t-1}, S_t):*

$$\sum_{S_t} q_{S_t} = \mathbb{P}(\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})} c_i)$$

Proof. By Definition A.1.2 of the sampling record, we have $S_t \subset \Gamma(S_{t-1})$. The relationship of its complement would be:

$$\mathcal{C}\backslash\Gamma(S_{t-1}) \subset \mathcal{C}\backslash S_t.$$

Using the above result, we have:

$$\mathbb{P}\left(\bigwedge_{c_j\in\mathcal{C}\backslash S_t} c_j \wedge \bigwedge_{c_k\in\mathcal{C}\backslash\Gamma(S_{t-1})} c_k\right) = \mathbb{P}\left(\bigwedge_{c_j\in\mathcal{C}\backslash S_t} c_j\right) \tag{A.13}$$

Based on Remark A.2.3 and Baye's theorem, we have:

$$
\begin{aligned}
&\mathbb{P}(S_t| \wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})} c_i) \\
&= \mathbb{P}\left(\bigwedge_{c_i\in S_t}\neg c_i \wedge \bigwedge_{c_j\in\mathcal{C}\backslash S_t} c_j \middle| \wedge_{c_k\in\mathcal{C}\backslash\Gamma(S_{t-1})} c_k\right) && \text{By Equation (A.12)} \\
&= \frac{\mathbb{P}\left(\bigwedge_{c_i\in S_t}\neg c_i \wedge \bigwedge_{c_j\in\mathcal{C}\backslash S_t} c_j \wedge \bigwedge_{c_k\in\mathcal{C}\backslash\Gamma(S_{t-1})} c_k\right)}{\mathbb{P}\left(\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})}c_i\right)} && \text{By Bayes's formula} \\
&= \frac{\mathbb{P}\left(\bigwedge_{c_i\in S_t}\neg c_i \wedge \bigwedge_{c_k\in\mathcal{C}\backslash S_t} c_k\right)}{\mathbb{P}\left(\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})}c_i\right)} && \text{By Equation (A.13)} \\
&= \frac{q_{S_t}}{\mathbb{P}\left(\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})}c_i\right)}. && \text{By definition of } p_{S_t} \text{ in Equation (A.11)}
\end{aligned}
\tag{A.14}
$$

Since LHS of Equation (A.14) sums over all possible S_t is one: $\sum_{S_t}\mathbb{P}(S_t|\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})} c_i) = 1$. Thus, summing over S_t for the RHS of Equation (A.14), we have:

$$
1 = \sum_{S_t}\frac{q_{S_t}}{\mathbb{P}\left(\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})}c_i\right)} = \frac{\sum_{S_t} q_{S_t}}{\mathbb{P}\left(\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})}c_i\right)} \tag{A.15}
$$

In the second equality, the reason that we can move the summation operator to the numerator is that the denominator is a constant *w.r.t.* all possible S_t. To be specific, given $S_t \subseteq \Gamma(S_{t-1})$, we have S_t is independent to $\mathcal{C}\backslash\Gamma(S_{t-1})$. Based on Equation (A.15), we finally obtain:

$$
\sum_{S_t} q_{S_t} = \mathbb{P}(\wedge_{c_i\in\mathcal{C}\backslash\Gamma(S_{t-1})}c_i).
$$

The proof is finished.

Lemma A.2.5. *The probability of observing the sampling record $S_1,\dots,S_T$ by Algorithm 2 under Condition 3.2.1 is:*

$$
\mathbb{P}(S_1,\dots,S_T) = q_{S_T}\prod_{t=1}^{T-1} p_{S_t} \tag{A.16}
$$

Proof. Given sampling record $S_1, \ldots, S_{t-1}$, the conditional probability of observing the next record S_t, S_t' can be expanded based on Lemma A.1.6,

$$\frac{\mathbb{P}(S_t|S_1,\ldots,S_{t-1})}{\mathbb{P}(S_t'|S_1,\ldots,S_{t-1})} = \frac{\mathbb{P}(S_t|\wedge_{c_i\in\mathcal{C}\setminus\Gamma(S_{t-1})} c_i)}{\mathbb{P}(S_t'|\wedge_{c_i\in\mathcal{C}\setminus\Gamma(S_{t-1})} c_i)}$$

Based on Equation (A.14), we can simplify the RHS of the above ratio equality and obtain:

$$\frac{\mathbb{P}(S_t|S_1,\ldots,S_{t-1})}{\mathbb{P}(S_t'|S_1,\ldots,S_{t-1})} = \frac{q_{S_t}}{q_{S_t'}}$$

Because of $\sum_{S_t}\mathbb{P}(S_t|S_1,\ldots,S_{t-1}) = 1$ and Equation (A.15), we can get:

$$\mathbb{P}(S_t|S_1,\ldots,S_{t-1}) = \frac{q_{S_t}}{\mathbb{P}(\wedge_{c_i\in\mathcal{C}\setminus\Gamma(S_{t-1})}c_i)} \tag{A.17}$$

We finally compute the probability of observing the sampling record $S_1 \ldots, S_T$ by:

$$\begin{aligned}
\mathbb{P}(S_1 \ldots, S_T) =&\mathbb{P}(S_1)\prod_{t=2}^{T}\mathbb{P}(S_t|S_1,\ldots,S_{t-1}) && \text{By Chain rule}\\
=&q_{S_1}\prod_{t=2}^{T}\frac{q_{S_t}}{\mathbb{P}(\wedge_{c_i\in\mathcal{C}\setminus\Gamma(S_{t-1})}c_i)} && \text{By Equation (A.17)}\\
=&q_{S_T}\prod_{t=2}^{T}\frac{q_{S_{t-1}}}{\mathbb{P}(\wedge_{c_i\in\mathcal{C}\setminus\Gamma(S_{t-1})}c_i)} && \text{Left shift the numerator from } S_t \text{ to } S_{t-1}\\
=&q_{S_T}\prod_{t=1}^{T-1}p_{S_t} && \text{Plugin Lemma A.2.2}
\end{aligned}$$

The proof is finished.

A.2.2 An Upper Bound on Expected Running Time

Suppose the expected number of samplings of constraints c_i is $\mathbb{E}(T_i)$, then the total running time will be:

$$\mathbb{E}(T) \leq \sum_{i=1}^{n}\mathbb{E}(T_i)$$

Since each random variable has equal status, then the question comes down to the computation of individual T_i's expectation. Let $S_1, \ldots, S_T$ be any record of the algorithm that successfully terminates, and $T_i(S_1, \dots, S_T)$ be the total number of sampling related to constraint c_i throughout this record. Based on Lemma A.2.5, we have:

$$\mathbb{E}(T_i) = \sum_{S_1,\dots,S_T} \mathbb{P}(S_1, \dots, S_T) T_i(S_1, \dots, S_T)$$

By far, we have shown the original proof of our work. We leave the difference between our proof with the existing one in Appendix A.2.3.

The rest of the computation can be done in the same way as the proof in Guo *et al.* Thus we cite the necessary intermediate steps in the existing work and finish the proof logic for the coherence of the whole running time analysis.

Lemma A.2.6 (Guo *et al.* Lemma 12). *Let $q_\emptyset$ be a non-zero probability of all the constraints are satisfied. Let $q_{\{c_j\}}$ denote the probability that only constraint c_j is broken and the rest all hold. If $q_\emptyset > 0$, then $\mathbb{E}(T_i) = q_{\{c_j\}}/q_\emptyset$.*

After incorporating our fix, we can conclude the upper bound on the expected running time in Theorem A.2.7.

Theorem A.2.7 (Guo *et al.* Theorem 13). *Under Condition 3.2.1, the total number of re-samplings throughout the algorithm is then $\frac{1}{q_\emptyset} \sum_{j=1}^{L} q_{\{c_j\}}$.*

A.2.3 Difference to the Existing Proof

The main difference in the above proof to the existing proof in [95, Theorem 13] is that: based on Equation (A.17) and (A.14), we show

$$\mathbb{P}(S_t | S_1, \dots, S_{t-1}) = \mathbb{P}(S_t | \wedge_{c_i \in \mathcal{C} \setminus \Gamma(S_{t-1})} c_i)$$

In Guo *et al.*'s Equation (9), the first step cannot hold without the above equality. The original paper uses this result directly without providing enough justification.

A.3 Constrained MRF Model

A.3.1 Single Variable Form of Constrained MRF

Here we provide an example of transforming MRF with pairwise and single potential functions into a single potential form by introducing extra variables. Given random variables X_1, X_2, X_3, we have the following example MRF model:

$$\begin{aligned}\phi_\theta(x_1, x_2, x_3) &= \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 \\ P_\theta(x) &= \frac{\exp(\phi_\theta(x_1, x_2, x_3))}{Z(\theta)}\end{aligned}$$

In the above formula, we have a cross term $x_1 x_2$. Two Boolean variables can have 4 different assignments in total. Therefore we can construct 4 extra Boolean variables to encode all these assignments. To illustrate, we introduce extra random variables $\hat{X}_{00}$, $\hat{X}_{01}$, $\hat{X}_{10}$, $\hat{X}_{11}$. We further introduce extra constraints: When $X_1 = 0, X_2 = 0$, the extra variable must take values: $\hat{X}_{00} = 1$, $\hat{X}_{01} = 0$, $\hat{X}_{10} = 0$, $\hat{X}_{11} = 0$. See the rest constraints in Table A.2.

X_1	X_2	$\hat{X}_{00}$, $\hat{X}_{01}$, $\hat{X}_{10}$, $\hat{X}_{11}$
0	0	$1, 0, 0, 0$
0	1	$0, 1, 0, 0$
1	0	$0, 0, 1, 0$
1	1	$0, 0, 0, 1$

Table A.2. 4 constraints for converting pairwise terms in the potential function into single variable form.

Then the new potential function, including extended variables and pairwise to single variable constraints $\mathcal{C}$, is reformulated as:

$$\begin{aligned}\hat{\phi}_\theta(x_1, x_2, x_3, \hat{x}_{00}, \hat{x}_{01}, \hat{x}_{10}, \hat{x}_{11}) &= \theta_1 x_1 + \theta_2 x_2 + \theta_3 \hat{x}_{00} + \theta_3 \hat{x}_{01} + \theta_3 \hat{x}_{10} + \theta_3 \hat{x}_{11} \\ P_\theta(x|\mathcal{C}) &= \frac{\exp(\hat{\phi}_\theta(x_1, x_2, x_3, \hat{x}_{00}, \hat{x}_{01}, \hat{x}_{10}, \hat{x}_{11}))}{Z_{\mathcal{C}}(\theta)}\end{aligned}$$

For clarity, the newly added constraints do not impact Condition 3.2.1. Since the single variable transformation in the MRFs model originates from Sang *et al.*, thus is not considered as our contribution.

A.3.2 Gradient of log-Partition Function

We use the Chain rule of the gradient to give a detailed deduction of Equation (3.4).

$$
\begin{aligned}
\nabla \log Z_{\mathcal{C}}(\theta) = \frac{\nabla Z_{\mathcal{C}}(\theta)}{Z_{\mathcal{C}}(\theta)} &= \frac{1}{Z_{\mathcal{C}}(\theta)} \nabla \sum_{x \in \mathcal{X}} \exp\left(\phi_\theta(x)\right) C(x) \\
&= \sum_{x \in \mathcal{X}} \frac{\exp(\phi_\theta(x)) C(x)}{Z_{\mathcal{C}}(\theta)} \nabla \phi_\theta(x) \\
&= \sum_{x \in \mathcal{X}} P_\theta(x|\mathcal{C}) \nabla \phi_\theta(x) \\
&= \mathbb{E}_{x \sim P_\theta(\tilde{x}|\mathcal{C})} \left[\nabla \phi_\theta(x)\right]
\end{aligned}
\tag{A.18}
$$

The above result shows the gradient of the constrained partition function is equivalent to the expectation of the gradient of the potential function $\nabla \phi_\theta$ over the model's distribution $P_\theta(\tilde{x}|\mathcal{C})$. Therefore, we transform the gradient estimation problem into the problem of sampling from the current MRF model.

A.4 Experiment Settings and Configurations

A.4.1 Implementation Details

Implementation of Nelson. The proposed sampler can be implemented with Numpy, Pytorch, or Jax. We further offer a "batch version" implementation, that draws a batch of samples in parallel on GPU. The batched sampler is useful for those tasks that require a huge number of samples to estimate the gradient with a small approximation error.

In Section 3.4.1, we define vector of assignment $x^t = (x_1^t, \dots, x_n^t)$, where x_i^t is the assignment of variable X_i in the t-th round of Algorithm 2. $x_i^t = 1$ denotes variable X_i takes value

1 (or true). In the batch version, we define the matrix for a batch of assignments. Let b be the batch size, we have

$$x^t = \begin{bmatrix} x^t_{11} & \dots & x^t_{1n} \\ \vdots & \ddots & \vdots \\ x^t_{b1} & \dots & x^t_{bn} \end{bmatrix}$$

In the following part, we provide the detailed computation pipeline for the batch version of the proposed algorithm.

Initialization. The first step is to sample an initial assignment of X from the given the marginal probability vector P:

$$x^1_{li} = \begin{cases} 1 & \text{if } u_{li} > P_i, \\ 0 & \text{otherwise.} \end{cases}, \quad \text{if } 1 \leq i \leq n, 1 \leq l \leq b \tag{A.19}$$

Here u_{li} is sampled from the uniform distribution over $[0, 1]$.

Check Constraint Satisfaction The second step extracts which constraint is violated. Given an assignment x^t at round $t \geq 1$, tensor W and matrix b, the computation of tensor Z^t is:

$$Z^t_{lik} = \sum_{i=1}^{n} W_{ikj} x^t_{lj} + b_{lik},$$

The special multiplication between tensor and matrix can be efficiently implemented with the Einstein summation[2]

$$S^t_{lj} = 1 - \max_{1 \leq k \leq K} Z_{ljk}, \quad \text{for } 1 \leq j \leq L, 1 \leq l \leq b$$

Here $S^t_{lj} = 1$ indicates x^t_l violates j-th clause. We can check $\sum_{l=1}^{b} \sum_{j=1}^{L} S^t_{lj} \neq 0$ to see if any clause is violated for the current batch of assignments, which corresponds to $\sum_{i=1}^{b} C(x_l) = 0$.

[2] ↑https://github.com/dgasmith/opt_einsum. Note that $Z^t_{ljk} = 1$ indicates for l-th batched assignment x_l, the k-th literal of j-th clause is true (takes value 1). Next, we compute S^t_{lj} as:

Extract Variables in Violated Clauses. We extract all the variables that require resampling based on vector S^t computed from the last step. The vector of the resampling indicator matrix A^t can be computed as:

$$A^t_{li} = \mathbf{1}\left(\sum_{j=1}^{L} S^t_{lj} V_{ji} \geq 1\right), \qquad \text{for } 1 \leq i \leq n, 1 \leq l \leq b$$

where $\sum_{j=1}^{L} S^t_{lj} V_{ji} \geq 1$ implies X_{li} requires resampling.

Resample. Given the marginal probability vector P, resample indicator matrix A^t and assignment matrix x^t, we draw a new random sample x^{t+1}.

$$x^{t+1}_{li} = \begin{cases} (1 - A^t_{li})x^t_{li} + A^t_{li} & \text{if } u_{li} > P_i, \\ (1 - A^t_{li})x^t_{li} & \text{otherwise.} \end{cases} \qquad \text{for } 1 \leq i \leq n, 1 \leq l \leq b$$

where u_{li} is drawn from the uniform distribution in $[0, 1]$.

Since GPUs are more efficient at computing tensor, matrix, and vector operations but are slow at processing for loops. Drawing a batch of samples using the above extended computational pipeline is much faster than using a for loop over the computational pipeline in Section 3.4.1.

The sampler is involved with one hyper-parameter T_{tryout}. Nelson would terminate when it reaches T_{tryout} number of re-samples. This practice is commonly used to handle randomized programs that might run forever in rare cases.

Implementation of Algorithm 3 We first use the Constraints $\mathcal{C}$ and parameters θ^t to build the current Nelson module. Then we draw m samples from Nelson module $\{\tilde{x}^j\}_{j=1}^m$ and draw from dataset randomly m samples $\{x^j\}_{j=1}^m$. Continuing from that point, we compute the potential value from the two sets of inputs, i.e., $\{\phi_\theta(\tilde{x}^j)\}_{j=1}^m$ and $\{\phi_\theta(\tilde{x}^j)\}_{j=1}^m$. Pytorch would be slow if we compute each potential's gradient using a for-loop. To bypass this problem, we instead compute the following:

$$\overline{\ell_{\mathcal{C}}(\theta)} = \frac{1}{m}\sum_{j=1}^{m} \phi_\theta(x^j) - \frac{1}{m}\sum_{j=1}^{m} \phi_\theta(\tilde{x}^j). \tag{A.20}$$

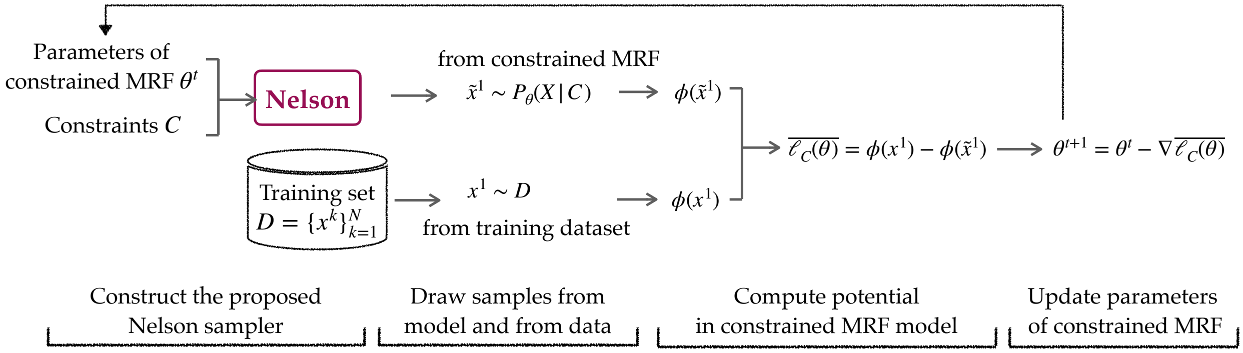

Parameters of constrained MRF θ^t
Constraints C
Nelson
from constrained MRF
$\tilde{x}^1 \sim P_\theta(X \mid C)$
$\phi(\tilde{x}^1)$
Training set $D = \{x^k\}_{k=1}^N$
$x^1 \sim D$
from training dataset
$\phi(x^1)$
$\overline{\ell_C(\theta)} = \phi(x^1) - \phi(\tilde{x}^1)$
$\theta^{t+1} = \theta^t - \nabla\overline{\ell_C(\theta)}$
Construct the proposed Nelson sampler
Draw samples from model and from data
Compute potential in constrained MRF model
Update parameters of constrained MRF

Figure A.1. Implementation pipeline of the NELSON-CD algorithm with $m = 1$. The proposed NELSON can be efficiently adapted to a Pytorch-based machine learning library and enforces constraint satisfaction during learning.

Following that, we call the PyTorch library's gradient function, which computes exactly

$$\nabla\overline{\ell_{\mathcal{C}}(\theta)} = \nabla\left(\frac{1}{m}\sum_{j=1}^{m}\phi_\theta(x^j) - \frac{1}{m}\sum_{j=1}^{m}\phi_\theta(\tilde{x}^j)\right) = \frac{1}{m}\sum_{j=1}^{m}\nabla\phi_\theta(x^j) - \frac{1}{m}\sum_{j=1}^{m}\nabla\phi_\theta(\tilde{x}^j)$$

Note that $\nabla\overline{\ell_{\mathcal{C}}(\theta)}$ recovers the result in Equation (3.4). Finally, we update the parameters θ. The proposed NELSON module and the neural network are computed on the same GPU device. This allows us to exploit the parallel computing power of modern GPUs and remove time for the data transfer from CPU to GPU or vice versa. See Figure A.1 for a visualized overview of the implementation with Pytroch.

A.4.2 Learn Random K-SAT Solutions with Preference

Task Definition We are given a training set $\mathcal{D}$ containing some preferred assignments $\mathcal{D} = \{x^j\}_{j=1}^N$ for the corresponding CNF formula $c_1 \wedge \ldots \wedge c_L$. We require the CNF formula to be true. This means, by the definition of CNF formulas, that every clause has to be satisfied. These clauses become our set of constraints. Under the constrained MRF model, the learning task is to maximize the log-likelihood of the assignments seen in the training set $\mathcal{D}$. The inference task is to generate valid solutions from the learned model's distribution [116, 289].

Dataset We denote the Boolean variables' size in K-SAT as the "problem size". We consider several datasets of different problem sizes generated from CNFGen[3] [117] random K-SAT functions. K is fixed as 5; the number of variables and clauses are kept the same, ranging from 10 to 1500. We generate 100 different CNF formulas for every problem size. To generate the training set $\mathcal{D}$, we use the Glucose4 solver from PySAT[4] library [290] to generate 200 assignments randomly as the preferred assignments for every formula.

Note that we don't consider datasets like SATLIB and SAT competitions. This is mainly because these datasets are hard instances with a much larger input space but a limited number of solutions. Nelson would generally take exponential time to find these solutions, just like finding needles in a haystack. The other reason is that using neural networks to learn these limited assignments is straightforward since we can simply hard-wire the network to memorize all the valid assignments. The main purpose of this work is to let a constrained MRF learn a representation for the underlying preference pattern, not create a neural solver that can generate valid assignments for any CNF formula. Thus, we conform to the settings of the easy formula where obtaining valid solutions is easy.

A.4.3 Learn Sink-Free Orientation in Undirected Graphs

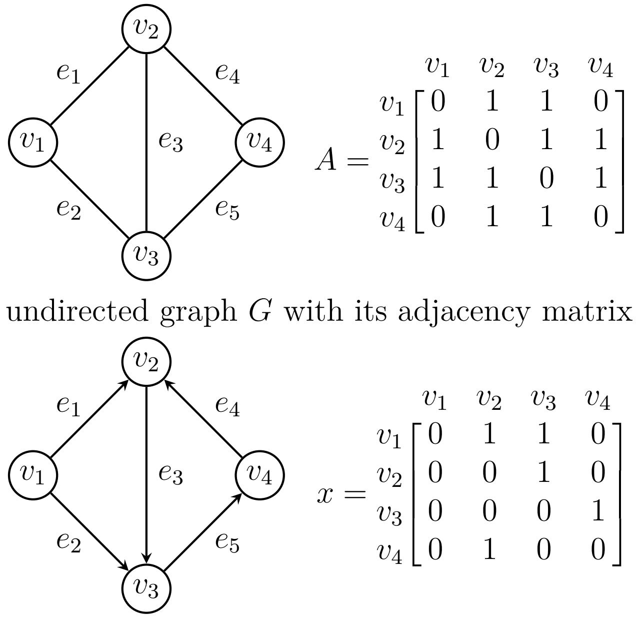
Task Definition In graph theory, a *sink-free* orientation of an undirected graph is a choice of orientation for each edge such that every vertex has at least one outgoing edge [130]. It has wide applications in robotics routing and IoT network configuration [131]. The constraint for this problem is that every vertex has at least one outgoing edge after orientation. As stated in [95], these constraints satisfy Condition 3.2.1.

See Figure A.2 for an example graph and one possible sink-free edge orientation. We define binary variables $X_1, \ldots, X_m$, and associate variable X_i to edge e_i for $1 \leq i \leq m$. Variable X_i takes value 1 if the edge orientation is $v_i \rightarrow v_j$ where $i < j$. Otherwise, X_i takes value 0. The constraints are:

$$\mathcal{C} = (X_1 \vee X_2) \wedge (\neg X_1 \vee X_3 \vee \neg X_4) \wedge (\neg X_2 \vee \neg X_3 \vee X_5) \wedge (X_4 \vee \neg X_5)$$

[3] ↑https://github.com/MassimoLauria/cnfgen

[4] ↑https://pysathq.github.io/



$$A = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 1 & 1 & 0 \\ v_2 & 1 & 0 & 1 & 1 \\ v_3 & 1 & 1 & 0 & 1 \\ v_4 & 0 & 1 & 1 & 0 \end{array}$$

(a) An undirected graph G with its adjacency matrix A

$$x = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 1 & 1 & 0 \\ v_2 & 0 & 0 & 1 & 0 \\ v_3 & 0 & 0 & 0 & 1 \\ v_4 & 0 & 1 & 0 & 0 \end{array}$$

(b) An orientation of the edges and the orientation matrix x

Figure A.2. **(a)** An un-directed graph $G(V, E)$ where the vertices are $V = \{v_1, v_2, v_3, v_4\}$ and the un-directed edges are $E = \{e_1 = (v_1, v_2), e_2 = (v_1, v_3), e_3 = (v_2, v_3), e_4 = (v_2, v_4), e_5 = (v_3, v_4)\}$. **(b)** A possible sink-free orientation of the edges in the graph and its matrix representation x, where every vertex has at least one outgoing edge.

where the single constraint $c_1 = (X_1 \vee X_2)$ corresponds to vertex v_1, constraint $c_2 = (\neg X_1 \vee X_3 \vee \neg X_4)$ corresponds to vertex v_2, constraint $c_3 = (\neg X_2 \vee \neg X_3 \vee X_5)$ corresponds to vertex v_3, and constraint $c_4 = (X_4 \vee \neg X_5)$ corresponds to vertex v_4. The orientation assignment matrix x shown in Figure A.2(b) implies: $X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 0, X_5 = 1$.

Notations Let graph $G(V, E)$ be an un-directed graph; its adjacency matrix A that represents graph connectivity is:

$$A_{ij} = \begin{cases} 1 & \text{If } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{A.21}$$

A possible assignment for the orientation of every edge can be represented as a matrix $x \in \{0, 1\}^{|V| \times |V|}$:

$$x_{ij} = \begin{cases} 1 & \text{if the edge orientation is } v_i \rightarrow v_j \\ 0 & \text{otherwise} \end{cases} \tag{A.22}$$

In the constrained MRF model defined in Equation (3.6), the potential function of one orientation of all edges is

$$\phi_\theta(x) = \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} \theta_{ij} A_{ij} x_{ij}$$

A single constraint for vertex v_k is $c_k(x) = \mathbf{1}\left(\sum_{j=1}^{n} A_{k,j} x_{k,j} = 1\right)$. If there is no ongoing edge of vertex v_k. The constraint function $C(x)$ is defined as: $\prod_{i=1}^{n} c_k(x)$. In Algorithm 2 step 1, edge (v_i, v_j) will pick the orientation $v_i \to v_j$ with probability:

$$\frac{\exp(\theta_{ij} A_{ij} x_{ij})}{\exp(\theta_{ji} A_{ji} x_{ji}) + \exp(\theta_{ij} A_{ij} x_{ij})}$$

Dataset We use the NetworkX[5] package to generate random Erdos Renyi graph with edge probability 0.55. The problem size refers to the number of vertices in the graph, we range the problem size from 10 to 100. For each problem size, we generate 100 different random undirected graphs. We then convert the graph into CNF form using the above edge-variable conversion rule. Afterward, we follow the same processing steps as the previous problem that learn preferential solution distribution for random K-SAT. .

A.4.4 Learn Vehicle Delivery Routes

Given a set of locations to visit, the task is to generate a sequence to visit these locations in which each location is visited once and only once and the sequence closely resembles the trend presented in the training data. The training data are such routes collected in the past. The dataset is constructed from TSPLIB, which consists of 29 cities in Bavaria, Germany. In Figure 3.3, we see NELSON can obtain samples of this delivery problem highly efficiently.

A possible travel plan can be represented as a matrix $x \in \{0,1\}^{|V|\times|V|}$:

$$x_{ij} = \begin{cases} 1 & \text{if edge } v_i \to v_j \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{A.23}$$

The constraints are that every routing plan should visit every location once and only once.

[5] ↑https://networkx.org/

Similarly, in the constrained MRF model defined in Equation (3.6), the potential function of the vehicle routing plan is

$$\phi_\theta(x) = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \theta_{ij} A_{ij} x_{ij}$$

A.4.5 Detailed Baselines Configuation

In terms of sampling-based methods, we consider:

- Gibbs sampler [118], a special case of MCMC that is widely used in training MRF models. In each step, the Gibbs algorithm samples one dimension based on a conditional marginal distribution. We follow this implementation[6].
- Weighted SAT samplers, including WAPS[7] [119], WeightGen[8] [120] and XOR sampler[9] [121, 122].
- Uniform SAT samplers, including UniGen[10] [125], QuickSampler[11] [124], CMSGen[12] [123] and KUS[13] [126].

Currently, there are only GPU-based SAT solvers [127, 291] and model counters [128], GPU-based SAT samplers are not available by far.

A.4.6 Detailed Definition of Evaluation Metrics

In terms of evaluation metrics, we consider

- Training time per epoch. The average time for the whole learning method to finish one epoch with each sampler.
- Validness. The learned model is adopted to generate assignments and we evaluate the percentage of generated assignments that satisfy the constraints.

[6] ↑https://github.com/Fading0924/BPChain-CD/blob/master/mrf.py
[7] ↑https://github.com/meelgroup/waps
[8] ↑https://bitbucket.org/kuldeepmeel/weightgen/src/master/
[9] ↑https://cs.stanford.edu/~ermon/code/srcPAWS.zip
[10] ↑https://github.com/meelgroup/unigen
[11] ↑https://github.com/RafaelTupynamba/quicksampler
[12] ↑https://github.com/meelgroup/cmsgen
[13] ↑https://github.com/meelgroup/KUS

- Mean Averaged Precision (MAP@10). This is a ranking-oriented metric that can evaluate the closeness of the learned MRF distribution to the goal distribution. If the model learns the goal distribution in the training set, then it would assign a higher potential value to those assignments in the training set than all the rest unseen assignments. Based on this principle, we randomly pick two sets of inputs in those valid assignments: seen assignments from the training set and unseen assignments that are randomly generated. We use the value of factor potential $\phi(x)$ to rank those assignments in ascending order. Next, we check how many preferred solutions can fall into the Top-10 by computing the following

$$\text{MAP@10} = \sum_{k=1}^{10} \frac{\text{\#preferred assignments among top-}k}{k}$$

- log-likelihood of assignments in the training set $\mathcal{D}$. The model that attains the highest log-likelihood learns the closest distribution to the training set. Specifically, given a training set $\mathcal{D} = \{x^k\}_{k=1}^N$ and parameters θ, the log-likelihood value is:

$$\frac{1}{N}\sum_{k=1}^{N} \log P_\theta(X = x^k|\mathcal{C}) = \frac{1}{N}\sum_{k=1}^{N} \phi_\theta(x^k) - \log Z_\mathcal{C}(\theta) \tag{A.24}$$

  We use the ACE algorithm to compute the approximated value of $\log Z_\mathcal{C}(\theta)$[14].

- Approximation Error of $\nabla \log Z_\mathcal{C}(\theta)$, that is the L_1 distance between the exact gradient of $\log Z_\mathcal{C}(\theta)$ in Equation (3.4) and the empirical gradient from the sampler. For small problem sizes, we enumerate all $x \in \mathcal{X}$ to get the exact gradient and draw samples $\{\tilde{x}^j\}_{j=1}^m$ with $m = 2000$ from every sampler for approximation.

$$\left| \underbrace{\sum_{x\in\mathcal{X}} \frac{\exp\left(\sum_{j=1}^n \theta_j x_j\right) C(x)}{Z_\mathcal{C}(\theta)} x_i}_{\text{Exact gradient term}} - \underbrace{\sum_{j=1}^m \tilde{x}_i^j}_{\text{Estimated gradient with sampler}} \right|$$

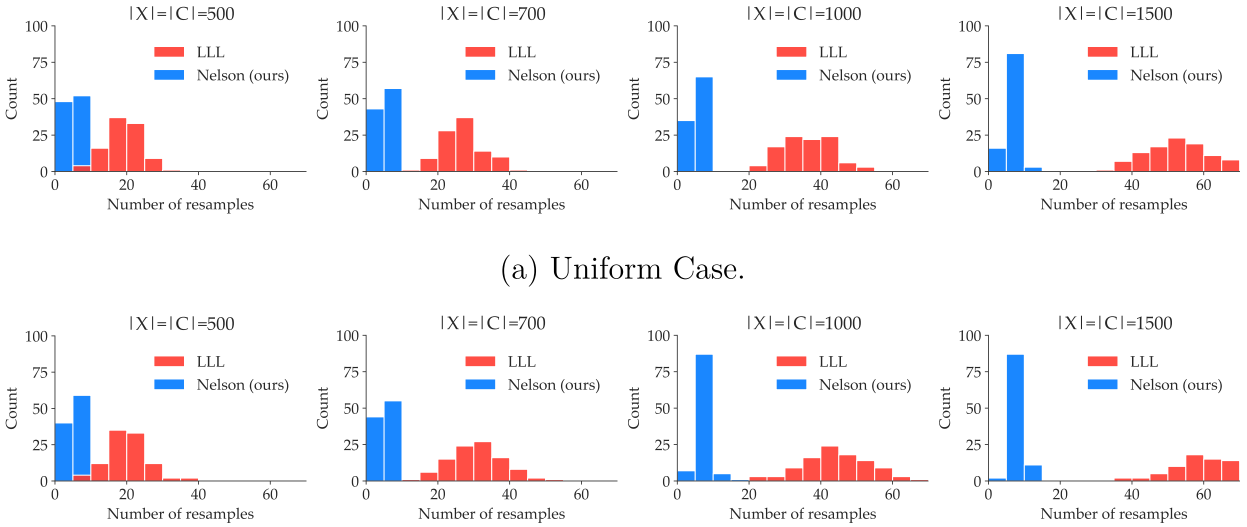
  For fixed parameter θ, the best sampler would attain the smallest approximation error.

[14] ↑http://reasoning.cs.ucla.edu/ace/moreInformation.html

A.4.7 Hyper-parameter Settings

In the implementation of Nelson, we set the maximum tryout of resampling as $T_{tryout} = 1000$ for all the experiments and all the datasets.

For the hyper-parameters used in learning the constrained MRF, we set the number of samples from the model to be $m = 200$, the learning rate η is configured as 0.1 and the total learning iterations are $T_{\max} = 1000$.



(a) Uniform Case.

(b) Weighted Case.

Figure A.3. The distribution of resampling steps in the Nelson and Algorithmic-LLL [94]. Both of them get a valid sample within $T_{tryouts}$. Nelson takes much fewer resamples than Algorithmic-LLL because it resamples all the violated clauses at every iteration while Algorithmic-LLL only resamples one of them.

B. Appendix for Chapter 4

B.1 Detailed Experiment Settings

In this section, we detail our experimental settings for interrogative, imperative, and sentimental sentence generation tasks, along with the process of human evaluation.

In the expression of stationary distribution Equation(4.1), the first term $P_{\text{LM}}(x)$ is evaluated by the BERT model, which is based on the huggingface's BERT implementation [292]. We use BERT-base in our experiments, with hyper-parameters: L=12, H=768, A=12, Total Parameters=110M. To evaluate the term $P_{\text{LM}}(x)$ with the BERT model, we multiply the BERT score of masking and querying the conditional probability of each word in sentence x, close in form of the pseudo-likelihood [293]. Since we only require $\pi(x)$ to be proportional to $P_{\text{LM}}(x)$ times the constraint score, $P_{\text{LM}}(x)$ does not need to be normalized.

B.1.1 Interrogative Sentences Generation

According to the adapted definition of interrogative sentence grammar, the first word should be a question word, and there should be an auxiliary verb at a suitable position. In our actual implementation, we also enforce that there should be only one question word and one auxiliary verb in the sentence in order to improve the quality of generated sentences. The question words include *what, when, where, which, who, whom, whose, why, how*; the auxiliary verbs include *do, does, did, be, am, are, is, was, were, shall, will, should, would, can, could, may, might, must.*

For the task of generating interrogative sentences with keywords, we also enforce that the keyword only appears once in the sentence.

The dataset of this task is based on the SQuAD 2.0 dataset [150], where we select 600 questions and remove the stop words using the Rake toolkit [294].

B.1.2 Imperative Sentences Generation

The dataset for generating imperative sentences is retrieved from[1]. We select 300 sentences and extract the keywords from the sentences as our input. According to the grammar of imperative sentences, we need to verify if the word is a present-tense verb. In the implementation, we use the POS tag information in WordNet and Stanford CoreNLP as the criterion for deciding the word POS tag of the given word. We first select all the words with at least one verb meaning in WordNet [295], then use Stanford CoreNLP [296] to get POS tags for each word and only preserve the present tense form of verbs.

B.1.3 Sentiment Sentence Generation

This application requires a set of input keywords and an external sentiment classifier, which is used to estimate whether the sentiment of the sentence is positive or not. To estimate the sentiment score of the sentences, we train a sentiment analysis model with fastText [297] on Yelp Review Polarity dataset [298]. The input keywords are extracted from 300 selected sentences in the Yelp test set. Half of the original sentences are positive, and the other half are negative (which is harder to transform into positive sentences).

With input keywords of positive and negative sentiment, we enforce the model to generate sentences with positive sentiment. The second sub-task with negative sentiment keywords is much more difficult than the sub-task with positive sentiment keywords, as it requires transforming from negative to positive sentiment.

B.2 Case Studies

As shown in Table B.1, we compare some output sentences of our method with the baseline using the same inputs and keywords. From these cases, we can see that the baseline sometimes generates awkward or disordered sentences. For example, the baseline generates one sentence:"*how was lower Normandy ever truly founded?*". Although this sentence seems to satisfy the constraints of an interrogative sentence, its meaning is awkward. The sentence

[1] ↑https://github.com/lettergram/sentence-classification

Keywords	university warsaw established
TSMH (Ours)	when was the technical university of warsaw first formally established?
CGMH	Polish Polytechnical Institute - university of technology warsaw - was established here in 1964?
Keywords	organization charge running
TSMH (Ours)	who would charge her with running such an organization?
CGMH	who else would charge him with running a very profitable business?
Keywords	tribes khan fight
TSMH (Ours)	what tribes would fight back against the Genghis khans?
CGMH	why else would tribesmen like gen. and gen. genghis khan fight them off?
Keywords	European travel amazon
TSMH (Ours)	why did early european explorers not travel to amazonia?
CGMH	see below, also : did any european settlers ever travel to build the ”first North American sailing canoes”?
Keywords	economic growth schooling
TSMH (Ours)	how do economic growth rates in the united states make children receive high - quality schooling?
CGMH	what good is economic growth in comparison with being among the best in public schooling?

(1) Interrogative Sentences

Keywords	seat
TSMH (Ours)	please get up from your seat
CGMH	go on in and take your seat
Keywords	careful
TSMH (Ours)	please be so very very careful.
CGMH	and please be a very very careful
Keywords	turn, lights
TSMH (Ours)	turn on the lights all the time
CGMH	turn on near all the main lights
Keywords	close, window
TSMH (Ours)	stay close enough to the window
CGMH	stick close enough to meet the window
Keywords	nice, weekend
TSMH (Ours)	have yourself a very nice private weekend
CGMH	please be nice about spending the weekend

(2) Imperative Sentences

Table B.1. Case study of generating interrogative and imperative sentences with keywords.

generated by our method is "*when was the duchy of Normandy founded?*", which is more realistic. Also, the sentence from the baseline "*and please be a very very careful*" does not follow imperative grammar, and "*the Catholics are now mainly concentrated there*" is not a question.

C. Appendix for Chapter 5

C.1 Convergence Analysis of the Maximum Likelihood Learning

C.1.1 Basic Definitions and Properties

The following definitions are related to the convergence analysis of the gradient-descent-based approaches.

Definition C.1.1 (L-Smoothness). A function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is noted as L-smooth with smoothness constant $L \geq 0$ if and only if for all $\theta_1, \theta_2 \in \mathbb{R}^d$,

$$\|\nabla \mathcal{L}(\theta_1) - \nabla \mathcal{L}(\theta_2)\|_2 \leq L\|\theta_1 - \theta_2\|_2 \tag{C.1}$$

Remark C.1.2. In the above definition about the L-smoothness, Equation (C.1) can be equivalently reformulated as:

$$\mathcal{L}(\theta_1) \leq \mathcal{L}(\theta_2) + \langle \nabla \mathcal{L}(\theta_2), \theta_1 - \theta_2 \rangle + \frac{L}{2}\|\theta_1 - \theta_2\|_2^2 \tag{C.2}$$

Lemma C.1.3 (Mean value property). *Let function $\mathcal{L} : [\theta_1, \theta_2] \to \mathbb{R}$ be continuous on the closed interval $[\theta_1, \theta_2]$, and differentiable on the open interval (x_1, x_2), where $\theta_1 < \theta_2$. Then there exists $\tilde{\theta} \in (\theta_1, \theta_2)$ such that*

$$\nabla \mathcal{L}(\tilde{\theta}) = \frac{\mathcal{L}(\theta_2) - \mathcal{L}(\theta_1)}{\theta_2 - \theta_1} \tag{C.3}$$

The following definitions and properties are matrix-related terms and notations.

Definition C.1.4 (Covariance Matrix). The covariance matrix of a random vector $X \in \mathbb{R}^d$ with mean vector μ is defined via:

$$\mathtt{Cov}(X) = \mathbb{E}[(X - \mu)(X - \mu)^\top] \tag{C.4}$$

For the diagonal term, $(i, i)^{th}$ of this covariance matrix is the variance of X_i:

$$\mathtt{Var}(X_i) = \mathbb{E}[(X_i - \mu_i)^2] \tag{C.5}$$

The off-diagonal $(i,j)^{th}$ element (where $i \neq j$) of this covariance matrix $\texttt{Cov}(X)$ is given by

$$\texttt{Cov}(X)_{ij} = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)] \tag{C.6}$$

Definition C.1.5 (Matrix L_2-Norm). Given matrix $A \in \mathbb{R}^{m\times n}$, the matrix L_2-norm is defined as:

$$\|A\|_2 = \max_{x\neq 0} \frac{\|Ax\|}{\|x\|_2} = \max_{\|x\|_2=1} \|Ax\|_2 \tag{C.7}$$

Lemma C.1.6. *If matrix A is symmetric and positive definitive, then all the eigenvalues are positive. Let $\lambda_{\max}$ be the largest eigenvalue of A, then*

$$\|A\|_2 = \max_{1\leq i\leq d} |\lambda_i| = |\lambda_{\max}| = \lambda_{\max} \tag{C.8}$$

Definition C.1.7 (Leibniz formula for determinants). The Leibniz formula for the determinant of the square matrix $\det(A)$ is

$$\det(A) = \sum_{\pi} \prod_{i=1}^{d} \texttt{sgn}(\pi) A_{i,\pi_i} \tag{C.9}$$

where π is a permutation for the index set $\{1,\ldots,n\}$; the sum in the RHS is over all permutations π of $\{1,\ldots,n\}$; $\texttt{sgn}$ is the sign function of permutations in the permutation group, which returns $+1$ and -1 for even and odd permutations[1], respectively; A_{i,π_i} is the element in i-th row and π_i-th column of matrix A.

Definition C.1.8 (Trace of Matrix). The trace of a square matrix $A \in \mathbb{R}^{d\times d}$, noted as $tr(A)$, is the sum of the diagonal elements of A: $\texttt{Tr}(A) = \sum_{i=1}^{d} A_{i,i}$.

Lemma C.1.9. *The trace of a symmetric matrix $A \in \mathbb{R}^{d\times d}$ is equal to the sum of its eigenvalues:* $\texttt{Tr}(A) = \sum_{i=1}^{d} \lambda_i$.

Proof. Recall the characteristic equation of matrix A is:

$$\det(x\mathbf{I} - A) = (x - \lambda_1)\ldots(x - \lambda_d) \tag{C.10}$$

[1] ↑https://en.wikipedia.org/wiki/Parity_of_a_permutation

Observe that the coefficient of x^{d-1} in the RHS is equal to $-\sum_{i=1}^{d} \lambda_i$. To prove the claim above, it is enough to show that the coefficient of x^{d-1} is the negative of the trace of A.

According to Definition C.1.7, the Leibniz formula for the determinant of the square matrix $\det(x\mathbf{I} - A)$ is

$$\det(x\mathbf{I} - A) = \sum_{\pi} \prod_{i=1}^{d} \operatorname{sgn}(\pi)(x\mathbf{I} - A)_{i,\pi_i} \tag{C.11}$$

Observe that for every permutation π in the RHS either $\pi_i = i$ for all i or there exists at least two indices i, j such that $\pi_i = i$ and $\pi_j = j$. But the latter case does not give any monomial of degree $d-1$ in x. It can only give monomials of degree at most $d-2$. Now, consider the terms coming from the identity permutation π as the coefficient of x^{n-1} comes from this permutation. It follows that such a permutation has sign $+1$. So we just need to figure out the coefficient of x^{d-1} coming from the product of diagonal entries of the matrix $x\mathbf{I} - A$,

$$\prod_{i=1}^{d} (x\mathbf{I} - A)_{i,i} = \prod_{i=1}^{d} (x\mathbf{I} - A_{i,i}) \tag{C.12}$$

The RHS is exactly the negative of the sum of diagonal entries of A. The proof is completed.

C.2 The Relationship between Variance and L-smoothness

Lemma C.2.1. *$\mathcal{L}_{LB}(\theta)$ is $Var_{P_\theta}(\phi(a))$-smooth w.r.t. θ.*

Proof. According to Definition C.1.1, to show $\mathcal{L}_{LB}(\theta)$ L-smooth, it is equivalent to show that

$$\|\nabla \mathcal{L}_{LB}(\theta_1) - \nabla \mathcal{L}_{LB}(\theta_2)\|_2 \leq L \|\theta_1 - \theta_2\|_2,$$

where $\theta_1, \theta_2 \in \mathbb{R}^d$ and L is a constant. Based on the mean value lemma C.1.3, there exists a point $\tilde{\theta} \in (\theta_1, \theta_2)$ such that

$$\nabla \mathcal{L}_{LB}(\theta_1) - \nabla \mathcal{L}_{LB}(\theta_2) = \left(\nabla^2 \mathcal{L}_{LB}(\tilde{\theta})\right)(\theta_1 - \theta_2).$$

where $\nabla^2\mathcal{L}_{LB}(\theta) \in \mathbb{R}^{d\times d}$ is the gradient for the function $\nabla\mathcal{L}_{LB}(\theta)$. $\nabla^2\mathcal{L}_{LB}(\theta)$ is also the hessian matrix for the function $\mathcal{L}_{LB}(\theta)$. Taking the L_2 norm on both sides, we have

$$\|\nabla\mathcal{L}_{LB}(\theta_1) - \nabla\mathcal{L}_{LB}(\theta_2)\|_2 = \|\nabla^2\mathcal{L}_{LB}(\tilde{\theta})(\theta_1 - \theta_2)\|_2 \leq \|\nabla^2\mathcal{L}_{LB}(\tilde{\theta})\|_2 \; \|\theta_1 - \theta_2\|_2$$

Then, the task is to bound the matrix L_2-norm $\|\nabla^2\mathcal{L}_{LB}(\tilde{\theta})\|_2$.

Lemma C.2.2. *By definition of the* log-*partition function* $\log Z_\phi = \log\sum_{a'\in\mathcal{A}}\exp(\phi(a'))$*, its gradient vector* $\nabla\log Z_\phi$ *is the expectation of the function* $\phi(a)$ *under probability distribution* $Pr_\theta(a)$*. The hessian matrix of* $\nabla^2\log Z_\phi$ *is the covariance matrix of function* $\phi(a)$ *under probability distribution* $Pr_\theta(a)$*. That is,*

$$\begin{aligned}\nabla\log Z_\phi &= \mathbb{E}_{a\sim Pr_\theta(a)}[\phi(a)],\\ \nabla^2\log Z_\phi &= Cov_\theta[\phi(a)] = \mathbb{E}_{a\sim Pr_\theta(a)}\left(\left(\phi(a) - \nabla\log Z_\phi\right)\left(\phi(a) - \nabla\log Z_\phi\right)^\top\right).\end{aligned} \tag{C.13}$$

Since we know the explicit form of $\mathcal{L}_{LB}(\theta)$, which is

where $\nabla^2\mathcal{L}_{LB}(\theta)$ is the co-variance matrix. Denote $\mathrm{Cov}_\theta[\phi(a)]$, which is symmetric and positive semi-definite. We have

$$\|\nabla^2\mathcal{L}_{LB}(\tilde{\theta})\|_2 = \|\mathrm{Cov}_\theta[\phi(a)]\|_2 = \lambda_{\max}, \text{By Lemma C.1.6} \tag{C.14}$$

where $\lambda_{\max}$ is the maximum eigenvalue of the matrix $\mathrm{Cov}_\theta[\phi(a)]$. Then because of the positive semi-definiteness of the co-variance matrix, all the eigenvalues are non-negative. By Lemma C.1.9, we can bound $\lambda_{\max}$ as

$$\lambda_{\max} \leq \sum_{i=1}^{d}\lambda_i = \mathtt{Tr}(\mathtt{Cov}_\theta[\phi(a)]),$$

where $\texttt{Tr}(\texttt{Cov}_\theta[\phi(a)])$ is the trace of matrix $\texttt{Cov}_\theta[\phi(a)]$. By Definition C.1.8, $\texttt{Tr}(\texttt{Cov}_\theta[\phi(a)])$ is equal to the summation of diagonal terms in the covariance matrix, which is the summation of all the variance terms $\sum_{i=1}^{d} \texttt{Var}(\phi(a))_i Pr_\theta(a)$. We have

$$\|\nabla^2 \mathcal{L}_{LB}(\tilde{\theta})\|_2 = \|\texttt{Cov}_{\tilde{\theta}}[\phi(a)]\|_2 \leq \sum_{i=1}^{d} \texttt{Var}(\phi(a))_i Pr_\theta(a)$$

Finally, we have

$$\|\nabla \mathcal{L}_{LB}(\theta_1) - \nabla \mathcal{L}_{LB}(\theta_2)\|_2 \leq L\ \|\theta_1 - \theta_2\|_2.$$

where $L = \sum_{i=1}^{d} \texttt{Var}(\phi(a))_i Pr_\theta(a)$. This completes the proof.

C.3 Proof of Theorem 5.3.2

Lemma C.3.1 (Recursion). *For all* $1 \leq t \leq T$,

$$\mathbb{E}[\mathcal{L}_{LB}(\theta_{t+1})] - \mathcal{L}_{LB}(\theta^*) \leq \mathbb{E}[\frac{1}{2\eta}(\|\theta_t - \theta^*\|_2^2 - \|\theta_{t+1} - \theta^*\|_2^2)] + \frac{\eta \texttt{Cov}^2}{M}.$$

Proof. According to the gradient update rule, we have

$$\theta_{t+1} = \theta_t - \eta g_t$$

By L-smooth of $\mathcal{L}_{LB}$ (in Remark C.1.2), we have for the t-th iteration,

$$\begin{aligned}\mathcal{L}_{LB}(\theta_{t+1}) &\leq \mathcal{L}_{LB}(\theta_t) + \langle \nabla \mathcal{L}_{LB}(\theta_t), \theta_{t+1} - \theta_t \rangle + \frac{L}{2}\|\theta_{t+1} - \theta_t\|_2^2, \\ &= \mathcal{L}_{LB}(\theta_t) - \eta \langle \nabla \mathcal{L}_{LB}(\theta_t), g_t \rangle + \frac{L\eta^2}{2}\|g_t\|^2.\end{aligned}$$

Because of $\mathbb{E}[g_t]^2 = \mathbb{E}[\|g_t\|_2^2] - Var(g_t)$, by taking expectation on both sides *w.r.t.* g_t we get

$$\begin{aligned}
\mathbb{E}[\mathcal{L}_{LB}(\theta_{t+1})] &= \mathcal{L}_{LB}(\theta_t) - \eta\mathbb{E}[g_t]^2 + \frac{L\eta^2}{2}\mathbb{E}[\|g_t\|_2^2], \\
&= \mathcal{L}_{LB}(\theta_t) - \eta(\mathbb{E}[\|g_t\|_2^2] - Var(g_t)) + \frac{L\eta^2}{2}\mathbb{E}[\|g_t\|_2^2], \\
&\leq \mathcal{L}_{LB}(\theta_t) - \eta(1 - \frac{L\eta}{2})\mathbb{E}[\|g_t\|_2^2] + \frac{\eta\texttt{Cov}^2}{M}, \\
&\leq \mathcal{L}_{LB}(\theta_t) - \frac{\eta}{2}\mathbb{E}[\|g_t\|_2^2] + \frac{\eta\texttt{Cov}^2}{M}.
\end{aligned}$$

where the last inequality follows as $L\eta \leq 2$. Because $\mathcal{L}_{LB}$ is convex, we get

$$\mathcal{L}_{LB}(\theta_t) - \mathcal{L}_{LB}(\theta^*) \leq \langle \nabla\mathcal{L}_{LB}(\theta_t), \theta_t - \theta^* \rangle$$

We further have,

$$\begin{aligned}
\mathbb{E}[\mathcal{L}_{LB}(\theta_{t+1})] &\leq \mathcal{L}_{LB}(\theta^*) + \langle \nabla\mathcal{L}_{LB}(\theta_t), \theta_t - \theta^* \rangle - \frac{\eta}{2}\mathbb{E}[\|g_t\|_2^2] + \frac{\eta\texttt{Cov}^2}{M}, \\
&= \mathcal{L}_{LB}(\theta^*) + \langle \mathbb{E}[g_t], \theta_t - \theta^* \rangle - \frac{\eta}{2}\mathbb{E}[\|g_t\|_2^2] + \frac{\eta\texttt{Cov}^2}{M}, \\
&= \mathcal{L}_{LB}(\theta^*) + \mathbb{E}[\langle g_t, \theta_t - \theta^* \rangle - \frac{\eta}{2}\|g_t\|_2^2] + \frac{\eta\texttt{Cov}^2}{M}.
\end{aligned}$$

we now repeat the calculations by completing the square for the middle two terms to get

$$\begin{aligned}
\mathbb{E}[\mathcal{L}_{LB}(\theta_{t+1})] &\leq \mathcal{L}_{LB}(\theta^*) + \mathbb{E}[\frac{1}{2\eta}(\|\theta_t - \theta^*\|_2^2 - \|\theta_t - \theta^* - \eta g_t\|_2^2)] + \frac{\eta\texttt{Cov}^2}{M}, \\
&= \mathcal{L}_{LB}(\theta^*) + \mathbb{E}[\frac{1}{2\eta}(\|\theta_t - \theta^*\|_2^2 - \|\theta_{t+1} - \theta^*\|_2^2)] + \frac{\eta\texttt{Cov}^2}{M}.
\end{aligned}$$

Theorem 5.3.2 states the function value of the output of PALM, in expectation converges to the true optimum within a small constant distance at a linear speed *w.r.t.* the number of iterations T.

Proof. Summing the above equations for $t = 0, \dots, T-1$, we get

$$\begin{aligned}\sum_{t=0}^{T-1} \mathbb{E}[\mathcal{L}_{LB}(\theta_{t+1}) - \mathcal{L}_{LB}(\theta^*)] &\leq \frac{1}{2\eta}(\|\theta_0 - \theta^*\|_2^2 - \mathbb{E}[\|\theta_T - \theta^*\|_2^2]) + T\frac{\eta\mathtt{Cov}^2}{M} \\ &\leq \frac{\|\theta_0 - \theta^*\|_2^2}{2\eta} + T\frac{\eta\mathtt{Cov}^2}{M}.\end{aligned}$$

Finally, by Jensen's inequality, $T\mathcal{L}_{LB}(\overline{\theta_T}) \leq \sum_{t=1}^{T} \mathcal{L}_{LB}(\theta_t)$, thus,

$$\begin{aligned}\sum_{t=0}^{T-1} \mathbb{E}[\mathcal{L}_{LB}(\theta_{t+1}) - \mathcal{L}_{LB}(\theta^*)] &= \mathbb{E}[\sum_{t=1}^{T} \mathcal{L}_{LB}(\theta_t)] - T\mathcal{L}_{LB}(\theta^*) \\ &\geq T\mathbb{E}[\mathcal{L}_{LB}(\overline{\theta_T})] - T\mathcal{L}_{LB}(\theta^*).\end{aligned}$$

Combining the above equations we get

$$\mathbb{E}[\mathcal{L}_{LB}(\overline{\theta_T})] - \mathcal{L}_{LB}(\theta^*) \leq \frac{\|\theta_0 - \theta^*\|_2^2}{2\eta T} + \frac{\eta\mathtt{Cov}^2}{M}.$$

This completes the proof.

D. Appendix for Chapter 6

D.1 Proof of Lemma 6.3.2

Proof. Because all operands are binary, a symbolic expression tree of l nodes has $\frac{l+1}{2}$ leaves and $\frac{l-1}{2}$ internal nodes. The number of binary trees of $\frac{l-1}{2}$ internal nodes is given by the Cantalan number $C_{(l-1)/2} = \frac{l-1}{l+1}\binom{l-1}{(l-1)/2}$, which asymptotically scales at $\frac{2^{l-1}}{(\frac{l-1}{2})^{3/2}\sqrt{\pi}}$. A symbolic expression replaces each internal node of a binary tree with an operand and replaces each leaf with either a constant or one of the input variables. Because there are o operands and m input variables, the total number of different symbolic expression trees involving l nodes is given by:

$$\begin{aligned} A(l) = C_{(l-1)/2}(m+1)^{\frac{l+1}{2}} o^{\frac{l-1}{2}} &= \frac{l-1}{l+1}\binom{l-1}{(l-1)/2}(m+1)^{\frac{l+1}{2}} o^{\frac{l-1}{2}} \\ &\sim \frac{(4(m+1)o)^{\frac{l-1}{2}}}{\left(\frac{l-1}{2}\right)^{3/2}}. \end{aligned} \tag{D.1}$$

Hence, the total number of trees up to l nodes is:

$$S(l) = \sum_{i=0}^{(l-1)/2} A(2i+1) \sim \sum_{i=0}^{(l-1)/2} \frac{(4(m+1)o)^i}{i^{3/2}}. \tag{D.2}$$

When i is sufficiently large,

$$(4(m+1)o)^{i/2} \leq \frac{(4(m+1)o)^i}{i^{3/2}} \leq (4(m+1)o)^i. \tag{D.3}$$

Therefore,

$$S(l) \leq \sum_{i=0}^{(l-1)/2} (4(m+1)o)^i \in \mathcal{O}((4(m+1)o)^{(l-1)/2}),$$

and

$$S(l) \geq (4(m+1)o)^{(l-1)/2}/((l-1)/2)^{3/2} \geq (4(m+1)o)^{(l-1)/4},$$

which implies

$$S(l) \in \Omega((4(m+1)o)^{\frac{l-1}{4}}).$$

D.2 Experiment Settings

D.2.1 Dataset Configuration

Synthesised datasets We generated several families of multiple-variable ground-truth expressions, and use these expressions to generate the datasets for control variable experiments $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$. We label the datasets by 1) the set of mathematical operators that can be included in the ground-truth expression that generates the dataset; 2) the number of independent variables m; 3) the number of single terms of the ground-truth expression; 4) the number of cross terms of the ground-truth expression. We write items 2), 3), and 4) into a tuple in various charts and tables to represent the dataset.

These four characteristics of the ground-truth expression determine the complexity of learning the ground-truth expression from the dataset. To give an example, one ground-truth expression that generates a dataset labeled by the "$\texttt{inv}$, $+, -, \times$" operators with configuration $(2, 1, 1)$ can be:

$$0.4967 - \frac{0.6824}{x_1} - \frac{0.7346x_1}{x_0}$$

This expression contains two variables x_1, x_2, a cross term $\frac{x_1}{x_0}$ with a constant coefficient 0.7346, a single term $1/x_1$ with a constant coefficient 0.6824, and a constant 0.4967. We give more examples of such ground-truth expressions in Table D.1.

Dataset Configs	Example expression
(2,1,1)	$0.497 - 0.682/x_1 - 0.735x_1/x_0$
(3,2,2)	$-0.603x_0x_1 + 0.744x_0 + 0.09x_1/x_2 + 0.562 + 0.582/x_2$
(a) Datasets containing operands $\{\texttt{inv}, +, -, \times\}$.	
(2,1,1)	$0.259x_0 \sin(x_1) + 0.197x_1 - 0.750$
(3,2,2)	$-0.095x_0x_2 + 0.012x_2 \sin(x_1) - 0.576x_2 - 0.214 \cos(x_0) - 0.625$
(b) Datasets containing operands $\{\sin, \cos, +, -, \times\}$.	
(2,1,1)	$0.7272 \sin(x_0) - 0.3866 + 0.183/x_0$
(3,2,2)	$0.7167x_0/x_2 - 0.0632x_1 + 0.2746x_2 \cos(x_1) - 0.7293 + 0.0627/x_2$
(c) Datasets containing operands $\{\sin, \cos, \texttt{inv}, +, -, \times\}$.	

Table D.1. Example expressions used in our experiments with different dataset configurations and the set of operands.

Noisy Dataset Setting. In real scientific experiments, the datasets often contain noises. We add Gaussian noise $\mathcal{N}(0, \sigma^2)$ to the output y in the dataset and control the noise rate by varying the values of σ in $\{0.02, 0.04, 0.08, 0.1, 0.12, 0.14\}$.

D.2.2 Evaluation Metrics

Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ generated from the ground-truth expression ϕ, where n indicates the number of test samples. The empirical variance of the target values σ_y is defined as:

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \frac{\sum_{i=1}^n y_i}{n}\right)^2 \tag{D.4}$$

During training and testing, we measure the goodness-of-fit of a candidate expression $\bar{\phi}$, by evaluating the following Mean-square error (MSE), Negative Mean-square error (MSE), normalized Mean-square error (NMSE), normalized root Mean-squared error (NRMSE), Invese normalized root Mean-squared error (InvNRMSE):

$$\begin{aligned}
\mathrm{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2, \\
\mathrm{NegMSE} &= -\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2, \\
\mathrm{NMSE} &= \frac{1}{n}\frac{\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2}{\sigma_y^2}, \\
\mathrm{RMSE} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2}, \\
\mathrm{NRMSE} &= \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2} \\
\mathrm{InvNRMSE} &= \frac{1}{\frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2}}
\end{aligned} \tag{D.5}$$

D.2.3 Baselines implementation

GP The implementation is based on the deap package[1]. However, we re-implemented the code following the concept of their package.

[1] ↑https://github.com/DEAP/deap

	CVGP	GP	DSR	PQT	GPMeld	Eureqa
Reward function	NegMSE	NegMSE	InvNRMSE	InvNRMSE	InvNRMSE	NegRMSE
Training set size	25, 600	25, 600	50, 000	50, 000	50, 000	50, 000
Testing set size	256	25, 600	256	256	256	256
Batch size	256	256	1024	1024	1024	*N/A*
#CPUs for training	1	1	4	4	4	1
ϵ-risk-seeking policy	N/A	0.02	0.02	0.02	N/A	N/A
#genetic generations	100	100	N/A	N/A	60	10,000
#Hall of fame	10	10	25	25	25	N/A
Mutation Probability	0.5	0.5	0.5	N/A	N/A	N/A
Mating Probability	0.5	0.5	0.5	N/A	N/A	N/A
training time (hours)	$\sim$0.5	$\sim$0.5	$\sim$0.5	$\sim$0.5	$\sim$6	$\sim$0.5

Table D.2. Major hyper-parameters settings for all the algorithms considered in the experiment.

CVGP Our method is implemented on top of the **GP** following Algorithm 5.

Eureqa This algorithm is currently maintained by the DataRobot webiste[2]. We use the python API provided [3] to send the training dataset to the DataRobot website and collect the predicted expression after 30 minutes. This website only allows us to execute their program under a limited budget. Due to budgetary constraints, we were only able to test the datasets for the noiseless settings. For the Eureqa method, the fitness measure function is negative RMSE. We generated large datasets of size 10^5 in training each benchmark.

DSR, PQT, GPMeld These algorithms are evaluated based on an implementation in [4]. For every ground-truth expression, we generate a dataset of sizes 10^5 training samples. Then we execute all these baselines on the dataset with the configurations listed in Table D.2. For the four baselines (*i.e.*, PQT, VPG, DSR, GPMeld), the reward function is INV-NRMSE, which is defined as $\frac{1}{1+\text{NRMSE}}$.

Note that Wolfram was not considered in this research, because the current "`FindFormula`" function in the Wolfram language only supports searching for single variable expressions.

[2] ↑https://docs.datarobot.com/en/docs/modeling/analyze-models/describe/eureqa.html
[3] ↑https://pypi.org/project/datarobot/
[4] ↑https://github.com/brendenpetersen/deep-symbolic-optimization

D.2.4 Hyper-parameter Configurations

We list the major hyper-parameter setting for all the algorithms in Table D.2. Note that if we use the default parameter settings, the GPMeld algorithm takes more than 1 day to train on one dataset. Because of such slow performance, we cut the number of genetic programming generations in GPMeld by half to ensure fair comparisons with other approaches.

D.3 Extended Experimental Analysis

Quantile of NMSE Metric. In Figure D.1, we show the five-number summary for the NMSE metric in Table 6.1, *i.e.*, the minimum, 25% quartile, median, 75% quartile, and maximum. In Figure D.2, we show the five-number summary for the NMSE metric in Table 6.2. For the datasets considered, our CVGP shows a consistent improvement over all the baselines.

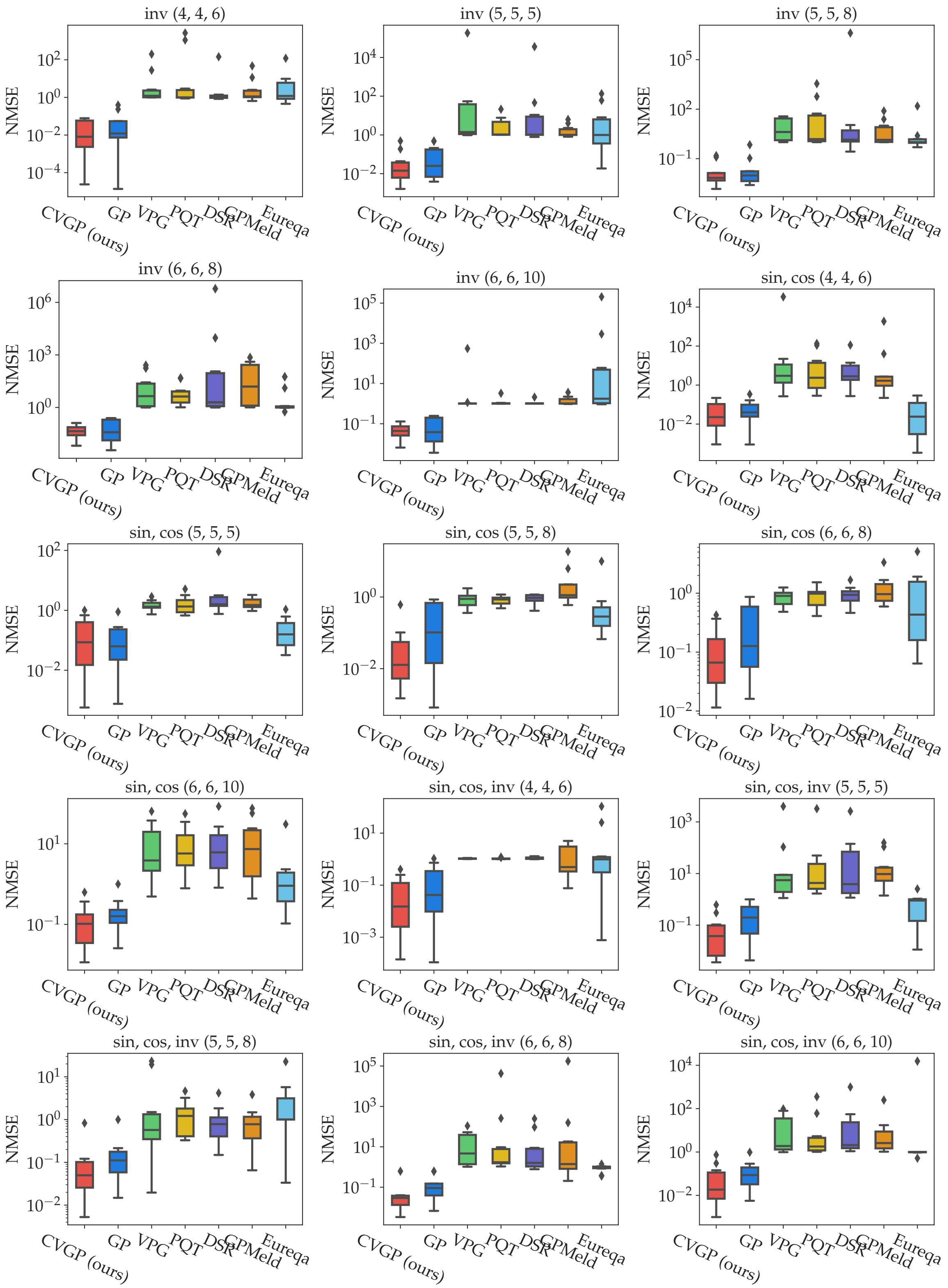


Figure D.1. Quartiles of NMSE values of all the methods over several *noiseless* datasets. Our CVGP shows a consistent improvement over all the baselines considered, among all the datasets.

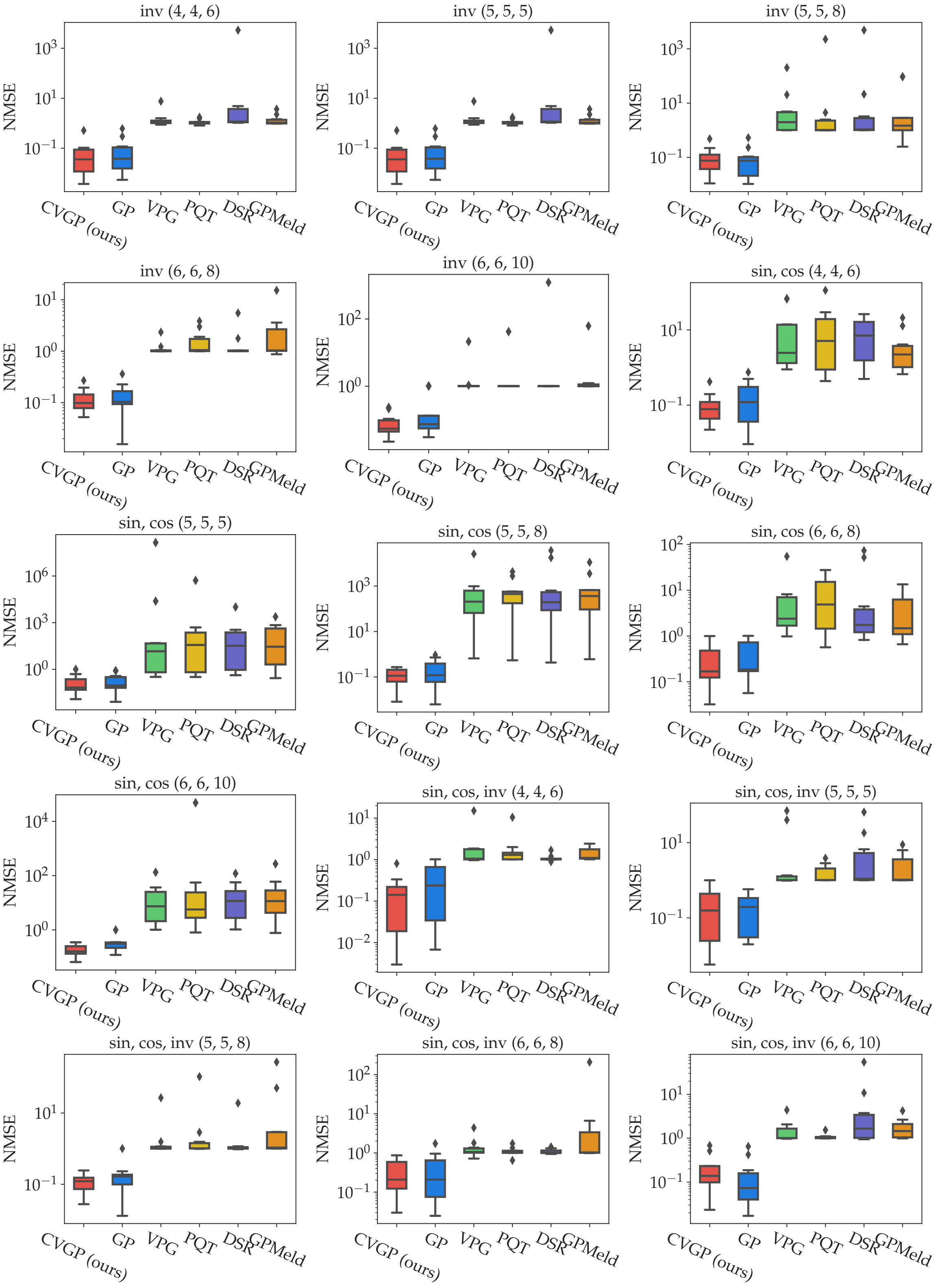


Figure D.2. Quartiles of NMSE values of all the methods over several *noisy* datasets. Our CVGP shows a consistent improvement over all the baselines considered, among all the datasets.

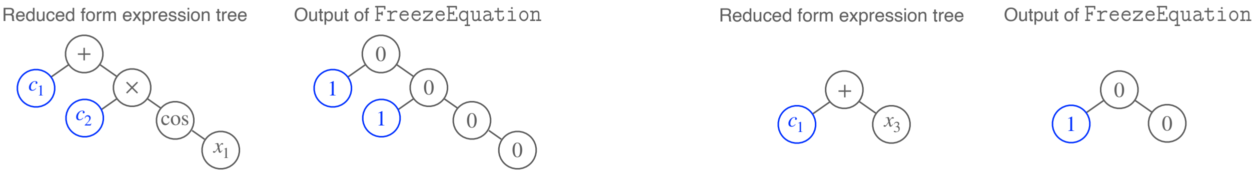
E. Appendix for Chapter 7

E.1 Implementation

Please find our code repository. It contains 1) the implementation of our Racing-CVGP method, 2) the list of datasets, and 3) the implementation of several baseline algorithms.

E.2 Genetic Programming Algorithm in Racing-CVGP

For the `FreezeEquation` function used in Algorithm 7, we use Figure E.1 to demonstrate the output. The `FreezeEquation` function will reduce the size of candidate nodes to be edited in the GP algorithms and increase the probability of finding expression trees with close-to-zero fitness scores.



(a) controlled variable $\mathbf{x}_c = \{x_2, x_3\}$.

(b) controlled variable $\mathbf{x}_c = \{x_1, x_2\}$.

Figure E.1. Visualization of the `FreezeEquation` function. 0 implies the node is non-editable and 1 implies the node is editable, by the GP algorithm. The `FreezeEquation` function will increase the probability of finding expression trees with close-to-zero fitness scores.

The pipeline of Genetic Programming in our Racing-CVGP framework is presented in Algorithm 8. It is a minimally modified genetic programming algorithm for symbolic regression.

For the `Mutate` step, the algorithm will apply one of the operations for the chosen expression tree:

1. Find a leaf node that is not frozen and then replace the node with a generate a full expression tree of maximum depth involving variables only in $\mathbf{x} \setminus \mathbf{x}_c$.

2. Find a node and replace it with a node of the same arity. Here arity is the number of operands taken by an operator. For example, the arity of binary operators $\{+, -, \times, \div\}$ is 2 and the arity of unary operators $\{\sin, \cos, \log, \exp\}$ is 1.

3. Inserts a node at a random position, and the original subtree at the location becomes one of its subtrees.
4. Delete a node that is not frozen, use one of its children to replace its position.

For the `Mate` step, we will pick two expressions ϕ_l, ϕ_j from the population pool $\mathcal{P}$ that has the same control variables $\mathbf{x}_{c,l} = \mathbf{x}_{c,j}$. Then we exchange two randomly chosen subtrees in the expressions. Because applying mating over two expressions with different control variables does not necessarily result in two better expressions.

Algorithm 8: $\texttt{GP}(\mathcal{P}, \texttt{DataOracle}, K, M, \texttt{\#Gen}, \texttt{\#Hof}, P_{mu}, P_{ma}, O_p)$

```
Input: Initial GP Pool $\mathcal{P}$; DataOracle; # control variable trials $K$; GP pool size
       $N_p$; # of generations #Gen; #expressions in hall-of-fame set #Hof; mutation
       node library $O_p$.
Output: The GP pool and the updated hall of fame set.
Parameters: mutate probability $P_{mu}$; mate probability $P_{ma}$;
for $i \leftarrow 1$ to #Gen do
    $\mathcal{P}_{new} \leftarrow \emptyset$; for $\langle \phi_{new}, \pi, \mathbf{x}_c \rangle \in \mathcal{P}$ do
        if with probability $P_{mu}$ then
            //mutation;
            $\phi_{new} \leftarrow \texttt{Mutate}(\phi_{new}, O_p, \mathbf{x} \setminus \mathbf{x}_c)$;
            $\{\mathcal{D}_k\}_{k=1}^{K} \leftarrow \texttt{DataOracle}(\phi_{new}, \mathbf{x}_c, K)$;
            $\mathbf{o}, \mathbf{c} \leftarrow \texttt{Optimize}(\phi_{new}, \{\mathcal{D}_k\}_{k=1}^{K})$;
        $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\langle \phi_{new}, \mathbf{o}, \mathbf{c}, \pi, \mathbf{x}_c \rangle\}$;
    $\mathcal{P} \leftarrow \mathcal{P}_{new}$; $\mathcal{P}_{new} \leftarrow \emptyset$;
    for $\langle \phi_l, \pi_l, \mathbf{x}_{c,l} \rangle, \langle \phi_j, \pi_j, \mathbf{x}_{c,j} \rangle \in \mathcal{P}$ do
        //mating;
        if with probability $P_{ma}$ and $\mathbf{x}_{c,l} = \mathbf{x}_{c,j}$ then
            // pick two expressions with the same $\mathbf{x}_c$;
            $\phi_l, \phi_j \leftarrow \texttt{Mate}(\phi_l, \phi_j)$;
            $\{\mathcal{D}_k\}_{k=1}^{K} \leftarrow \texttt{DataOracle}(\phi_l, \mathbf{x}_{c,l}, K)$;
            $\mathbf{o}_l, \mathbf{c}_l \leftarrow \texttt{Optimize}(\phi_l, \{\mathcal{D}_k\}_{k=1}^{K})$;
            $\{\mathcal{D}_k\}_{k=1}^{K} \leftarrow \texttt{DataOracle}(\phi_j, \mathbf{x}_{c,j}, K)$;
            $\mathbf{o}_j, \mathbf{c}_j \leftarrow \texttt{Optimize}(\phi_j, \{\mathcal{D}_k\}_{k=1}^{K})$;
        $\mathcal{P}_{new} \leftarrow \mathcal{P}_{new} \cup \{\langle \phi_l, \mathbf{o}_l, \mathbf{c}_l, \pi_l, \mathbf{x}_{c,l} \rangle, \langle \phi_j, \mathbf{o}, \mathbf{c}_j, \pi_j, \mathbf{x}_{c,j} \rangle\}$;
    $\mathcal{H} \leftarrow \texttt{TopK}(\mathcal{P}_{new} \cup \mathcal{H}, K = \texttt{\#Hof})$; ;        // Update the hall of fame set
return GP pool and hall-of-fame $\mathcal{P}_{new}, \mathcal{H}$.
```

E.3 Experiment Settings

E.3.1 Dataset Configuration

Livermore2 Dataset The list of Livermore2 datasets is summarized at[1]. In Tables E.1, E.2, E.3, we details the exact equation of Livermore2 [180]. The operator set for each expression is available in the codebase.

The list of Feynamn datasets is collected from[2]. In Tables E.4, E.5. We only use a subset of the expressions in the original Feynman dataset. The challenging part for this dataset is the ranges of input variables vary greatly. For example, in one equation with ID "ICh34Eq8" the ranges of all the variables are:

$$x_1 \in (10^{-11}, 10^{-9}), x_2 \in (10^5, 10^7), x_3 \in (10, 10^3), x_4 \in (10^9, 10^{11}) \tag{E.1}$$

In comparison, the input ranges of the Livermore2 dataset are $x_i \in (0.01, 10)$. The operator set for each expression is available in the codebase.

E.3.2 Evaluation Metrics

We mainly consider two evaluation criteria for the learning algorithms tested in our work: 1) the goodness-of-fit measure and 2) the total running time of the learning algorithms. The goodness-of-fit indicates how well the learning algorithms perform in discovering unknown symbolic expressions. Given a testing dataset $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ generated from the ground-truth expression ϕ, we measure the goodness-of-fit of a predicted expression $\bar{\phi}$, by evaluating the mean-squared-error (MSE) and normalized-mean-squared-error (NMSE):

$$\begin{aligned}
\text{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2, & \text{NMSE} &= \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2}{\sigma_y^2} \\
\text{RMSE} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2}, & \text{NRMSE} &= \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{\phi}(\mathbf{x}_i))^2}
\end{aligned} \tag{E.2}$$

[1] ↑https://github.com/brendenpetersen/deep-symbolic-optimization/blob/master/dso/dso/task/regression/benchmarks.csv

[2] ↑https://github.com/omron-sinicx/srsd-benchmark/blob/main/datasets/feynman.py

Table E.1. Detailed equation in Livermore2 datasets (part-1).

	Livermore2 ($n = 4$)	
Equation ID	n	Expression
Vars4-1	4	$x_1 - x_2x_3 - x_2 - x_4$
Vars4-2	4	$x_1\sqrt{x_2}x_4/x_3$
Vars4-3	4	$2x_1 + x_4 - 0.01 + x_3/x_2$
Vars4-4	4	$x_1 - x_4 - (-x_1 + \sin(x_1))^4/(x_1^8x_2^2x_3^2)$
Vars4-5	4	$x_1 + \sin(x_2/(x_1x_2^2x_4^2(-3.22x_2x_4^2 + 13.91x_2x_4 + x_3)/2 + x_2))^2$
Vars4-6	4	$(-x_1 - 0.54\exp(x_1 sqrt(x_4 + \cos(x_2))\exp(-2x_1)))/x_3$
Vars4-7	4	$x_1 + \cos(x_2/\log(x_2^2x_3 + x_4))$
Vars4-8	4	$x_1(x_1 + x_4 + \sin((-x_1\exp(\exp(x_3)) + x_2)/(-4.47x_1^2x_3 + 8.31x_3^3 + 5.27x_3^2))) - x_1$
Vars4-9	4	$x_1 - x_4 + \cos(x_1(x_1 + x_2)(x_1^2x_2 + x_3) + x_3)$
Vars4-10	4	$x_1 + (x_1(x_4 + (\sqrt{x_2} - \sin(x_3))/x_3))^{1/4}$
Vars4-11	4	$2x_1 + x_2(x_1 + \sin(x_2x_3)) + \sin(2/x_4)$
Vars4-12	4	$x_1x_2 + 16.97x_3 - x_4$
Vars4-13	4	$x_4(-x_3 - \sin(x_1^2 - x_1 + x_2))$
Vars4-14	4	$x_1 + \cos(x_2^2(-x_2 + x_3 + 3.23) + x_4)$
Vars4-15	4	$x_1(x_2 + \log(x_3 + x_4 + \exp(x_2^2) - 0.28/x_1)) - x_3 - \frac{x_4}{2x_1x_3}$
Vars4-16	4	$x_3(-x_4 + 1.81/x_3) + \sqrt{x_2(-x_1^2\exp(x_2) - x_2)} - 2.34x_4/x_1$
Vars4-17	4	$x_1^2 - x_2 - x_3^2 - x_4$
Vars4-18	4	$x_1 + \sin(2x_2 + x_3 - x_4\exp(x_1) + 2.96\sqrt{-0.36x_2^3 + x_2x_3^2 + 0.94} + \log((-x_1 + x_2)\log(x_2)))$
Vars4-19	4	$(x_1^3x_2 - 2.86x_1 + x_4)/x_3$
Vars4-20	4	$x_1 + x_2 + 6.21 + 1/(x_3x_4 + x_3 + 2.08)$
Vars4-21	4	$x_1(x_2 - x_3 + x_4) + x_4$
Vars4-22	4	$x_1 - x_2x_3 + x_2\exp(x_1) - x_4$
Vars4-23	4	$-x_1/x_2 - 2.23x_2x_3 + x_2 - 2.23x_3/\sqrt{x_4} - 2.23\sqrt{x_4} + \log(x_1)$
Vars4-24	4	$-4.81\log(\sqrt{x_1\sqrt{\log(x_1(x_1x_2 + x_1 + x_4 + log(x_3))))}})$
Vars4-25	4	$0.38 + (-x_1/x_4 + cos(2x_1x_3/(x_4(x_1 + x_2x_3)))/x_4)/x_2$

where the empirical variance $\sigma_y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2}$. Note that the coefficient of determination (R^2) metric [299, 300] is equal to $(1 - \text{NMSE})$ and therefore omitted in the experiments.

E.3.3 Baseline Implementation

Racing-CVGP Our method is implemented on top of the **GP** following Algorithm 6. See the codebase for more details.

GP The implementation is based on the version in "baselines" of DSO package[3]. However, we re-implemented the code following the concept of their package.

CVGP The implementation is available at[4].

[3] ↑https://github.com/DEAP/deap

[4] ↑https://github.com/jiangnanhugo/cvgp

Table E.2. Detailed equation in Livermore2 datasets (part-2).

	Livermore2 ($n = 5$)	
Equation ID	n	Expression
Vars5-1	5	$-x_1 + x_2 - x_3 + x_4 - x_5 - 4.75$
Vars5-2	5	$x_3\left(x_1 + x_5 + \frac{0.27}{(x_3^2 + \frac{(x_2+x_4)}{(x_1x_2+x_2)})}\right)$
Vars5-3	5	$2x_1x_2x_3 + x_5 - \sin(x_1\log(x_2) - x_1 + x_4)$
Vars5-4	5	$x_2 + x_3x_4 + x_5^2 + \sin(x_1)$
Vars5-5	5	$x_5 + 0.36\sqrt{(\log(x_1x_2 + x_3 + \log(x_2 + x_4)))}$
Vars5-6	5	$x_1x_4 + x_1 + x_2 + x_5 + \sqrt{(0.08x_1/(x_3x_5) + x_3)}$
Vars5-7	5	$x_1x_5 + \sqrt{(x_1x_2\cos(x_1) - x_1/(x_2 + x_3 + x_4 + 8.05))}$
Vars5-8	5	$\sqrt{(x_2)}x_3 - x_4 - 0.07(x_1 + (x_1 - x_2)\sqrt{(x_2 - 0.99)})\cos(x_5)$
Vars5-9	5	$x_1(x_3 + (x_1 + x_2)/(x_2x_4 + x_5))$
Vars5-10	5	$x_1/(x_4(-0.25x_1x_3x_4 + x_2 - 8.43x_4x_5)\sin(x_3 + \log(x_2))) + x_4x_5$
Vars5-11	5	$-x_4^2 + \sqrt{\frac{x_1(x_3+x_5)-x_2+x_5}{x_3}} + 0.47\sqrt{x_3\frac{x_1-\sqrt{x_2}+x_2}{x_2}}$
Vars5-12	5	$x_1\left(x_2 - \frac{1}{x_3(x_4+x_5)}\right)$
Vars5-13	5	$\sqrt{(x_1(x_5(x_2 - 1.52) - \cos(4.03x_3 + x_4)))}$
Vars5-14	5	$-x_1/(x_2x_5) + \cos(x_1x_3x_4\exp(-x_2))$
Vars5-15	5	$-x_4 + \log(x_1/\log(11.06x_2x_5^2) + x_3) - \cos(x_2 + x_5 + \sqrt{(x_2x_5)})$
Vars5-16	5	$x_2 + 0.33x_5(x_1/(x_1^2 + x_2) + x_3x_4^(3/2))$
Vars5-17	5	$x_1 - \sin(x_2) + \sin(x_3) - \cos(-x_2 + \sqrt{(x_4)} + x_5) + 0.78$
Vars5-18	5	$x_1x_2 - x_4 - (\sqrt{(x_3^2/(x_1(x_3 + x_4)))} - 1.13/x_3)/x_5$
Vars5-19	5	$4.53x_1x_2 + x_1 - x_1\cos(\sqrt{(x_2)})/x_2 - x_3 - x_4 - x_5$
Vars5-20	5	$-\exp(x_1 + x_5) + \sin(x_1 - 4.81)/(0.21(x_5 - \log(x_3 + x_4) - \exp(x_5))/x_2)$
Vars5-21	5	$\sqrt{(x_4)}(2x_1 + \cos(x_1(x_3x_4\exp(x_1x_2) + x_3 - \log(x_3) - 3.49))/x_5)$
Vars5-22	5	$-x_1 - x_2 + x_3 + \sqrt{(x_1 - x_2(\sin(x_3) - \log(x_1x_5/(x_2^2 + x_4))/x_4))} - 0.73$
Vars5-23	5	$x_1(x_2/(x_3 + \sqrt{(x_2(x_4 + x_5))}(-x_3 + x_4)) - x_5)$
Vars5-24	5	$-x_2x_5 + \sqrt{(x_1 + x_2(-x_1 + x_4\cos(\sqrt{x_3} + x_3) - \frac{x_2+7.84x_3^2x_5}{x_5}) + \frac{x_2}{x_3}}$
Vars5-25	5	$x_1 + \log(x_1(-3.57x_1^2x_2 + x_1 + x_2 + x_3\log(-x_1x_4\sin(x_3)/x_5 + x_3)))$

Eureqa This algorithm is currently maintained by the DataRobot webiste[5]. We use the python API provided [6] to send the training dataset to the DataRobot website and collect the predicted expression after 30 minutes. This website only allows us to execute their

[5] ↑https://docs.datarobot.com/en/docs/modeling/analyze-models/describe/eureqa.html

[6] ↑https://pypi.org/project/datarobot/

Table E.3. Detailed equation in other small-scale datasets (part-3). n stands for the number of maximum variables.

	Livermore2 ($n = 6$)	
Equation ID	n	Expression
Vars6-1	6	$x_1 - x_6 + (x_1 + x_4 + x_5)\sqrt{(x_1^2 + x_2 - x_3)}$
Vars6-2	6	$x_1(2x_2 + x_2/x_3 + x_4 + \log(x_1x_5x_6))$
Vars6-3	6	$\sqrt{(x_2 + x_5 - x_6 + x_3^4x_4^4/(x_1x_2^4))}$
Vars6-4	6	$x_1(x_2(x_1^2 + x_1) - x_2 + x_3^2 - x_3 - x_5 - x_6 - \sin(x_4) - \cos(x_4))^2$
Vars6-5	6	$x_2\sqrt{(x_1x_2)(x_1x_3 - x_3 - x_4) + x_5 + x_6}$
Vars6-6	6	$(x_1/(x_2x_3 + \log(\cos(x_1))^2) - x_2x_4 + \sin((x_2x_4 + x_5)/x_6) + \cos(x_3))\log(x_1)$
Vars6-7	6	$x_1\sqrt{(x_1 - x_6^2 + \sin((x_1\exp(-x_2) - x_4(x_2 + x_3^2))/(x_2 + x_5)))}$
Vars6-8	6	$x_1 + x_2^2 + 0.34x_3x_5 - x_4 + x_6$
Vars6-9	6	$x_4(x_1 + \exp(13.28x_3^2x_6 - x_5^2\log(x_2^4))/(x_1x_3 - x_2^2 + x_2 - x_6 - \log(x_3)))$
Vars6-10	6	$x_1 + 61.36x_2^6 + x_2/(x_1x_3(x_4 - \cos(x_4(2x_1x_2x_6/x_5 + x_5))))$
Vars6-11	6	$(x_1 + x_1/(x_2 + x_4(8.13x_1^2x_6 + x_1x_2x_3 + 2x_2 + x_5 + x_6)))^2$
Vars6-12	6	$(\sqrt{2}\sqrt{(x_1)} - x_2 - x_3/\sqrt{(x_4(8.29x_1x_3^2 + x_1x_5) + x_4 + x_6)})/x_6$
Vars6-13	6	$x_1 + x_5 + 0.21\sqrt{(x_1/(x_2^2x_3^2\sqrt{(x_6)}(\sqrt{(x_3)} + x_3 + 2x_6 + (x_2 + x_4 + x_5)/x_5)))}$
Vars6-14	6	$-2.07x_6 + \log(x_2 - x_6 - \sqrt{(x_3(x_5 + \log(-x_1 + x_5 + 1))/x_4)})$
Vars6-15	6	$x_1(x_1 + \cos(x_2^2x_3x_4(x_5 - 0.43x_6^2)))/x_4$
Vars6-16	6	$-\sqrt{(x_1)} - x_1 + x_2 - x_4 - x_5 - \sqrt{(x_6/x_3)} - 3.26$
Vars6-17	6	$x_1/(x_2x_4(-x_5 + \log(x_6^2\cos(2x_2 + x_3^2 - x_4)^2))(129.28x_1^2x_2^2 + x_3))$
Vars6-18	6	$\sqrt{x_5}(2x_1 + \cos(x_1(x_3x_4\exp(x_1x_2) + x_3 - \log(x_3) - 3.49))/x_6)$
Vars6-19	6	$x_1 + x_2 + x_3 + 0.84\sqrt{(-x_3x_6 + x_4 - x_5 + \sqrt{((x_2 + \log(x_3 + \exp(x_2)))/(x_2 - x_4))})}$
Vars6-20	6	$(x_1 - 0.97x_1/(x_5 - x_6(x_1^{(}3/2)x_4 + x_6)) - x_2 + x_3 + \sin(x_1^2)/x_1)^2$
Vars6-21	6	$x_1 + x_3 + (x_1 + \sin(-3.47x_2\log(x_6)/x_5 + x_4 + 25.56\exp(x_5^2)/x_2)^2)\sin(x_2)$
Vars6-22	6	$x_1 + (x_4 + \sin(-0.22(x_3 - x_4 + 1.0))\cos(x_6))\cos(x_2 + 2.27x_5)$
Vars6-23	6	$x_1 + x_4 + \log(x_1^2 + x_1(-x_6 + 1.88\sqrt{(0.71x_1 + x_2 + 0.28(x_3 - x_4/x_5))}))$
Vars6-24	6	$-0.59(1.42(0.24x_2 + \sqrt{x_3}/(x_6\sqrt{(-x_4 + x_5)}))^{(}1/4) + \sin(x_1))/x_6$
Vars6-25	6	$x_1 - x_2^2 - x_3 + x_5\cos(x_3) + x_5 + x_6 - 2.19\sqrt{(-x_3 - 0.44/x_4)}$

program under a limited budget. Due to budgetary constraints, we were only able to test the datasets for the noiseless settings. For the Eureqa method, the fitness measure function is negative RMSE. We generated large datasets of size 10^5 in training each benchmark.

DSR, PQT, GPMeld These algorithms are evaluated based on an implementation in[7]. For every ground-truth expression, we generate a dataset of sizes 10^5 training samples. Then we execute all these baselines on the dataset with the configurations listed in Table E.6.

[7] ↑https://github.com/brendenpetersen/deep-symbolic-optimization

Table E.4. Detailed equations in Feynman datasets ($n = 4$). n stands for the number of maximum variables.

	Feynman ($n = 4$)	
Equation ID	n	Expression
I.8.14	4	$\sqrt{(x_0 - x_1)^2 + (x_2 - x_3)^2}$
I.13.4	4	$0.5x_0(x_1^2 + x_2^2 + x_3^2)$
I.13.12	4	$6.6743e - 11x_0x_1(-1/x_3 + 1/x_2)$
I.18.4	4	$(x_0x_1 + x_2x_3)/(x_0 + x_2)$
I.18.16	4	$x_0x_1x_2\sin(x_3)$
I.24.6	4	$0.25x_0x_3^2(x_1^2 + x_2^2)$
I.29.16	4	$\sqrt{x_0^2 + 2x_0x_1\cos(x_2 - x_3) + x_1^2}$
I.32.17	4	$0.0035\pi x_0^2x_1^2x_2^4/(x_2^2 - x_3^2)^2$
I.34.8	4	$x_0x_1x_2/x_3$
I.40.1	4	$x_0\exp(-7.10292768111229e + 23x_1x_2/x_3)$
I.43.16	4	$x_0x_1x_2/x_3$
I.44.4	4	$1.38e - 23x_0x_1\log(x_2/x_3)$
I.50.26	4	$x_0(x_3\cos(x_1x_2)^2 + \cos(x_1x_2))$
II.11.20	4	$2.41e + 22x_0x_1^2x_2/x_3$
II.34.11	4	$x_0x_1x_2/(2x_3)$
II.35.18	4	$x_0/(\exp(7.24e + 22x_1x_2/x_3) + \exp(-7.24e + 22x_1x_2/x_3))$
II.35.21	4	$x_0x_1\tanh(7.24e + 22x_1x_2/x_3)$
II.38.3	4	$x_0x_1x_2/x_3$
III.10.19	4	$x_0\sqrt{x_1^2 + x_2^2 + x_3^2}$
III.14.14	4	$x_0(\exp(7.24e + 22x_1x_2/x_3) - 1)$
III.21.20	4	$-x_0x_1x_2/x_3$
BONUS.1	4	$3.32e - 57x_0^2x_1^2/(x_2^2\sin(x_3/2)^4)$
BONUS.3	4	$x_0(1 - x_1^2)/(x_1\cos(x_2 - x_3) + 1)$
BONUS.11	4	$4x_0\sin(x_1/2)^2\sin(x_2x_3/2)^2/(x_1^2\sin(x_3/2)^2)$
BONUS.19	4	$-1872855580.36049(8.07e + 33x_0/x_1^2 + 8.98e + 16x_2^2(1 - 2x_3))/\pi$

The official implementation of Symbolic Physics Learner (SPL) [240][8] does not support open constants. Thus SPL is not considered in this research.

For the four baselines (*i.e.*, PQT, VPG, DSR, GPMeld), the reward function is INV-NRMSE, which is defined as $\frac{1}{1+\text{NRMSE}}$.

[8] ↑https://github.com/isds-neu/SymbolicPhysicsLearner

Table E.5. Detailed equations in Feynman datasets ($n = 5$). n stands for the number of maximum variables.

		Feynman ($n = 5$)
Equation ID	n	Expression
I.12.11	5	$x_0(x_1 + x_2x_3\sin(x_4))$
II.2.42	5	$x_0x_3(x_1 - x_2)/x_4$
II.6.15a	5	$84707476846.623x_0x_1\sqrt{(x_3^2 + x_4^2)}/(\pi x_2^5)$
II.11.3	5	$x_0x_1/(x_2(x_3^2 - x_4^2))$
II.11.17	5	$x_0(7.24e + 22x_1x_2\cos(x_3)/x_4 + 1)$
II.36.38	5	$7.24e + 22x_0x_1/x_2 + 9.10e + 16x_0x_3x_4/x_2$
III.9.52	5	$1.21e + 34\pi x_0x_1\sin(x_2(x_3 - x_4)/2)^2/(x_2(x_3 - x_4)^2)$
bonus.4	5	$\sqrt{2}\sqrt{(x_1 - x_2 - x_3^2/(2x_0x_4^2))/x_0}$
bonus.12	5	$x_0(-x_0x_2^3x_4/(x_2^2 - x_4^2)^2 + 4pix_1x_3x_4)/(4\pi x_1x_2^2)$
bonus.13	5	$x_1/(4\pi x_0\sqrt{x_2^2 - 2x_2x_3\cos(x_4) + x_3^2})$
bonus.14	5	$x_0(-x_2 + x_3^3(x_4 - 1)/(x_2^2(x_4 + 2)))\cos(x_1)$
bonus.16	5	$x_1x_4 + 8.98e + 16\sqrt{x_3^2 + 1.11e - 17(x_0 - x_1x_2)^2}$

E.3.4 Optimizers

We consider several optimizers CG [301] Nelder-Mead [302], BFGS [243], Basin Hopping [244], SHGO [303], Dual Annealing [304]. The list of local and global optimizers shown in Figure E.6 are from Scipy library[9].

E.3.5 Hyper-parameter Configuration

We list the major hyper-parameter setting for all the algorithms in Table E.6. Note that if we use the default parameter settings, the GPMeld algorithm takes more than 1 day to train on one dataset. Because of such slow performance, we cut the number of genetic programming generations in GPMeld by half to ensure fair comparisons with other approaches.

[9] ↑https://docs.scipy.org/doc/scipy/reference/optimize.html

Table E.6. Major hyper-parameters settings for all the algorithms considered in the experiment.

	Racing-CVGP	GP	DSR	PQT	GPMeld	Eureqa
Reward function	NegMSE	NegMSE	InvNRMSE	InvNRMSE	InvNRMSE	NegRMSE
Training set size	25, 600	25, 600	50, 000	50, 000	50, 000	50, 000
Testing set size	256	25, 600	256	256	256	256
Batch size	256	256	1024	1024	1024	*N/A*
#CPUs	1	1	4	4	4	1
ϵ-risk-seeking policy	N/A	0.02	0.02	0.02	N/A	N/A
#generations	100	100	N/A	N/A	60	10,000
#Hall of fame	10	10	25	25	25	N/A
Mutation Prob.	0.5	0.5	0.5	N/A	N/A	N/A
Mating Prob	0.5	0.5	0.5	N/A	N/A	N/A
training time (hours)	~0.5	~0.5	~0.5	~0.5	~6	~0.5

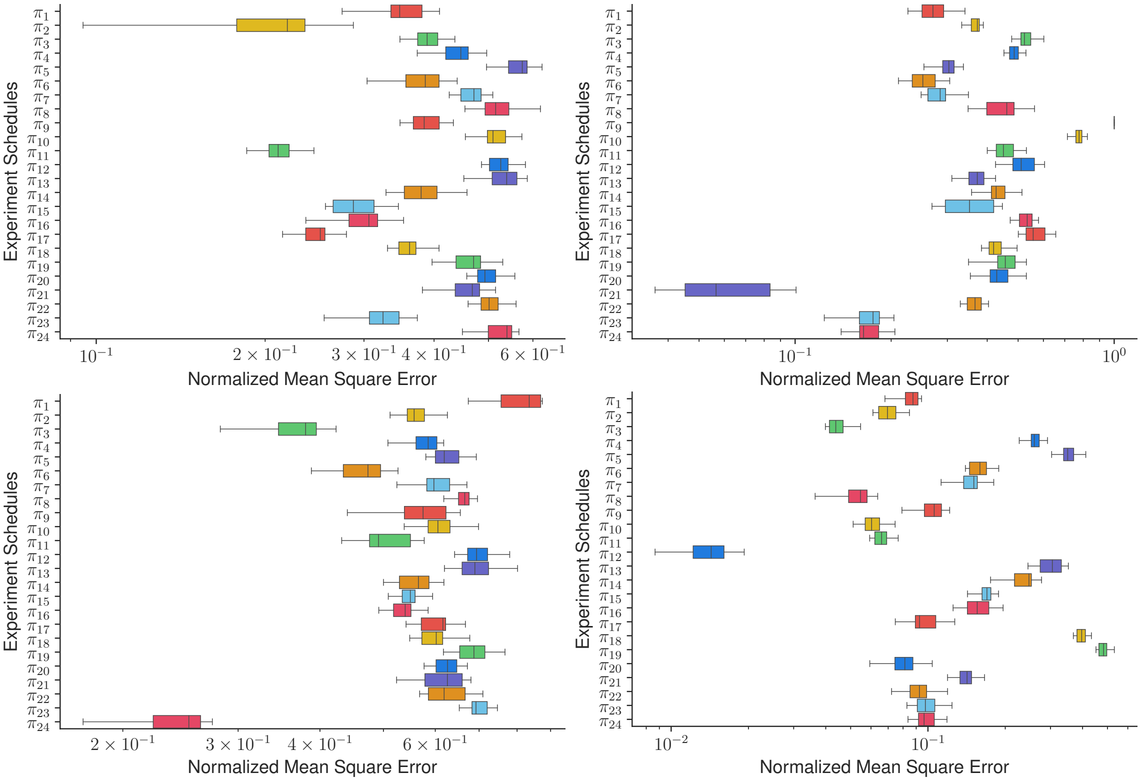


Figure E.2. Impact of experiment schedules (noted as π) on learning performance of control variable genetic programming, on the Trigonometric $(4, 4, 6)$ with operator set $\{+, -, \times, \div, \sin, \cos\}$ dataset. For the discovery of 10 different expressions with 4 variables, there always exists a better experiment schedule than the default one (*i.e.*, π_1), in terms of normalized mean square error.

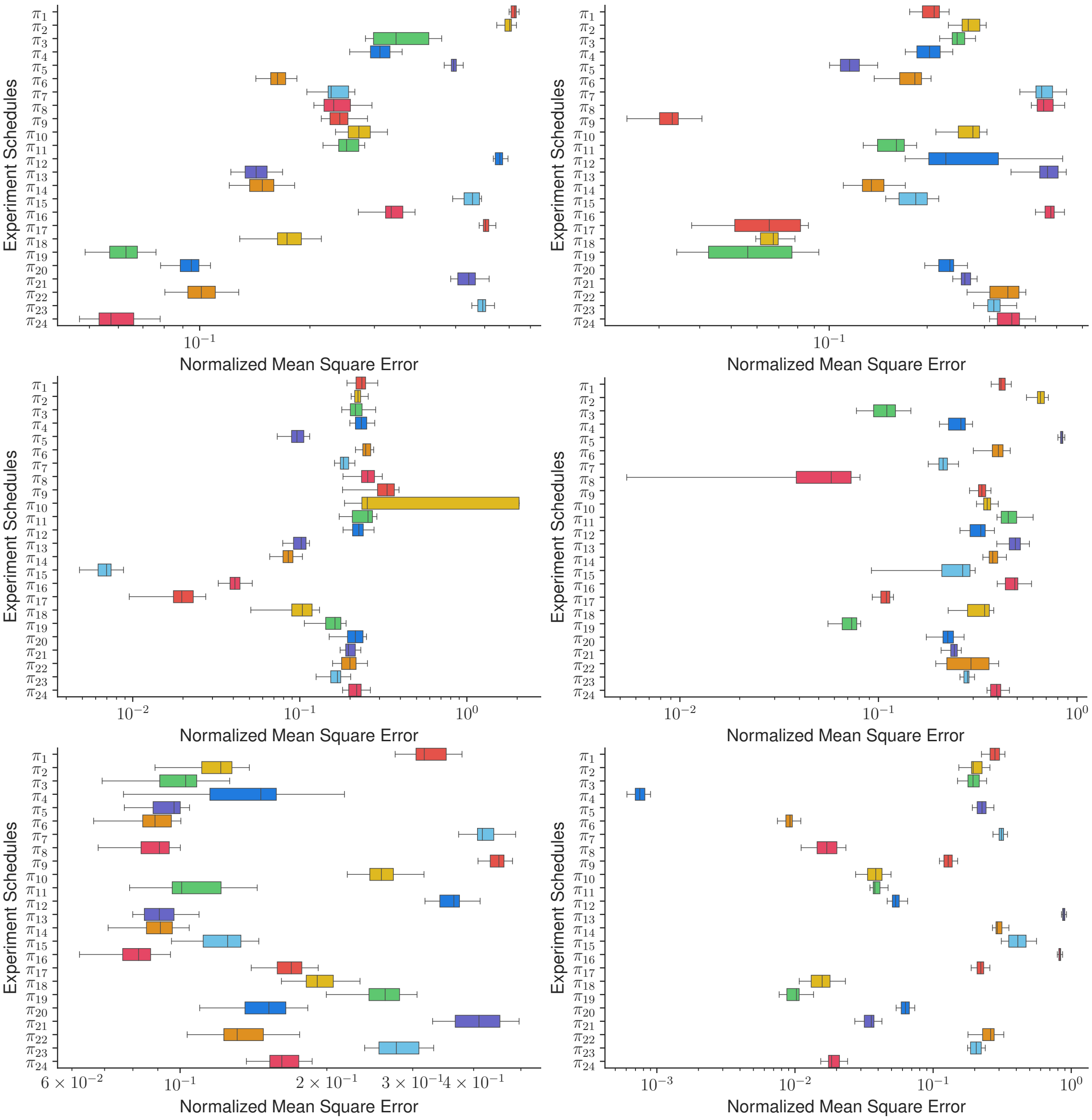


Figure E.3. (Continued) Impact of experiment schedules (noted as π) on learning performance of control variable genetic programming. For the discovery of expression with 4 variables, there always exists a better experiment schedule than the default one (*i.e.*, π_1), in terms of normalized mean square error.

E.4 Extra Experimental Analysis

E.4.1 Impact of Experiment Schedules: See Figure E.2,E.3

E.4.2 Empirical Running Time: See Figure E.4

E.4.3 Impact of Evaluation Metrics: See Figure E.5

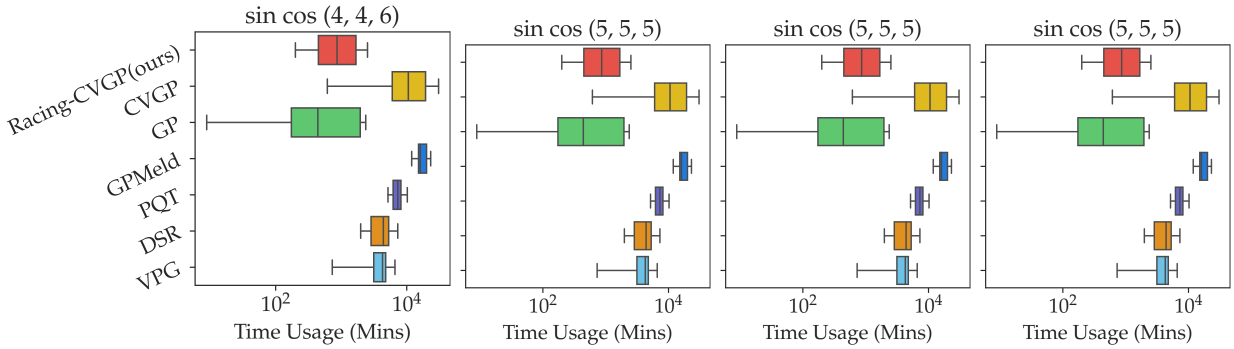


Figure E.4. On Trigonometric datasets, quartiles of the total running time of all the methods. Our Racing-CVGP method takes less time than CVGP by early stopping those unfavorable experiment schedules.

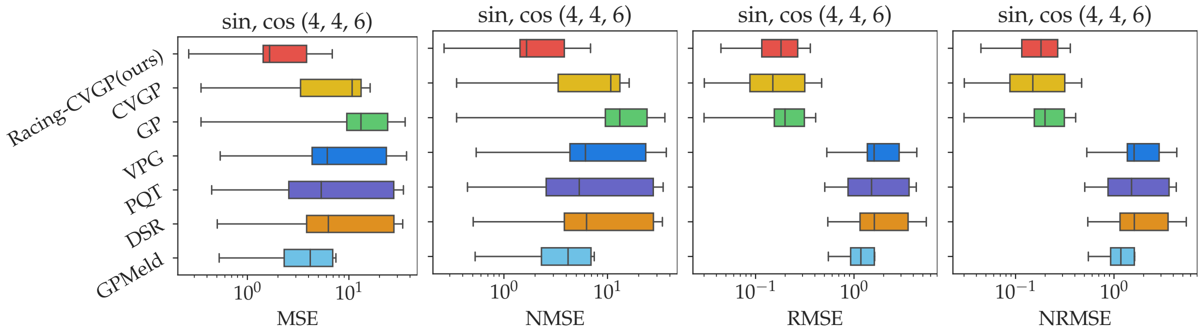


Figure E.5. On selected Trigonometric datasets, MSE, NMSE, RMSE, and NRMSE evaluation metrics of the expressions found by different algorithms.

E.4.4 Impact of Optimizers

Here we study the impact of using global and local optimizers over those non-convex expressions. With the introduction of control variable experiments, fitting the open constants in the expressions is solving more and more non-convex optimization problems.

For those expressions in the populations, an optimizer might find a set of open constants for a structurally correct expression with large NMSE errors, resulting in a low ranking in the whole population. Such structurally correct expressions will not be included after several rounds of genetic operations.

We summarize the experimental result in Figure E.6. In general, the list of global optimizers (SHGO, Direct, Basin-Hopping, and Dual-Annealing) fits better for the open constants

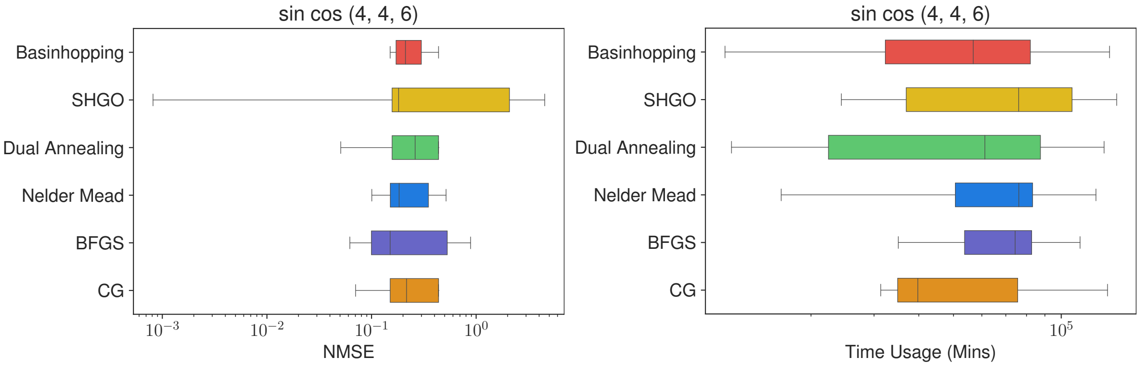


Figure E.6. Impact of optimizers on finding the values of open constants for non-convex expressions. Over 10 randomly generated expressions involving 4 variables, SHGO can find better solutions (in terms of NMSE metric) than local optimizers (including Nelder-Mead, BFGS, CG), while the time taken by SHGO is higher than local optimizers.

than the list of local optimizers but they take significantly more CPU resources and time for computations.

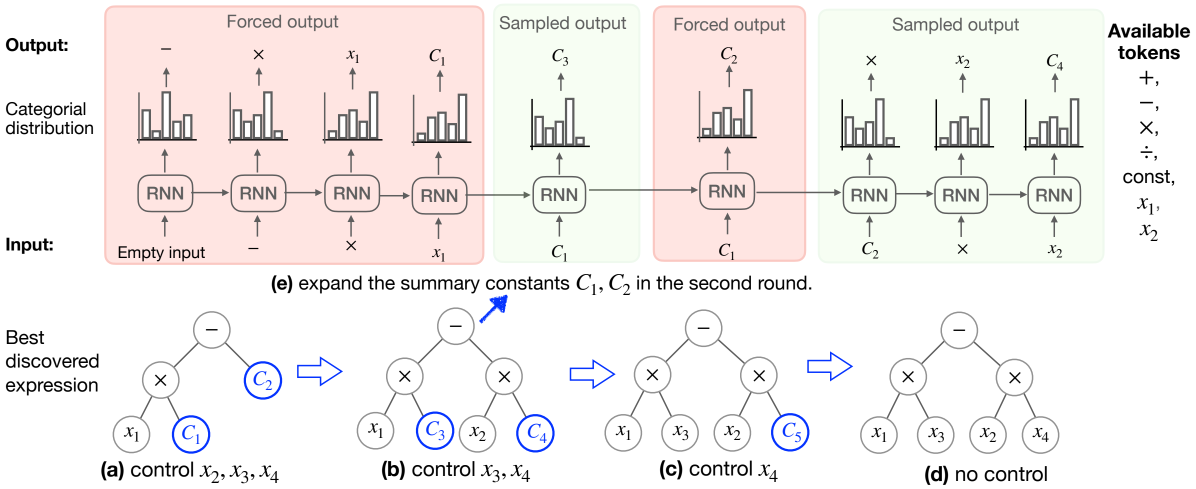
F. Appendix for Chapter 8

F.1 Direct Integration of Vertical Symbolic Regression with Deep Policy Gradient

Here we provide two possible pipelines for integrating the idea of vertical symbolic regression with deep reinforcement learning, using the binary tree representation of symbolic expressions. We will show the limitations of each integration. The fundamental cause is the tree representation of expression.

Tree Representation for Symbolic Expression. A symbolic expression can be represented as an *expression tree*, where variables and constants correspond to leaves, and operators correspond to the inner nodes of the tree. An inner node can have one or multiple child nodes depending on the arity of the associated operator. For example, a node representing the addition operation (+) has 2 children, whereas a node representing trigonometric functions like cos operation has a single child node. The preorder traversal sequence of the expression tree uniquely determines a symbolic expression. Figure F.1(a) presents an example of such an expression tree of the expression $x_1 \times C_1 - C_2$. Its preorder traversal sequence is $(-, \times, x_1, C_1, C_2)$. This traversal sequence uniquely determines a symbolic expression.

Genetic Programming for Symbolic Regression. Genetic Programming (GP) [305] has been a popular algorithm for symbolic regression. The core idea of GP is to maintain a pool of expressions represented as *expression trees*, and iteratively improve this pool according to the fitness score. The fitness score of a candidate expression measures how well the expression fits a given dataset. Each generation of GP consists of 3 basic operations – *selection, mutation* and *crossover*. In the *selection* step, candidate expressions with the highest fitness scores are retained in the pool, while those with the lowest fitness scores are discarded. In the *mutation* step, sub-expressions of some randomly selected candidate expressions are altered with some probability. In the *crossover* step, the sub-expressions of different candidate expressions are interchanged with some probability. In implementation, *mutation* changes a node of the expression tree while *crossover* is the exchange of subtrees between a pair of trees. This whole process repeats until we reach the final generation. We obtain a pool of expressions with high fitness scores, i.e., expressions that fit the data well, as our final solutions.



(e) expand the summary constants C_1, C_2 in the second round.

(a) control x_2, x_3, x_4 **(b)** control x_3, x_4 **(c)** control x_4 **(d)** no control

Figure F.1. Constraint-based integration of deep reinforcement learning with vertical symbolic regression. The constraints enforce the output of RNN output the given token at each step. It has limitations in passing the gradient to the parameters of RNN and also requires heavy engineering of different constraints. **(a)** Initially, the RNN to learn a reduced form equation with variables x_2, x_3, x_4 controlled. The RNN learns to sample the best preorder traversal of the reduced form expression tree from the *available tokens*. No constraints are applied in the first round. **(b, e)** Given the best-predicted expression ϕ_1 represented as $(-, \times, x_1, C_1, C_2)$ at the first round, the RNN is used to predict an expression with control variables x_3, x_4. For the first four steps, the constraints are applied to mask out other tokens in the output, to enforce that the output must be $-, \times, x_1, C_1$. Since C_1 is a summary constant, the RNN samples a sub-expression with no constraints starting at the 5th step, which is C_3. In 6-th step, with the termination of the prior sub-expression, constraints are applied to enforce the RNN outputs C_2. Starting at the 7th step, we sample a subexpression $x_2 \times C_4$. **(c,d)** The rest of the steps in the pipeline of vertical symbolic regression using expression tree representation.

Genetic Programming for Vertical Symbolic Regression (VSR-GP). VSR-GP uses GP as a sub-routine to predict the best expression at every round. At the end of every round, for an expression in the pool with close-to-zero MSE metric, VSR-GP marks the inner nodes for mathematical operators and leaf nodes for standalone constants and variables as non-mutable. Only the leaf nodes for summary constants as marked as mutable, because summary constants are those sub-expressions containing controlled variables. During *mutation* and *crossover*, VSR-GP only alters the mutable nodes of the candidate expression trees. In comparison, all the nodes in the expression tree are mutable in classic GP.

Deep Policy Gradient for Symbolic Regression The deep reinforcement learning-based approaches predict the expression by sampling the pre-order traversal sequence of the expression using RNN. The parameters of the RNN are trained through a policy gradient-based objective. The original work [180] proposes a relatively complex RL-based symbolic regression framework, where those extra modules are omitted in this part to ensure the main idea is clearly delivered. In this work, the policy gradient and the RL learner is used interchangeably, which refers to the sample algorithm.

In the following, we present two possible integrations for the vertical discovery path with policy gradient-based symbolic regressor, based on the expression tree representation.

F.1.1 Constraint-based Integration

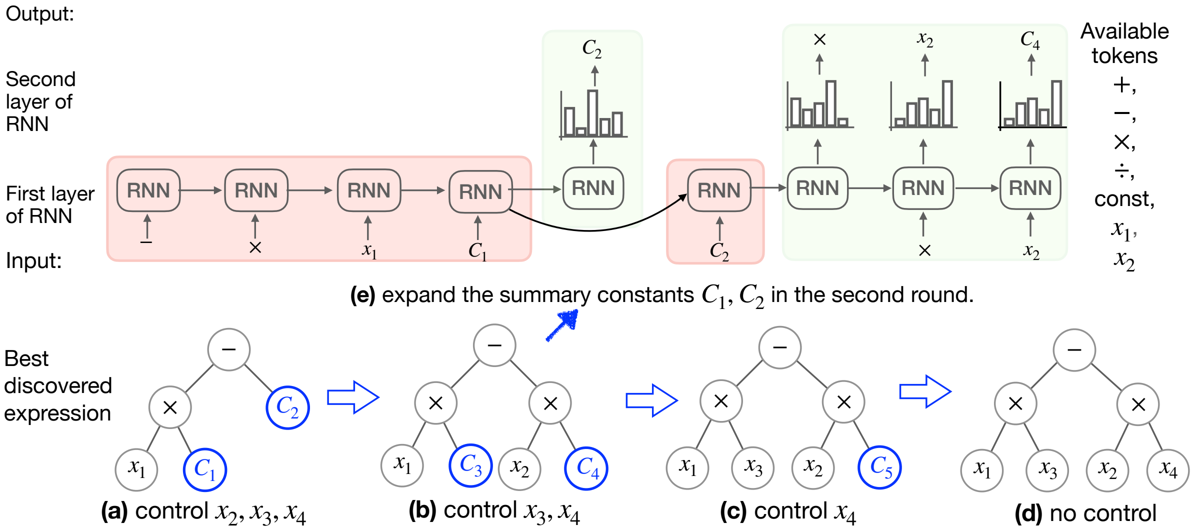
To force the predicted expression at the current round to be close to the previously predicted expression, the first idea is to apply constraints to limit the output vocabulary at every step of sampling tokens.

Take Figure F.1 as an example. Given the best-predicted expression ϕ_1 represented as pre-order traversal sequence $(-, \times, x_1, C_1, C_2)$ at the first round, we want to use the RNN to predict a new expression ϕ that:

1. has close-to-zero MSE value on the data with control variables x_3, x_4,
2. similar to ϕ_1 under controlled variables x_2, x_3, x_4.

It could be achieved by forcing the RNN to predict the subtract token "$-$" at the first step, where the rest tokens are masked out by the designed constraint. Similarly, we force the RNN to predict the rest of two tokens $\times$, `const` with the designed constraints. Since we know the 4-th step output is a summary constant, we would have no constraint at 5th step and sample a token from the categorical distribution over all tokens. In the 6th step, the constraint is applied to force the RNN to output C_2, because the previous sub-expression has been completed. This constraint-based approach will force the sampled expression, like $\phi_2 = x_1 \times C_3 - x_2 \times x_4$, to be close to the best expression ϕ_1 of the prior round.

The limitations of constraint-based integration are:



(e) expand the summary constants C_1, C_2 in the second round.

(a) control x_2, x_3, x_4 **(b)** control x_3, x_4 **(c)** control x_4 **(d)** no control

Figure F.2. Concatenation-based integration of deep reinforcement learning with vertical symbolic regression. Multiple layers of RNN are concatenated together to implement the vertical symbolic regression. The limitation is we need to store all the parameters of previously trained RNN, leading to a joint model with massive memory consumption. **(a,b,c,d)** The pipeline of vertical symbolic regression using expression tree representation. **(e)** The first layer takes the input of the best-predicted expression ϕ, and the second layer uses the hidden vectors of the 4-th step and 5-th step of the first layer, as input to predict two separated sequence C_3 and $\times, x_2, C_4$. The parameters of the first layer are frozen while the parameters of the second layer are trained.

1. heavy engineering of designing the constraints and checking if the sub-expression has been completed. In Figure F.1(e), every step of constraints is different from each others.

2. Gradient computation issue. Only when the first sub-expression is done can we then apply constraints to enforce the RNN to output C_2. This step can be implemented with various if-then checking which does not have a properly defined gradient. It causes the gradient computation of the loss function to the parameters of RNN.

F.1.2 Concatenation-based Integration

The second possible idea is concatenating multiple layers of RNNs. The first layer of RNN is the trained RNN at the 1st round. We use the first layer of the RNN to take in the best sequence of the first round. When we read in a summary constant, we use the updated hidden vector of the first layer as the initial vector of the second layer RNN.

Take Figure F.2 as an example. The first layer RNN takes the sequence $(-, \times, \texttt{const}, x_1, \texttt{const})$. Because the 4th and 5th step input of the first layer is summary constant type, we use the 4th and 5th step output vectors to initialize the hidden state vector of the second layer RNN. The second layer of RNN predicts two separated sub-expressions: `const` and $\times, x_2, \texttt{const}$. The whole sequence corresponds to the sampled expression ϕ_2 from the concatenated RNNs. The parameters inside the second layer RNN need to be trained by the policy gradient algorithm. Similarly, at the last round, we re-use pre-trained $n-1$ layers of RNN to take in the best-predicted expression ϕ_{n-1} and use one more layer to expand the summary constants in expression ϕ_{n-1}. The whole predicted sequence of tokens is the final predicted expression ϕ_n. Notice that the parameters of the prior layers of RNN can be frozen to reduce the number of parameters for training.

The main limitation of this idea is that: (1) we need to store all the trained RNNs. This does not scale up to many input variable cases. At the last round, we will have $n-1$ frozen layers of RNN and one trainable layer of RNN. (2) Due to multiple layers of RNNs and the sequence of input becoming longer and longer, then the training speed of the whole model will be slower and slower with fewer and fewer controlled variables.

F.2 Extended Explanation of Vsr-Dpg method

Data-availability Assumption. A crucial assumption behind the success of vertical symbolic regression is the availability of a `DataOracle` that returns a (noisy) observation of the dependent output with input variables in $\mathbf{x}_c$ controlled. Such a data oracle represents conducting control variable experiments in the real world, which can be expensive. This differs from the horizontal symbolic regression, where a dataset is obtained prior to learning with no variable controlled [197].

The vertical discovery path is to build algorithms that mimic human scientific discovery, which has achieved tremendous success in early works [189–191]. Recent work [275–277, 286] also pointed out the importance of having a data oracle that can actively query data points, rather than learning from a fixed dataset. In cases where it is difficult to obtain such a data oracle, Keren *et al.* proposed the use of deep neural networks to learn a data generator for the given set of controlled variables.

F.2.1 Sequential Decision Making Formulation

We consider an undiscounted MDP with a finite horizon. The state space $\mathcal{S}$ is the set of all possible sequences of rules with maximum steps. The action space $\mathcal{A}$ is the set of grammar rules. The t-th step state s_t is the sequence of sampled rules before the current step t, i.e., $s_t := (\tau_1, \dots, \tau_t)$. The action a_t is the sampled single rule, $a_t := \tau_{t+1}$.

Objective and its Gradient. The loss function of VSR-DPG is informed by the REINFORCE algorithm [239], which is based on the log-derivative property:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau),$$

where $p_\theta(\tau) \in (0,1)$ represents a probability distribution over input τ with parameters θ and notation ∇_θ is the partial derivative with respect to θ. In our formulation, $p_\theta(\tau)$ denotes the probability of sampling a sequence of grammar rules τ and $\texttt{reward}(\tau) = 1/(1 + \texttt{NMSE}(\phi))$. Here ϕ is the corresponding expression constructed from the rules τ following the procedure in Section 8.3.1. The probability $p_\theta(\tau)$ is modeled by the RNN modules. The learning objective is to maximize the expected reward of the sampled expressions from the RNN:

$$\arg\max_\theta \ \mathbb{E}_{\tau\sim p_\theta(\tau)}[\texttt{reward}(\tau)]$$

Based on the REINFORCE algorithm, the gradient of the objective can be expanded as:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\tau\sim p_\theta(\tau)}[\texttt{reward}(\tau)] &= \nabla_\theta \sum_{\tau\in\Sigma} \texttt{reward}(\tau) p_\theta(\tau) \\
&= \sum_{\tau\in\Sigma} \texttt{reward}(\tau) \nabla_\theta p_\theta(\tau) \\
&= \sum_{\tau\in\Sigma} \texttt{reward}(\tau) p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \\
&= \mathbb{E}_{\tau\sim p_\theta(\tau)} \left[\texttt{reward}(\tau) \nabla_\theta \log p_\theta(\tau)\right]
\end{aligned}$$

Algorithm 9: Vertical Symbolic Regression via Deep Policy Gradient.

```
Input: #input variables $n$; Mathematical Operators $O_p$; Draw data with
       controlled variables DataOracle
Output: The best-predicted expression
$\mathbf{x}_c \leftarrow \{x_1, \ldots, x_n\}$;                        // controlled variables
$S = A$ ;                                                            // start symbol
$\mathcal{Q} \leftarrow \emptyset$ ;                                  // best expressions across all rounds
$D_{global} \leftarrow$ DataOracle($\emptyset$) ;                     // data oracle with no control variable
draw a batch of data $T_g \leftarrow$ GenData($D_{global}$);
for $x_i \in \{x_1, \ldots, x_n\}$ do
    set controlled variables $\mathbf{x}_c \leftarrow \mathbf{x}_c \setminus \{x_i\}$;
    construct data oracle $D_o \leftarrow$ DataOracle($\mathbf{x}_c$);
    obtain the best predicted equation $\phi \leftarrow$ DPG($S, D_o, O_p \cup \{$const$, x_i\}$);
    for $k = 1$ to $K$ do
        //multiple control variable trails;
        draw a batch of data $T_k \leftarrow$ GenData($D_o$);
        fitness score $o_k$, fitted constant values $c_k$, fitted expression
          $\phi_k \leftarrow$ Optimize($\phi, T_k$);
    decide summary or standalone type for every constant in $\phi$ using $\{(o_k, c_k)\}_{k=1}^{K}$;
    construct start symbol $S$ for next round from $\phi$;
    fitness score $o_g$, fitted constant values $c_g$, fitted expression $\phi_g \leftarrow$ Optimize($\phi, T_g$);
    saving $\langle o_g, c_g, \phi_g \rangle$ into $\mathcal{Q}$.
return the equation with best fitness score in $\mathcal{Q}$;
Function DPG:
    Input: start symbol $S$, data oracle $D_o$, allowed operators and variables $O_p$
    initialize $Q =[]$. construct grammar rules from $O_p$. set input and output
      vocabulary for RNN with the grammar rules. sets the initial input of RNN as
      the start symbol $S$.
    for $t \leftarrow 1$ to #epochs do
        //In Section 8.3.2;
        sample $N$ sequences of grammar rules $\{\tau_i\}_{i=1}^{N}$ from RNN;
        //In Section 8.3.1;
        construct expressions $\{\phi_i\}_{i=1}^{N}$ from grammar rules $\{\tau_i\}_{i=1}^{N}$;
        for $i = 1$ to $N$ do
            draw data $T_i \leftarrow$ GenData($D_o$) ;               // optimize open constants
            fitness score $o_i$, fitted constants $c_i$, fitted expression $\leftarrow$ Optimize($\phi, T_i$);
            compute reward($\phi_i$) using fitness score $o_i$;
            saving $\langle o_i, c_i, \phi_i \rangle$ into $\mathcal{Q}$;
        compute estimated base $b = \sum_{i=1}^{N}$ reward($\tau^i$);
        compute the estimated policy gradient
          $g_t \leftarrow \frac{1}{N}\sum_{i=1}^{N}($reward$(\tau^i) - b)\nabla_\theta \log p_\theta(\tau^i)$;
        update parameters of RNN by gradient descent $\theta^t \leftarrow \theta^{t-1} + \alpha g_t$.
    return the expression in $\mathcal{Q}$ with best fitness score.
```

where Σ represents all possible sequences of grammar rules sampled from the RNN. The above expectation can be estimated by computing the empirical average over samples drawn from the distribution $p_\theta(\tau)$. We first sample several times from the RNN module and obtain N sequences $(\tau^1, \dots, \tau^N)$, an unbiased estimation of the gradient of the objective is computed as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \texttt{reward}(\tau^i)\nabla_\theta \log p_\theta(\tau^i)$$

In practice, the above computation has a high variance. To reduce variance, it is common to subtract a baseline function b from the reward. In this study, we choose the baseline function as the average of the reward of the current sampled batch expressions. Thus we have:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} (\texttt{reward}(\tau^i) - b)\nabla_\theta \log p_\theta(\tau^i),$$

where $b = \sum_{i=1}^{N} \texttt{reward}(\tau^i)$.

Based on the description of the execution pipeline of the proposed Vsr-Dpg, we summarize every step in Algorithm 9.

F.2.2 Implementation of VSR-DPG

Neural Network Configuration. In the experiments, we use Long short-term memory (LSTM) as the RNN layer and we configure the number of RNN layers as 3. The dimension of the input embedding layer and the hidden vector in LSTM is configured as 512. We use the Adam optimizer as the gradient descent algorithm with a learning rate of 0.009. The learning epoch for each round is configured 30. The maximum sequence of grammar rules is fixed to be 20. The number of expressions sampled from the RNN is set as 1024.

Coefficient Fitting. Afterward, we decide the optimal value of open constants in each expression. Assume the expression has m open constants. We first sample a batch of data D with the controlled variable $\mathbf{x}_c$ and then use a gradient-based optimizer to fit those open constants, by minimizing the objective:

$$\min_{\mathbf{c}\in\mathbb{R}^m} \frac{1}{N}\sum_{i=1}^{N} \ell(\phi(\mathbf{x}_i, \mathbf{c}), y_i)$$

We then obtain the fitness score o, the fitted constants $\mathbf{c}$, and the fitted equation ϕ. When fitting the values of open constants in each expression, we sample a batch of data with batch size 1024 from the data Oracle. The open constants in the expressions are fitted on the data using the BFGS optimizer[1]. We use a multi-processor library to fit multiple expressions using 8 CPU cores in parallel. This greatly reduced the total training time.

An expression containing placeholder symbol A or containing more than 20 open constants is not evaluated on the data, the fitness score of it is $-\infty$. In terms of the reward function in the policy gradient objective, we use $\texttt{reward}(\tau) = \frac{1}{1+\texttt{NMSE}(\phi)}$. The normalized mean-squared error metric is further defined in Equation F.1.

The deep network part is implemented using the most recent version of TensorFlow, the expression evaluation is based on the Sympy library, and the step for fitting open constants in expression with the dataset uses the Scipy library.

F.3 Experiment Settings

F.3.1 Evaluation Metrics

The goodness-of-fit indicates how well the learning algorithms perform in discovering unknown symbolic expressions. Given a testing dataset $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ generated from the ground-truth expression, we measure the goodness-of-fit of a predicted expression ϕ, by evaluating the mean-squared-error (MSE) and normalized-mean-squared-error (NMSE):

$$\begin{aligned}\text{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \phi(\mathbf{x}_i))^2,\\ \text{NMSE} &= \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \phi(\mathbf{x}_i))^2}{\sigma_y^2},\end{aligned} \tag{F.1}$$

The empirical variance $\sigma_y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2}$. We use the NMSE as the main criterion for comparison in the experiments and present the results on the remaining metrics in the case studies. The main reason is that the NMSE is less impacted by the output

[1] ↑https://docs.scipy.org/doc/scipy/reference/optimize.minimize-bfgs.html

range. The output ranges of expression are dramatically different from each other, making it difficult to present results uniformly if we use other metrics.

Prior work [180] further proposed coefficient of determination R^2-based Accuracy over a group of expressions in the dataset, as a statistical measure of whether the best-predicted expression is almost close to the ground-truth expression. An R^2 of 1 indicates that the regression predictions perfectly fit the data [299]. Given a threshold value `thresh` (we use $\texttt{thresh} = 0.9999$), for a dataset containing fitting tasks of N expressions, the algorithm finds a group of best expressions $[\phi_1, \ldots, \phi_N]$ correspondingly. The R^2-based accuracy is computed as follows:

$$R^2\text{- based Accuracy} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(R^2(\phi_i) \geq \texttt{thresh}),$$

where $R^2(\phi_i) = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \phi(\mathbf{x}_i))^2}{\sigma_y^2}$ and $\mathbf{1}(\cdot)$ is an indicator function that outputs 1 when the $R^2(\phi_i)$ exceeds the threshold τ.

F.3.2 Symbolic Regression on Algebraic Equations

Baselines. We consider a list of current popular baselines based on genetic programming[2]:

- Genetic Programming (GP) maintains a population of candidate symbolic expressions, in which this population *evolves* between generations. In each generation, candidate expressions undergo *mutation* and *crossover* with a pre-configured probability value. Then in the *selection* step, expressions with the highest fitness scores (measured by the difference between the ground truth and candidate expression evaluation) are selected as the candidates for the next generation, together with a few randomly chosen expressions, to maintain diversity. After several generations, expressions with high fitness scores, *i.e.*, those expressions that fit the data well survive in the pool of candidate solutions. The best expressions in all generations are recorded as hall-of-fame solutions.

[2] ↑https://github.com/jiangnanhugo/cvgp

- Vertical symbolic regression with genetic programming (VSR-GP) builds on top of GP. It discovers the ground-truth expression following the vertical discovery path. In the t-th round, it controls variables $x_{t+1}, \ldots, x_n$ as constant, and only discovers the expression involving the rest variables $x_1, \ldots, x_n$.

Eureqa [237] is the current best commercial software based on evolutionary search algorithms. Eureqa works by uploading the dataset $\mathcal{D}$ and the set of operators as a configuration file to its commercial server. This algorithm is currently maintained by the DataRobot webiste[3]. Computation is performed on its commercial server and only the discovered expression will be returned after several hours. We use the provided Python API to send the training dataset to the DataRobot website and collect the predicted expression from the server-returned result. For the Eureqa method, the fitness measure function is negative RMSE. We generated large datasets of size 10^5 in training each dataset.

A line of methods based on reinforcement learning[4]:

- Deep Symbolic Regression (DSR) [180] uses a combination of recurrent neural network (RNN) and reinforcement learning for symbolic regression. The RNN generates possible candidate expressions, and is trained with a risk-seeking policy gradient objective to generate better expressions.

- Priority queue training (PQT) [238] also uses the RNN similar to DSR for generating candidate expressions. However, the RNN is trained with a supervised learning objective over a data batch sampled from a maximum reward priority queue, focusing on optimizing the best-predicted expression.

- Vanilla Policy Gradient (VPG) [239] is similar to DSR method for the RNN part. The difference is that VPG uses the classic REINFORCE method for computing the policy gradient objective.

- Neural-Guided Genetic Programming Population Seeding (GPMeld) [181] uses the RNN to generate candidate expressions, and these candidate expressions are improved by a genetic programming (GP) algorithm.

[3] ↑https://docs.datarobot.com/en/docs/modeling/analyze-models/describe/eureqa.html

[4] ↑https://github.com/dso-org/deep-symbolic-optimization

(a) Genetic Programming-based methods.

	VSR-GP	GP	Eureqa
Fitness function	NegMSE	NegMSE	NegRMSE
Testing set size	256	256	50,000
#CPUs for training	1	1	N/A
#genetic generations	200	200	10,000
Mutation Probability	0.8	0.8	
Crossover Probability	0.8	0.8	

(b) Monte Carlo Tree Search-based methods.

	MCTS
Fitness function	NegMSE
Testing set size	256
#CPUs for training	1

(c) Deep reinforcement learning-based methods.

	DSR	PQT	GPMeld
Reward function	1/(1+NRMSE)		
Training set size	50,000		
Testing set size	256		
Batch size	1024		
#CPUs for training	8		
ϵ-risk-seeking policy	0.02	N/A	N/A
#genetic generations	N/A	N/A	60
#Hall of fame	N/A	N/A	25
Mutation Probability	N/A	N/A	0.5
Crossover Probability	N/A	N/A	0.5

Table F.1. Major hyper-parameters settings for all the algorithms considered in the experiment.

Symbolic Physics Learner (SPL) is a heuristic search algorithm based on Monte Carlo Tree Search for finding optimal sequences of production rules using context-free grammars [240, 258][5]. It employs Monte Carlo simulations to explore the search space of all the production rules and determine the value of each node in the search tree. SPL consists of four steps in each iteration: 1) Selection. Starting at a root node, recursively select the optimal child (*i.e.*, one of the production rules) until reaching an expandable node or a leaf

[5] ↑https://github.com/isds-neu/SymbolicPhysicsLearner

node. 2) Expansion. If the expandable node is not the terminal, create one or more of its child nodes to expand the search tree. 3) Simulation. Run a simulation from the new node until achieving the result. 4) Backpropagation. Update the node sequence from the new node to the root node with the simulated result. To balance the selection of optimal child node(s) by exploiting known rewards (exploitation) or expanding a new node to explore potential rewards exploration, the upper confidence bound (UCB) is often used.

End to End Transformer for symbolic regression (E2ETransformer) [208][6]. They propose to use a deep transformer to pre-train on a large set of randomly generated expressions. We load the shared pre-trained model. We provide the given dataset and the E2ETransformer infers 10 best expressions. We choose to report the expression with the best NMSE scores.

We list the major hyper-parameter settings for all the algorithms in Table F.1. Note that if we use the default parameter settings, the GPMeld algorithm takes more than 1 day to train on one dataset. Because of such slow performance, we cut the number of genetic programming generations in GPMeld by half to ensure fair comparisons with other approaches.

Dataset for Algebraic Equations The dataset is available at the code repository with the folder name:

`data/algebraic_equations/equations_trigonometric`.

The expressions used for comparison have the same mathematical operators $O_p = \{+, -, \times, \sin, \cos\}$. One configuration $(2, 1, 1)$ is shown in Table F.2.

For the extended analysis, where we consider many more input variables, they are available in the folder with the name:

$$\texttt{data/algebraic\_equations/large\_scale\_}n$$

where the value of n is the number of total variables in, which can be $10, 20, 30, 40, 50$.

The original expression is: $-0.4156x_0x_1 - 0.1399x_2\cos(x_1) + 0.0438x_2 + 0.9508x_3\sin(x_1) + 0.2319x_3 - 0.6808x_4\cos(x_3) - 0.4468x_4 + 0.0585\sin(x_0) + 0.6224\cos(x_1) - 0.8638\cos(x_2)\cos(x_3) + 0.959$. We extend this expression by choosing 5 variables from the total $n = 10$ variables and

[6] ↑https://github.com/facebookresearch/symbolicregression

Equation ID	Exact Expression
prog-0	$-0.167\sin(x_0)\cos(x_1)+0.4467\cos(x_0)-0.2736$
prog-1	$0.6738x_0-0.5057\sin(x_0)\sin(x_1)+0.8987$
prog-2	$-0.5784x_0x_1+0.556\cos(x_1)+0.8266$
prog-3	$0.0882x_0-0.7944\sin(x_0)\sin(x_1)+0.4847$
prog-4	$-0.7262\sin(x_1)\cos(x_0)-0.006\cos(x_1)-0.9218$
prog-5	$0.189x_0x_1-0.7125\cos(x_1)-0.4207$
prog-6	$0.2589x_0\sin(x_1)+0.1977x_1-0.7504$
prog-7	$-0.2729x_0\sin(x_1)-0.7014x_1+0.3248$
prog-8	$-0.2582x_0-0.8355x_1\cos(x_0)-0.5898$
prog-9	$0.1052x_0x_1+0.0321x_0-0.9554$

Table F.2. 10 randomly drawn expressions with 2 variables, 1 single term, and 1 cross term with operators $\{\sin, \cos, +, -, \times\}$.

mapping the selected variables to the variables $x_0, \ldots, x_4$. Here are 10 randomly generated expressions:

$$\begin{aligned}
\phi_1 =& -0.4156x_3x_9-0.1399x_1\cos(x_3)+0.0438x_1+0.9508x_0\sin(x_3)+0.2319x_0-0.6808x_4\cos(x_0)\\
&-0.4468x_4+0.0585\sin(x_9)+0.6224\cos(x_3)-0.8638\cos(x_0)\cos(x_1)+0.959\\
\phi_2 =& -0.4156x_0x_5-0.1399x_3\cos(x_0)+0.0438x_3+0.9508x_1\sin(x_0)+0.2319x_1-0.6808x_7\cos(x_1)\\
&-0.4468x_7+0.0585\sin(x_5)+0.6224\cos(x_0)-0.8638\cos(x_1)\cos(x_3)+0.959\\
\phi_3 =& -0.4156x_5x_8-0.1399x_1\cos(x_5)+0.0438x_1+0.9508x_4\sin(x_5)+0.2319x_4-0.6808x_0\cos(x_4)\\
&-0.4468x_0+0.0585\sin(x_8)+0.6224\cos(x_5)-0.8638\cos(x_1)\cos(x_4)+0.959\\
\phi_3 =& -0.4156x_2x_6-0.1399x_3\cos(x_2)+0.0438x_3+0.9508x_7\sin(x_2)+0.2319x_7-0.6808x_9\cos(x_7)\\
&-0.4468x_9+0.0585\sin(x_6)+0.6224\cos(x_2)-0.8638\cos(x_3)\cos(x_7)+0.959\\
\phi_4 =& -0.4156x_3x_7-0.1399x_8\cos(x_3)+0.0438x_8+0.9508x_2\sin(x_3)+0.2319x_2-0.6808x_9\cos(x_2)\\
&-0.4468x_9+0.0585\sin(x_7)+0.6224\cos(x_3)-0.8638\cos(x_2)\cos(x_8)+0.959\\
\phi_5 =& -0.4156x_1x_3-0.1399x_6\cos(x_3)+0.0438x_6+0.9508x_2\sin(x_3)+0.2319x_2-0.6808x_0\cos(x_2)\\
&-0.4468x_0+0.0585\sin(x_1)+0.6224\cos(x_3)-0.8638\cos(x_2)\cos(x_6)+0.959\\
\phi_6 =& -0.4156x_4x_5-0.1399x_7\cos(x_5)+0.0438x_7+0.9508x_6\sin(x_5)+0.2319x_6-0.6808x_8\cos(x_6)\\
&-0.4468x_8+0.0585\sin(x_4)+0.6224\cos(x_5)-0.8638\cos(x_6)\cos(x_7)+0.959
\end{aligned}$$

$$
\begin{aligned}
\phi_7 = &-0.4156x_3x_8 - 0.1399x_5\cos(x_3) + 0.0438x_5 + 0.9508x_0\sin(x_3) + 0.2319x_0 - 0.6808x_7\cos(x_0) \\
&- 0.4468x_7 + 0.0585\sin(x_8) + 0.6224\cos(x_3) - 0.8638\cos(x_0)\cos(x_5) + 0.959
\end{aligned}
$$

$$
\begin{aligned}
\phi_8 = &-0.4156x_0x_3 - 0.1399x_2\cos(x_0) + 0.0438x_2 + 0.9508x_5\sin(x_0) + 0.2319x_5 - 0.6808x_6\cos(x_5) \\
&- 0.4468x_6 + 0.0585\sin(x_3) + 0.6224\cos(x_0) - 0.8638\cos(x_2)\cos(x_5) + 0.959
\end{aligned}
$$

$$
\begin{aligned}
\phi_9 = &-0.4156x_0x_5 - 0.1399x_8\cos(x_5) + 0.0438x_8 + 0.9508x_7\sin(x_5) + 0.2319x_7 - 0.6808x_2\cos(x_7) \\
&- 0.4468x_2 + 0.0585\sin(x_0) + 0.6224\cos(x_5) - 0.8638\cos(x_7)\cos(x_8) + 0.959
\end{aligned}
$$

The rest expressions are available in the data folder.

F.3.3 Extra Results

We show in Fig F.3 the full quartiles of the experiments.

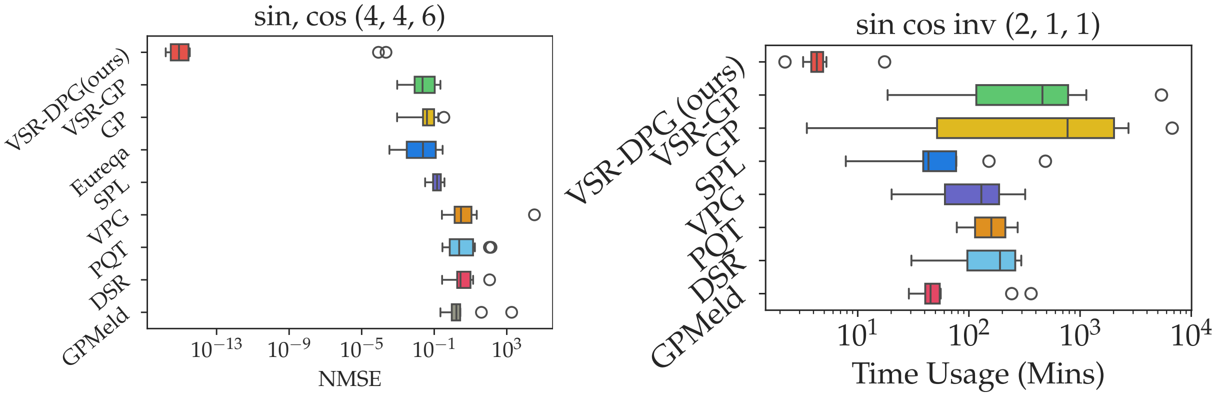


Figure F.3. Measurement of variability for the experiments. quantiles (25%, 50%, 75%) of (Left) Normalized MSE values of discovered equations and (Right) execution time of the learning algorithm.

F.3.4 Symbolic Regression on Ordinary Differential Equations

The temporal evolution of the system is modeled by the time derivatives of the state variables. Let $\mathbf{x}$ be the n-dimensional vector of state variables, and $\mathrm{d}\mathbf{x}/\mathrm{d}t$ is the vector of their time derivatives, which is noted as $\dot{\mathbf{x}}$ for abbreviation. The ordinary differential equation (ODEs) is of the form $\dot{\mathbf{x}} = \phi(\mathbf{x}, \mathbf{c})$, where constant vector $\mathbf{c} \in \mathbb{R}^m$ are parameters of the ODE model. Given the initial state $\mathbf{x}(t_0)$, the finite time difference Δt and the expression $\phi(\mathbf{x}, \mathbf{c})$, the ODEs are numerically simulated to obtain the state trajectory $\mathbf{x}(t_1), \dots, \mathbf{x}(t_N)$, where $\mathbf{x}(t_i) = \mathbf{x}(t_{i-1}) + \phi(\mathbf{x}, \mathbf{c})\Delta t$ and $t_i = t_{i-1} + \Delta t$.

Task Definition. Following the definition of symbolic regression on differential equation in [240, 265], given a trajectory dataset of state variable and its time derivatives $\{(\mathbf{x}(t_i), \dot{\mathbf{x}}(t_i))\}_{i=1}^N$, $\dot{\mathbf{x}}(t_i)$ represents the value of the derivative of variable $\mathbf{x}$ at time t_i, the symbolic regression task is to predict the best expression $\phi(\mathbf{x}, \mathbf{c})$ that minimizes the average loss on trajectory data:

$$\arg\min_{\phi} \frac{1}{N} \sum_{i=1}^{N} \ell(\dot{\mathbf{x}}(t_i), \phi(\mathbf{x}(t_i), \mathbf{c}))$$

Other formulations of this problem assume we have no access to its time derivatives, that is $\{(t_i, \mathbf{x}(t_i))\}_{i=1}^N$ [266]. This formulation is tightly connected to our setting and relatively more challenging. We can still estimate the finite difference between the current and next state variables as its approximated time derivative: $\dot{\mathbf{x}}(t_i) = \frac{\mathbf{x}(t_i) - \mathbf{x}(t_{i-1})}{t_i - t_{i-1}}$.

Baselines. For the baselines on the differentiable equations, we consider

- SINDy [214][7] is a popular method using a sparse regression algorithm to find the differential equations.
- ODEFormer [266][8] is the most recent framework that uses the transformer for the discovery of ordinary differential equations. We use the provided pre-trained model to predict the governing expression with the dataset. We execute the model 10 times and pick the expression with the smallest NMSE error. The dataset size is 500, which is the largest dataset configuration for the ODEFormer.

[7] ↑https://github.com/dynamicslab/pysindy

[8] ↑https://github.com/sdascoli/odeformer

- ProGED [264][9] uses probabilistic context-free grammar to search for differential equations. ProGED first samples a list of candidate expressions from the defined probabilistic context-free grammar for symbolic expressions. Then ProGED fits the open constants in each expression using the given training dataset. The equation with the best fitness scores is returned.

Dataset for Differential Equations. We collect a set of real-world ordinary differential equations of multiple input variables from the SINDy codebase[10].

- Lorenz Attractor. Let x_0, x_1, x_2 be functions of time $x_0(t), x_1(t), x_2(t)$ and stands for the position in the (x, y, z) coordinates. Here we consider 3-dimensional Lorenz system whose dynamical behavior (x_0, x_1, x_2) is governed by

$$\begin{aligned}\dot{x}_0 &= \sigma(x_1 - x_0),\\ \dot{x}_1 &= x_0(\rho - x_2) - x_1,\\ \dot{x}_2 &= x_0 x_1 - \beta x_2,\end{aligned}$$

  with parameters $\sigma = 10, \beta = 8/3, \rho = 28$.

[9] ↑https://github.com/brencej/ProGED

[10] ↑https://github.com/dynamicslab/pysindy/blob/master/pysindy/utils/odes.py

Variable	Biological Definition	Range	Standard deviation
x_0	Glucose	$[0.15, 1.60]$	0.4872
x_1	Glyceraldehydes-3-phosphate and dihydroxyacetone phosphate pool	$[0.19, 2.16]$	0.6263
x_2	1,3-bisphosphoglycerate	$[0.04, 0.20]$	0.0503
x_3	Cytosolic pyruvate and acetaldehyde pool	$[0.10, 0.35]$	0.0814
x_4	NADH	$[0.08, 0.30]$	0.0379
x_5	ATP	$[0.14, 2.67]$	0.7478
x_6	Extracellular pyruvate and acetaldehyde pool	$[0.05, 0.10]$	0.0159

Table F.3. Biological definition of variables in Glycolysis Oscillations. The allowed range of initial states for the training data set and the standard deviation of the limit cycle are also included.

- Glycolysis Oscillations. The dynamic behavior of yeast glycolysis can be described as a set of 7 variables $x_0, \dots, x_6$. The biological definition of each variable from Brechmann and Rendall is provided in Table F.3. The governing equations are:

$$\begin{aligned}
\dot{x}_0 &= J_0 - \frac{(k_1 x_0 x_5)}{(1 + (x_5/K_1)^q)}, \\
\dot{x}_1 &= \frac{2(k_1 x_0 x_5)}{1 + (x_5/K_1)^q} - k_2 x_1 (N - x_4) - k_6 x_1 x_4, \\
\dot{x}_2 &= k_2 x_1 (N - x_4) - k_3 x_2 (A - x_5), \\
\dot{x}_3 &= k_3 x_2 (A - x_5) - k_4 x_3 x_4 - \kappa (x_3 - x_6), \\
\dot{x}_4 &= k_2 x_1 (N - x_4) - k_4 x_3 x_4 - k_6 x_1 x_4, \\
\dot{x}_5 &= \frac{-2 k_1 x_0 x_5}{1 + (x_5/K_1)^q} + 2 k_3 x_2 (A - x_5) - k_5 x_5, \\
\dot{x}_6 &= \phi \kappa (x_3 - x_6) - K x_6
\end{aligned}$$

where the parameters $J_0 = 2.5, k_1 = 100, k_2 = 6, k_3 = 16, k_4 = 100, k_5 = 1.28, k_6 = 12, K = 1.8, \kappa = 13, q = 4, K_1 = 0.52, \phi = 0.1, N = 1, A = 4$. The rest of the differential equations from this Glycolysis family can be found at [306].

- MHD turbulence. The following equations describe the dynamic behavior of the Carbone and Veltri triadic MHD model:

$$
\begin{aligned}
\dot{x}_0 &= -2\nu x_0 + 4(x_1x_2 - x_4x_5), \\
\dot{x}_1 &= -5\nu x_1 - 7(x_0x_2 - x_3x_5), \\
\dot{x}_2 &= -9\nu x_2 + 3(x_0x_1 - x_3x_4), \\
\dot{x}_3 &= -2\mu x_4 + 2(x_5x_1 - x_2x_4), \\
\dot{x}_4 &= -5\mu x_4 + \sigma x_5 + 5(x_2x_3 - x_0x_5), \\
\dot{x}_5 &= -9\mu x_5 + \sigma x_4 + 9(x_4x_0 - x_1x_3),
\end{aligned}
$$

  where the parameters $\nu = 0, \mu = 0, \sigma = 0$. [307] define x_0, x_1, x_2 as the velocity and x_3, x_4, x_5 as to the magnetic field. ν, μ represents, respectively, the kinematic viscosity and the resistivity.

Evaluation Metrics. We use the R^2-based Accuracy metric to evaluate if the whole set of predicted expressions has a R^2 score higher than 0.9999.

F.4 Extra Experiments

F.4.1 Discovered Algebraic Equations by the Learning Algorithms

The predicted expression by Vsr-Dpg (ours) for configuration $(4, 4, 6)$. 60% of the predicted expression has a $\leq 10^{-6}$ NMSE score.

The predicted result for prog-0:

$$
\begin{aligned}
&-0.3012000175544417x_0x_3 - 0.23479995033497178x_0 + \\
&0.045433905730119135x_1 + 0.10966141816565093x_2 + \\
&0.22430013864298073x_3 + 0.9739999857983681\sin(x_2) + \\
&0.3581998363171518\cos(x_2)\cos(x_3) + \\
&0.2862218136669438\cos(x_3) + 3.126115887545009
\end{aligned}
$$

The predicted result for prog-1:

$$
\begin{gathered}
-0.5807073848480102x_0 - 0.09660000567273663x_1 - \\
0.9748000148040502x_2x_3 - 0.4638000163793846x_3\cos(x_0) \\
-0.4221638801953578x_3 - 0.012754904995223835\sin(x_2) \\
+0.15999997730356633\cos(x_2) + 0.2524999760074076\cos(x_3) \\
+0.3830840657508305
\end{gathered}
$$

The predicted result for prog-2:

$$
\begin{gathered}
0.5974706919691478x_0 + 0.8783029159486363x_1 + \\
0.584599994337829x_2\cos(x_1) - 0.8430097368938334x_3 - \\
0.4739999968689642\sin(x_2) - 3.3558075600032683e - 8\sin(x_3) \\
-0.3244634752208093\cos(x_2) + 0.5068000094901586\cos(x_3) \\
-0.787302540612925
\end{gathered}
$$

The predicted result for prog-3:

$$
\begin{gathered}
-0.8973000084985939x_0 - 7.242399512792391x_1 - \\
1.2513833693643626\cos(x_0) - 1.5175517989615754 \\
0.03273456882139954 4x_0 + 0.928299994219054x_1 sin(x_0) \\
+0.11740000072851x_1 - 1.6081211674938465x_2 + \\
0.5674704740296996x_3 + 0.1769999997564291\sin(x_2)\cos(x_3) \\
-0.4200518345694358
\end{gathered}
$$

The predicted result for prog-4:

$$
\begin{aligned}
&0.10499999077952751x_0\sin(x_2) + 0.8918999999268585x_3\sin(x_1)\\
&+ 0.11399999910836027x_3\cos(x_2) - 0.38250000586131516x_3\\
&- 0.14609999633658752x_4\sin(x_0) + 0.9090999941858626x_4\sin(x_1)\\
&- 0.6846999999068688\sin(x_0) + 0.9993000283241971\sin(x_2)\\
&- 0.19519999212829273\cos(x_1) + 0.6172999945789425\cos(x_4)\\
&- 0.4587999974860775
\end{aligned}
$$

The predicted result for prog-5:

$$
\begin{aligned}
&0.3900029487047949x_0 + 0.15013453625258577x_2 + 0.7973748097464934sin(x_2)\\
&+ 0.6004443541983869cos(x_1) + 1.4041023040405819
\end{aligned}
$$

F.4.2 Discovered Differential Equations by each Learning Algorithm

We collect the best-predicted expression by each algorithm for the MHD turbulence instance.

SINDy.

$$
\begin{aligned}
\dot{x}_0 = {}& 0.195 + 0.009x_0 + 0.025x_1 + 0.045x_2 + 0.001x_4 - 0.012x_5 - 3.772x_0^2 - 0.002x_0x_2\\
&+ 1.157x_0x_3 + 0.002x_0x_4 - 0.011x_0x_5 - 2.016x_1^2 + 3.976x_1x_2 - 0.001x_1x_3 + 1.158x_1x_4\\
&+ 0.003x_1x_5 + 0.306x_2^2 - 0.005x_2x_3 - 0.007x_2x_4 + 1.164x_2x_5 + 0.602x_3^2\\
&+ 0.011x_3x_5 + 0.437x_4^2 - 3.996x_4x_5 - 1.426x_5^2
\end{aligned}
$$

$$\begin{aligned}
\dot{x}_1 = {} & -1.046 + 0.011x_0 - 0.008x_1 + 0.01x_2 - 0.005x_3 - 0.003x_4 - 0.012x_5 - 0.686x_0^2 \\
& + 0.007x_0x_1 - 7.015x_0x_2 + 0.030x_0x_3 - 0.004x_0x_4 + 0.011x_0x_5 + 0.075x_1^2 + 0.013x_1x_2 \\
& - 0.003x_1x_3 + 0.035x_1x_4 + 0.001x_1x_5 + 1.108x_2^2 + 0.010x_2x_3 + 0.003x_2x_4 + 0.043x_2x_5 \\
& - 0.370x_3^2 + 0.001x_3x_4 + 6.997x_3x_5 + 0.518x_4^2 + 0.005x_4x_5 - 0.015x_5^2
\end{aligned}$$

$$\begin{aligned}
\dot{x}_2 = {} & 0.098 + 0.003x_0 - 0.007x_1 - 0.007x_2 - 0.007x_3 + 0.002x_5 - 0.965x_0^2 + 2.993x_0x_1 \\
& - 0.004x_0x_2 + 0.248x_0x_3 + 0.002x_0x_5 - 0.582x_1^2 + 0.007x_1x_2 + 0.255x_1x_4 + 0.001x_1x_5 \\
& - 0.050x_2^2 + 0.008x_2x_3 + 0.001x_2x_4 + 0.248x_2x_5 + 0.486x_3^2 - 2.997x_3x_4 - 0.001x_3x_5 \\
& + 0.161x_4^2 - 0.002x_4x_5 - 0.340x_5^2
\end{aligned}$$

$$\begin{aligned}
\dot{x}_3 = {} & -0.027 + 0.004x_0 - 0.003x_1 - 0.012x_2 + 0.001x_3 + 0.001x_4 + 0.002x_5 - 2.958x_0^2 \\
& - 0.013x_0x_2 + 0.750x_0x_3 + 0.019x_0x_5 - 1.610x_1^2 + 0.009x_1x_2 - 0.003x_1x_3 + 0.751x_1x_4 \\
& + 1.986x_1x_5 + 0.198x_2^2 + 0.007x_2x_3 - 2.004x_2x_4 + 0.749x_2x_5 + 1.185x_3^2 - 0.013x_3x_5 \\
& + 0.589x_4^2 + 0.005x_4x_5 - 0.989x_5^2
\end{aligned}$$

$$\begin{aligned}
\dot{x}_4 = {} & -0.434 + 0.024x_0 + 0.008x_1 - 0.001x_2 - 0.002x_3 - 0.006x_4 - 0.015x_5 + 3.462x_0^2 \\
& + 0.002x_0x_1 - 0.039x_0x_2 - 1.182x_0x_3 - 0.005x_0x_4 - 4.975x_0x_5 + 2.168x_1^2 - 0.019x_1x_2 \\
& - 1.179x_1x_4 + 0.033x_1x_5 + 0.455x_2^2 + 5.005x_2x_3 + 0.018x_2x_4 - 1.162x_2x_5 - 1.890x_3^2 \\
& - 0.012x_3x_5 - 0.482x_4^2 - 0.009x_4x_5 + 1.269x_5^2
\end{aligned}$$

$$\begin{aligned}
\dot{x}_5 = {} & -1.775 - 0.015x_0 - 0.022x_1 + 0.121x_2 - 0.032x_3 + 0.009x_4 - 0.035x_5 + 21.145x_0^2 \\
& + 0.013x_0x_1 + 0.016x_0x_2 - 5.838x_0x_3 + 8.978x_0x_4 + 0.010x_0x_5 + 11.874x_1^2 + 0.023x_1x_2 \\
& - 8.993x_1x_3 - 5.863x_1x_4 - 0.580x_2^2 + 0.028x_2x_3 - 0.008x_2x_4 - 5.810x_2x_5 - 6.541x_3^2 \\
& + 0.003x_3x_4 + 0.004x_3x_5 - 2.911x_4^2 - 0.003x_4x_5 + 7.768x_5^2
\end{aligned}$$

ODEFormer

$$\begin{aligned}
\dot{x}_0 &= 0.0093x_0(-0.1332 - x_2)^2 \\
\dot{x}_1 &= -4.8118x_2 \\
\dot{x}_2 &= 2.4147x_1 - 1.3145\sin(-0.1171 + 15.0423x_1) \\
\dot{x}_3 &= -3.6859x_1x_2 \\
\dot{x}_4 &= 0.9808x_2 - 3.7675x_5 \\
\dot{x}_5 &= \frac{0.0105}{-11.23 + 7.7065x_2} + 8.2969x_4 - 2.2755x_1
\end{aligned}$$

SPL

$$\begin{aligned}
\dot{x}_0 &= -0.2x_0 + 4x_1x_2 - 4x_4x_5 \\
\dot{x}_1 &= -7x_0x_2 - 0.5x_1 + 6.99x_3x_5 \\
\dot{x}_2 &= 2.95x_0x_1 - 3.02x_3x_4 \\
\dot{x}_3 &= -2.07x_2x_4 + 0.435 \\
\dot{x}_4 &= -4.97x_0x_5 + 5.0x_2x_3 + 0.045x_2x_5 + 0.025x_3 + 0.032x_4x_5 - 0.993x_4 \\
\dot{x}_5 &= 9.076x_0x_4 - 0.0116x_0 - 8.996x_1x_3 - 1.758x_5
\end{aligned}$$

Vsr-Dpg (ours)

$$\begin{aligned}
\dot{x}_0 &= -0.2x_0 + 4.0x_1x_2 - 4.0x_4x_5 \\
\dot{x}_1 &= -7.0x_0x_2 - 0.5x_1 + 7.0x_3x_5 \\
\dot{x}_2 &= 3.0x_0x_1 - 0.9x_2 - 3.0x_3x_4 \\
\dot{x}_3 &= 2.x_1x_5 - 2.x_2x_4 - 0.40x_4 \\
\dot{x}_4 &= -5.0x_0x_5 + 5.0x_2x_3 - 1.0x_4 + 0.3x_5 \\
\dot{x}_5 &= 9.0x_0x_4 - 9.0x_1x_3 + 0.3x_4 - 1.8x_5
\end{aligned}$$

G. Appendix for Chapter 9

We summarize the supplementary material as follows: Section G.1 offers a detailed explanation of the phase portrait for ODE and the concepts of active learning; Section G.2 provides the extended explanation of the proposed Apps method; Section G.3 details the experimental settings; Section G.3 collects the extra experimental result.

G.1 Extended Preliminaries

Phase Plane. The phase plane is a visual display of solutions of differential equations. Given an ODE, its solutions are a family of functions, which can be graphically plotted in the phase plane. At point (x_1, x_2), we draw a vector representing the derivatives of the point with respect to the time variable, that is $(dx_1/dt, dx_2/dt)$. With sufficient of these arrows in place the system behavior over the regions of the place can be visualized and the long-term behavior can be quantitatively determined, like the limit cycles and attractors. The obtained entire figure is known as the *phase portrait*. It is a geometric representation of all possible trajectories from the corresponding ODE. One can interpret the phase plane in terms of dynamics. The solution of ODE corresponds to a trajectory of a point moving on the phase plane with velocity.

Butterfly Effect. In the context of ordinary differential equations (ODEs), the butterfly effect implies the *sensitive dependence on initial conditions* in dynamical systems. It means that small differences in the initial state of a system can lead to vastly different trajectories over time.

Mathematically, let $\mathbf{x}_0$ and $\mathbf{x}'_0$ be two initial conditions that are very close to each other, i.e., $\|\mathbf{x}_0 - \mathbf{x}'_0\| \leq \delta$. The butterfly effect refers to the situation where the distance between the corresponding trajectories, $\mathbf{x}(t)$ and $\mathbf{x}'(t)$, grows exponentially over time:

$$\|\mathbf{x}(t) - \mathbf{x}'(t)\| \approx \exp(\lambda t)\|\mathbf{x}_0 - \mathbf{x}'_0\|,$$

where λ quantifies the rate at which two nearby trajectories diverge. If λ is positive, small initial differences grow exponentially over time, implying that even a tiny perturbation (like a butterfly flapping its wings) can lead to dramatically different outcomes in a chaotic system.

In this research, where the task is to query informative data to rank a given list of ODEs, this phenomenon implies the drawn data (i.e., initial conditions) are less informative but its close neighbor is highly informative. To avoid this, we can consider evaluating more initial conditions, which demand huge space usage and slow time computation.

Different trajectories never intersect in the phase plane. Solutions of ODEs are uniquely defined by initial conditions except at some special points in the phase plane. Trajectories in the phase plane cannot cross (except at some special points) as this would be equivalent to non-uniqueness of solutions. The special points are fixed points or singular points where trajectories start or end.

The existence and uniqueness theorem has an important corollary: different trajectories never intersect. If two trajectories did intersect, then there would be two solutions starting from the same point (the crossing point), and this would violate the uniqueness part of the theorem. In other words, a trajectory cannot move in two directions at once. Because trajectories cannot intersect, phase portraits always have a well-groomed look to them.

Theorem G.1.1 (Existence and Uniqueness [308]). *Consider the initial value problem* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$*,* $\mathbf{x}(0) = \mathbf{x}_0$*. Suppose that* $\mathbf{f}$ *is continuous and that all its partial derivatives* $\partial f_i / \partial x_i$*,* $i, j = 1, \ldots, n$*, are continuous for* $\mathbf{x}$ *in some open connected set* $D \subset \mathbb{R}^n$*. Then for* $\mathbf{x}_0 \in D$*, the initial value problem has a solution* $\mathbf{x}(t)$ *on some time interval* (a, b) *about* $t = 0$*, and the solution is unique.*

Active Learning is a machine learning method in which the learning algorithm inspects unlabeled data and interactively chooses the most informative data points to learn. The goal of active learning algorithms is to learn an accurate predictor with as little total data as possible. There are two standard settings: pool-based and streaming-based settings. In the pool-based setting, the learner is provided with a pool of unlabeled data, from which the interactively selects the most informative points and asks for their label. In the streaming-based setting, the learner receives a sequence of unlabeled points and decides on whether to request the label of the current point. In this research, we only consider pool-based active learning algorithms.

According to the active learning [270, 273, 286, 309], the input is the initial condition $\mathbf{x}_0 \in \mathbb{R}^n$ and the output is the obtained trajectory $(\mathbf{x}_{t_1}, \ldots, \mathbf{x}_{t_k}) \in \mathbb{R}^{n\times k}$. In the formulation of the Query-by-Committee method, they define the uncertainty at an input point as the variance of the predictions of the committee members.

G.2 Extended Explanation of Apps method

Data-availability Assumption. A crucial assumption behind the success of Apps is the availability of a Data Oracle $\mathcal{O}$ that returns a (noisy) observation of the trajectory with a specified initial condition and a sequence of discrete times. Such a data oracle represents conducting controlled experiments in the real world, which can be expensive. This differs from the current symbolic regression, where a dataset is obtained prior to learning.

Vocabulary Construction. Given the set of math operators and variables:

$$O_m = \{+, -, \times, \div, \sin \ldots\} \cup \{x_1, \ldots, x_n, \texttt{const}\},$$

where `const` indidates the coefficients in the expressions. Following the definition of grammar in Section 9.3. For example, given $O_m = \{+, -, \times, \div\} \cup \{x_1, x_2, \texttt{const}\}$, we construct the following grammar rules:

```
A -> (A + A)
A -> (A - A)
A -> A * A
A -> A / A
A -> x1
A -> x2
A -> const
B -> (B + B)
B -> (B - B)
```

```
B -> B * B

B -> B / B

B -> x1

B -> x2

B -> const
```

The non-terminal symbol "A" denotes a subexpression in dx_1 and similarly "B" denotes a subexpression in dx_2. Each of them will be a unique token of the input vocabulary of the decoder and will be mapped to a distinct vector in the first embedding layer of the decoder. The above rules also form the output vocabulary of the decoder, where the neural decoder predicts a categorical distribution over the list of grammar rules.

The discovery path is to build algorithms that mimic human scientific discovery, which has achieved tremendous success in early works [189–191]. Recent work [275–277, 286] also pointed out the importance of having a data oracle that can actively query data points, rather than learning from a fixed dataset.

Sequential Decision Making Formulation. We consider an undiscounted MDP with a finite horizon. The state space $\mathcal{S}$ is the set of all possible sequences of rules with maximum steps. The action space $\mathcal{A}$ is the set of grammar rules. The t-th step state s_t is the sequence of sampled rules before the current step t, i.e., $s_t := (s_1, \ldots, s_t)$. The action a_t is the sampled single rule, $a_t := s_{t+1}$.

The loss function of Apps is informed by the REINFORCE algorithm [239], which is based on the log-derivative property:

$$\nabla_\theta p_\theta(s) = p_\theta(s) \nabla_\theta \log p_\theta(s)$$

where $p_\theta(s) \in (0,1)$ represents a probability distribution over input s with parameters θ and notation ∇_θ is the partial derivative with respect to θ. In our formulation, let $p_\theta(s)$ denote the probability of sampling a sequence of grammar rules s and $\texttt{reward}(s) = 1/(1+\texttt{NMSE}(\phi))$. Here ϕ is the corresponding expression constructed from the rules s following the procedure in Section 9.3. The probability $p_\theta(s)$ is modeled by the decoder modules. The objective described in Equation 9.1 is translated to maximize the expected reward of the sampled sequences from the decoder:

$$\arg\max_{\theta}\ \mathbb{E}_{s\sim p_\theta(s)}[\texttt{reward}(s)]$$

Based on the REINFORCE algorithm, the gradient of the objective can be expanded as:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{s\sim p_\theta(s)}[\texttt{reward}(s)] &= \nabla_\theta \sum_{s\in\Sigma} \texttt{reward}(s) p_\theta(s) \\
&= \sum_{s\in\Sigma} \texttt{reward}(s) \nabla_\theta p_\theta(s) \\
&= \sum_{s\in\Sigma} \texttt{reward}(s) p_\theta(s) \frac{\nabla_\theta p_\theta(s)}{p_\theta(s)} \\
&= \sum_{s\in\Sigma} \texttt{reward}(s) p_\theta(s) \nabla_\theta \log p_\theta(s) \\
&= \mathbb{E}_{s\sim p_\theta(s)} \left[\texttt{reward}(s) \nabla_\theta \log p_\theta(s)\right]
\end{aligned}$$

where Σ represents all possible sequences of grammar rules sampled from the decoder. The above expectation can be estimated by computing the averaged over samples drawn from the distribution $p_\theta(s)$. We first sample several times from the decoder module and obtain N sequences $(s^1, \dots, s^N)$, an unbiased estimation of the gradient of the objective is computed as: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \texttt{reward}(s^i) \nabla_\theta \log p_\theta(s^i)$. In practice, the above computation has a high variance. To reduce variance, it is common to subtract a baseline function b from the reward. In this study, we choose the baseline function as the average of the reward of the current sampled batch expressions. Thus we have:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} (\texttt{reward}(s^i) - b) \nabla_\theta \log p_\theta(s^i),$$

where $b = \sum_{i=1}^{N} \texttt{reward}(s^i)$. Based on the description of the execution pipeline of the proposed Apps, we summarize every step in Algorithm 10.

Algorithm 10: Active Discovery of Ordinary Differential Equations via Phase Portrait Sketching.

```
Input: Defined expression grammar; neural sequential decoder; data oracle for the
       ground-truth ODE $\mathcal{O}$; maximum learning epoch #epochs; discrete time
       steps $T = (t_1, \ldots, t_k)$.
Output: The best-predicted ODE.
initialize the set of best predicted ODEs $\mathcal{Q} \leftarrow \emptyset$; ;     // initialization
randomly draw data $D$ from Oracle $\mathcal{O}$;
for $t \leftarrow 1$ to #epochs do
    //in Figure 9.2(a);
    sample $N$ sequences $\{s_1, \ldots, s_N\}$ from the sequential decoder;
    //in Figure 9.2(b);
    construct $N$ ODEs $\{\phi_i\}_{i=1}^N$ for each sequence from defined grammar;
    fit coefficients in each expression with data $c_i \leftarrow$ Optimize$(\phi_i, D)$;
    //in Figure 9.2(c);
    find a region $u$ of high uncertainty with phase portrait sketching;
    draws trajectory data from Oracle in the selected region $D_u \leftarrow \mathcal{O}(u, T)$;
    computes reward using data $D_u$;
    save all expressions into $\mathcal{Q}$ ;
    save new data $D_u$ into $D$;
    applies policy gradient to the parameters of the decoder;
return the expression in $\mathcal{Q}$ with best goodness-of-fit on data $D$.
```

G.2.1 Implementation of Apps

In the experiments, we use an embedding layer, a multi-head self-attention layer, and finally softmax layer as the decoder. The dimension of the input embedding layer and the hidden vector is configured as 256. We use the Adam optimizer as the gradient descent algorithm with a learning rate of 0.009. The learning epoch is configured as 50. The maximum sequence of grammar rules is fixed to be 20. The batch size of expressions sampled from the decoder is set as 100.

When fitting the values of coefficients in each expression, we sample a batch of data with batch size 1024 from the data Oracle. The open constants in the expressions are fitted on the data using the BFGS optimizer[1].

[1] ↑https://docs.scipy.org/doc/scipy/reference/optimize.minimize-bfgs.html

```python
import numpy as np
from sympy.parsing.sympy_parser import parse_expr
from sympy import lambdify, symbols

expr_odes = [parse_expr(one_expr) for one_expr in expr_strs]
t = symbols('t')
func = lambdify((t, input_var_Xs), expr_odes)
pred_trajectories = []
for one_x_init in x_init_conds:
    one_solution = runge_kutta4(func, t_evals, one_x_init)
    pred_trajectories.append(one_solution)
pred_trajectories = np.asarray(pred_trajectories)
```

Figure G.1. Implemented 4th order Runge Kutter method.

We use a multi-processor Python library `pathos` to fit multiple expressions in parallel using 20 CPU cores. This greatly reduced the total time of the coefficients fitting step. The rest of the implementation details are available in the code implementation.

In terms of numerical integration, we use the fourth-order Runge–Kutta (RK45) method to compute the trajectory data of the specified ODEs. Other numerical integration algorithms in `sicpy.integrate` are implemented with adaptive time step size, which makes it very slow when fitting the coefficients in the candidate ODEs.

Every candidate ODE is represented by the Sympy Lambdify function. We implement the Runge-Kutta function instead of using the `Scipy.integrate.solve_ivp` function, where the latter has internally used the adaptive step size and is extremely slow for specific candidate ODEs. In our experiments, we find that the `Scipy.integrate.solve_ivp` API cannot return a trajectory when running together with the coefficient fitting steps for more than 12 hours.

In Figure G.1, given one predicted ODE `expr_strs`, which is represented as an array of expressions of length n, the following lines of code compute the predicted trajectory `pred_trajectories` for a batch of initial conditions `x_init_conds`. `input_var_Xs` the list of symbols of variables: $[x_1, \dots, x_n]$. We can then compare if the predicted trajectories are close to the ground-truth trajectories using the NMSE metric, given the same set of initial conditions.

Hyper-parameter Configuration. An expression containing placeholder symbol A or containing more than 20 open constants is not evaluated on the data, the fitness score of it is $-\infty$. In terms of the reward function in the policy gradient objective, we use $\texttt{reward}(s) = \frac{1}{1+\texttt{NMSE}(\phi)}$. The normalized mean-squared error metric is further defined in Equation E.2. We set the relative size of the interval of the region to be 1/4, meaning the length of every edge of the region is 1/4 of the original interval for each variable. We set the number of regions to be 10. The region is randomly generated by first generating the leftmost point in the original variable interval and then applying each edge of the region with a given relative length.

The deep network part is implemented using the most recent version of Pytorch, the expression evaluation is based on the Sympy library, and the step for fitting open constants in expression with the dataset uses the Scipy library. The visualization of the phase portrait is from [2] and the part of the experimental visualization is borrowed from [3].

G.2.2 Limitation and Broader Impact

Limitation The proposed Apps that generates phase portraits can suffer from resolution issues. Fine details of the dynamics might be missed if the region width is too large or the number of sampled points in each region is too small. It is also unclear if the proposed phase portrait sketching idea is applicable to partial differential equations.

Broad Impact The proposed Apps can be useful for scientists to actively discover governing laws from data. It will accelerate the discovery process compared to passive learning algorithms.

G.3 Experiment Settings

G.3.1 Baselines

For the baselines of the ODE discovery task, we consider

[2] ↑https://phaseportrait.github.io/

[3] ↑https://github.com/sdascoli/odeformer/blob/main/ODEFormer_demo.ipynb

- SINDy [214][4] is a popular method using a sparse regression algorithm to find the differential equations.

- ProGED [264][5] uses probabilistic context-free grammar to search for differential equations. ProGED first samples a list of candidate expressions from the defined probabilistic context-free grammar for symbolic expressions. Then ProGED fits the open constants in each expression using the given training dataset. The equation with the best fitness scores is returned.

- ODEFormer [266][6] is the most recent framework that uses the transformer for the discovery of ordinary differential equations. We use the provided pre-trained model to predict the governing expression with the dataset. We execute the model 10 times and pick the expression with the smallest NMSE error. The dataset size is 500, which is the largest dataset configuration for the ODEFormer.

In principle, the ODE discovery task can be formulated as a symbolic regression task where the input is $\mathbf{x}_t$ and the output is directly $\dot{\mathbf{x}}_t$. Given the trajectory data $(\mathbf{x}_0, \mathbf{x}(t_1), \ldots, \mathbf{x}(t_n))$, its output label is approximated by computing:

$$\dot{\mathbf{x}}(t_i) \approx \frac{(\mathbf{x}(t_{i+1}) - \mathbf{x}(t_i))}{(t_{i+1} - t_i)}$$

In literature, this approach is called symbolic regression with gradient matching. We consider two representative baselines in the symbolic regression task:

- Symbolic Physics Learner (SPL) is a heuristic search algorithm based on Monte Carlo Tree Search for finding optimal sequences of production rules using context-free grammars [240, 258][7]. It employs Monte Carlo simulations to explore the search space of all the production rules and determine the value of each node in the search tree. SPL consists of four steps in each iteration: 1) Selection. Starting at a root node, recursively select the optimal child (*i.e.*, one of the production rules) until reaching an expandable node or a leaf node.

[4] ↑https://github.com/dynamicslab/pysindy

[5] ↑https://github.com/brencej/ProGED

[6] ↑https://github.com/sdascoli/odeformer

[7] ↑https://github.com/isds-neu/SymbolicPhysicsLearner

	Apps	ProGED	ODEFormer	SPL
goodness-of-fit function	NegMSE	NegMSE	NegRMSE	NegRMSE
Training setting	1 sec with step-size 0.001 sec and 100 random initial conditions			
Testing setting	10 sec with step-size 0.001 sec and 100 random initial conditions			
CPUs Type	CPU is Milan CPUs @ 2.45GHz			
#CPU for training	20			

Table G.1. Major hyper-parameters settings for all the algorithms considered in the experiment.

2) Expansion. If the expandable node is not the terminal, create one or more of its child nodes to expand the search tree. 3) Simulation. Run a simulation from the new node until achieving the result. 4) Backpropagation. Update the node sequence from the new node to the root node with the simulated result. To balance the selection of optimal child node(s) by exploiting known rewards (exploitation) or expanding a new node to explore potential rewards exploration, the upper confidence bound (UCB) is often used.

- End to End Transformer for symbolic regression (E2ETransformer) [208][8]. They propose to use a deep transformer to pre-train on a large set of randomly generated expressions. We load the shared pre-trained model. We provide the given dataset and the E2ETransformer infers 10 best expressions. We choose to report the expression with the best NMSE scores.

Note that the open link in NSODE [310] has no code implementation by far. Thus, the NSODE approach is not included for comparison. Also, a recent method [273] proposes a new learning framework for actively drawing data. However, their approaches are designed for searching algebraic equations, and the integration into differential equations is unclear. So their approach is also not considered for comparison in this work.

G.3.2 Evaluation Metrics

The goodness-of-fit indicates how well the learning algorithms perform in discovering unknown ODEs. The testing set contains new trajectories with new initial conditions, generated from the ground-truth ODE. we measure the goodness-of-fit of a predicted expres-

[8] ↑https://github.com/facebookresearch/symbolicregression

sion $\phi = [\phi_1, \dots, \phi_m]$, by evaluating the mean-squared-error (MSE) and normalized-mean-squared-error (NMSE):

$$\text{NMSE} = \frac{1}{\sigma^2}\frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}(t_i) - \hat{\mathbf{x}}(t_i))^2, \tag{G.1}$$

The empirical variance $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i - \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\right)^2}$. We use the NMSE as the main criterion for comparison in the experiments and present the results on the remaining metrics in the case studies. The main reason is that the NMSE is less impacted by the output range. The output ranges of expression are dramatically different from each other, making it difficult to present results uniformly if we use other metrics.

Prior work [180] further proposed a coefficient of determination R^2-based score over a group of expressions in the dataset, as a statistical measure of whether the best-predicted expression is almost close to the ground-truth expression. An R^2 of 1 indicates that the regression predictions perfectly fit the data [299]. The R^2 score is computed as follows:

$$R^2(\phi_i) = 1 - \texttt{NMSE}(\phi_i) \tag{G.2}$$

G.3.3 Computational Resource

All the methods are running with Python 3.10 and on the same set of hardware where the CPU is Milan CPUs @ 2.45GHz, the RAM is set as 8GB, and the maximum running time is set as 24 hours. The extra necessary configuration is collected in the anonymous code repository. The hyper-parameter configurations of baselines are listed in Table G.1.

G.3.4 Extended Experimental Results

The given set of best-predicted ODEs for Table 9.3 is shown in Figure G.2.

The Kendall tau distance is computed with two ranked lists, one of them is evaluated on full data and another is evaluated on the chosen region with high uncertainty. We use library[9] to compute the ranking score.

[9] ↑https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html

$$
\begin{aligned}
\phi_1 &= (0.088x_1, 0.146\cos(x_1)) \\
\phi_2 &= (-0.0107\sin(x_1), 0.712) \\
\phi_3 &= (2/\cos(x_1) + 0.1759\sin(x_1), 0.820) \\
\phi_4 &= (0.645\sin(x_0) - 0.893\cos(x_0), \\
&\quad 0.2891\frac{x_1}{(x_1 - 0.153\cos(x_0))} + 0.213\frac{\sin(x_0)}{(x_1 - 0.153\cos(x_0))} - 0.0443\frac{\cos(x_0)}{(x_1 - 0.15\cos(x_0))}) \\
\phi_5 &= (0.95\sin(x_0) - 0.29, 0.14153x_1 - 0.691\frac{\sin(x_1)}{\cos(x_0)} - 0.0673\cos(x_0) + 0.8825\frac{\cos(x_1)}{\cos(x_0)}) \\
\phi_6 &= (1.63\sin(x_0), x_0) \\
\phi_7 &= (1.15\sin(x_0), -70.56\cos(x_1)) \\
\phi_8 &= (0.71, -70.56\cos(x_1)) \\
\phi_9 &= (0.99\cos(x_0), -1.0x_0 - 39.53\sin(x_1) - 152.57\cos(x_1)) \\
\phi_{10} &= (-0.871\sin(x_1), -3.826\sin(x_0) - 0.212\sin(x_1) + 4.311) \\
\phi_{11} &= (0.088x_1, 0.146\cos(x_1)) \\
\phi_{12} &= (-0.010\sin(x_1), 0.712) \\
\phi_{13} &= (1.477e - 5x_12/\cos(x_1) + 0.1759\sin(x_1), 0.8205) \\
\phi_{14} &= (0.645\sin(x_0) - 0.893\cos(x_0), \\
&\quad \frac{0.289x_1}{(x_1 - 0.15\cos(x_0))} + \frac{0.21\sin(x_0)}{(x_1 - 0.153\cos(x_0))} - \frac{0.044\cos(x_0)}{(x_1 - 0.1532\cos(x_0))}) \\
\phi_{15} &= (0.953\sin(x_0) - 0.2922, 0.1415x_1 - 0.691\frac{\sin(x_1)}{\cos(x_0)} - 0.0673\cos(x_0) + 0.882\frac{\cos(x_1)}{\cos(x_0)}) \\
\phi_{16} &= (1.631\sin(x_0), x_0) \\
\phi_{17} &= (1.152\sin(x_0), -70.56\cos(x_1)) \\
\phi_{18} &= (0.710, -70.56\cos(x_1)) \\
\phi_{19} &= (0.99\cos(x_0), \\
&\quad - 1.0x_0 - 39.53\sin(x_1) - 152.57\cos(x_1)) \\
\phi_{20} &= (-0.871\sin(x_1), -3.826\sin(x_0) - 0.211\sin(x_1) + 4.3117)
\end{aligned}
$$

Figure G.2. The given set of best-predicted ODEs for Table 9.3.

G.3.5 Dataset

We present the visualized phase portraits and the explicit forms of the ODEs considered in this research in the following figures and tables.

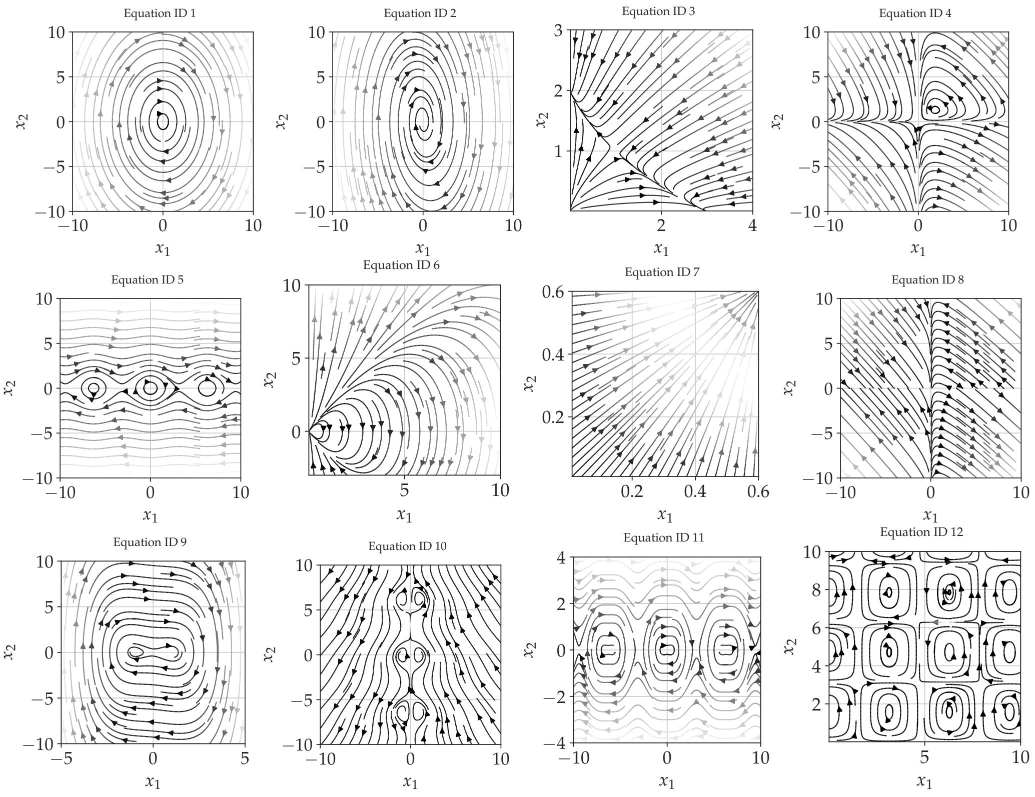


Figure G.3. Selected phase portrait of Strogatz dataset with variables $n = 2$.

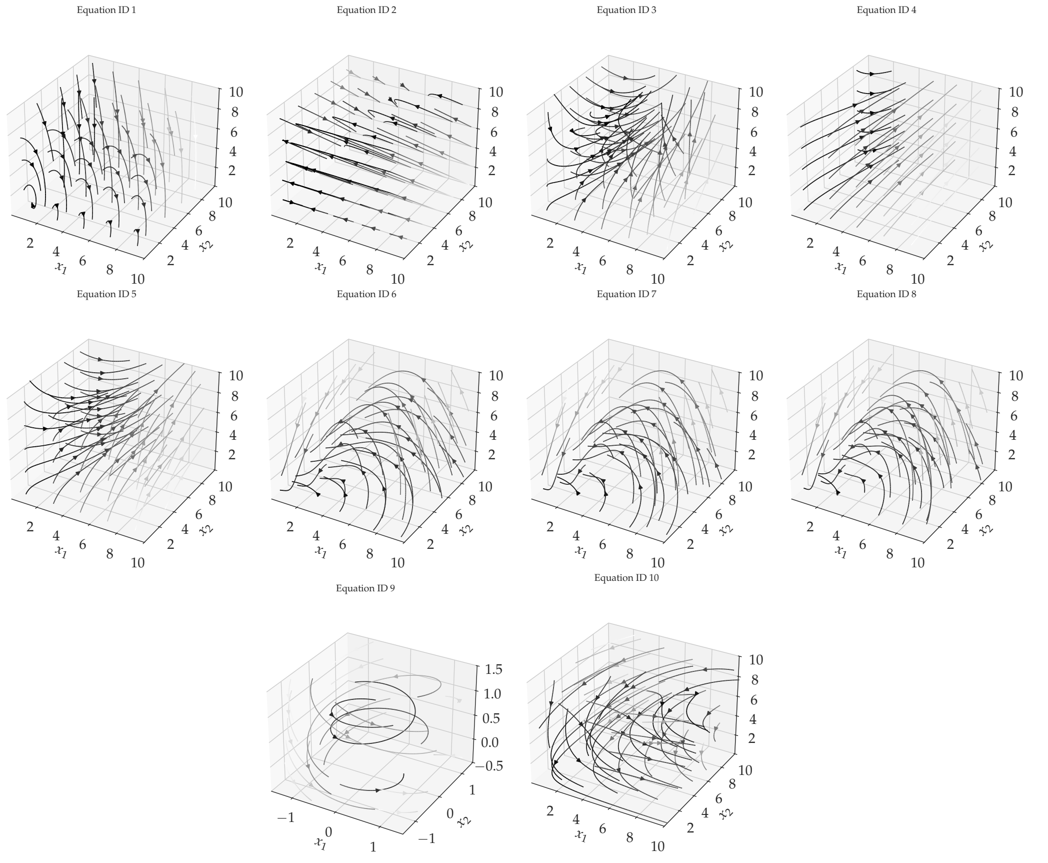


Figure G.4. Selected phase portrait of Strogatz dataset with variables $n = 3$.

ID	Equation
1	RC-circuit (charging capacitor) $\dot{x}_0 = (0.7 - x_0/1.2)/2.31$
2	Population growth (naive) $\dot{x}_0 = 0.23x_0$
3	Population growth with carrying capacity $\dot{x}_0 = 0.79x_0(1 - x_0/74.3)$
4	RC-circuit with non-linear resistor (charging capacitor) $\dot{x}_0 = 1/(1 + \exp(0.5 - x_0/0.96)) - 0.5$
5	Velocity of a falling object with air resistance $\dot{x}_0 = 9.81 - 0.0021175x_0^2$
6	Autocatalysis with one fixed abundant chemical $\dot{x}_0 = 2.1x_0 - 0.5x_0^2$
7	Gompertz law for tumor growth $\dot{x}_0 = 0.032x_0 \log(2.29x_0)$
8	Logistic equation with Allee effect $\dot{x}_0 = 0.14x_0(1 - x_0/130.0)(x_0/4.4 - 1)$
9	Language death model for two languages $\dot{x}_0 = (1 - x_0)0.32 - x_0 0.28$
10	Refined language death model for two languages $\dot{x}_0 = (1 - x_0)0.2x_0^{1.2} - x_0(1 - 0.2)(1 - x_0)^{1.2}$
11	Naive critical slowing down (statistical mechanics) $\dot{x}_0 = -x_0^3$
12	Photons in a laser (simple) $\dot{x}_0 = 1.8x_0 - 0.1107x_0^2$
13	Overdamped bead on a rotating hoop $\dot{x}_0 = 0.0981 \sin(x_0)(9.7 \cos(x_0) - 1)$
14	Budworm outbreak model with predation $\dot{x}_0 = 0.78x_0(1 - x_0/81.0) - 0.9x_0^2/(21.2^2 + x_0^2)$
15	Budworm outbreak with predation (dimensionless) $\dot{x}_0 = 0.4x_0(1 - x_0/95.0) - x_0^2/(1 + x_0^2)$
16	Landau equation (typical time scale tau = 1) $\dot{x}_0 = 0.1x_0 - -0.04x_0^3 - 0.001x_0^5$
17	Logistic equation with harvesting/fishing $\dot{x}_0 = 0.4x_0(1 - x_0/100.0) - 0.3$
18	Improved logistic equation with harvesting/fishing $\dot{x}_0 = 0.4x_0(1 - x_0/100.0) - 0.24x_0/(50.0 + x_0)$
19	Improved logistic equation with harvesting/fishing (dimensionless) $\dot{x}_0 = x_0(1 - x_0) - 0.08x_0/(0.8 + x_0)$
20	Autocatalytic gene switching (dimensionless) $\dot{x}_0 = 0.1 - 0.55x_0 + x_0^2/(1 + x_0^2)$
21	Dimensionally reduced SIR infection model for dead people (dimensionless) $\dot{x}_0 = 1.2 - 0.2x_0 - \exp(-x_0)$

Table G.2. Selected Strogatz dataset with variable $n = 1$.

ID	Explicit Equations
1	Harmonic oscillator without damping $\dot{x}_0 = x_1$ $\dot{x}_1 = -2.1x_0$
2	Harmonic oscillator with damping $\dot{x}_0 = x_1$ $\dot{x}_1 = -4.5x_0 - 0.43x_1$
3	Lotka-Volterra competition model (Strogatz version with sheeps and rabbits) $\dot{x}_0 = x_0(3.0 - x_0 - 2.0x_1)$ $\dot{x}_1 = x_1(2.0 - x_0 - x_1)$
4	Lotka-Volterra simple (as on Wikipedia) $\dot{x}_0 = x_0(1.84 - 1.45x_1)$ $\dot{x}_1 = -x_1(3.0 - 1.62x_0)$
5	Pendulum without friction $\dot{x}_0 = x_1$ $\dot{x}_1 = -0.9\sin(x_0)$
6	Dipole fixed point $\dot{x}_0 = 0.65x_0x_1$ $\dot{x}_1 = x_1^2 - x_0^2$
7	RNA molecules catalyzing each others replication $\dot{x}_0 = x_0(x_1 - 1.61x_0x_1)$ $\dot{x}_1 = x_1(x_0 - 1.61x_0x_1)$
8	SIR infection model only for healthy and sick $\dot{x}_0 = -0.4x_0x_1$ $\dot{x}_1 = 0.4x_0x_1 - 0.314x_1$
9	Damped double well oscillator $\dot{x}_0 = x_1$ $\dot{x}_1 = -0.18x_1 + x_0 - x_0^3$
10	Glider (dimensionless) $\dot{x}_0 = -\sin(x_1) - 0.08x_0^2$ $\dot{x}_1 = x_0 - \cos(x_1)/x_0$
11	Frictionless bead on a rotating hoop (dimensionless) $\dot{x}_0 = x_1$ $\dot{x}_1 = \sin(x_0)(\cos(x_0) - 0.93)$
12	Rotational dynamics of an object in a shear flow $\dot{x}_0 = \cot(x_1)\cos(x_0)$ $\dot{x}_1 = \sin(x_0)(\cos(x_1)^2 + 4.2\sin(x_1)^2)$
13	Pendulum with non-linear damping, no driving (dimensionless) $\dot{x}_0 = x_1$ $\dot{x}_1 = -\sin(x_0) - x_1 - 0.07\cos(x_0)x_1$
14	Van der Pol oscillator (standard form) $\dot{x}_0 = x_1$ $\dot{x}_1 = -x_0 - 0.43(x_0^2 - 1)x_1$
15	Van der Pol oscillator (simplified form from Strogatz) $\dot{x}_0 = 3.37(x_1 - x_0^3/3 + x_0)$ $\dot{x}_1 = -x_0/3.37$
16	Glycolytic oscillator, e.g., ADP and F6P in yeast (dimensionless) $\dot{x}_0 = -x_0 + 2.4x_1 + x_0^2x_1$ $\dot{x}_1 = 0.07 - 2.4x_0 - x_0^2x_1$
17	Duffing equation (weakly non-linear oscillation) $\dot{x}_0 = x_1$ $\dot{x}_1 = -x_0 + 0.886x_1(1 - x_0^2)$
18	Cell cycle model by Tyson for interaction between protein cdc2and cyclin (dimensionless) $\dot{x}_0 = 15.3(x_1 - x_0)(0.001 + x_0^2) - x_0$ $\dot{x}_1 = 0.3 - x_0$
19	Reduced model for chlorine dioxide-iodine-malonic acid rection (dimensionless) $\dot{x}_0 = 8.9 - x_0 - 4.0x_0x_1/(1 + x_0^2)$ $\dot{x}_1 = 1.4x_0(1 - x_1/(1 + x_0^2))$
20	Driven pendulum with linear damping / Josephson junction (dimensionless) $\dot{x}_0 = x_1$ $\dot{x}_1 = 1.67 - \sin(x_0) - 0.64x_1$
21	Driven pendulum with quadratic damping (dimensionless) $\dot{x}_0 = x_1$ $\dot{x}_1 = 1.67 - \sin(x_0) - 0.64x_1^1$

Table G.3. Selected Strongatz dataset with variables $n = 2$.

ID	Explicit Equations
1	Maxwell-Bloch equations (laser dynamics) $\dot{x}_0 = 0.1(x_1 - x_0)$ $\dot{x}_1 = 0.21(x_0 x_2 - x_1)$ $\dot{x}_2 = 0.34(3.1 + 1 - x_2 - 3.1 x_0 x_1)$
2	Model for apoptosis (cell death) $\dot{x}_0 = 0.1 - 0.4 x_1 x_0/(0.1 + x_0) - 0.05 x_0$ $\dot{x}_1 = 0.6 x_2 (0.1 + x_1) - 0.2 x_1/(0.1 + x_1) - 7.95 x_0 x_1/(2.0 + x_1)$ $\dot{x}_2 = -0.6 x_2 (0.1 + x_1) + 0.2 x_1/(0.1 + x_1) + 7.95 x_0 x_1/(2.0 + x_1)$
3	Lorenz equations in well-behaved periodic regime $\dot{x}_0 = 5.1(x_1 - x_0)$ $\dot{x}_1 = 12.0 x_0 - x_1 - x_0 x_2$ $\dot{x}_2 = x_0 x_1 - 1.67 x_2$
4	Lorenz equations in complex periodic regime $\dot{x}_0 = 10.0(x_1 - x_0)$ $\dot{x}_1 = 99.96 x_0 - x_1 - x_0 x_2$ $\dot{x}_2 = x_0 x_1 - 2.6666666666666665 x_2$
5	Lorenz equations standard parameters (chaotic) $\dot{x}_0 = 10.0(x_1 - x_0)$ $\dot{x}_1 = 28.0 x_0 - x_1 - x_0 x_2$ $\dot{x}_2 = x_0 x_1 - 2.6666666666666665 x_2$
6	Rössler attractor (stable fixed point) $\dot{x}_0 = 5.0(-x_1 - x_2)$ $\dot{x}_1 = 5.0(x_0 - 0.2 x_1)$ $\dot{x}_2 = 5.0(0.2 + x_2(x_0 - 5.7))$
7	Rössler attractor (periodic) $\dot{x}_0 = 5.0(-x_1 - x_2)$ $\dot{x}_1 = 5.0(x_0 + 0.1 x_1)$ $\dot{x}_2 = 5.0(0.2 + x_2(x_0 - 5.7))$
8	Rössler attractor (chaotic) $\dot{x}_0 = 5.0(-x_1 - x_2)$ $\dot{x}_1 = 5.0(x_0 + 0.2 x_1)$ $\dot{x}_2 = 5.0(0.2 + x_2(x_0 - 5.7))$
9	Aizawa attractor (chaotic) $\dot{x}_0 = x_0(x_2 - 0.7) - 3.5 x_1$ $\dot{x}_1 = 3.5 x_0 + x_1(x_2 - 0.7)$ $\dot{x}_2 = 0.65 + 0.95 x_2 - x_2^3/3. - (x_0^2 + x_1^2)(1 + 0.25 x_2) + 0.1 x_2 x_0^3$
10	Chen-Lee attractor $\dot{x}_0 = 5 x_0 - x_1 x_2$ $\dot{x}_1 = -10.0 x_1 + x_0 x_2$ $\dot{x}_2 = -3.8 x_2 + x_0 x_1/3.0$

Table G.4. The Strogatz dataset with variables $n = 3$.

ID	Equations
1	Norel1990 - MPF and Cyclin Oscillations $\dot{x}_0 = 1.0x_0^2x_1 - 10.0x_0/(x_0 + 1.0) + 3.466x_1$ $\dot{x}_1 = 1.2 - 1.0x_0$
2	Chrobak2011 - A mathematical model of induced cancer-adaptive immune system competition $\dot{x}_0 = -0.03125x_0^2 - 0.125x_0x_1 + 0.0625x_0$ $\dot{x}_1 = -0.08594x_0x_1 - 0.03125x_1^2 + 0.03125x_1$
3	FitzHugh1961-NerveMembrane $\dot{x}_0 = -1.0x_0^3 + 3.0x_0 + 3.0x_1 - 1.2$ $\dot{x}_1 = -0.3333x_0 - 0.2667x_1 + 0.2333$
4	Clarke2000 - One-hit model of cell death in neuronal degenerations $\dot{x}_0 = -0.278x_0$ $\dot{x}_1 = -0.223x_1$
5	Wodarz2018/1 - simple model $\dot{x}_0 = 0.004x_0 + 0.004x_1/(0.01x_0^1.0 + 1.0)$ $\dot{x}_1 = 0.006x_0 - 0.003x_1 - 0.004x_1/(0.01x_0^1.0 + 1.0)$
6	Ehrenstein2000 - Positive-Feedback model for the loss of acetylcholine in Alzheimer's disease $\dot{x}_0 = -0.007x_0x_1$ $\dot{x}_1 = -0.004x_0 - 0.01x_1 + 0.33$
7	Cao2013 - Application of ABSIS method in the bistable Schlagl model $\dot{x}_0 = 0.12x_0^2 - 3.071x_0 + 12.5 - 0.00192/x_0$ $\dot{x}_1 = -0.12x_0^2 + 3.071x_0 - 12.5 + 0.00192/x_0$
8	Chaudhury2020 - Lotka-Volterra mathematical model of CAR-T cell and tumour kinetics $\dot{x}_0 = 0.002x_0x_1 - 0.16x_0$ $\dot{x}_1 = 0.15x_1$
9	Baker2013 - Cytokine Mediated Inflammation in Rheumatoid Arthritis $\dot{x}_0 = -x_0 + 3.5x_1^2/(x_1^2 + 0.25)$ $\dot{x}_1 = 1.0x_1^2/(x_0^2x_1^2 + x_0^2 + x_1^2 + 1.0) - 1.25x_1 + 0.025/(x_0^2 + 1.0)$
10	Somogyi1990-CaOscillations $\dot{x}_0 = -5.0x_0x_1^4.0/(x_1^4.0 + 81.0) - 0.01x_0 + 2.0x_1$ $\dot{x}_1 = 5.0x_0x_1^4.0/(x_1^4.0 + 81.0) + 0.01x_0 - 3.0x_1 + 1.0$
11	Cucuianu2010 - A hypothetical-mathematical model of acute myeloid leukaemia pathogenesis $\dot{x}_0 = -0.1x_0 + 0.3x_0/(0.5x_0 + 0.5x_1 + 1.0)$ $\dot{x}_1 = -0.1x_1 + 0.3x_1/(0.5x_0 + 0.5x_1 + 1.0)$
12	Wang2016/3 - oncolytic efficacy of M1 virus-SN model $\dot{x}_0 = -0.2x_0x_1 - 0.02x_0 + 0.02$ $\dot{x}_1 = 0.16x_0x_1 - 0.03x_1$
13	Chen2011/1 - bone marrow invasion absolute model $\dot{x}_0 = -0.2x_0^2 + 0.1x_0$ $\dot{x}_1 = -1.0x_0x_1 - 0.8x_1^2 + 0.7x_1$
14	Cao2013 - Application of ABSIS method in the reversible isomerization model $\dot{x}_0 = -0.12x_0 + 1.0x_1$ $\dot{x}_1 = 0.12x_0 - 1.0x_1$

Table G.5. Selected ODEBase dataset with variables $n = 2$.

1	Turner2015-Human/Mosquito ELP Model $\dot{x}_0 = 600.0 - 0.411x_0$ $\dot{x}_1 = 0.361x_0 - 0.184x_1$ $\dot{x}_2 = 0.134x_1 - 0.345x_2$
2	Al-Tuwairqi2020 - Dynamics of cancer virotherapy - Phase I treatment $\dot{x}_0 = -1.0x_0x_2$ $\dot{x}_1 = 1.0x_0x_2 - 1.0x_1$ $\dot{x}_2 = -0.02x_0x_2 + 1.0x_1 - 0.15x_2$
3	Fassoni2019 - Oncogenesis encompassing mutations and genetic instability $\dot{x}_0 = 0.01 - 0.01x_0$ $\dot{x}_1 = 0.03x_1$ $\dot{x}_2 = -0.5x_2^2 + 0.034x_2$
4	Zatorsky2006-p53-Model5 $\dot{x}_0 = -3.7x_0x_1 + 2.0x_0$ $\dot{x}_1 = -0.9x_1 + 1.1x_2$ $\dot{x}_2 = 1.5x_0 - 1.1x_2$
5	Lenbury2001-InsulinKineticsModel-A $\dot{x}_0 = -0.1x_0x_2 + 0.2x_1x_2 + 0.1x_2$ $\dot{x}_1 = -0.01x_0 + 0.01 + 0.01/x_2$ $\dot{x}_2 = -0.1x_1x_2 + 0.257x_1 - 0.1x_2^2 + 0.331x_2 - 0.3187$
6	Zatorsky2006-p53-Model1 $\dot{x}_0 = -3.2x_0x_1 + 0.3$ $\dot{x}_1 = -0.1x_1 + 0.1x_2$ $\dot{x}_2 = 0.4x_0 - 0.1x_2$
7	Smallbone2013 - Colon Crypt cycle - Version 1 $\dot{x}_0 = -0.002207x_0^2 - 0.002207x_0x_1 - 0.002207x_0x_2 + 0.1648x_0$ $\dot{x}_1 = -0.01312x_0^2 - 0.0216x_0x_1 - 0.01312x_0x_2 + 1.574x_0 - 0.008477x_1^2 - 0.008477x_1x_2 + 0.5972x_1$ $\dot{x}_2 = -0.04052x_0x_1 - 0.04052x_1^2 - 0.04052x_1x_2 + 4.863x_1 - 1.101x_2$
8	Cortes2019 - Optimality of the spontaneous prophage induction rate. $\dot{x}_0 = -0.99x_0^2/(x_0 + x_1) - 1.0x_0x_1/(x_0 + x_1) + 0.99x_0$ $\dot{x}_1 = -0.99x_0x_1/(x_0 + x_1) - 1.0x_1^2/(x_0 + x_1) + 1.0x_1$ $\dot{x}_2 = -0.001x_2$
9	Figueredo2013/2 - immunointeraction model with IL2 $\dot{x}_0 = -1.0x_0x_1/(x_0 + 1.0) + 0.18x_0$ $\dot{x}_1 = 0.05x_0 + 0.124x_1x_2/(x_2 + 20.0) - 0.03x_1$ $\dot{x}_2 = 5.0x_0x_1/(x_0 + 10.0) - 10.0x_2$
10	A mathematical model of cytotoxic and helper T cell interactions in a tumour microenvironment $\dot{x}_0 = -10.0x_0^2 - 2.075x_0x_2 + 10.0x_0$ $\dot{x}_1 = 0.19x_0x_1/(x_0^2 + 0.0016) - 1.0x_1 + 0.5$ $\dot{x}_2 = -2.075x_0x_2 + 1.0x_1x_2 - 1.0x_2 + 2.0$
11	Munz2009 - Zombie SIZRC $\dot{x}_0 = -0.009x_0x_1 + 0.05$ $\dot{x}_1 = 0.004x_0x_1$ $\dot{x}_2 = 0.005x_0x_1$

Table G.6. Selected ODEBase dataset with variables $n = 3$.