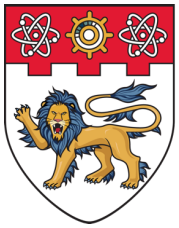
Empowering Distributed Constraint Optimization with Deep Learning



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Yanchen Deng

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

14/08/2023

Date

Yanchen Deng

Yanchen Deng

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

14/08/2023

Date

Prof. Bo An

Authorship Attribution Statement

This thesis contains material from five papers published in the following peer-reviewed conferences and journal in which I am the first author.

Chapter 3 is published as

- **Yanchen Deng** and Bo An. Speeding up incomplete GDL-based algorithms for multi-agent optimization with dense local utilities. Proceedings of the 29th International Joint Conferences on Artificial Intelligence (**IJCAI'20**), pp. 31-38, 2020.
- **Yanchen Deng** and Bo An. Utility distribution matters: Enabling fast belief propagation for multi-agent optimization with dense local utility function. Autonomous Agents and Multi-Agent Systems, Vol.35, No.2, Article 24, 2021.

The contributions of the authors are as follows:

- I proposed the idea and implemented the algorithms.
- I conducted the experiments and drafted the manuscripts.
- Prof. Bo An provided insightful comments and carefully revised the manuscripts.

Chapter 4 is published as **Yanchen Deng**, Runsheng Yu, Xinrun Wang and Bo An. Neural regret matching for distributed constraint optimization problems. Proceedings of the 30th International Joint Conference on Artificial Intelligence (**IJCAI'21**), pp.146-153, 2021.

The contributions of the authors are as follows:

- Prof. Bo An proposed the initial idea of leveraging Deep Learning to solve DCOPs.
- Dr. Xingrun Wang, Runsheng Yu and I investigated and implemented neural regret-matching for DCOPs.
- Dr. Xingrun Wang and I proposed prioritized training scheme.
- I conducted theoretical and empirical analyses, and drafted the initial manuscript.
- Prof. Bo An, Dr. Xingrun Wang and Runsheng Yu provided insightful comments and carefully revised the manuscripts.

Chapter 5 is published as **Yanchen Deng**, Shufeng Kong and Bo An. Pretrained cost model for distributed constraint optimization problems. Proceedings of the 36th AAAI Conference on Artificial Intelligence (**AAAI'22**), pp.9331-9340, 2022.

The contributions of the authors are as follows:

- I proposed the idea of pretrained cost model for DCOPs.
- Dr. Shufeng Kong and I refined the idea and implemented the algorithms.
- I proposed the Distributed Embedding Schema (DES) and proved its soundness. Dr. Shufeng Kong discussed with me about the theoretical analysis.
- I conducted the empirical evaluations, and wrote the main manuscript. Dr. Shufeng Kong wrote a part of the manuscript.
- Prof. Bo An and Dr. Shufeng Kong provided insightful comments and carefully revised the manuscripts.

Chapter 6 is published as **Yanchen Deng**, Shufeng Kong, Caihua Liu and Bo An. Deep attentive belief propagation: Integrating reasoning and learning for solving constraint optimization problems. Proceedings of 36th Conference on Neural Information Processing Systems (**NeurIPS'22**), pp.25436-25449, 2022.

The contributions of the authors are as follows:

- Dr. Shufeng Kong proposed the idea of learning to improve the performance of Min-sum BP.
- I refined the idea by proposing to use attention mechanism to reason about the dynamic hyperparameters of Min-sum BP.
- I proposed smoothed cost as a learning objective, and an efficient online self-supervised learning algorithm.
- Dr.Shufeng Kong and I implemented the algorithm.
- I conducted the empirical evaluations, and wrote the main manuscript. Dr. Shufeng Kong wrote a part of the manuscript.
- Prof. Bo An, Dr. Shufeng Kong and Dr. Caihua Liu provided insightful comments and carefully revised the manuscripts.

14/08/2023

Yanchen Deng

.....

Date

.....

Yanchen Deng

Acknowledgements

Reaching the end of my PhD study, I want to express my sincere gratitude to the excellent people who taught me, help me and inspire me. The past three years of study is definitely the most wonderful journey that I have experienced, and I am fortunate enough to work with brilliant and excellent people, who always inspire me to become professional, mature, and humble.

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Bo An. Without your kind help, visionary instructions and endless encouragements, this point can never be reached. Thank you very much for constantly providing me insightful discussions, constructive feedback, and the freedom that allows me to explore interesting research problems. Your attitude, enthusiasm and profession to the research deeply impress and inspire me, and will profoundly influence my research philosophy in future career. I will be forever grateful to your supervision and I hope I can let you be proud of me someday in the future.

I would like to thank my Thesis Advisory Committee (TAC) members: Prof. Xiaohui Bei and Prof. Zinovi Rabinovich. The meetings and discussions help me to improve the works and build a better vision of my research.

During the three years of PhD study, I am grateful to have the privilege to collaborate with many extraordinary excellent researchers: Dr. Xinrun Wang, Dr. Qingyu Guo, Dr. Shufeng Kong, Mr. Runshen Yu and Prof. Rina Dechter. Thank you for your professional mentoring and instructions. Together, we have experienced many unforgettable moments: tensions before conference deadlines, busyness during rebuttal periods and anxiousness before notification. It is always a pleasure to work with you.

I would like to thank all the AMI members who accompanied my pleasant life these years. They are Jiuchuan Jiang, Youzhi Zhang, Xinrun Wang, Lei Feng, Xu He, Aye Phyu, Hongxin Wei, Rundong Wang, Wei Qiu, Jakub Cerny, Runsheng Yu,

Shuxin Li, Wanqi Xue, Feifei Lin, Zhuyun Yin, Xinyu Cai, Shuo Sun, Pengdeng Li, Hao Chen, Ruhao Jiang, Yin Zheng, Chaojie Wang and Shufeng Kong. Thank you all for the joy, the support and the excitement you gave me.

I would like to express my special thanks to my precious friends, who are not around me but always here to support me, accompany me and comfort me, especially during my tough time. They are Tengfei Wu, Dingding Chen, Decheng Zheng, Wenxin Zhang, Siting Hong and Lizhen Liu.

Lastly but most importantly, I want to thank my parents and my brother for your unconditional love and supports. Needless to say, this thesis would not have been possible without your encouragement along the way. This thesis is dedicated to all of you.

Abstract

Distributed Constraint Optimization Problems (DCOPs) are a fundamental formalism for multi-agent coordination, in which a set of autonomous agents cooperatively find assignments to optimize a global objective. Due to its ability of capturing the essentials of cooperative distributed problem solving, DCOPs have been successfully applied to model the problems in many real-world domains like radio channel allocation and smart grids. However, solving DCOPs is notoriously difficult due to its NP-hardness, and many existing techniques either deliver a poor solution or incur prohibitive overheads when facing large-scale problem instances.

This doctoral thesis is devoted to improve the existing DCOP solving techniques in terms of both the *scalability* and the *efficiency*, by leveraging the power of heuristic search and deep learning to handle the high-dimensional solution space of DCOPs.

The first two parts of this thesis focus on improving the *scalability* of two important class of DCOP algorithms: Belief Propagation (BP) and context-based sampling algorithms. Specifically, in the first part, we investigate the defects of GDP, a state-of-the-art acceleration technique for BP, and point out the algorithm can fail if the local utility values are densely distributed. In light of this, we propose a class of techniques to offer excellent acceleration performance by providing better lower bound quality and search space organization, especially on the problems with dense local utility values. Extensive experiments on both synthetic and real-world problems demonstrate the great advantages of our algorithms over state-of-the-art. In the second part, we develop a memory-efficient context-based sampling algorithm for DCOPs by leveraging deep neural networks (DNNs) to approximate the high-dimensional tables during the solving process. In more detail, we first introduce a new context-based sampling algorithm built upon regret-matching, with a novel regret rounding scheme for incentivizing exploration. Because the regret values are associated with each context, which could be exponential in the induced width of the problem. Therefore, we propose to maintain a DNN, whose size is polynomial in the induced width, for each non-leaf agent to estimate the regret

values, by employing online learning. We theoretically show the regret bounds and demonstrate the advantages of our algorithms on various setting, especially on the scenario where the memory is limited.

The last two parts of this thesis aim to enhance the performance of existing DCOP algorithms. In the third part, we present the first general-purpose deep model for DCOPs, called GAT-PCM, to generate efficient heuristics for a wide range of DCOP algorithms by learning to estimate the optimal cost of a target assignment given a partially-instantiated DCOP instance. The model is first pretrained with a large amount of optimally-labelled data offline, then implements fully-decentralized inference when generating heuristics for DCOP algorithms. We use local search and backtracking search for DCOPs as examples and show the great superiority of GAT-PCM-boosted algorithms through extensive empirical evaluations. In the final part, we investigate using DNNs to boost the performance of BP for solving COPs, by learning the optimal dynamic hyperparameters. Our model, DABP, seamlessly integrates BP and DNNs within the message-passing framework to reason about dynamic neighbor weights and damping factors for composing new BP messages. Furthermore, unlike existing neural-based BP variants, we propose a novel self-supervised learning algorithm for DABP with a smoothed cost, which does not require expensive training labels and also avoids the common out-of-distribution issue through efficient online learning. Extensive experiments show that our model significantly outperforms state-of-the-art baselines.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Distributed Constraint Optimization Problems (DCOPs) [1, 2] are a fundamental formalism for multi-agent coordination, in which a set of autonomous agents cooperatively find assignments to optimize a global objective through localized communication. Due to its ability of capturing the essentials of cooperative distributed problem solving, DCOPs have been successfully applied to model the problems in many real-world domains, including radio channel allocation [3, 4], vessel navigation [5, 6], meeting scheduling [7–9], mobile sensor team [10, 11], task allocation [12–14], smart home automation [15–17] and smart grids [18, 19].

Over the past few decades, many algorithms for solving DCOPs have been proposed and can be generally classified as complete and incomplete according to whether they guarantee to find the optimal solution. While distributed backtracking search [2, 20–24] aims to find the optimal solution by explicitly exhausting the solution space, complete techniques based on Dynamic Programming (DP) [25–27] perform iterative variable elimination. However, due to the inherent NP-hardness of DCOPs, complete algorithms often incur exponential coordination overheads. On the other hand, incomplete algorithms trade the optimality for practical computational efforts and can scale up to large problems. Local search [28–32] is a typical class of incomplete algorithms which iteratively optimize the objective by exploring the neighborhoods of the incumbent solution, while Belief Propagation (BP) [9, 33–36] solves DCOPs by computing the marginal costs/utilities for each

variable. Besides, sampling algorithms like DUCT [4] and D-Gibbs [37] are emerging incomplete algorithms which solve DCOPs by constructing upper confidence bounds for each context and finding the maximum likelihood solution with Gibbs sampling [38], respectively.

1.2 Motivation

Many real-world scenarios require to deliver *high-quality* solutions within *practicable* budgets (e.g., runtime, network load, memory, etc.). Consider the scenario of maritime navigation [6] where multiple vessels are required to coordinate their headings and speeds to avoid collision and minimize the total travel time. In such case, high-quality solutions are desired because otherwise the vessels are at the risk of collision, while the computations are limited to a short time-frame because the environments (e.g., vessels' course and positions) are continuously changing. Consequently, there is an urgent need to develop *efficient* and *scalable* algorithms for large-scale DCOPs.

However, although numerous algorithms have been proposed for DCOPs, their performance and scalability are still far from satisfactory when solving large-scale problems. In fact, many existing techniques rely on *static* and *hand-crafted* rules or heuristics, which fails to explore the solution space efficiently. For example, distributed backtracking search usually explores the domain of variables blindly, which may require considerable computational efforts before finding a high-quality bound to prune the search space. Local search, on the other hand, can explore only a small neighborhood of the incumbent solution and often leads to trivial local optima (e.g., 1-opt solutions). DUCT finds better solutions but requires exponential memory to perform context-based sampling on a pseudo-tree. Finally, although BP can often achieve state-of-the-art performance after combining with anytime mechanism [39] and function splitting [36], it needs laborious hyperparameter tuning and requires exponential runtime to compute function-to-variable messages.

On the other hand, deep learning has achieved tremendous successes in many challenging tasks, ranging from Computer Vision (CV) [40, 41], Natural Language

Processing (NLP) [42, 43] to learning optimal control [44, 45], thanks to the high-capacited Deep Neural Networks (DNNs) and modern computing devices. With the advancements of Graph Neural Networks (GNNs) [46, 47], recent years have seen a surge of interest in applying deep learning to solve combinatorial optimization problems, i.e., Neural Combinatorial Optimization (NCO) [48, 49]. By learning generalizable representations automatically, NCO eliminates the need of manually-engineered heuristics which often require extensive expertise and domain knowledge to construct, and has achieved state-of-the-art performance in many NP-hard problems, such as Travelling Salesman Problem (TSP) [50, 51], Maximum Independent Set (MIS) [52, 53], Mixed Integer Programming (MIP) [54, 55] and Boolean Satisfibility (SAT) [56, 57]. However, since they usually require the full knowledge about the problems to be solved, these methods cannot be directly applied to a distributed setting like DCOPs where centralization is not realistic due to geographical limitations or privacy concerns.

Given the above backgrounds, we aim to improve the existing DCOP solving techniques in terms of both the *scalability* and the *efficiency*, by leveraging the power of heuristic search and deep learning to efficiently explore the high-dimensional solution space of DCOPs.

1.3 Contributions

The key contributions of the thesis are summarized as follows.

- We improve the scalability of Max-sum BP by combining branch-and-bound (B&B) [13, 58, 59] and domain pruning [60]. We first propose GD^{2}P to accelerate BP on the problems with densely distributed utility functions by continuously tightening a running lower bound. We further enhance GD^{2}P by merging the entries with the same utility value into a search tree to enable efficient B&B on tied entries, resulting in a new algorithm called ST-GD^{2}P. To balance the reconstruction overhead and the domain pruning efficiency, we propose a discretization mechanism for ST-GD^{2}P which merges small search trees into a bigger one. Finally, we introduce a Partial Tree-sorting Scheme (PTS) with different sorting criteria to reduce the memory consumption of ST-GD^{2}P. We theoretically show the soundness and the superiority

in terms of pruning capability of our algorithms. Extensive empirical evaluations indicate that the proposed algorithms significantly outperform the state-of-the-art acceleration techniques in various benchmarks.

- We develop a novel scalable context-based sampling algorithm for solving DCOPs by leveraging Deep Neural Networks (DNNs) to approximate the high-dimensional tables generated during the solving process. Specifically, we first introduce a new context-based sampling algorithm built upon regret-matching [61], with a novel regret rounding scheme for incentivizing exploration. Compared to the existing regret-based local search algorithms [62], our method performs sampling on a pseudo tree and stores regret values for each context separately, which allows an agent to devise different strategies for different contexts. Then we present a neural-based scheme to improve the scalability of the proposed sampling algorithm by using DNNs to approximate the regret values, which is the first attempt to leverage DNNs to solve DCOPs. Finally, we theoretically analyze the regret bounds of our algorithms, and demonstrate the superiority over the state-of-the-art baselines through extensive empirical evaluations.

- We develop the first general-purpose deep model for DCOPs, called GAT-PCM, to generate efficient heuristics for a wide range of DCOP algorithms by learning to estimate the optimal cost of a target assignment given a partially-instantiated DCOP instance. Specifically, by introducing a novel directed tripartite graph representation based on microstructure [63], we effectively encode a partially instantiated DCOP instance and use Graph Attention Networks (GATs) [64] to learn generalizable embeddings. We then pretrain the model with the optimally labelled data generated by an exact DCOP solver. To enable decentralized model inference without disclosing local constraint costs, we propose a Distributed Embedding Schema (DES) where each agent exchanges only the embedded vectors via localized communication. As a case study, we develop two efficient heuristics for local search and backtracking search for DCOPs based on GAT-PCM, respectively. Finally, extensive empirical evaluations indicate that GAT-PCM-boosted algorithms significantly outperform the state-of-the-art methods in various standard benchmarks.

- We investigate using DNNs to boost the performance of BP. Our model, DABP, seamlessly integrates BP, Gated Recurrent Units (GRUs) [65], and

GATs within the message-passing framework to reason about dynamic neighbor weights and damping factors for composing new BP messages. Specifically, given a factor graph and the BP messages in previous iterations, we infer the optimal hyperparameters for current iteration through GRUs and GATs, followed by a multi-head attention layer [66]. Furthermore, unlike existing neural-based BP variants, we propose a novel self-supervised learning algorithm for DABP with a smoothed cost, which does not require expensive training labels and also avoids the common out-of-distribution issue through efficient online learning. Extensive experiments show that our model significantly outperforms state-of-the-art baselines.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we review preliminaries including DCOPs, DCOP algorithms and neural combinatorial optimization. Chapter 3 presents a class of acceleration algorithms that enable fast BP for DCOPs with dense utility functions. In Chapter 4, we develop a novel scalable regret-based sampling algorithm for solving DCOPs by leveraging DNNs to approximate high-dimensional regret tables. Chapter 5 proposes the first general-purpose deep model for DCOPs by learning to estimate the optimal cost of a target assignment given a partially-instantiated DCOP instance. In Chapter 6 we boost the performance of BP by integrating learning and reasoning to infer the optimal hyperparameters for each iteration. Finally, we conclude and discuss the future work in Chapter 7.

Chapter 2

Preliminaries

2.1 Distributed Constraint Optimization Problems

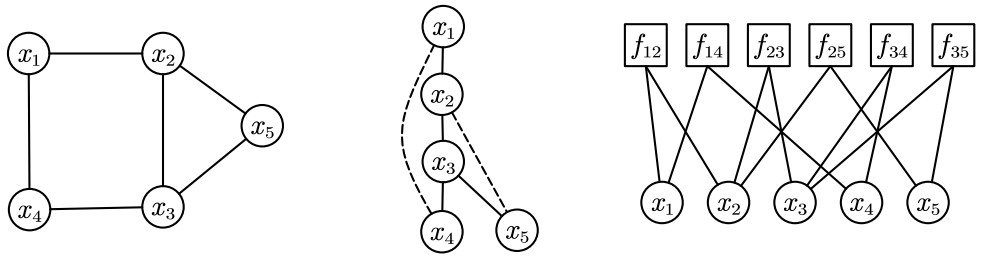
2.1.1 Definition

A Distributed Constraint Optimization Problem (DCOP) [1, 2] can be defined by a tuple $\langle I, X, D, F \rangle$ where

- $I = \{1, \ldots, |I|\}$ is the set of agents.
- $X = \{x_1, \ldots, x_{|X|}\}$ is the set of variables.
- $D = \{D_1, \ldots, D_{|X|}\}$ is the set of domains. Each variable $x_i \in X$ is associated with a discrete and finite domain $D_i \in D$.
- $F = \{f_1, \ldots, f_{|F|}\}$ is the set of constraint functions. Each constraint function $f_i : D_{i,1} \times \cdots \times D_{i,k} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ specifies the cost or utility[1] for each combination of variables in its scope $scp(f_i) = \{x_{i,1}, \ldots, x_{i,k}\} \subseteq X$.

For sake of simplicity, in this thesis we make the standard assumption that each agent only controls one variable (i.e., $|I| = |X|$), and thus the term "agent" and

[1] We sometimes use "utility functions" or "cost functions" to refer constraint functions if there is no ambiguity.



(a) constraint graph (b) pseudo tree (c) factor graph

FIGURE 2.1: DCOP visualizations

"variable" can be used interchangeably. Besides, each agent $i \in I$ is aware of only the local problem that involves the variable it controls, and can only interact with the set of neighboring agents N_i that share a constraint function with agent i through message-passing.

The objective of solving DCOPs is to find a solution $\tau^* = (d_1^*, \ldots, d_{|X|}^*) \in \prod_{x_i \in X} D_i$ such that the total cost is minimized (or the total utility is maximized):

$$\tau^* = \underset{\tau \in \Pi_{x_i \in X} D_i}{\arg\min} \sum_{f_j \in F} f_j(\tau|_{scp(f_j)}) \quad \vee \quad \tau^* = \underset{\tau \in \Pi_{x_i \in X} D_i}{\arg\max} \sum_{f_j \in F} f_j(\tau|_{scp(f_j)}), \tag{2.1}$$

where $\tau|_{scp(f_j)}$ is the projection of τ on $scp(f_j)$. In this thesis, we focus on minimization DCOPs unless otherwise stated.

2.1.2 Visualization

DCOPs can be visualized by constraint graphs, pseudo trees [67], and factor graphs [68]. The constraint graph $G = (X, E_G)$ of a DCOP is an undirected graph where each vertex corresponds to a variable in the DCOP, and there is an edge $e = (x_i, x_j) \in E_G$ for each pair of $x_i, x_j \in X$ if $\exists f_k \in F$ such that $\{x_i, x_j\} \subseteq scp(f_k)$. Figure 2.1(a) presents the constraint graph of a DCOP with 5 variables and 6 constraint functions.

A pseudo tree $PT = (X, E_T \cup E_B)$ is a tree arrangement to a constraint graph G such that constrained variables fall in the same branch, where E_T is the set of tree edges and $E_B = E_G \backslash E_T$ is the set of back edges, respectively. Typically, starting from a root, a pseudo tree can be generated by performing a Depth-first Search (DFS) traverse to the constraint graph. For each variable $x_i \in X$, we define the following notations in a pseudo tree:

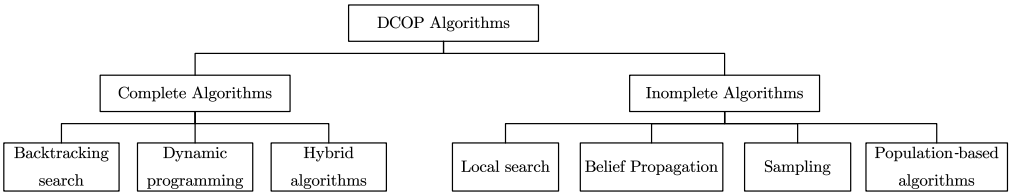


FIGURE 2.2: The taxonomy of DCOP algorithms

- $P(x_i)$ is the parent of x_i, i.e., the ancestor that connects x_i through a tree edge.
- $PP(x_i)$ is the pseudo parent(s) of x_i, i.e., the set of ancestor(s) that connect x_i through the back edges.
- $C(x_i)$ is the child(ren) of x_i, i.e., the descendent(s) that connect x_i through tree edges.
- $PC(x_i)$ is the pseudo child(ren) of x_i, i.e., the descendent(s) that connect x_i through back edges.
- $Sep(x_i)$ is the set of ancestor(s) that connect x_i and its descendent(s).

Besides, we also use $AP(x_i) = \{P(x_i)\} \cup PP(x_i)$ and $AC(x_i) = C(x_i) \cup PC(x_i)$ to denote all parents and all children of x_i, respectively. The induced width [69] of a pseudo tree $w^* = \max_{x_i \in X} |Sep(x_i)|$. It is also noteworthy that finding a pseudo tree with the minimum induced width is a NP-complete problem [70]. Figure 2.1(b) gives a possible pseudo tree of the constraint graph in Figure 2.1(a) where solid lines and dotted lines are tree edges and back edges, respectively.

The factor graph $FG = (V_X \cup V_F, E_{FG})$ is a bipartite graph representation of a DCOP, where variable nodes V_X and function nodes V_F correspond to variables and constraint functions of the DCOP, respectively. An edge between variable node x_i and function node f_j is established if $x_i \in scp(f_j)$. Figure 2.1(c) displays the factor graph of the DCOP instance corresponding to Figure 2.1(a).

2.2 DCOP Algorithms

In general, DCOP algorithms can be classified as complete and incomplete according to whether they guarantee to find the optimal solution. Based on the solving

strategy, complete algorithms can be further divided into backtracking search, dynamic programming and hybrid algorithms. Similarly, incomplete algorithms are categorized into local search, belief propagation, sampling and population-based algorithms. Figure 2.2 presents the taxonomy of DCOP algorithms.

2.2.1 Backtracking Search

Given an ordering of variables, distributed backtracking search thoroughly explores the solution space to find the optimal solutions. SyncBB [20] is the first complete search algorithm which performs synchronous branch-and-bound (B&B) on a static linear communication structure. AFB [21] extends SyncBB by allowing asynchronous bound checking, so as to detect suboptimality and initiate backtracking promptly. ConFB [23], on the other hand, runs multiple parallel versions of AFB concurrently on several disjoint sub-spaces. By sharing a global upper bound across all search processes, it can quickly detect suboptimal sub-space and thus gains a 2-3 orders of magnitude performance improvement with respect to AFB.

However, a chain-based structure could force unconstrained agents to communicate with each other and disallow parallel exploration of the search space. Alternatively, pseudo tree was introduced to create communication links among agents that share constraints and parallelize the computation in different branches. ADOPT [2] is a classical asynchronous algorithm operating on a pseudo tree, which uses a best-first search strategy. However, its search strategy incurs unnecessary reconstruction of abandoned solutions. Subsequently, BnB-ADOPT [22] was proposed to solve the issue by adopting a depth-first B&B search strategy. Finally, NCBB [71] and PT-FB [24] perform synchronous depth-first B&B with improved bounds via eager lower bound propagation and forward bounding on a pseudo tree, respectively.

2.2.2 Dynamic Programming

Dynamic Programming (DP) algorithms indirectly explore the solution space by performing iterative variable eliminations. As a distributed implementation of Bucket Elimination (BE) [72], DPOP [25] is a typical DP algorithm that performs

variable elimination on a pseudo tree. Since its memory consumption scales exponentially with respect to the induced width of the pseudo tree, several memory-bounded schemes like MB-DPOP [26] and ODPOP [73] were proposed to trade message count for smaller memory consumption. Recently, RMB-DPOP [27] refines MB-DPOP by incorporating a distributed enumeration mechanism, a iterative selection mechanism and a caching mechanism to reduce the number of redundant inferences. Besides, Action_GDL [74] generalizes DPOP by performing dynamic programming on a distributed junction tree to enhance the efficiency.

2.2.3 Hybrid Algorithms

Hybrid algorithms attempt to combine the advantages from both backtracking search and DP. ADOPT-BDP [75] hybridizes search and inference for the first time by using approximated inference (e.g., ADPOP [76]) to build lower bound lookup tables to accelerate ADOPT. DJAO [77] follows a similar strategy but instead focuses on searching over a junction tree [78]. PT-ISABB [79] combines a tailored version of ADPOP and a pseudo tree-based search algorithm to efficiently solve Asymmetric DCOPs (ADCOPs) [80] while protect the privacy of each agent. Instead of using approximated inference to generate static heuristics for search in preprocessing phase, HS-CAI [81] actively selects promising partial assignments to improve the quality of generated heuristics and reduce the overhead of iterative inference by introducing a context evaluation mechanism.

2.2.4 Local Search

Local search algorithms including DSA [29] and MGM [28] iteratively optimize the solution via local moves. In those algorithms, agents keep exchanging self states (i.e., assignments or gains) with their neighbors, and determine whether to replace and how to replace their incumbent assignment with the best responses based on the received states from their neighbors. Different algorithms adopt different assignment replacement strategies. For example, agents in DSA stochastically replace their assignment in every iteration, while only the agents who hold the maximum gain among their neighbors can replace their assignment in MGM. In [62], the authors analyzed the connection between DCOPs and potential games,

and presented a unifying framework for the local search algorithms that iteratively approximate best-responses, including DSA and MGM. Instead of exploring only the direct neighborhood of the incumbent solution, DLNS [82] explores a much larger neighborhood by destroying a part of the solution, and then repairs it by solving a tree-structure relaxation of the induced subproblem with DPOP in every iteration.

To improve the quality of local convergence, KOPT [30] coordinates the decisions of all agents within the K-size coalition and converges to a K-optimal [31] state ensuring that the solution quality cannot be improved if K agents or fewer change their assignment. Recently, RODA [83] was proposed as an algorithmic formalism to unify the existing region-optimal algorithms. Besides, Anytime Local Search (ALS) [39] framework was proposed to provide anytime property [84] for local search algorithms, which caches the best solution found so far by aggregating the local cost of each agent through a Breadth-first Search (BFS) spanning tree. To breakout local optima, GDBA [32] adapts the notion of constraint violation and the manner of cost increase in DBA [85], which is designed to solve Distributed Constraint Satisfaction Problems (DisCSPs) [86], to solve the general-valued DCOPs. Finally, Partial Decision Scheme (PDS) [87] helps local search algorithms to escape from local optima by ignoring the assignment of neighboring agents.

2.2.5 Belief Propagation

Based on the Generalized Distributive Law (GDL) [88], Belief Propagation (BP) [68, 89] is an important technique for many reasoning tasks over graphical models [90, 91]. When applied to solve DCOPs, BP is instantiated to Max-sum (resp. Min-sum)[2] [33] which computes the marginal utilities (resp. costs) for each variable by performing message-passing on a factor graph. Specifically, each variable node maintains beliefs about the marginal utilities (resp. costs) for its possible assignments, and keeps updating the beliefs according to the messages from the neighboring function nodes. In every iteration, each variable node selects the value with the highest marginal utility (resp. lowest marginal cost) as its incumbent assignment. However, it is widely acknowledged that Max-sum does not provide

[2]Hereafter, we will interchangeably use Max-sum and Min-sum to refer the BP for solving DCOPs, depending on the direction of optimization.

convergence guarantee and usually explores low-quality solutions in cyclic problems.

Bounded Max-sum (BMS) [34] provides both convergence and solution quality guarantee by relaxing the cyclic problem. In more detail, the algorithm first identifies the binary dependencies between variable nodes and function nodes which have the least impacts on the solution quality, and then removes the dependencies to obtain a tree-structured problem via minimization marginalization. Then standard Max-sum is used to solve the relaxed problem and provide a bound on the approximation of the optimal solution. To provide a tighter approximation ratio, Improved Bounded Max-sum (IBMS) [92] considers the relaxations of both minimization and maximization marginalization, and selects the best result to calculate the bounds. Nonetheless, all BMS algorithms inevitably introduce errors in relaxation phase. Thus, ED-IBMS [93] attempts to alleviate the pathology by decomposing binary dependencies into unary functions.

Different from BMS algorithms, Max-sum_AD [9] guarantees single phase convergence by performing message-passing on two directed acyclic graphs alternatively. That is, instead of sending messages to every neighbor in Max-sum, nodes in Max-sum_AD only send messages to the ones ordered after them. And the message-passing direction is alternated after the algorithm converges. Furthermore, value propagation is introduced to enforce the cross phase convergence and monotonicity, resulting a new algorithm called Max-sum_ADVP [9]. However, the exploitative nature of value propagation would degenerate belief propagation into a greedy local search algorithm. Therefore, a number of non-consecutive value propagation strategies were proposed in [35] to balance exploration and exploitation.

Damped Max-sum (DMS) [36] provides an alternative way to increase the chances for convergence of belief propagation by decreasing the effect of cyclic propagation with message damping. Instead of transforming factor graph or controlling message-passing direction, DMS manipulates the content of the messages sent from variable nodes to function nodes. That is, a message now is the weighted sum of new calculation performed in the current iteration and calculation performed in the previous iteration. Combining with anytime mechanism [39] and factor splitting [36], DMS with a high damping factor outperforms all other versions of Max-sum, as well as local search algorithms.

Scalability is a key challenge of applying Max-sum as the overhead of computing a function-to-variable message grows exponentially with respect to the arity of the constraint function. To date, a number of acceleration algorithms for Max-sum were proposed and can be generally divided into B&B- and sorting-based techniques. BnB-MS [58] and BnB-FMS [13] are typical B&B-based methods that incorporate a preprocessing phase to reduce the domain size by leveraging domain-specific knowledge and an online pruning phase to speed up the computation of function-to-variable messages by using branch-and-bound. However, the bounds used for B&B in these algorithms are computed by either brute-force or domain-specific knowledge, which cannot be applied to general cases. Recently, FDSP [59] was proposed as a generic B&B method for accelerating Max-sum by employing dynamic programming to efficiently compute domain-agnostic lower bounds.

Sorting-based methods, on the other hand, perform an one-shot pruning on the combinations of variable assignments given the (partially) sorted utility list of a constraint function. Specifically, G-FBP [94] only sorts for top $cd^{\frac{n-1}{2}}$ values of the search space and presumes that the highest utility can be found in the ranges. Here, c is a constant, d is the maximal domain size and n is the arity of a utility function, respectively. However, the algorithm has to perform an exhaustive search when the assumption fails. On the other hand, GDP [60] constructs a lower bound on the completely sorted utility list according to the entry with the maximum local utility and prunes any entries with local utility lower than the bound.

Orthogonally, Tarlow et al. [95] studied the problems with Tractable Higher Order Potentials (THOPs) and pointed out that the computational complexity of Max-sum on these problems can be reduced to polynomial time. For example, given a binary problem with cardinality factors where each variable can only take either 0 ("off") or 1 ("on") and the value of a constraint function depends solely on the number of on variables, Max-sum can be efficiently implemented in $O(n \log n)$ per function-node by using sorting and dynamic programming. For some specific types of THOPs, the belief propagation can even be implemented in linear time [96–98]. However, a major issue of THOP-based methods is the limited representational capability of THOPs, which disallows the use in general problems. Besides, applying THOP-based algorithms could be laborious: for each constraint function, one needs to first identify its pattern and then design a specified algorithm for it.

2.2.6 Sampling

Sampling-based techniques including DUCT [4] and D-Gibbs [37] are the emerging incomplete methods for DCOPs, which perform sequential sampling on a pseudo tree. Given a context a, an agent in DUCT first constructs a confidence bound for each value k in its domain, which is an optimistic estimation of the optimal cost for its subtree under context a and value k, and samples the value with the lowest bound. However, the memory requirement per agent in DUCT is exponential in the induced width of the pseudo tree, which makes it unsuitable for large real applications. Differently, D-Gibbs solves a DCOP by mapping it to a Maximum Likelihood Estimation (MLE) problem and using Gibbs sampling [38] to find a solution, which only requires a linear-space of memory.

2.2.7 Population-based Algorithms

Population-based algorithms try to optimize by maintaining a population of candidate solutions. ACO_DCOP [99] pioneers this line of research by introducing Ant Colony Optimization (ACO) [100] to solve DCOPs for the first time. Guided by the pheromone lines associated to each constraint function, agents in ACO_DCOP cooperatively construct solutions along a pseudo tree. AED [101] follows a similar strategy but instead uses evolutionary algorithm as the underlying optimization technique. LSGA [102], on the other hand, applies Genetic Algorithm (GA) [103] to a population constituted by the solutions produced by multiple local search algorithms. Finally, DPSA [104] learns the optimal temperature for distributed simulated annealing [105] for solving DCOPs by using cross-entropy sampling [106].

2.3 Neural Combinatorial Optimization

Neural Combinatorial Optimization (NCO) aims to leverage deep learning to construct flexible yet effective algorithms to solve combinatorial optimization problems, which can be generally categorized into three strategies, i.e., learning to construct solutions from scratch, learning to improve given solutions and learning effective heuristics to boost the performance of the existing algorithms.

2.3.1 Learning to Construct Solutions

Pointer Network (Ptr-Net) [107] is a seminal work that solves TSP by using a sequence-to-sequence network to map two-dimensional points to a tour with the minimum total length by outputting one unvisited city for each decoding step. The model is trained with the expert data from Concorde solver [108], and uses a beam search procedure to achieve better performance than greedy decoding. Instead of using supervised learning, Bello et al. [109] proposed to train Ptr-Net with Reinforcement Learning (RL) [110], eliminating the need of expensive labelled data.

Nonetheless, since Ptr-Net deals with sequences as its inputs and outputs, it fails to effectively capture the combinatorial structure of graph problems. In light of this, Dai et al. [111] leveraged `structure2vec` architecture [112] to learn generalizable embeddings for combinatorial optimization problems over graphs, and employed Deep Q Networks (DQN) [44] to learn the solution construction heuristics. Li et al. [52], on the other hand, used Graph Convolutional Networks (GCNs) [113] to embed combinatorial optimization problems and construct solutions via a guided tree search technique. Differently, Kool et al. [51] proposed to use an Attention Model (AM) [66] to embed a TSP instance and trained the solution construction policy with REINFORCE algorithm [114]. POMO [115] improves AM by forcing the algorithm to explore from multiple starting nodes so as to avoid local optima. Finally, DIMES [116] solves combinatorial optimization problems by introducing a compact continuous space for parameterizing the underlying distribution of candidate solutions, which allows stable REINFORCE-based training and fine-tuning via massively parallel sampling.

Beside routing problems, Selsam et al. [56] trained a message-passing neural network to predict the satisfibility of a propositional formula, and further decoded the satisfying assignment according to the literal votes. DG-DAGRNN [117] solves circuit-SATs in an unsupervised fashion by maximizing a differentiable soft evaluation function which replaces the operator of `AND`, `OR` and `NOT` in circuit-SATs. Finally, Nair et al. [55] trained a GCN model to construct high-quality partial assignments for a MIP so as to provide a tight upper bound for effective pruning.

2.3.2 Learning to Improve Solutions

Chen et al. [118] proposed NeuRewriter which aims to learn an optimal policy to schedule a portfolio of algorithms for improving solution quality. Starting from an initial solution, the method picks heuristics and rewrites the local components of the current solution to iteratively improve the incumbent solution until convergence. Similarly, Lu et al. [119] presented a Learning-to-Improve (L2I) framework, which learns an improvement policy to select improvement operators, a perturbation policy to select perturbation operators and a meta policy to coordinate these two policies. Instead of learning to select improvement operators, Li et al. [120] proposed to iteratively improve the incumbent solution by learning to identify appropriate subproblems and delegate their improvement to a black box subsolver. In the same vein, Cheng et al. [121] proposed training an RL agent to select and optimize a subproblem derived from the incumbent solution, and incorporating a Large Neighborhood Search (LNS) [122] scheme to avoid local optima. Finally, CVAE-Opt [123] first uses a Conditional Variational Auto-encoder (CVAE) [124] that learns to map points in a continuous latent search space to high-quality, instance-specific solutions, then employs an unconstrained continuous optimizer to perform search in the latent search space, and finally decodes the optimal latent vector found into a solution.

2.3.3 Learning Heuristics

In [125], the authors improved LNS for routing problems by learning repair heuristics. Specifically, they trained an RL agent to rebuild the part of the incumbent solution which is destructed by a rule-based destory operator. Fu et al. [126] proposed to solve large-scale TSP instances by first generating a heatmap for each subproblem sampled from a TSP instance with a pretrained model, then merging the heatmaps to build initial heuristics for subsequent Monte Carlo Tree Search (MCTS) [127]. Wu et al. [128] proposed to learn 2-opt improvement heuristics for routing problems by selecting a pair of nodes to apply local search operators. Finally, Hudson et al. [129] proposed to solve TSP by predicting the regret of including each edge of the problem graph in the solution to inform Guided Local Search (GLS) [130].

Branch-and-Cut (B&C) is an important technique for exactly solving MIP instances [131] while various decision choices like branching variable selection, node selection and cutting-plane selection in B&C can significantly impact its performance. In [132], the authors proposed to approximate the full Strong Branching (SB) [133] heuristic for variable selection, which is computationally demanding, by supervisely training the model with handcrafted problem features. Later, neural-based imitation methods [55, 134, 135] automatically learn the representations for MIP instances by leveraging GNNs, and achieve state-of-the-art performance in various benchmarks. Beside mimic effective but expensive heuristics, there are also attempts [136–138] that aim to discover new heuristics with performance better than traditional heuristics using RL in MIP solving.

In constraint and SAT solving, Yolcu and Póczos [139] proposed to use GNNs to encode a SAT instance and learn heuristics for WalkSAT [140] with REINFORCE. Similarly, Kurin et al. [141] proposed to learn branching heuristics for a Conflict-Driven Clause Learning (CDCL) solver [142] using GNNs and DQN. Recently, NLocalSAT [143] is proposed to boost the performance of the stochastic local search solver by guiding initialization assignments with a gated GCN. Finally, Song et al. [144] employed RL to learn the variable ordering heuristics for the backtracking search algorithms for Constraint Satisfaction Problems (CSPs) [145].

2.4 Chapter Summary

In this chapter, we review the preliminaries of the thesis, including DCOPs, DCOP algorithms and learning-based techniques for solving combinatorial optimization problems. We demonstrate that although incomplete algorithms are promising methods for solving large-scale DCOPs, they can still suffer from scalability issues (e.g., Max-sum and context-based sampling) or poor performance (e.g., loopy BP and local search algorithms). On the other hand, NCO has emerged as an efficient technique for solving NP-hard combinatorial optimization problems by leveraging the representational power of deep neural networks, but cannot be directly applied to DCOPs as it usually requires the full knowledge of the problem to be solved. In Chapter 3 and 4, we will address the scalability issue of Max-sum and context-based

sampling with heuristic search and DNNs, respectively. Then we boost the performance of DCOP algorithms by learning effective heuristics and hyperparameters with GNNs in Chapter 5 and 6, respectively.

Chapter 3

Enabling Fast Belief Propagation for Multi-agent Optimization with Dense Local Utility Functions

3.1 Introduction

Belief propagation algorithms including Max-sum[1] and its variants [9, 33–35] are important methods for solving large-scale DCOPs built upon the Generalized Distributive Law (GDL) [88] and have been applied to many real-world domains [13, 33, 146]. However, these algorithms face a significant scalability challenge. In more detail, Max-sum implements belief propagation on a factor graph [68] by optimizing the sum of local utility functions and corresponding query messages. As a result, the computational effort grows exponentially with respect to the arity of each utility function, which prohibits Max-sum from scaling up to the problems with high-arity factors.

Therefore, a number of acceleration algorithms for belief propagation were proposed to improve the scalability and can be generally divided into B&B-based and sorting-based algorithms. B&B-based algorithms [13, 58, 59] construct an estimation for each partial assignment and employ branch-and-bound to reduce the search space. Differently, Generic Domain Pruning (GDP) [60] technique is a

[1] In this chapter, we deal with the DCOPs with maximization objective. Therefore, constraint function $f_i \in F$ now specifies a utility value for each combination of variables in $scp(f_i)$.

sorting-based approach that performs a one-shot pruning on a completely sorted local utility list.

However, a key issue of the existing methods is the bound quality. In more detail, B&B-based algorithms alphabetically exhaust the whole search space without using any *a priori* knowledge. As a result, the algorithms may not be able to find a high-quality lower bound promptly when utility functions are highly structured, leading to a poor acceleration performance. On the other hand, although GDP attempts to find a more efficient bound by sorting local utility values, the one-shot nature still cannot guarantee the quality of the lower bound.

The issue could be amplified when solving the problem with dense utility functions. Consider a utility function where the entries with high utility value are sparsely distributed (or, equivalently, the entries with low utility value are densely distributed). B&B-based algorithms may need to search a considerably large proportion of search space before finding a high-quality lower bound. Similarly, when the entries with high utility value are densely distributed, the pruned range returned by GDP may contain many entries whose utility value is no less than the one-shot lower bound. Even worse, GDP cannot prune at all if the entries have the same utility value. In fact, GDP relies on a linearly structured space (i.e., the sorted local utility list) to perform pruning and thus cannot discard the suboptimal tied entries in advance.

Unfortunately, densely distributed utility functions are ubiquitous in real-world scenarios. For example, in a NetRad system [146], multiple radars coordinate their scanning strategy to maximize the total utility of scanning weather phenomena. In such scenario, the utility of scanning a weather phenomenon does not necessarily grow linearly with respect to the scanning quality. Some phenomena may require high-quality observations and the entries with low utility value are very densely distributed in the corresponding utility functions. While some phenomena require less joint efforts and the corresponding utility functions have many entries with high utility value.

Against the background, in this chapter[2] we aim to cope with dense local utility functions from the perspectives of both bound quality and search space organization

[2]This chapter has been published in [147, 148].

and develop more efficient acceleration algorithms for Max-sum and its variants. Specifically, our main contributions are listed as follows.

- We propose a Generic Dynamic Domain Pruning (GD^{2}P) technique to ensure the bound quality. Instead of performing a one-shot pruning with the initial bound in GDP, we iteratively reduce the search space by continuously tightening a running lower bound. We further enhance GD^{2}P by merging the entries with the same utility value into a search tree to enable efficient branch-and-bound on tied entries. Therefore, the search space is collectively represented by a sorted array of search trees and the scheme is referred as Search Tree-based GD^{2}P (ST-GD^{2}P).
- We propose a discretization mechanism for ST-GD^{2}P to offer a tradeoff between the reconstruction overhead and the domain pruning efficiency. That is, we eliminate small search trees in ST-GD^{2}P by discretizing the utility range into slots according to a fixed step size and building a search tree for each slot. In this way, we can reduce both the preprocessing and reconstruction overheads when the utility functions are extremely dense.
- We introduce a K-depth Partial Tree-sorting Scheme (PTS) with different sorting criteria to reduce the memory consumption of ST-GD^{2}P. Given a utility function and a total ordering on the involved variables, the scheme builds a sorted array of search trees to represent the search space of the first K variables according to certain sorting criterion. For the remaining variables, the vanilla branch-and-bound is carried out to exhaust the search space. Finally, we introduce a built-in termination detection mechanism to allow domain pruning in PTS.
- We theoretically show that the proposed algorithms are correct and both GD^{2}P and ST-GD^{2}P outperform GDP in terms of pruning capability. Our empirical evaluations indicate that the proposed algorithms significantly outperform the state-of-the-art in various benchmarks.

The reset of this chapter is organized as follows. We present backgrounds in Section 3.2 and motivate our research in Section 3.3, respectively. In Section 3.4, we introduce the proposed GD^{2}P, ST-GD^{2}P and discretization mechanism. K-depth partial-tree sorting scheme is presented in Section 3.5. We present empirical evaluations in Section 3.6 and conclude the chapter in Section 3.7.

3.2 Backgrounds

In this section, we review preliminaries including Max-sum, GDP, FDSP and search tree.

3.2.1 Max-sum

Max-sum [33] is a GDL-based incomplete message-passing algorithm for DCOPs operating on a factor graph. Starting from arbitrarily initialized messages, Max-sum implements belief propagation via query and response messages. A query message is sent from a variable node to a neighboring function node, which is computed by summing up the latest messages received from the neighboring function nodes except the target function node. Formally, the query message from variable node $x_i \in scp(f_j)$ to function node f_j in iteration m is given by

$$q^m_{x_i \to f_j}(x_i) = \sum_{f_k \in N_{x_i} \setminus \{f_j\}} r^{m-1}_{f_k \to x_i}(x_i) - \alpha, \tag{3.1}$$

where N_{x_i} is the set of neighboring function nodes of x_i, $r^{m-1}_{f_k \to x_i}(x_i)$ is the response message from f_k in the previous iteration and α is a normalization term to control the magnitude of each entry in the message. A response message is sent by a function node to a neighboring variable node, which contains the best utility value for each value in the domain of target variable under current belief. Formally, the response message from function node f_j to variable node $x_i \in \mathbf{x_j}$ in iteration m is given by

$$r^m_{f_j \to x_i}(x_i) = \max_{\mathbf{x_j} \setminus \{x_i\}} \left(f_j(\mathbf{x_j}) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q^{m-1}_{x_k \to f_j}(x_k) \right), \tag{3.2}$$

where $\mathbf{x_j}$ is the shorthand for $scp(f_j)$. A variable node x_i makes a decision in iteration m by choosing the assignment with the highest utility under the current belief.[3] That is,

$$d^*_i = \arg\max_{d_i \in D_i} \sum_{f_k \in N_{x_i}} r^m_{f_k \to x_i}(d_i).$$

[3]We hereafter drop the notion of m for sake of simplicity.

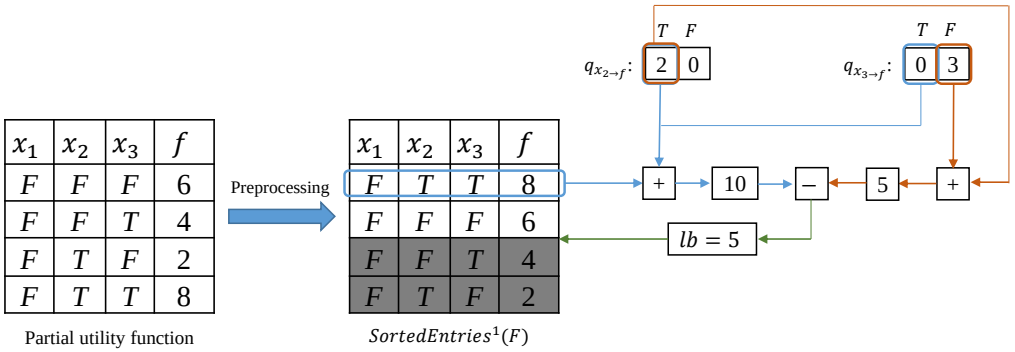


FIGURE 3.1: The trace of GDP

3.2.2 Generic Domain Pruning

It can be seen that the computational overhead of Eq. (3.2) is exponential in the arity of utility function f_j, which prohibits Max-sum from solving the problems with high-arity factors. Generic Domain Pruning (GDP) [60] is a state-of-the-art acceleration algorithm operating on completely sorted utility lists. In more detail, for each variable $x_i \in \mathbf{x_j}$ and each value $d_i \in D_i$, f_j sorts the space corresponding to $\mathbf{x_j} \wedge \{x_i = d_i\}$ into $SortedEntries^i_j(d_i)$ according to the utility value, where each entry $e \in SortedEntries^i_j(d_i)$ is an assignment to $\mathbf{x_j}$ such that x_i takes d_i. When computing the response utility for $x_i = d_i$, f_j constructs a one-shot lower bound lb according to the entry with the highest local utility value. That is,

$$lb = f_j(e_0) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e_0[x_k]) - \max_{d_k \in D_k} q_{x_k \to f_j}(d_k),$$

where e_0 is the first element in $SortedEntries^i_j(d_i)$ and $e_0[x_k]$ is the assignment of x_k in entry e_0. Clearly, an entry e is proven to be suboptimal if its local utility $f_j(e)$ is lower than lb, since its total value is lower than the one produced by e_0 even if the maximum query message utility is attained. Therefore, GDP performs binary search on $SortedEntries^i_j(d_i)$ to find the maximum index q such that $f_j(e_q) \geq lb$, and returns all entries between e_0 and e_q (both are inclusive) as the pruned range.

Figure 3.1 illustrates the trace of GDP when computing response utility for $x_1 = F$. Starting with a preprocessing phase, which sorts the utility function in descending order, GDP uses the first entry in the sorted utility list $\{F, T, T\}$ to compute an initial value 10. Then the value is shifted by subtracting the maximum message utility of each non-target variable, resulting in the lower bound of 5. Finally, any

entry with local utility value smaller than 5 is pruned and the algorithm returns $\{F, T, T\}$ and $\{F, F, F\}$ as the pruned range.

3.2.3 Function Decomposing and State Pruning

A prominent drawback of GDP is the prohibitively expensive sorting overhead. In fact, for each utility function f_j, sorting for a variable requires $O(nD^n_{\max})$ operations, where $n = |\mathbf{x_j}|$ and $D_{\max} = \max_{x_i \in \mathbf{x_j}} |D_i|$. Therefore, Function Decomposing and State Pruning (FDSP) [59] was proposed to reduce the search space by directly performing branch-and-bound without sorting the utility values.

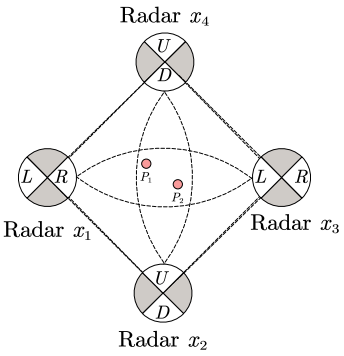
Specifically, FDSP assumes a static ordering over the involved variables of a utility function, e.g., a lexicographical order. When f_j is computing a response utility for a target assignment $\mathbf{x_{j,i}} = d$ where $1 \leq i \leq |\mathbf{x_j}|$, it performs backtrack search by maintaining a partial assignment PA. Pruning happens whenever the upper bound of PA is smaller than the known lower bound. Here, the upper bound is given by

$$ub^{PA} = \sum_{1 \leq k \leq l, k \neq i} q_{\mathbf{x_{j,k}} \to f_j}(PA[\mathbf{x_{j,k}}]) + \sum_{l < k \leq |\mathbf{x_j}|, k \neq i} \max_{d_k} q_{\mathbf{x_{j,k}} \to f_j}(d_k) + FunEst_l(PA), \quad (3.3)$$

where i is the index of the target variable, l is the index of the last assigned variable of PA in $\mathbf{x_j}$ and $FunEst_l(PA)$ is the most optimistic estimation on how much local utility we will get given prefix PA. In particular, FDSP considers two types of function estimation. When $i < l$, the estimation is nothing but a maximization to f_j over all unassigned variables. Otherwise, the estimation is enhanced by considering the assignment of the target variable $\mathbf{x_{j,i}}$. That is,

$$FunEst_l(PA) = \begin{cases} \max\limits_{z = \{\mathbf{x_{j,k}} | l < k \leq |\mathbf{x_j}|\}} f_j(PA, z) & i < l \\ \max\limits_{z = \{\mathbf{x_{j,k}} | l < k \leq |\mathbf{x_j}| \wedge k \neq i\}} f_j(PA, \mathbf{x_{j,i}} = d, z) & i > l \end{cases}. \quad (3.4)$$

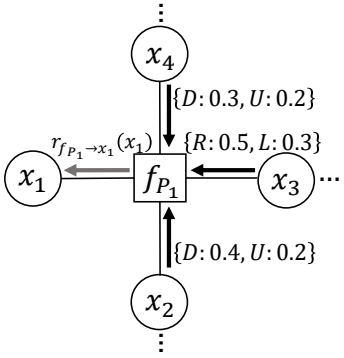
To compute the estimations efficiently, FDSP performs dynamic programming (i.e., the so-called "function decomposing") to back up the maximum local utility values in preprocessing phase, which incurs a much smaller overhead than GDP.



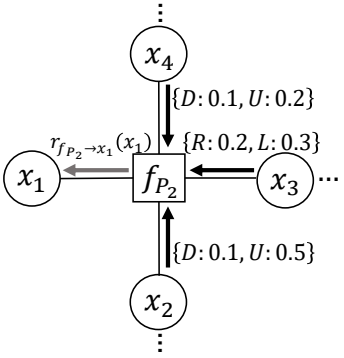
(a) the NetRad system

x_1	x_2	x_3	x_4	f_{P_1}	f_{P_2}
R	D	R	D	0.98	0.14
R	D	R	U	0.97	0.13
R	D	L	D	0.99	0.15
R	D	L	U	0.98	0.14
R	U	R	D	0.99	0.15
R	U	R	U	0.98	0.14
R	U	L	D	1	1
R	U	L	U	0.99	0.15

(b) partial utility tables



(c) local interactions of f_{P_1}



(d) local interactions of f_{P_2}

FIGURE 3.2: The NetRad system example

3.2.4 Search Trees

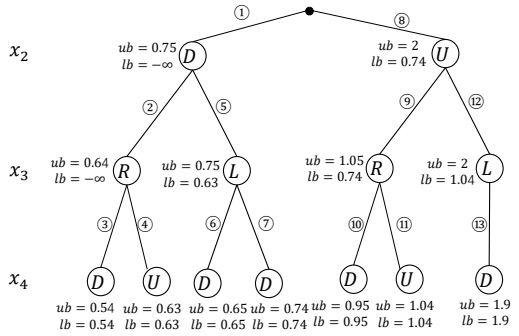
Search trees [149, 150] are a powerful tool for representing search spaces. In a search tree, each level is labelled with a variable and each node corresponds to a possible assignment the variable can take. Therefore, a path from a root to a leaf uniquely determines a full assignment, and enumerating all possible solutions of the search space is equivalent to performing a traverse of the search space. For example, Figure 3.3(b) presents a search tree over variables x_2, x_3 and x_4 where x_2 and x_4 can take either D or U while x_3 takes a value from $\{R, L\}$. A path $D - R - D$ corresponds to an assignment $\{x_2 = D, x_3 = R, x_4 = D\}$.

Naturally, the search space of Eq. (3.2) can be represented by a search tree where each level is assigned to a non-target variable and each node is the value the associated variable can take. In fact, FDSP implicitly exhausts the search tree by branching on each non-target variable and discards subtrees when the corresponding partial assignments are proven to be suboptimal.

x_1	x_2	x_3	x_4	f_{P_1}	
R	U	L	D	1	e_0
R	D	L	D	0.99	
R	U	R	D	0.99	
R	U	L	U	0.99	
R	D	R	D	0.98	
R	D	L	U	0.98	
R	U	R	U	0.98	
R	D	R	U	0.97	

$lb = 0.6$

(a) the trace of GDP on f_{P_1}



(b) the trace of FDSP on f_{P_2}

FIGURE 3.3: Traces of GDP and FDSP on the NetRad system example

3.3 Motivation

In this section, we demonstrate that the existing acceleration algorithms could perform poorly when solving the problems with dense utility functions. Consider a part of a NetRad system shown in Figure 3.2 consisting of four radars and two weather phenomena P_1, P_2. For simplicity, assume that each radar can only scan one of two sectors (i.e., $\{R, L\}$ for x_1 and x_3, $\{D, U\}$ for x_2 and x_4). Phenomenon P_1 requires less joint observation efforts and the utility is high even if there is only one radar scanning P_1, while observing P_2 is demanding and all radars must scan simultaneously to obtain a high utility.

Figure 3.2 (b-d) give the partial utility functions (w.r.t. $x_1 = R$) and local interactions for P_1 and P_2, respectively. We now aim to compute the response utility value for $x_1 = R$ from function node f_{P_1} and f_{P_2}. When applying GDP to f_{P_1}, a one-shot lower bound is constructed according to the entry $e_0 = \{R, U, L, D\}$ since it has the highest local utility value. That is,

$$lb = 1 + 0.2 - 0.4 + 0.3 - 0.5 + 0.3 - 0.3 = 0.6.$$

Therefore, any entry with local utility value less than 0.6 is filtered out. Unfortunately, as demonstrated in Figure 3.3(a), since all entries are densely distributed in the range of $[0.97, 1]$, GDP actually cannot prune any combination from the search space in this case and fails to accelerate Max-sum.

On the other hand, when applying FDSP to f_{P_2}, the algorithm sequentially extends the partial assignment and performs branch-and-bound to reduce the search space. Figure 3.3(b) presents the trace of FDSP on f_{P_2} when computing response utility for $x_1 = R$ where the number labelled in each edge denotes the sequence of exploration. The algorithm explores the search space in a depth-first fashion by extending a partial assignment. After 3 steps of extension, FDSP reaches the first full assignment $\{R, D, R, D\}$[4] and updates the lower bound by

$$lb = 0.14 + 0.1 + 0.2 + 0.1 = 0.54.$$

Unfortunately, the lower bound fails to reduce the search space in this case. In fact, it can be clearly seen that the pruning only happens after visiting $\{R, U, L, D\}$. Therefore, in this case FDSP needs to explore a considerably large proportion of search space to obtain a high-quality lower bound. In fact, FDSP almost degenerates to vanilla Max-sum and only prunes one partial assignment in this case. That is because it sequentially explores the search space, and high utility values are sparsely distributed in the utility function.

In fact, dense utility values are ubiquitous in real-world scenarios due to the law of diminishing marginal utility. Therefore, we aim to develop a series of efficient techniques for accelerating belief propagation algorithms by alleviating the negative effect of dense utility values.

3.4 Dynamic Domain Pruning Techniques

In this section, we present our dynamic domain pruning techniques for accelerating Max-sum. We begin with introducing GD^{2}P, a variant of GDP that integrates both search and pruning to iteratively reduce the search space. To prune entries with the same utility value effectively, we then present ST-GD^{2}P which merges the tied entries to search trees to enable branch-and-bound. Finally, we propose a discretization mechanism to offer a tradeoff between the reconstruction overhead and the domain pruning efficiency. We conclude by discussing the modifications when applying our methods to other versions of belief propagation.

[4]We have omitted $x_1 = R$ from the search tree.

3.4.1 GD²P

As demonstrated earlier, the quality of the lower bound is the key of efficient pruning, especially when local utility values are dense. Therefore, we propose to integrate both search and pruning and iteratively reduce the search space by continuously tightening a running lower bound. Thus, the pruning is no longer a one-shot procedure and the scheme is referred as Generic Dynamic Domain Pruning (GD²P). Alg. 1 presents the sketch of GD²P.

Similar to GDP, GD²P also begins with complete sorting of each utility function (line 1-3). When computing a response message for a variable node $x_i \in \mathbf{x_j}$, it searches for the highest utility for each assignment $d_i \in D_i$ by exhausting the sorted entries whose local utility value is no less than the running lower bound lb in a sequential order (line 8-13). Here, the lower bound is updated whenever a higher utility is found (line 10-11).

Algorithm 1: GD²P for function node f_j

When `Initialization`**:**
1 **foreach** $x_i \in \mathbf{x_j}$ **do**
2 **foreach** $d_i \in D_i$ **do**
3 $SortedEntries^i_j(d_i) \leftarrow$ arg sort $f_j(x_i = d_i, \cdot)$ in non-increasing order
When *computing a response message for* $x_i \in \mathbf{x_j}$**:**
4 $msgUB \leftarrow \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k)$
5 **foreach** $d_i \in D_i$ **do**
6 $e \leftarrow$ the first element in $SortedEntries^i_j(d_i)$
7 $u \leftarrow f_j(e)$, $util^* \leftarrow -\infty, lb \leftarrow -\infty$
8 **while** $e \neq NIL$ *and* $u \geq lb$ **do**
9 $util \leftarrow u + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e[x_k])$
10 $util^* \leftarrow \max(util^*, util)$
11 $lb \leftarrow util^* - msgUB$
12 $e \leftarrow$ the next element in $SortedEntries^i_j(d_i)$
13 $r_{f_j \to x_i}(d_i) \leftarrow util^*$
14 **return** $r_{f_j \to x_i}(x_i)$

To look deeper into how dynamic domain pruning works, consider the instance in Figure 3.2(c). The upper bound of query messages' utility is given by

$$msgUB = \max\{0.4, 0.2\} + \max\{0.5, 0.3\} + \max\{0.3, 0.2\} = 1.2.$$

TABLE 3.1: The trace of GD^{2}P on f_{P_1}

e	$f_{P_1}(e)$	$util$	$util^*$	lb
$\{R, U, L, D\}$	1	1.8	1.8	0.6
$\{R, D, L, D\}$	0.99	1.99	1.99	0.79
$\{R, U, R, D\}$	0.99	1.99	1.99	0.79
$\{R, U, L, U\}$	0.99	1.69	1.99	0.79
$\{R, D, R, D\}$	0.98	2.18	2.18	0.98
$\{R, D, L, U\}$	0.98	1.88	2.18	0.98
$\{R, U, R, U\}$	0.98	1.88	2.18	0.98
$\{R, D, R, U\}$	0.97	Pruned	2.18	0.98

Then GD^{2}P sequentially explores the rows of sorted utility list (i.e., Figure 3.3(a)) and updates the lower bound until the current local utility value is less than the lower bound. Table 3.1 presents the detailed trace of GD^{2}P.

Next, we theoretically show its correctness and superiority over GDP.

Theorem 3.1. *GD^{2}P guarantees the optimality of Eq. (3.2).*

Proof. Assume that the optimal entry e^* is pruned by GD^{2}P when $x_i = d_i$. According to line 4, 8 and 11, we have

$$f_j(e^*) < lb = util^* - msgUB$$

where $util^*$ is the highest utility returned by GD^{2}P. Combing with query messages, we have

$$f_j(e^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e^*[x_k]) < util^* - msgUB + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e^*[x_k]).$$

Since

$$\sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e^*[x_k]) \leq \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k) = msgUB,$$

it must be the case that

$$f_j(e^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(e^*[x_k]) < util^*,$$

which results in e^* is not an optimal entry, and is contradictory to the assumption. Therefore, GD^{2}P must preserve entry e^*. According to the assumption that e^* is

the optimal solution to Eq. (3.2) and the fact that $util^*$ is the maximum value of all visited entries (line 9-10), GD^{2}P must return the value corresponding to e^* and therefore guarantees the optimality of Eq. (3.2). □

Theorem 3.2. *GD^{2}P never explores the entries pruned by GDP.*

Proof. Assume that GD^{2}P explores an entry e which is pruned by GDP at iteration m. Denote the lower bound constructed by GDP as lb^{GDP} and the current lower bound of GD^{2}P as $lb^{GD^2P}(m)$. According to line 8, since e is explored by GD^{2}P but pruned by GDP, it must be the case that

$$lb^{GDP} > f_j(e) \geq lb^{GD^2P}(m). \tag{3.5}$$

W.l.o.g., assume that sorting is stable, i.e., both GDP and GD^{2}P have the same sorted local utility list and construct the initial lower bound using the same entry. Therefore, $lb^{GDP} = lb^{GD^2P}(0)$, where $lb^{GD^2P}(0)$ is the lower bound of GD^{2}P in iteration 0. Since $lb^{GD^2P}(m)$ is non-decreasing (i.e., line 10-11), we have

$$lb^{GD^2P}(m) \geq lb^{GD^2P}(0) = lb^{GDP},$$

which is contradictory to Eq. (3.5). □

The preprocessing overhead of GD^{2}P is the same as that of GDP. Formally, we have the following proposition.

Proposition 3.1. *GD^{2}P requires $O(n^2 D_{\max}^n)$ operations and $O(nD_{\max}^n)$ space to preprocess a utility function f_j, where $n = |\mathbf{x_j}|$ is the arity of the function and $D_{\max} = \max_{x_i \in \mathbf{x_j}} |D_i|$ is the maximum domain size of the involved variables.*

Proof. Since the function has $O(D_{\max}^n)$ utility values, for each involved variable and each value it requires $O(D_{\max}^{n-1} \log D_{\max}^{n-1}) = O((n-1)D_{\max}^{n-1})$ operations to perform sorting (line 1-3). Therefore, GD^{2}P needs $O((n-1)D_{\max}^n)$ operations for each variable and $O(n^2 D_{\max}^n)$ operations in total to perform preprocessing.

For space complexity, we first note that an assignment to the involved variables takes $O(n)$ space and thus storing all combinations of the involved variables requires $O(nD_{\max}^n)$ space for a variable. Since we could share these assignments across different variables via pointers in practice, it takes $O(D_{\max}^n)$ space for each variable

to store the addresses of the data structures corresponding to the assignments. Therefore, the overall space complexity of GD^{2}P is $O(nD^n_{\max}+nD^n_{\max}) = O(nD^n_{\max})$. □

To implement such sharing mechanism, we maintain an array of length $\Pi_{x_i \in \mathbf{x_j}}|D_i|$, in which each element is the address of the data structure corresponding to an assignment to all involved variables. To lookup the address of a specific assignment, we first convert the assignment into an index according to a pre-defined dimension order, then get the corresponding address by indexing the array. Therefore, the extra overhead of the mechanism lies in computing the index for each assignment, which requires $O(n)$ operations. Given n variables and $O(D^n_{\max})$ assignments for each variable to be processed, the total extra overhead is in $O(n^2 D^n_{\max})$, which does not increase the overall time complexity of GD^{2}P.

3.4.2 ST-GD^{2}P

As demonstrated by Figure 3.3(a) and Table 3.1, both GDP and GD^{2}P perform poorly when there are ties in utility functions since they use a linear structure to organize the search space. Consequently, they have to exhaust all the tied entries whose local utility is no less than the lower bound. Thus, we propose to organize tied entries into a search tree so as to enable efficient branch-and-bound when the domain pruning technique fails to reduce the search space. The scheme is referred as Search Tree-based GD^{2}P (ST-GD^{2}P) and Alg. 2 presents the sketch.

Different than completely sorting a utility function in GDP and GD^{2}P, ST-GD^{2}P only sorts for *distinct* local utility values for each assignment of each variable in the preprocessing phase (line 1-4). After that, it groups the entries with the same local utility value (line 5-9, 22-29) by inserting these entries to a search tree (line 22-29). In other words, instead of maintaining a linear list in GDP and GD^{2}P, ST-GD^{2}P uses a sorted array of search trees to collectively represent the search space. Note that the construction of search trees can be done by a sequential iteration over the utility function as the trees can be built incrementally. Similar to GD^{2}P, it maintains a running lower bound lb and terminates whenever the local utility value is less than the lower bound (line 12-19).

Algorithm 2: ST-GD^{2}P for function node f_j

When `Initialization`**:**

1 **foreach** $x_i \in \mathbf{x_j}$ **do**
2 **foreach** $d_i \in D_i$ **do**
3 $SortedUtil^i_j(d_i) \leftarrow$ all distinct utility values of $f_j(x_i = d_i, \cdot)$
4 sort $SortedUtil^i_j(d_i)$ in descending order
5 **foreach** $u \in SortedUtil^i_j(d_i)$ **do**
6 $t \leftarrow$ create an empty tree
7 **foreach** *entry* e *such that* $e[x_i] = d_i$ *and* $f_j(e) = u$ **do**
8 `Insert`(t, e)
9 $tree^i_j(d_i, u) \leftarrow t$

When *computing a response message for* $x_i \in \mathbf{x_j}$**:**

10 $msgUB \leftarrow \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k)$
11 **foreach** $d_i \in D_i$ **do**
12 $u \leftarrow$ the first element in $SortedUtil^i_j(d_i)$
13 $util^* \leftarrow -\infty, lb \leftarrow -\infty$
14 **while** $u \neq NIL$ *and* $u \geq lb$ **do**
15 $t \leftarrow trees^i_j(d_i, u)$
16 $util \leftarrow$ `Branch-and-bound`$(t, util^*)$
17 $util^* \leftarrow \max(util^*, util)$
18 $lb \leftarrow util^* - msgUB$
19 $u \leftarrow$ the next element in $SortedUtil^i_j(d_i)$
20 $r_{f_j \to x_i}(d_i) \leftarrow util^*$
21 **return** $r_{f_j \to x_i}(x_i)$

Function `Insert`(t, e)**:**

22 $n \leftarrow$ the root of t
23 **foreach** $x_i \in \mathbf{x_j}$ **do**
24 **if** *there is a child* c *of* n *such that* $val(c) = e[x_i]$ **then**
25 $n \leftarrow c$
26 **else**
27 $c \leftarrow$ create a new node
28 $val(c) \leftarrow e[x_i]$, mark c as a child of n
29 $n \leftarrow c$

It is noteworthy that instead of iterating over the sorted entries in GDP and GD^{2}P, ST-GD^{2}P iterates over the distinct sorted utility values in descending order and performs branch-and-bound to reduce the search space of tied entries. In more detail, for each distinct local utility u, ST-GD^{2}P exhausts the corresponding search tree $tree^i_j(d_i, u)$ in a depth-first fashion. For a node in the search tree, the path from the root to that node corresponds to a *partial assignment*. Therefore, by constructing a upper bound ub^{PA} for each partial assignment PA, we can efficiently explore the search tree by discarding the subtree of a node whenever the upper bound of corresponding partial assignment is less than the known highest utility

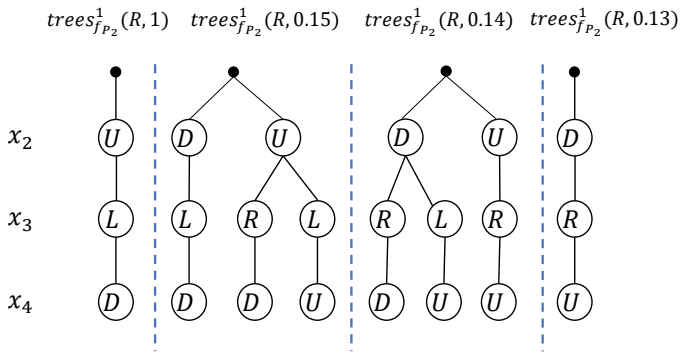


FIGURE 3.4: Search trees for $x_1 = R$ when applied to Figure 3.2(d)

$util^*$ (line 15-16). Formally, the upper bound is given by

$$ub^{PA} = u + \sum_{x_k \in PA} q_{x_k \to f_j}(PA[x_k]) + \sum_{x_k \notin PA} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k). \tag{3.6}$$

Take Figure 3.2(d) as an example. Given $x_1 = R$, ST-GD^{2}P first sorts distinct local utility values $\{1, 0.15, 0.14, 0.13\}$ and builds a search tree for each of them. Figure 3.4 presents the search trees. When computing the response utility for $x_1 = R$, ST-GD^{2}P first performs branch-and-bound to exhaust the search tree $tree^1_{f_{P_2}}(R, 1)$. It can be concluded that the algorithm terminates after reaching the first full assignment $\{R, U, L, D\}$ since it finds the highest utility 1.9, which results in a pruned rate of 87.5%.

We now show its correctness and its superiority over GD^{2}P.

Theorem 3.3. *ST-GD^{2}P guarantees the optimality of Eq. (3.2).*

Proof. There are two lines that prune the search space, i.e., line 14 and line 16. We have shown that line 14 cannot prune the highest utility in Theorem 3.1. Thus, we only need to show that line 16 never prunes the optimal assignment. Assume that the optimal full assignment A^* is pruned by line 16. Thus, it must exist a prefix $PA \subset A^*$ such that

$$ub^{PA} < util^*,$$

where $util^*$ is the highest utility found by ST-GD^{2}P. In fact,

$$\begin{aligned} ub^{PA} &= f_j(A^*) + \sum_{x_k \in PA} q_{x_k \to f_j}(PA[x_k]) + \sum_{x_k \notin PA} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k) \\ &\geq f_j(A^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A^*[x_k]). \end{aligned}$$

Thus, we have

$$f_j(A^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A^*[x_k]) < util^*,$$

implying that there is another full assignment $A' \neq A^*$ yielding higher utility than A^*, which is contradictory to the assumption that A^* is optimal. The optimality is hereby guaranteed. □

Theorem 3.4. *ST-GD^{2}P never explores the assignments pruned by GD^{2}P.*

Proof. Assume that ST-GD^{2}P explores one full assignment A which is pruned by GD^{2}P. Denote the lower bound of ST-GD^{2}P when exploring A as lb^{ST} and the lower bound of GD^{2}P when pruning A as lb^{GD^2P}, respectively. It must be the case that

$$lb^{GD^2P} > f_j(A) \geq lb^{ST}. \tag{3.7}$$

Since GD^{2}P sequentially exhausts the search space, it must exist an assignment $A' \prec A$ such that

$$\begin{aligned} lb^{GD^2P} &= f_j(A') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'[x_k]) - \max_{d_k \in D_k} q_{x_k \to f_j}(d_k) \\ &> f_j(A), \end{aligned} \tag{3.8}$$

which indicates that $f_j(A') > f_j(A)$. On the other hand, ST-GD^{2}P explores distinct local utility values in a descending order (line 4, 12 and 19 of Alg. 2). According to Eqs.(3.7-3.8), A' produces a higher utility than A. Thus, ST-GD^{2}P must have explored an assignment A'' such that

$$f_j(A'') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A''[x_k]) \geq f_j(A') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'[x_k]),$$

which implies

$$lb^{ST} \geq f_j(A') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'_{x_k}) - \max_{d_k \in D_k} q_{x_k \to f_j}(d_k) = lb^{GD^2P}. \tag{3.9}$$

Therefore, Eq. (3.9) is contradictory to Eq. (3.7) and the theorem is concluded. □

The additional overheads of ST-GD^{2}P mainly lie in creating and maintaining the sorted array of search trees. Specifically, we have the following result.

Proposition 3.2. *For each utility function, ST-GD^{2}P requires $O(n^2 D^n_{\max})$ operations and space to perform preprocessing.*

Proof. For a utility function with n variables, inserting an assignment to a search tree requires $O(n)$ operations (line 22-29). Therefore, building search trees for each variable requires $O(nD^n_{\max})$ operations, and sorting these trees requires $O((n-1)D^n_{\max})$ operations in the worst case where all utility values are different. Given n variables to be preprocessed, the overall time complexity is $O(n^2 D^n_{\max} + (n^2 - n)D^n_{\max}) = O(n^2 D^n_{\max})$.

Similarly, in the worst case each search tree corresponds to a single assignment and takes $O(n)$ space. Therefore, ST-GD^{2}P needs $O(nD^n_{\max})$ space for a variable and $O(n^2 D^n_{\max})$ space in total in order to perform preprocessing. □

3.4.3 Discretization Mechanism

Since ST-GD^{2}P builds a search tree for each distinct local utility value, each tree would correspond to a small search space when the local utility values are extremely dense, which would incur unnecessary solution reconstructions. Consider the example in Figure 3.5. Assume that all the assignments with prefix $\{F, T\}$ are pruned by line 16 and the highest utility is attained on $\{F, F, F, F\}$. Obviously, ST-GD^{2}P has to visit the suboptimal partial assignment $\{F, T\}$ for four times in this case. Besides, storing small search trees is also memory-consuming due to the repetitive substructures. In this example, since each search tree only represents an assignment, substructures like $\{F, F, F\}$, $\{F, F, T\}$, $\{F, T, F\}$ and $\{F, T, T\}$ repeat twice, which results in the worst-case memory consumption. Finally, a large number of unique utility values would also incur a high sorting overhead.

We overcome the issues by introducing a discretization mechanism. That is, instead of sorting and building search trees for distinct local utility values directly, we first group the utility values into several discrete slots. In more detail, given a step size

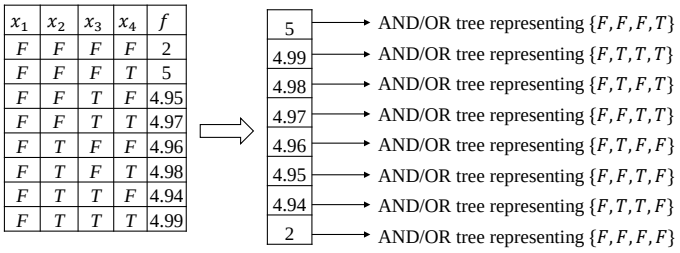


FIGURE 3.5: ST-GD^{2}P operating on extremely dense utilities

$\eta > 0$, each utility value u of a utility function is transformed to a discretized value $s = \eta\lceil u/\eta \rceil$, where $\lceil \cdot \rceil$ is the ceiling operation, which returns the least integer greater than or equal to a number. Then we sort all the distinct discretized values into a list $S = \{s_1, \ldots, s_{|S|}\}$ such that $s_i > s_j, \forall 1 \leq i < j \leq |S|$. For each $s \in S$, we merge all the entries in the utility function whose discretized value is s into a search tree $tree(d_i, s)$, and update the discretized value s to the maximum local utility value in the search space specified by $tree(d_i, s)$. Again, note that since search trees can be built incrementally, we only need a single scan to the utility function in order to perform preprocessing.

When computing a response message, we sequentially explore the discretized value list S, perform branch-and-bound on the corresponding search trees, and terminate if the current discretized value s is smaller than the running lower bound. It is worth noting that since discretized value s could be larger than the utility value of an entry in the search space specified by $tree(d_i, s)$, directly plugging s to Eq. (3.6) may not result in a tight upper bound and would jeopardize the performance of branch-and-bound. Therefore, for each node in the search tree $tree(d_i, s)$, we maintain an estimation $est_s(\cdot)$ which is the maximum local utility achieved by any full assignment derived from that node. Now, the upper bound of a partial assignment PA is given by [5]

$$ub^{PA} = est_s(PA) + \sum_{x_k \in PA} q_{x_k \to f_j}(PA[x_k]) + \sum_{x_k \notin PA} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k). \tag{3.10}$$

Figure 3.6 presents the search trees after performing discretization with a step size of 1 on the instance in Figure 3.5, where the number associated with each

[5]Recall that a path from the root to an internal node in a search tree specifies a partial assignment. We therefore slightly abuse $est_s(PA)$ to denote the estimation of PA in $tree(d_i, s)$ which is stored in the node that corresponds to PA.

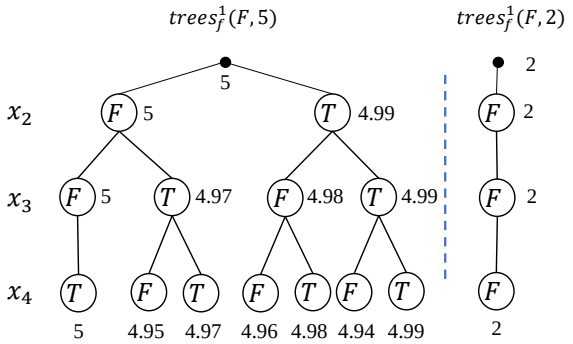


FIGURE 3.6: Discretization mechanism operating on extremely dense utilities

node is the estimation. It can be seen that both the number of search trees and total memory consumption are reduced significantly. In fact, if we directly apply ST-GD^{2}P on Figure 3.5, it would require 24 units of memory to store 8 search trees. While we only need 16 units of memory to maintain 2 search trees after discretization, since all small search trees with utility value greater than 4.94 are merged into a single large search tree (i.e., $tree_f^1(F,5)$).

It is noteworthy that beside helping to reduce the preprocessing overhead by restricting the number of discretized utility values, discretization mechanism also offers a tradeoff between domain pruning efficiency and reconstruction overhead. In more detail, a small η produces fine-grained slots and could prune the suboptimal slots promptly (line 12 of Alg. 2). In contrast, a large η tends to build search trees corresponding to large search spaces, which reduces the reconstruction overhead since we are less likely to revisit the same prefix when the search trees are large enough.

We now show the soundness and the complexity of discretization mechanism.

Theorem 3.5. *ST-GD^{2}P with discretization mechanism guarantees the optimality of Eq. (3.2).*

Proof. We first show that each discretized value s is no less than the local utility value of any entry in the space specified by $tree(d_i, s)$. Assume by contradiction that an entry e of utility function f_j belongs to the space specified by $tree(d_i, s)$ but $f_j(e) > s$. Its discretized value s_e is given by

$$s_e = \eta\lceil\frac{f_j(e)}{\eta}\rceil \geq f_j(e) > s,$$

which implies $s \neq s_e$ and thus e cannot belong to the space specified by $tree(d_i, s)$.

Assume that the optimal entry e^* belongs to a search tree which is pruned by the domain pruning part of the algorithm under the discretization mechanism. Since the discretized value is used to perform domain pruning, it must be the case that

$$f_j(e^*) \leq s_{e^*} < util^* - msgUB,$$

where s_{e^*} is the discretized value for entry e^*. Applying the same argument in the proof of Theorem 3.1, we conclude that e^* cannot be optimal and the domain pruning part preserves the optimality.

Similarly, assume that the optimal assignment A^* is pruned by the branch-and-bound part of the algorithm under the discretization mechanism. It must exist a prefix $PA \subset A^*$ such that

$$ub^{PA} = est_{s_{A^*}}(PA) + \sum_{x_k \in PA} q_{x_k \to f_j}(PA[x_k]) + \sum_{x_k \notin PA} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k) < util^*,$$

where s_{A^*} is the discretized value for assignment A^*. Since by definition $f_j(A^*) \leq est_{s_{A^*}}(PA)$, we have

$$\begin{aligned} ub^{PA} &\geq f_j(A^*) + \sum_{x_k \in PA} q_{x_k \to f_j}(PA[x_k]) + \sum_{x_k \notin PA} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k) \\ &\geq f_j(A^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A^*[x_k]), \end{aligned}$$

and

$$f_j(A^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A^*[x_k]) < util^*.$$

Applying the same argument in the proof of Theorem 3.3, we conclude that A^* cannot be optimal and the branch-and-bound part preserves the optimality. Therefore, ST-GD^{2}P with discretization mechanism guarantees the optimality of Eq. (3.2). □

Proposition 3.3. *Given a utility function f_j, let $\delta_j = \max_e f_j(e) - \min_e f_j(e)$ be the difference between the maximum and minimum utility values, and $\theta_j = \min_{e,e':f_j(e)\neq f_j(e')} |f_j(e) - f_j(e')|$ be the minimum difference between two distinct utility values. ST-GD^{2}P with discretization mechanism requires $O(n^2 D^n_{\max})$ operations and $O(\frac{\delta_j}{\eta} n^2 + n D^n_{\max})$ space to store search trees if $\eta > \theta_j$.*

Proof. We first note that ST-GD^{2}P with discretization mechanism degenerates to ST-GD^{2}P if step size η is small enough, i.e., $\eta \leq \theta_j$. Hereafter we will focus on the case $\eta > \theta_j$.

Like ST-GD^{2}P, ST-GD^{2}P with discretization mechanism also needs to insert $O(D^n_{\max})$ entries into search trees for each variable, which requires $O(n^2 D^n_{\max})$ operations in total. Since the number of search trees in discretization mechanism is bounded by δ_j/η, the overhead of sorting search trees is $O(n\frac{\delta_j}{\eta}\log\frac{\delta_j}{\eta})$, which is bounded by $O(nD^n_{\max}\log D^n_{\max}) = O(n^2 D^n_{\max})$. Therefore, the overall time complexity is $O(n^2 D^n_{\max})$.

For space complexity, note that these search trees span the total search space. Therefore, for each variable there are $O(D^n_{\max})$ leaves. On the other hand, since there are at most δ_j/η search trees according to the discretization mechanism, any partial assignment can only appear at most δ_j/η times (i.e., the case that every search tree contains the partial assignment). Since there are $n-2$ variables[6] in a partial assignment in the worst case, ST-GD^{2}P requires $O(\frac{\delta_j}{\eta}(n-2) + D^n_{\max}) = O(\frac{\delta_j}{\eta}n + D^n_{\max})$ space for a variable and $O(\frac{\delta_j}{\eta}n^2 + nD^n_{\max})$ space for all involved variables. □

3.4.4 Discussion

We note that our proposed algorithms can be easily adapted to the variants of Max-sum and other versions of belief propagation. For example, we could adapt our algorithms to speed up Max-product [151], which is an important instantiation of GDL to find maximum *a posteriori* (MAP) assignment of a probabilistic graphic model, by changing the summation operations to the product operations (i.e., line 4, 9 and 11 of Alg. 1, line 10, 14 and 16 of Alg. 2). That is, $msgUB$ now is the product of maximum value in the query messages from non-target variables, the result is computed by multiplying the function value with the corresponding value of query messages, and lb is computed by the best result found so far dividing $msgUB$. Alternatively, one can cast Max-product belief propagation to Max-sum and directly use our methods by operating on the log space of the problem.

[6]Note that the target variable is omitted from the search tree.

When combining with Max-sum variants like damped Max-sum [36], bounded Max-sum [34] and Max-sum_AD [9], our algorithms require few modifications since these variants still use Eq. (3.2) to compute response messages. However, there are variants which do not exactly follow Eq. (3.2). For example, a function node in Max-sum_ADSSVP [35] needs to consider the assignments of some neighbors when computing the maximum utility for Eq. (3.2). In this case, an additional verification procedure needs to be introduced to filter out the incompatible entries when performing branch-and-bound on a search tree. That is, when visiting a node whose variable needs to be fixed, we first check its compatibility before adding it to the partial assignment. If the assignment of the node is not consistent with the given assignment, we discard all the subsequent search space by backtracking to its parent.

Besides, it is also noteworthy that our techniques do not make any assumption on the sign of utility values (cf. Theorem 3.1, Theorem 3.3 and Theorem 3.5) and thus can be naturally generalized to accelerate Min-sum BP. In more detail, when applied to Min-sum BP, our domain pruning part sorts the entries in ascending order and maintains a running upper bound on local cost value, so as to discard entries (or search trees) with local cost higher than the upper bound. Similarly, in B&B part of our algorithms, we perform dynamic programming to backup the minimum local cost a partial assignment would attain. During depth-first search process, we construct a lower bound for each partial assignment by computing the sum of the corresponding minimum local cost, attained belief costs and the minimum belief cost for unassigned variables. Then a partial assignment is discarded if its lower bound is higher than the known upper bound.

3.5 Partial Tree-sorting Scheme

In this section, we present a K-depth partial tree-sorting scheme to reduce the preprocessing overhead of the proposed ST-GD^{2}P. We begin with the motivation of using limited-depth tree-sorting scheme. Then we detail the proposed K-depth partial tree-sorting scheme. Besides, we introduce several sorting criteria for ranking search spaces. Finally, we propose a built-in termination detection mechanism to avoid unnecessary enumeration.

3.5.1 Motivation

A major drawback of ST-GD^{2}P is the high preprocessing overhead incurred by creating and maintaining the full search trees. Although discretization mechanism attempts to alleviate the issue by merging small search trees, the complexities still depend on the step size and the gap between the maximum utility and the minimum utility which varies in different utility functions. Moreover, even in the best case, i.e., there is only one search tree for each variable-assignment pair, ST-GD^{2}P still requires $O(nD^n_{\max})$ memory per function node to maintain all trees, which is undesirable in memory-limited scenarios such as coordinating low-power embedded devices [33].

Therefore, we consider to overcome the defect by limiting the depth of the search trees. That is, given a utility function with the arity of n, instead of maintaining the full search trees, we only consider the search trees with the depth of $K < n$, resulting the worest-case memory overhead of $O(nKD^K_{\max})$ which allows the tradeoff between the memory consumption and orderliness of search space. However, since each search tree now only specifies a subspace, several key issues arise when realizing such K-depth tree sorting scheme:

- *How to implement branch-and-bound on limited-depth search trees?*
- *How to rank search trees when each search tree specifies a subspace rather than a concrete utility value?*
- *How to enforce domain pruning technique if the search trees are not ranked according to the maximum utility value in their subspace?*

We address the first issue in Section 3.5.2, by presenting a general framework called PTS. Then we present several sorting criterion to rank search trees to solve the second issue in Section 3.5.3. Finally, we propose a built-in termination detection mechanism to enforce the domain pruning technique in Section 3.5.4.

3.5.2 K-Depth Partial Tree-sorting Scheme

In this section, we aim to develop a configurable K-Depth Partial Tree-sorting Scheme (PTS) to provide a tight bound on the memory consumption, and perform preprocessing phase within a user-specified memory budget.

Algorithm 3: K-depth PTS for function node f_j

When `Initialization`**:**
foreach $x_i \in \mathbf{x_j}$ **do**
  foreach $d_i \in D_i$ **do**
    foreach $e \in \Pi_{x_k \in SV_j^i} D_k$ **do**
      $w_e \leftarrow$ digest the subspace $f_j(e, x_i = d_i, \cdot)$ by using a **sorting criterion** γ
      insert e to $Tree_j^i(d_i, w_e)$
    $SortedTree_j^i(d_i) \leftarrow$ sort $Tree_j^i(d_i, \cdot)$ by w in decreasing order
When *computing a response message for* $x_i \in \mathbf{x_j}$**:**
  $msgUB \leftarrow \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} \max_{d_k \in D_k} q_{x_k \to f_j}(d_k)$
  foreach $d_i \in D_i$ **do**
    $t \leftarrow$ the first element in $SortedTree_j^i(d_i)$
    $util^* \leftarrow -\infty, lb \leftarrow -\infty$
    while ***termination condition*** *is not met* **do**
      if $est(t) \geq lb$ **then**
        $util \leftarrow$`Branch-and-bound`$(t, util^*)$
        $util^* \leftarrow \max(util^*, util)$
        $lb \leftarrow util^* - msgUB$
      $t \leftarrow$ the next element in $SortedTree_j^i(d_i)$
    $r_{f_j \to x_i}(d_i) \leftarrow util^*$
  return $r_{f_j \to x_i}(x_i)$

Alg. 3 presents the sketch of the partial tree-sorting scheme. Instead of sorting and maintaining full search trees over $n-1$ variables for each variable-assignment pair in ST-GD^{2}P, PTS only considers the combinations of the first K non-target variables. Specifically, given the lexicographical ordering of the involved variables of a utility function f_j, PTS first computes sorting variables SV_j^i for target variable $x_i \in \mathbf{x_j}$ according to

$$SV_j^i = \begin{cases} \{\mathbf{x_{j,k}} | 1 \leq k \leq K\} & x_i \succ \mathbf{x_{j,K}} \\ \{\mathbf{x_{j,k}} | 1 \leq k \leq K+1 \wedge \mathbf{x_{j,k}} \neq x_i\} & \text{Otherwise} \end{cases},$$

where $x_i \succ \mathbf{x_{j,K}}$ means x_i is ordered after $\mathbf{x_{j,K}}$. Intuitively, SV_j^i is the first K variables in the scope of f_j, except target variable x_i. Then the algorithm iterates over all possible combinations of SV_j^i to build search trees (line 3-5). For each combination e, a *sorting criterion* γ is applied to assess the quality of the combination by summarizing the utility values in the subspace of $\prod_{x_i \in \mathbf{x_j}} D_i$ given the prefix of $e \cup \{x_i = d_i\}$ into a scalar weight w_e, which will be detailed in Section 3.5.3. Optionally, like discretizing utility values in ST-GD^{2}P, the weight also can be discretized by a step size of η. Then the combination is merged into the search tree which has the same weight (or same discretized weight if discretization is used). After exhausting the combinations of SV_j^i, PTS sorts all the search trees by their weight (line 6). Like in the discretization mechanism, when building search trees we also maintain an estimation for each node in a search tree for efficient branch-and-bound.

It is noteworthy that both weight and estimation are indispensable for efficient branch-and-bound. Horizontally, we sort the partial search trees by their weight for obtaining high-quality lower bound promptly; vertically, estimations are used to compute a tight upper bound for a partial assignment so as to discard suboptimal solutions as soon as possible.

When computing a response message for variable node $x_i \in \mathbf{x_j}$, PTS sequentially explores the sorted search trees by performing branch-and-bound to reduce the search space until a *termination condition* is reached (line 9-17). Since we only build search trees for the first K variables, the procedure `Branch-and-bound` (line 13) needs to be specialized to handle two halves of search spaces. Specifically, when performing branch-and-bound on a search tree (i.e., the first half of the search space), we follow exactly the same method in discretization mechanism and use Eq. (3.10) to compute upper bound for a partial assignment.

After reaching a leaf of a search tree, the procedure `Branch-and-bound` needs to exhaust the remaining subspace (i.e., the second half of the search space). In fact, the path from the root to a leaf of a search tree now only corresponds to a prefix rather than a concrete full assignment. In other words, we still need to solve a smaller optimization problem of Eq. (3.2) defined over $\mathbf{x_j} \backslash (SV_j^i \cup \{x_i\})$.

Technically, all the existing approaches (e.g., exhaustive enumeration, GDP, GD^{2}P, FDSP, etc.) can be used to exhaust the subspace. In our implementation, we use

FDSP due to its lower computational overhead. To this end, we maintain function estimations [59] for the remaining $n-K-1$ variables that do not appear in the search trees (i.e., the variables in $\mathbf{x_j} \backslash (SV_j^i \cup \{\mathbf{x_i}\})$). When exhausting the remaining subspace, the upper bound of a partial assignment is computed by querying the corresponding function estimation (i.e., Eqs. (3.3-3.4)).

It is noteworthy that our PTS requires less space than FDSP in the worst case when K is small. Formally, we have the following results.

Proposition 3.4. *Given a utility function, PTS requires $O(nKD_{\max}^K)$ space to store search trees.*

Proof. Since PTS only builds search trees for the first K variables, a path from a root to a leaf corresponds to an assignment of length K. The total number of combinations of K variables is $O(D_{\max}^K)$. Therefore, PTS requires $O(KD_{\max}^K)$ space for one variable in the worst case, and $O(nKD_{\max}^K)$ to store all search trees. □

Corollary 3.6. *PTS' worst-case memory consumption is less than FDSP's worst-case memory consumption when $K < \frac{1+nd}{n+d}$.*

Proof. For the remaining $n-K-1$ variables that do not appear in the search trees, PTS and FDSP require the same space as both of them need to store function estimations for each of these variables. For the first K variables, FDSP needs at least $O([1+d(n-K)]D_{\max}^K)$ space since maintaining function estimations for variable $\mathbf{x_{j,k}}$ requires $O([1+d(n-k)]D_{\max}^k)$ space [59]. Letting $nKD_{\max}^K < [1+d(n-K)]D_{\max}^K$, we arrive at $K < \frac{1+nd}{n+d}$. □

3.5.3 Sorting Criteria

In ST-GD^{2}P, each path from a root to a leaf in an search tree uniquely determines a full assignment to the involved variables of a utility function, and hence the utility values can be directly used to rank these paths/assignments. However, since only a subset of variables are sorted in PTS, a path now could correspond to a partial assignment and specify a *subspace* rather than a concrete utility value. Therefore, we propose to rank a partial assignment by a scalar weight that summarizes the utility values in the corresponding subspace (i.e., line 4, 6 of Alg. 3). Specifically,

given a utility function f_j and a partial assignment e of length K and the target variable-assignment pair $x_i = d_i$, we consider the following criteria to compute the weight for summarizing subspace $f_j(e, x_i = d_i, \cdot)$.

- **Maximum utility value**. The most straightforward way to describe the subspace is the maximum utility value in the subspace. Formally,

$$\gamma_{\max}(f_j(e, x_i = d_i, \cdot)) = \max_z f_j(e, x_i = d_i, z),$$

  where z is the remaining unassigned variables that do not appear in the prefix $e \cup \{x_i = d_i\}$. Computing the maximum utility value requires $O(1)$ space to store the result and $O(D_{\max}^{n-K-1})$ operations to exhaust the subspace.

- **Mean utility value**. Concentrating on the maximum utility value solely is not desirable when the maximum utility value is sparsely distributed in the subspace. Therefore, this criterion tries to remedy the issue by considering the average value of the subspace. That is,

$$\gamma_{\text{mean}}(f_j(e, x_i = d_i)) = \frac{\sum_z f_j(e, x_i = d_i, z)}{|f_j(e, x_i = d_i, \cdot)|}.$$

  The criterion has the same complexities as maximum utility value criterion, since both of them need to exhaust the subspace.

- Q_3 **value**. This criterion considers the third quartile of the utility values in the subspace. Denote the sorted utility values (in ascending order) of $f_j(e, x_i = d_i, \cdot)$ as $\mathcal{U}_{f_j}^e$. Q_3 value is given by

$$\gamma_{Q_3}(f_j(e, x_i = d_i, \cdot)) = \mathcal{U}_{f_j}^e[m] + \alpha(\mathcal{U}_{f_j}^e[m+1] - \mathcal{U}_{f_j}^e[m]),$$

  where $m = \lfloor 0.75 \times |f_j(e, x_i = d_i, \cdot)| \rfloor$ and $\alpha = 0.75 \times |f_j(e, x_i = d_i, \cdot)| - m$. The criterion needs to sort the utility values in the subspace, which requires $O((n-K)D_{\max}^{n-K-1})$ operations and $O(D_{\max}^{n-K-1})$ space.

- **H-utility value**. Analogous to the H-index [152] measuring the impact of a scholar, this criterion attempts to capture simultaneously the quality and

0	1	2	3	3	4	4	5	5	6	7	7	8	9	10
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

FIGURE 3.7: Sorted utility list $\mathcal{U}$

quantity of utility values. Formally, given sorted utility values $\mathcal{U}^e_{f_j}$, the H-position of subspace $f_j(e, x_i = d_i, \cdot)$ is given by

$$h = \max\{i \in \mathbb{N}^+ | 1 - \frac{i}{|f_j(e, x_i = d_i, \cdot)|} \geq \frac{\mathcal{U}^e_{f_j}[i] - f_j^-}{\delta_j}\},$$

where δ_j is the difference between the maximum and minimum utility values of f_j, and $f_j^- = \min_{\mathbf{x_j}} f_j(\mathbf{x_j})$. Intuitively, an H-position of h means that there are at least $\frac{\mathcal{U}^e_{f_j}[h] - f_j^-}{\delta_j} \times 100\%$ entries in $f_j(e, x_i = d_i, \cdot)$ whose utility value ranks at the top $(1 - \frac{\mathcal{U}^e_{f_j}[h] - f_j^-}{\delta_j}) \times 100\%$ of the utility range. Then H-utility is defined as

$$\gamma_{\text{H-utility}}(f_j(e, x_i = d_i, \cdot)) = \mathcal{U}^e_{f_j}[h].$$

Like Q_3 value criterion, this criterion also operates on the sorted utility list, which requires $O((n-K)D_{\max}^{n-K-1})$ operations and $O(D_{\max}^{n-K-1})$ space.

As an example, consider the following utility values.

$$0 \quad 1 \quad 3 \quad 5 \quad 8 \quad 10 \quad 2 \quad 6 \quad 4 \quad 7 \quad 9 \quad 4 \quad 5 \quad 7 \quad 3$$

The maximum utility value criterion returns $\gamma_{\max} = 10$, while the mean utility value is $\gamma_{\text{mean}} = 4.93$. For Q_3 value and H-utility criteria, we need to first sort the utility values. Figure 3.7 gives the sorted utility list $\mathcal{U}$. The third quartile index $m = \lfloor 15 \times 0.75 \rfloor = 11$ and $\alpha = 15 \times 0.75 - 11 = 0.25$. Therefore, $\gamma_{Q_3} = 7 + 0.25 \times (7 - 7) = 7$. For H-utility criterion, assume that $\delta = 10$ and $f^- = 0$. We iterate over the sorted utility list and find the H-position $h = 7$ since $1 - 7/15 > 4/10$ and $1 - 8/15 < 5/10$. Therefore, $\gamma_{\text{H-utility}} = 4$.

3.5.4 Built-in Termination Detection Mechanism

Since the search trees now are not necessarily ranked according to the maximum utility value in their subspace, we cannot directly perform domain pruning and

discard the remaining search space when the estimation of current search tree is less than the lower bound like ST-GD^{2}P (line 11 of Alg. 2). In fact, when detected a suboptimal search tree, the only thing we can do in PTS is to simply skip the tree and continue to explore the next search tree (line 12, 16 of Alg. 3). In other words, to compute a response utility for a particular assignment, we must visit all search trees related to the assignment, which is not desirable when the number of trees is large.

Therefore, we develop a termination detection mechanism to enable domain pruning with few extra overheads. Alg. 4 presents the sketch of the proposed mechanism. The general idea behind the mechanism is keeping track of the set of visited search

Algorithm 4: Termination detection mechanism for K-depth PTS

Input: Target variable $x_i \in \mathbf{x_j}$, assignment $d_i \in D_i$

When `Initialization`**:**

1 $T \leftarrow$ sort the search trees in $SortedTree^i_j(d_i)$ according to their estimation

2 $P \leftarrow$ an array of length $|T|$

When *starting to compute the response utility*:

3 $\ell \leftarrow 1$, $r \leftarrow |T|$

4 clear the flags in P

When *an search tree t is exhausted*:

5 $i \leftarrow$ the index of t in T

6 mark $P[i]$ as visited

7 **while** $P[\ell]$ *is visited* **do**

8 $\ell \leftarrow \ell + 1$

When *lower bound lb is updated*:

9 $r \leftarrow$`Binary-search`(lb, ℓ, r)

When *querying termination condition*:

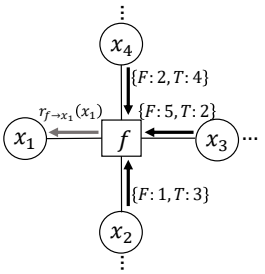
10 **if** $\ell > r$ **then**

11 **return** *"Terminate"*

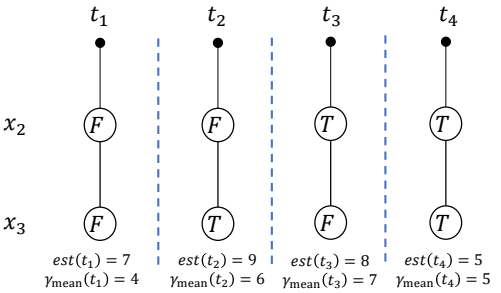
trees. If all search trees with estimation higher than current lower bound are visited, then PTS can terminate safely. To do this efficiently, we need to maintain another ordered search tree list T in which trees are sorted by their estimation (line 1), and a record array P. When PTS is invoked to compute a response utility, the mechanism clears the content of record array P, and initializes two pointers ℓ, r to the first position and the last position of T, respectively (line 3-4). After exhausting a search tree t, we mark the corresponding position as visited (line 5-6), and update the pointer ℓ to exclude all visited search trees (line 7-8). When the lower bound lb is updated, we update the pointer r by finding the last search tree in T

x_1	x_2	x_3	x_4	f
F	F	F	F	7
F	F	F	T	1
F	F	T	F	9
F	F	T	T	3
F	T	F	F	6
F	T	F	T	8
F	T	T	F	5
F	T	T	T	5

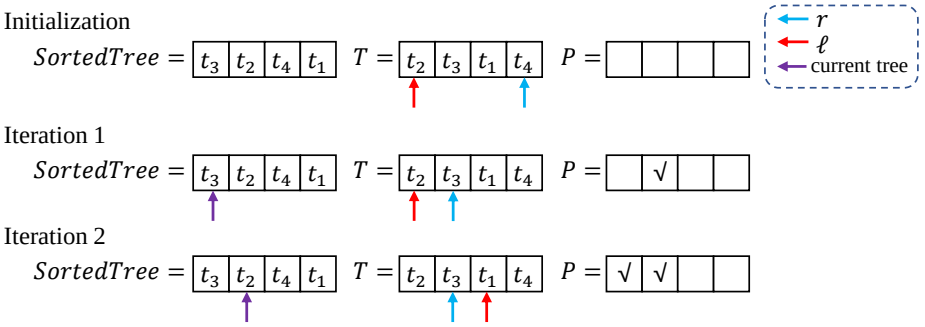
(a) partial utility table



(b) local interactions



(c) 2-depth search trees



(d) trace of PTS

FIGURE 3.8: The trace of PTS

whose estimation is no less than lb via binary search (line 9). Finally, PTS can terminate if $\ell > r$ (line 10-11).

Consider the example shown in Figure 3.8. Assume that we use 2-depth partial tree-sorting scheme with sorting criterion γ_{mean} to compute response utility for $x_1 = F$. Figure 3.8(c-d) give 2-depth search trees and sorted result (i.e., $SortedTree$), respectively. The algorithm begins with sorting the search trees by their estimation (i.e., T) and constructing an empty array P (the first row of Figure 3.8(d)). After exhausting the first search tree t_3, PTS updates lower bound $lb = 8$ (line 13-15 of Alg. 3). Therefore, termination detection mechanism marks t_3 as visited (line 5-6), and finds a new right pointer $r = 2$ which corresponds to the last search tree with estimation no less than 8 (line 9). The results are presented in the second row of Figure 3.8(d). Finally, after exhausting t_2, the mechanism marks the corresponding position as visited and updates the left pointer $\ell = 3$ (line 5-8). Since $\ell > r$, the algorithm terminates and prunes both t_1 and t_4.

We now are ready to show the correctness of PTS.

Theorem 3.7. *PTS guarantees the optimality of Eq. (3.2).*

Proof. There are three lines that prune the search space, i.e., line 11-13 of Alg. 3. Since we have already proved that line 12 and line 13 do not affect the optimality of Eq. (3.2) in Theorem 5, we only need to show that line 11 does not prune the optimal assignment.

Assume that the optimal assignment A^* belongs to a search tree t^* which is pruned by line 11 of Alg. 3. Denote the index of t^* in T as i^*. According to line 10-11 of Alg. 4, it must be the case that $i^* > r$ by the end of execution since otherwise PTS must have visited t^* (line 6-8 of Alg. 4), which implies that the estimation of t^* is less than lb (line 9 of Alg. 4). That is,

$$f_j(A^*) \leq est(t^*) < lb.$$

According to line 15 of Alg. 3, there must be an assignment $A' \neq A^*$ such that

$$\begin{aligned} f_j(A^*) &< f_j(A') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'[x_k]) - \max_{d_k \in D_k} q_{x_k \to f_j}(d_k) \\ &< f_j(A') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'[x_k]) - q_{x_k \to f_j}(A^*[x_k]) \end{aligned},$$

which implies

$$f_j(A^*) + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A^*[x_k]) < f_j(A') + \sum_{x_k \in \mathbf{x_j} \setminus \{x_i\}} q_{x_k \to f_j}(A'[x_k]).$$

Therefore, A^* cannot be an optimal assignment and PTS guarantees to return the optimal utility values. □

3.6 Empirical Evaluations

In this section, we empirically evaluate the proposed algorithms when accelerating Max-sum on various standard benchmark problems. We first present the results on random n-ary DCOPs to show our superiorities on general problems. To examine the performance on the problems with dense local utility values, we conduct experiments on the problems in which utility values follow a power-law distribution. Finally, we present the results on realistic benchmarks including NetRad systems and channel allocation problems.

3.6.1 Experimental Configurations

We benchmark algorithms with three types of problems, i.e., random n-ary DCOPs, NetRad system problems and channel allocation problems.

- *Random n-ary DCOPs* are the general form of the n-ary distributed constraint optimization problems. In the experiments, we vary the number of function nodes from 10 to 100. For each function node, we randomly sample the arity n from either [2,5] (low arity problems) or [5,8] (high arity problems), and connect the function node to n variable nodes. The total number of variable nodes is specified by variable tightness [59], which is randomly picked from either [0.1,0.5] (sparse problems) or [0.5,0.9] (dense problems). Further, we generate large domain problems and small domain problems by uniformly selecting a domain size from [5,8] and [2,5] for each variable, respectively. Finally, for each function node, the utility values are sampled either uniformly or according to a power-law from the range of $[0, 1000]$.

- *NetRad system problems* [146] aim to find a joint scanning strategy to maximize the total utility of scanning weather phenomena. Specifically, each radar is modelled as a variable whose assignment represents its scanning strategy (i.e., any combination of *E,N,W,S*). Weather phenomena with different sizes, weights and types are randomly generated across grids. Each phenomenon can be sensed by a subset of radars, and the scanning utility is determined by the joint strategy of the radars. In our experiments, the utility functions are defined according to [153]. We consider the NetRad systems with 48 and 96 radars which are arranged into 6×8 and 8×12 grids, respectively. In this set of experiments, the maximal utility of a function is 1, the maximal arity is 4 and the domain size of a variable can go up to 15.

- *Channel allocation problems* [4] try to coordinate a set of access points (APs) to minimize the radio interference. Let x_i be the channel allocated to AP i, P_i be the signal strength of i at the source, d_{ij} be the distance between AP i and j, and $Ap_i = \{j | P_j > N_b d_{ij}^2\}$ be the set of APs which can cause interference to the signal of i with background noise N_b. Then the utility

function for AP i is modelled as

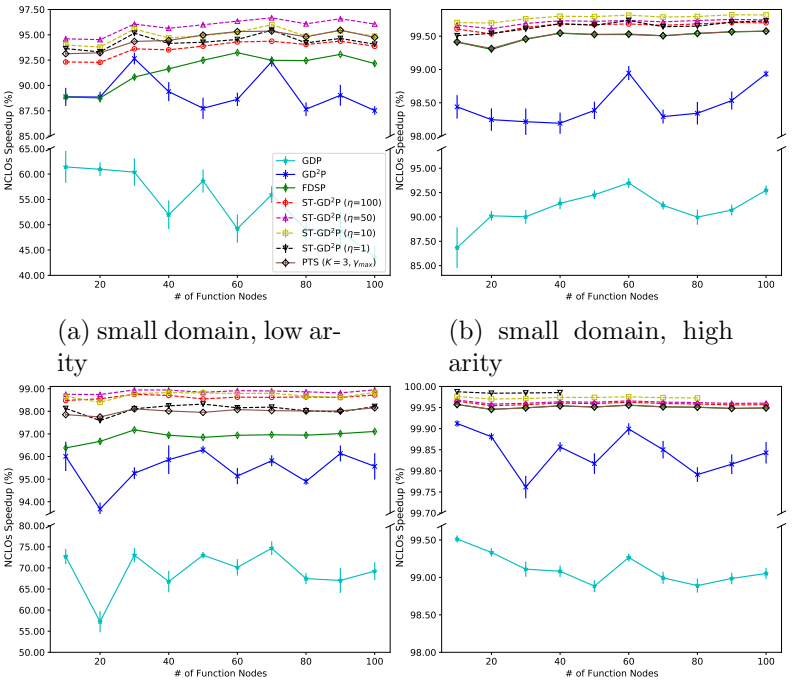
$$f_i(\mathbf{x_i}) = W \log_2 \left(1 + \frac{P_i}{1 + \sum_{j \in Ap_i} \frac{P_j}{d_{ij}^2} \mathbb{I}_{|x_i - x_j| \leq 3}} \right), \tag{3.11}$$

where W is a constant and $\mathbb{I}$ is the indicator function. In our experiments, we consider a 300×300 map, 10 available channels, and vary the number of APs from 60 to 100. For each AP i, we uniformly pick its power P_i from the range [490,510]. Finally, we set $W = 20$, and generate sparse problems and dense problems by setting $N_b = 1$ and $N_b = 0.5$, respectively.

The baselines we consider are GDP and FDSP since they are domain-independent state-of-the-art acceleration methods for belief propagation algorithms. Since channel allocation problems is a type of cardinality factors, we also consider THOP-based method [95] as an additional baseline.

The performance metrics we consider include NCLOs [23] speedup, pruned rate [60] and simulated runtime [154]. We consider a logical operation to be an access to a query message, and compute NCLOs speedup of an acceleration algorithm by dividing its total number of non-concurrent logical operations in an execution by the number of non-concurrent logical operations in vanilla Max-sum running for the same iterations. Then the NCLOs speedup is the remaining after subtracting the quotient from 1. This metric reflects the general acceleration capability of an algorithm in a distributed environment. Besides, we also consider the percentage of search space pruned (i.e., pruned rate) which is the improvement in terms of the total number of message accesses incurred in an execution. This metric drops the factor of concurrency and assesses purely the pruning capability of an algorithm. Finally, the simulated runtime directly reflects the overall performance of an acceleration algorithm in a distributed environment.

For each experiment, we generate 50 random instances and terminate algorithms after 2000 iterations with the timeouts of 1800s (in terms of simulated runtime), reporting the averaged metrics and standard errors as the results and the confidence intervals, respectively. For each instance, we host each function node randomly in one of its involved agents. All the experiments are conducted on a 32-cores Linux workstation with 384 GB memory. Finally, to avoid the potential precision loss incurred by floating-point numbers during an execution, we cast the utility values



(a) small domain, low arity

(b) small domain, high arity

(c) large domain, low arity

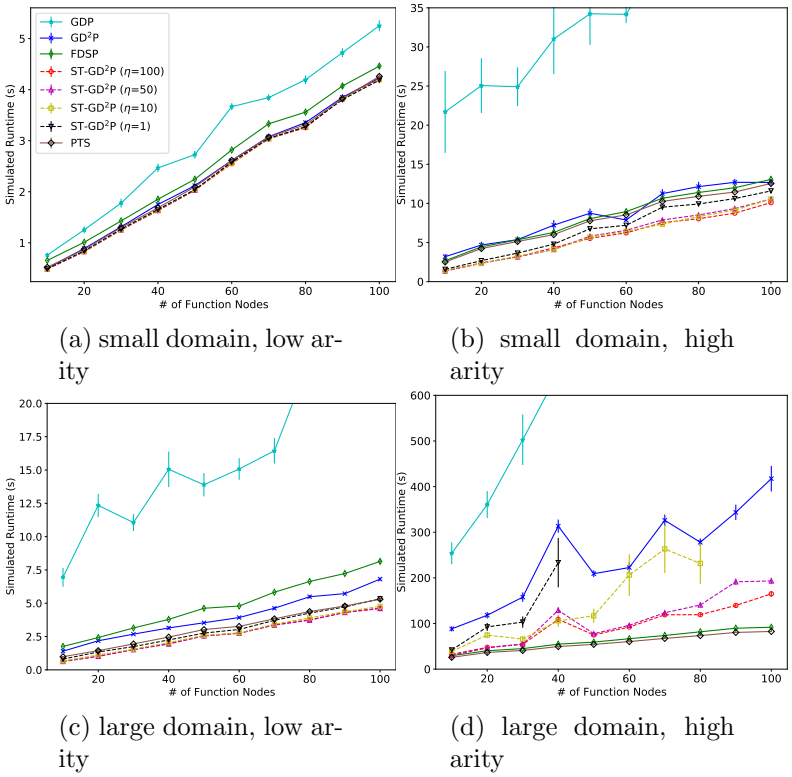
(d) large domain, high arity

FIGURE 3.9: NCLOs speedup of different algorithms on sparse random n-ary DCOPs

to big integers by multiplying by 10^6. That is important for a fair comparison since it ensures that every algorithm has the same query messages in every iteration and solves exactly the same optimization problem of Eq. (3.2) when computing response messages.

3.6.2 Evaluations of Dynamic Domain Pruning Techniques

Figure 3.9-3.10 present the results when solving sparse random n-ary DCOPs. It can be seen that FDSP performs worse than ST-GD^{2}P in terms of NCLOs speedup. That is due to the fact that FDSP performs a depth-first search on a huge search tree without any *a priori* knowledge. On the other hand, our ST-GD^{2}P performs branch-and-bound on a sorted array of search trees and can find an efficient lower bound more promptly, reducing about 95% - 99.9% of operations, which demonstrates great superiorities over the other competitors. It can be clearly seen from Figure 3.10 that the simulated runtime of all ST-GD^{2}P variants is lower than the one of FDSP, except the problems with large domain and high arity due to the excessive preprocessing overhead of building search trees. Nonetheless,



(a) small domain, low arity

(b) small domain, high arity

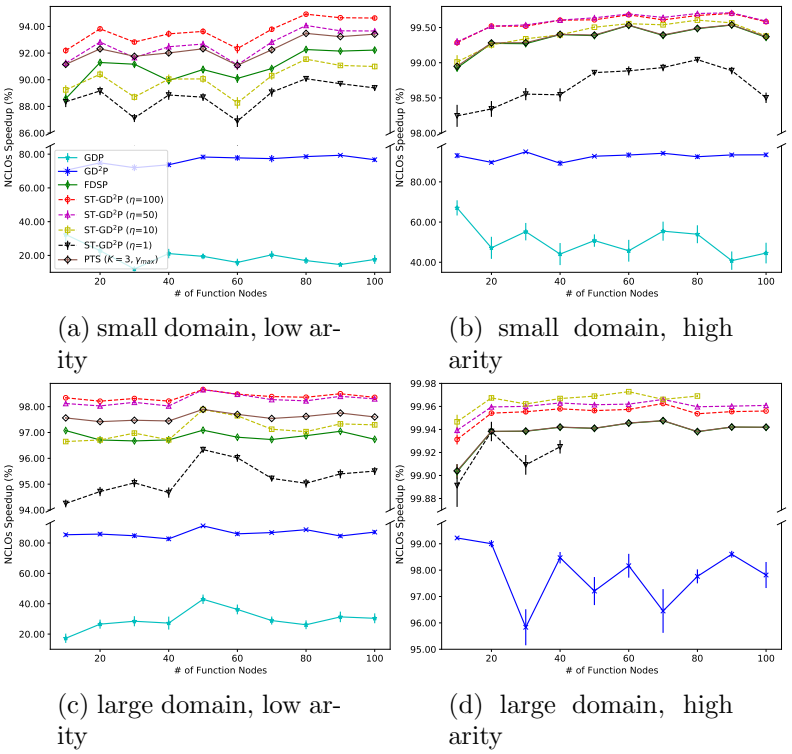
(c) large domain, low arity

(d) large domain, high arity

FIGURE 3.10: Simulated runtime of different algorithms on sparse random n-ary DCOPs

combining with K-depth partial tree-sorting scheme, our proposed PTS requires the lowest runtime on the problems with large domain and high arity.

Besides, our GD^{2}P maintains a running lower bound and significantly improves the performance of GDP, which highlights the importance of tight lower bounds in domain pruning techniques. In fact, GD^{2}P requires runtime lower than FDSP when solving the problems with low arity (i.e., Figure 3.10(a,c)). That might be due to the extra overhead FDSP needs to query function estimation to compute the upper bound for each partial assignment. Finally, step size plays an important role in the overall performance of ST-GD^{2}P. In more detail, ST-GD^{2}P with an appropriate step size (e.g., $\eta = 50$) has a higher NCLOs speedup than other versions of ST-GD^{2}P. That is because the algorithm with a small step size (e.g., $\eta = 1$) tends to build small search trees which lead to frequently solution reconstruction, while a large step size (e.g., $\eta = 100$) would produces coarse utility slots, preventing the algorithm from pruning suboptimal solutions promptly. Besides, a small step size can also incur a large preprocessing overhead. In fact, as shown in Figure 3.9(d) and 3.10(d), ST-GD^{2}P with $\eta = 1$ and $\eta = 10$ have significantly higher simulated



(a) small domain, low arity

(b) small domain, high arity

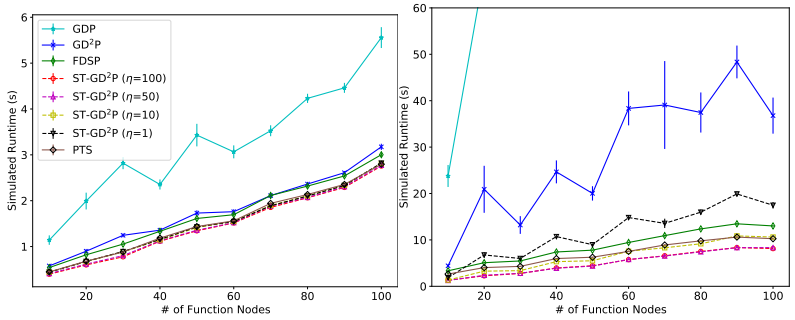
(c) large domain, low arity

(d) large domain, high arity

FIGURE 3.11: NCLOs speedup of different algorithms on dense random n-ary DCOPs

runtime than other variants, and run out of memory when solving the large-domain, high-arity problems with more than 40 and 80 function nodes, respectively.

Figure 3.11-3.12 present the results when solving dense random n-ary DCOPs. Compared to the ones in sparse problems, the variable nodes in these problems are over-constrained due to high variable tightness. Specifically, each variable node in this set of problems connects to 4 function nodes on average. As a result, query messages are amplified quickly due to the excessive cyclic information propagation in dense problems [155] as Max-sum proceeds, which may widen the difference between the maximum message utility and the message utility corresponding to the assignment with the highest local utility value. Therefore, GDP needs to explore more entries to cover the gap when solving a dense problem, and fails to solve the large-domain, high-arity problem in reasonable time. In fact, comparing to Figure 3.9 we can clearly see that the performance of GDP degenerates significantly and is strictly dominated by other competitors. In contrast, our dynamic domain pruning techniques are more robust to the high variable tightness due to the continuously tightening lower bound. Besides, it is interesting to find that different from the results on sparse problems, ST-GD^{2}P with small step size (e.g., $\eta = 1$)



(a) small domain, low arity

(b) small domain, high arity

(c) large domain, low arity

(d) large domain, high arity

FIGURE 3.12: Simulated runtime of different algorithms on dense random n-ary DCOPs

is dominated by FDSP when solving high-arity problems in terms of simulated runtime (i.e., Figure 3.12(b,d)). In addition to high preprocessing overhead, another possible reason for this could be that FDSP performs a depth-first search on search trees, which significantly reduces solution reconstructions. However, by limiting the depth of sorted tree, our proposed PTS reduces both preprocessing overhead and solution reconstruction overhead, which demonstrates its merits on general n-ary DCOPs.

To examine the general pruning capability on the problems with dense utility functions, we consider the problems with 100 function nodes, low arity, large domain size and the variable tightness of 0.5. For each problem, there are a number of dense function nodes whose utility values are selected according to a power-law distribution. More specifically, the probability of selecting a utility $u \in (0, 1000)$ is proportional to $(1000 - u)^{-\alpha}$. In our experiments, we set $\alpha = 1.1$. Figure 3.13 presents the performance comparison in terms of the pruned rate.

It can be seen that the performance of GDP decreases by near 20% w.r.t. growing dense function nodes, which indicates that GDP is sensitive to dense utility

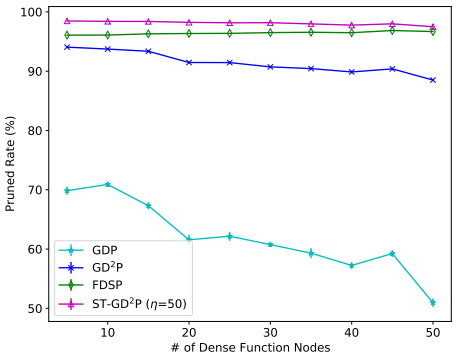
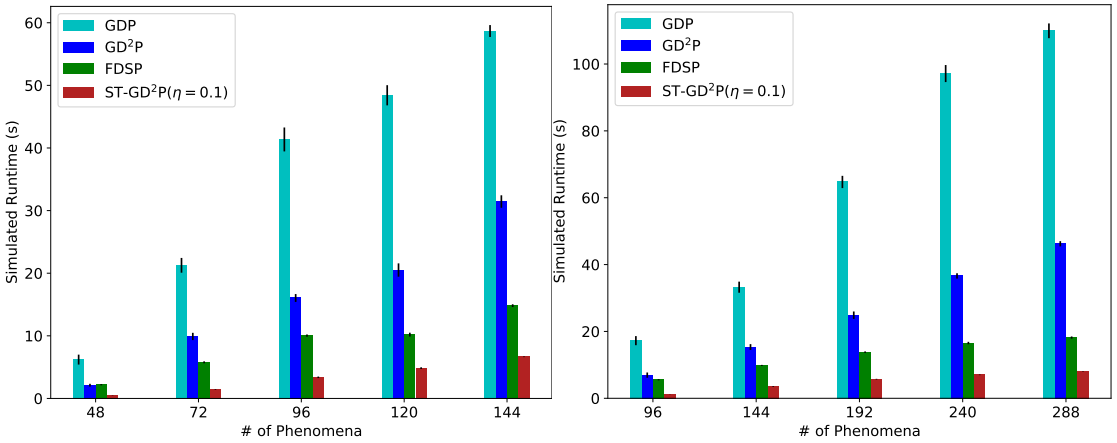


FIGURE 3.13: Performance comparison on the problems with dense utility functions



(a) 48 radars (b) 96 radars

FIGURE 3.14: Simulated runtime of different algorithms on NetRad system problems

functions. That is due to the fact that the pruned range could contain many entries since the utility values are close to each other in dense utility functions. On the other hand, although GD^{2}P also relies on domain pruning to reduce the search space, it is much more robust to the presence of dense function nodes due to the iteratively tighten lower bound. In fact, with the growing of dense function nodes its performance only drops by 4%. Finally, dense function nodes can hardly deteriorate the performance of ST-GD^{2}P since it can still perform effective branch-and-bound according to the query messages when local utility values are not distinguishable.

Figure 3.14 presents the results on NetRad system problems. It can be concluded that GDP performs poorly and the runtime grows quickly w.r.t. the number of phenomena in both cases. In contrast, our proposed GD^{2}P and ST-GD^{2}P significantly outperform GDP, only requiring about a half and one-tenth of its runtimes,

TABLE 3.2: Pruned rate (%) of PTS ($\eta = 0.1$) with different sorting criteria and depths on sparse channel allocation problems

# of access points		60	70	80	90	100
Average arity		3.24	3.28	3.29	3.39	3.51
$\gamma_{\max}$	$K = 1$	99.27±0.212	99.294±0.284	99.361±0.239	99.717±0.076	99.747±0.077
	$K = 2$	99.396±0.165	99.422±0.216	99.462±0.186	99.746±0.066	99.779±0.063
	$K = 3$	99.35±0.178	99.349±0.261	99.424±0.192	99.723±0.072	99.764±0.064
γ_{mean}	$K = 1$	99.343±0.191	99.354±0.258	99.42±0.219	99.737±0.073	99.778±0.067
	$K = 2$	99.41±0.164	99.427±0.215	99.47±0.185	99.75±0.067	99.786±0.061
	$K = 3$	99.347±0.18	99.34±0.264	99.413±0.197	99.717±0.074	99.761±0.065
γ_{Q_3}	$K = 1$	99.341±0.193	99.353±0.258	99.414±0.225	99.735±0.074	99.777±0.068
	$K = 2$	**99.417±0.163**	**99.436±0.212**	**99.476±0.183**	**99.751±0.066**	**99.789±0.06**
	$K = 3$	99.348±0.181	99.346±0.262	99.421±0.193	99.721±0.073	99.765±0.064
$\gamma_{\text{H-utility}}$	$K = 1$	99.303±0.209	99.313±0.286	99.395±0.232	99.731±0.076	99.772±0.07
	$K = 2$	99.229±0.245	99.242±0.347	99.368±0.238	99.721±0.078	99.765±0.07
	$K = 3$	98.974±0.329	98.96±0.513	99.189±0.303	99.64±0.104	99.699±0.092
FDSP		99.171±0.257	99.148±0.383	99.29±0.29	99.702±0.083	99.733±0.086
ST-GD^{2}P ($\eta = 0.1$)		97.8±0.549	98.471±0.286	98.744±0.112	99.013±0.11	99.17±0.073

respectively. On the other hand, although FDSP outperforms domain pruning variants, it is still dominated by ST-GD^{2}P. This is due to the fact that with the sorted search trees our proposed ST-GD^{2}P can find a high-quality lower bound quickly despite of highly-structured utility functions, which highlights the importance of search space sorting.

3.6.3 Evaluations of Partial Tree-sorting Scheme Variants

Table 3.2-3.3 tabulate the pruned rate of PTS with different sorting criteria and sorting depths on channel allocation problems. Compared to NetRad system problems, this set of problems are more challenging due to higher arity and more structured utility functions. In fact, both GDP and GD^{2}P fail to terminate in reasonable time. Therefore, we only present the results of PTS and FDSP. It can be concluded that both sorting depth K and sorting criterion γ can significantly affect the pruned rate. Specifically, PTS with a small K still can outperform FDSP. That is because FDSP sequentially explores all possible combinations starting from an all-one assignment, which is very inefficient in channel allocation problems since the first 4^{n-1} assignments correspond to low local utility values according to Eq. (3.11). On the other hand, our PTS tries to improve by ranking the most promising search trees at the top according to their weights, so as to find a high-quality lower bound more promptly. Therefore, a small permutation (e.g., $K = 1$) can greatly reduce

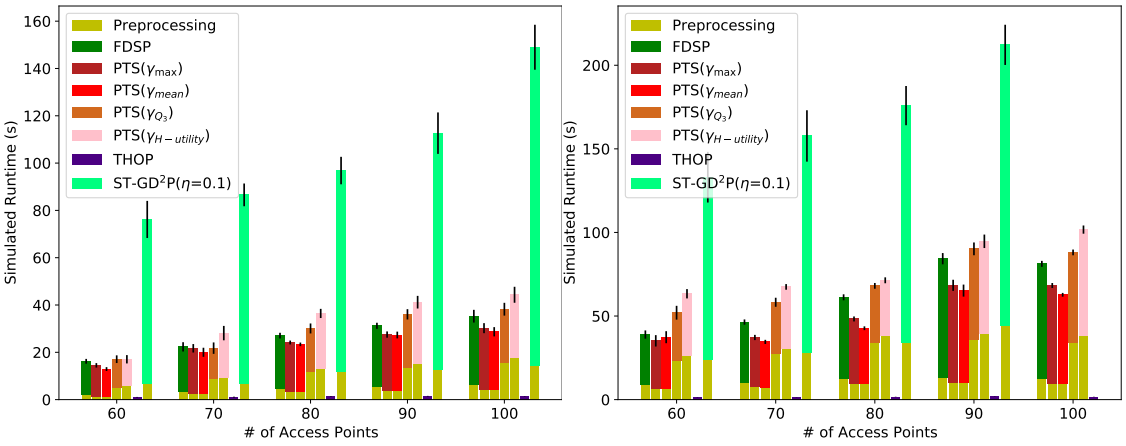
the search effort, and the benefit increases as the preprocessing budget K is increased. Besides, ST-GD^{2}P fails to solve the dense problems with 100 APs since it runs out of memory, and exhibits the worst performance on all remaining test cases. That might due to the large number of search trees after partitioning the search space of large factors. Consequently, solution reconstruction happens frequently and offsets the advantage of completely sorting.

For sorting criterion, it can be seen that directly using the maximum utility value to rank the search trees performs poorly when the sorting depth is small. That is not surprising since γ_{max} completely ignores the quantity of utility values in the remaining search space. However, the performance of γ_{max} is generally improved when K is large, especially on dense problems. That is because the size of the remaining search space is reduced if the search trees include more variables, and therefore the effect of quantity on pruning efficiency is reduced. Differently, γ_{mean} takes quantity into consideration, achieving better performance when K is small. However, its performance does not necessarily grow as the sorting depth increases. In fact, γ_{mean} is inferior to γ_{max} on the most of test cases when $K = 3$. Quartile criterion γ_{Q_3} considers the fine-grained statistical characteristics of utility value distribution, and is superior to all compared criteria as well as FDSP when $K = 2$. Finally, although $\gamma_{\text{H-utility}}$ tries to capture simultaneously the quality and quantity of utility values, it is strictly dominated by γ_{mean}. A possible reason could be that $\gamma_{\text{H-utility}}$ assumes a uniform distribution of utility values in calculating H-position, which may not be applicable to the highly-structured utility function of channel allocation problems.

Figure 3.15 presents simulated runtime of PTS with different sorting criteria on channel allocation problems when $K = 2$. Notably, THOP-based method exhibits the lowest runtime on all test cases. That is due to the fact that it only requires several linear scans over involved variables to perform sorting and dynamic programming when computing a response message. However, as we discussed earlier, THOP-based methods have a limited generality since it requires a utility function can be represented by a specific type of THOPs. Arguably, applying THOP-based methods also requires laborous expertise to identify the pattern of each utility function to implement a specific THOP-based method. Instead, our proposed algorithms make no assumption about the utility function and thus can be applied to general settings. It can be seen that compared to FDSP, PTS with γ_{max} and γ_{mean}

TABLE 3.3: Pruned rate (%) of PTS ($\eta = 0.1$) with different sorting criteria and depths on dense channel allocation problems

# of access points		60	70	80	90	100
Average arity		3.64	3.69	3.95	4.14	4.13
$\gamma_{\max}$	$K = 1$	99.559±0.272	99.824±0.106	99.879±0.031	99.89±0.027	99.878±0.046
	$K = 2$	99.639±0.208	99.85±0.087	99.893±0.027	99.903±0.022	99.893±0.038
	$K = 3$	99.653±0.199	99.856±0.082	99.894±0.024	99.906±0.02	99.896±0.034
γ_{mean}	$K = 1$	99.57±0.273	99.842±0.097	99.891±0.028	99.903±0.024	99.894±0.038
	$K = 2$	99.625±0.225	99.854±0.087	99.896±0.027	**99.909±0.021**	99.9±0.035
	$K = 3$	99.644±0.206	99.853±0.086	99.892±0.027	99.905±0.021	99.897±0.033
γ_{Q_3}	$K = 1$	99.578±0.266	99.842±0.097	99.891±0.029	99.903±0.024	99.894±0.039
	$K = 2$	99.651±0.202	99.857±0.084	**99.897±0.027**	**99.909±0.021**	**99.902±0.033**
	$K = 3$	**99.655±0.199**	**99.857±0.082**	99.893±0.026	99.906±0.021	99.898±0.032
$\gamma_{\text{H-utility}}$	$K = 1$	99.552±0.287	99.835±0.104	99.89±0.029	99.902±0.025	99.89±0.042
	$K = 2$	99.542±0.295	99.835±0.105	99.89±0.029	99.903±0.024	99.891±0.041
	$K = 3$	99.416±0.388	99.799±0.133	99.874±0.032	99.89±0.028	99.875±0.048
FDSP		99.464±0.349	99.804±0.129	99.877±0.032	99.886±0.031	99.875±0.045
ST-GD^{2}P ($\eta = 0.1$)		99.263±0.211	99.346±0.142	99.573±0.067	99.566±0.07	-



(a) sparse problems (b) dense problems

FIGURE 3.15: Simulated runtime of PTS ($\eta = 0.1$) different sorting criteria on channel allocation problems

has a smaller preprocessing overhead. That is because the calculation of maximum and mean utility value only requires linear time to scan entries, while FDSP needs to perform multiple phases of dynamic programming to compute the function estimations for each involved variables. On the other hand, the preprocessing phase of PTS with γ_{Q_3} and $\gamma_{\text{H-utility}}$ takes more time than FDSP. This is due to the higher complexity of utility sorting. In addition to lower preprocessing overhead, it is noteworthy that our proposed PTS with $\gamma_{\max}$ and γ_{mean} also significantly outperforms FDSP in terms of total runtime. In fact, our PTS is about 11% and 16% faster than FDSP on sparse problems when using $\gamma_{\max}$ and γ_{mean}, respectively.

3.7 Chapter Summary

Max-sum and its variants are important belief propagation algorithms for solving DCOPs but suffer from a high computational complexity. In this chapter, we demonstrate that the state-of-the-art algorithms for accelerating Max-sum could perform poorly when dealing with the problems with dense utility functions. To alleviate the negative effect of densely distributed utility values, we propose to iteratively update the lower bound and organize the tied entries to search trees. Further, we present a discretization mechanism to offer a tradeoff between the reconstruction overhead and domain pruning efficiency. Finally, we propose a K-depth partial tree-sorting scheme with different sorting criteria to reduce the preprocessing overhead. We theoretically show the correctness and superiority of our proposed algorithms. The extensive empirical evaluations confirm the advantages over the state-of-the-art methods on both synthetic and realistic benchmarks.

We note that all the existing acceleration methods are based on static knowledge and fail to exploit the characteristics of GDL algorithms or runtime information. For example, query messages in Damped Max-sum could change slowly when damping factor is high. Therefore, one can leverage such similarity between two consecutive iterations and give priority to search the neighborhood of optimal assignment found in the last iteration. In our future work, we plan to utilize the algorithmic characteristics or running history to further improve the performance of our algorithms.

Chapter 4

Neural Regret-Matching for Distributed Constraint Optimization Problems

4.1 Introduction

It can be concluded that most of incomplete algorithms for DCOPs are context-free, i.e., agents make a decision purely based on the state of their neighbors, which makes them prone to get trapped in poor local convergence since agents communicate their preferred decision based on the preferred decision of their neighbors [33]. While DUCT [4] tries to remedy the problem by exactly storing an upper confidence bound (i.e., an optimistic estimation of the optimal cost value for the subproblem of an agent) for each context and assignment, it may need considerable rounds to make the bounds informative due to the high variances of subproblem costs. Besides, exactly storing all confidence bounds requires exponential memory in the worst case, which severely limits its scalability.

Recent advances in Deep Reinforcement Learning (Deep RL) [156] have demonstrated the great potential of neural network for function approximation in handling large state space. Instead of storing information exactly in a table, deep RL uses deep neural networks to represent state or policy compactly and has led to tremendous success in various domains [44, 157, 158]. Unfortunately, there is

no previous work on employing function approximation to address the scalability limitation in the DCOP literature.

In this chapter[1], we aim to develop the first scalable and efficient neural-based sampling algorithm for DCOPs. We make the following key contributions:

- We first introduce a new context-based sampling algorithm for DCOPs built upon regret-matching [61]. Different from existing regret-based local search algorithms, our methods perform sampling on a pseudo tree and store regret values for each context separately, which allows an agent to devise different strategies for different contexts. Besides, compared to the existing sampling-based techniques, our method has a higher sample efficiency since it accumulates regret values according to the cost of the local problem rather than the subproblem of a variable.
- We present a neural-based scheme to improve the scalability of the proposed sampling algorithm by using deep neural networks to approximate the regret values. To the best of our knowledge, we are the first to leverage deep neural networks into solving DCOPs. To incentivize exploration, we propose a regret rounding scheme that rounds small regret values to positive numbers.
- We theoretically analyze the regret bounds of our algorithms, and extensive empirical evaluations indicate that our neural-based scheme can scale up to large-scale DCOPs and outperform the state-of-the-art methods.

The rest of this chapter is organized as follows. We review closely related works and the preliminaries of regret-matching in Section 4.2. Then we detail our context-based regret-matching for solving DCOPs in Section 4.3. In Section 4.4, We scale up our algorithm by leveraging DNNs to approximate regret values. Finally, we conduct empirical evaluations and conclude in Section 4.5 and Section 4.6, respectively.

4.2 Backgrounds

In this section, we review closely related work and present preliminaries for regret-matching.

[1]This chapter has been published in [159].

4.2.1 Related Work

Regret-based methods for DCOPs. In [62], Chapman et al. established the connection between DCOPs and potential games and presented a local search adaptation of regret-matching for DCOPs. In the algorithm, each agent accumulates regret based on the assignment of its neighbors and changes its assignment according to the regret values with a fixed probability p in each round. WRM-I [62, 160] is another regret-based local search scheme that converges to a pure-strategy Nash equilibrium. Different from the previous context-free methods, our methods perform sampling on a pseudo tree and store regret values for each context separately, which allows an agent to devise different strategies for different contexts. Additionally, we show that the negative regret values in vanilla regret-matching could result in poor performance and propose a novel regret rounding scheme.

Regret values approximation. Regression Regret-Matching (RRM) [161, 162] uses regression trees to estimate the accumulated regret. However, since a regression tree cannot be trained incrementally, RRM requires to re-train the model on *all* data as long as regret values are updated, which makes it very inefficient and scales up poorly. Deep CFR [163] estimate the regret by training a deep neural network on instantaneous regret values, which is not applicable to our case where instantaneous regret is not proportional to the total regret due to the non-linear regret rounding scheme.

4.2.2 Regret-matching

Consider a scenario in which an agent chooses a mixed strategy $\pi^t \in \Delta_{|A|}$[2] and observes a reward vector $f^t \in \mathbb{R}^{|A|}$ in each round t, where A is the set of actions. Then the instantaneous regret of action $a \in A$ is $r^t(a) = f^t(a) - \sum_{a'} \pi^t(a') f^t(a')$ and the accumulated regret is $R^t(a) = \sum_{t'=1}^{t} r^{t'}(a)$. To improve the total reward, the agent needs to minimize the accumulated regret. Regret-matching [61] is an efficient algorithm for regret minimization, which computes a mixed strategy according to

[2] $\Delta_{|A|}$ is the set of probability distributions over set A

the positive part of accumulated regret. That is,

$$\pi^{t+1}(a) = \begin{cases} \frac{R^{t,+}(a)}{\sum_{a' \in A} R^{t,+}(a')} & \sum_{a' \in A} R^{t,+}(a') > 0 \\ \frac{1}{|A|} & \text{otherwise} \end{cases}, \quad \forall a \in A, \tag{4.1}$$

where $R^{t,+}(a) = \max(0, R^t(a))$ is the positive part of $R^t(a)$. Regret-matching has the regret bound of $L\sqrt{T|A|}$, where L is the largest gap in reward vectors. Because the accumulated regret grows sublinearly w.r.t. the number of rounds, regret-matching is a so-called no-regret algorithm [164].

4.3 Context-based Regret-matching for DCOPs

In this section, we present context-based regret-matching schemes for DCOPs. We begin with formally introducing our proposed sampling algorithm in Section 4.3.1. Then we show that the vanilla regret-matching could perform poorly and present the regret rounding scheme in Section 4.3.2.

4.3.1 Context-based Regret-matching Scheme

The proposed context-based regret-matching scheme alternatively executes a sampling phase and a back-propagation phase on a pseudo tree. Starting from the root agent, each agent in sampling phase sequentially selects an assignment according to the regret vector associated with the received context, while back-propagation phase is a bottom-up procedure to refine the solution and update regret values for each agent.

Let us begin with introducing notations that will be used in the algorithm. For a variable x_i in a pseudo tree, we denote an assignment to the variables in $Sep(x_i)$ (i.e., a context) as X_i and an assignment to the variables in $AC(x_i)$ as Y_i, respectively. In each round t, after collecting X_i^t and Y_i^t, the hindsight local cost $S_i^t(d_i)$ of each assignment $d_i \in D_i$ is then computed by

$$S_i^t(d_i) = \sum_{x_j \in AP(x_i)} f_{ij}(d_i, X_i^t(x_j)) + \sum_{x_k \in AC(x_i)} f_{ik}(d_i, Y_i^t(x_k)), \tag{4.2}$$

where $X_i^t(x_j)$ denotes the assignment of x_j in X_i^t and $Y_i^t(x_k)$ denotes the assignment of x_k in Y_i^t, respectively. Given the mixed strategy π_i^t of x_i, the expected local cost $s_i^t = \sum_{d_i \in D_i} \pi_i^t(d_i) S_i^t(d_i)$. Finally, the accumulated regret of x_i in round t is defined as

$$R_i^t(X_i, d_i) = \begin{cases} R_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i) & X_i = X_i^t \\ R_i^{t-1}(X_i, d_i) & \text{otherwise} \end{cases}. \tag{4.3}$$

In each round t, the sampling phase is initiated by the root agent. After receiving the context X_i^t from its parent, a non-leaf variable x_i first computes the mixed strategy π_i^t by

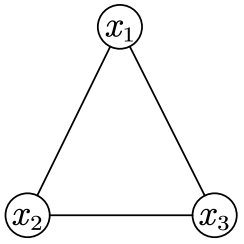
$$\pi_i^t(d_i) = \begin{cases} \frac{R_i^{t-1,+}(X_i^t, d_i)}{\sum_{d' \in D_i} R_i^{t-1,+}(X_i^t, d')} & \sum_{d' \in D_i} R_i^{t-1,+}(X_i^t, d') > 0 \\ \frac{1}{|D_i|} & \text{otherwise} \end{cases}, \ \forall d_i \in D_i. \tag{4.4}$$

Then it samples an assignment $d_i^t \sim \pi_i^t$ and sends the augmented context $X_i^t \cup \{x_i = d_i^t\}$ to its children $C(x_i)$. Leaf variables, however, do not have any subproblem and thus directly select the assignment that minimizes their local cost. The phase ends when all leaf agents finish sampling.

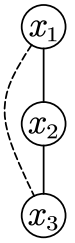
The back-propagation phase is a bottom-up procedure initiated by leaf agents sending the selected assignment to their direct ancestors. After collecting all the assignments of its direct descendants as Y_i^t, a variable x_i computes the hindsight local cost vector S_i^t and updates the regret R_i^t according to Eq. (4.2) and Eq. (4.3), respectively. Finally, x_i refines the solution by changing its assignment to the one with the lowest hindsight local cost and sends the assignment to its direct ancestors. The procedure ends after the root agent updates its regret and then the next round of sampling starts.

To look deeper into how context-based sampling avoids poor local convergence, consider the instance shown in Figure 4.1. In the first round of sampling, all non-leaf agents have no prior knowledge and sample randomly. Assume that x_1 assigns F, x_2 assigns T and thus leaf agent x_3 plays the best response F. If it does not differentiate contexts, x_2 would update its regret values by

$$\begin{cases} S_2^1 = [18, 16] \\ s_2^1 = 18 \times 0.5 + 16 \times 0.5 = 17 \end{cases} \quad \begin{cases} R_2^1(\mathrm{T}) = 17 - 18 = -1 \\ R_2^1(\mathrm{F}) = 17 - 16 = 1 \end{cases}$$



(a) The constraint graph



(b) A possible pseudo tree

x_2 \ x_1	T	F
T	0	10
F	7	15

x_3 \ x_1	T	F
T	0	10
F	3	1

x_3 \ x_2	T	F
T	0	10
F	8	1

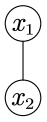
(c) Constraint functions

FIGURE 4.1: A DCOP and its pseudo tree.

which means that x_2 cannot assign T in the next round. In fact, one can easily verify that no matter what value x_1 assigns, if x_2 assigns F and x_3 plays the best response, then the instantaneous regret for T is always negative and x_2 cannot deviate to T, which precludes the possibility of reaching the optimal solution {T, T, T} in subsequent rounds.

The reason behind such mis-coordination is that in context-free method each agent fails to adapt the complex behavior of its lower priority neighbors since it uses the same regret values to make decision for different contexts. In the above example, given the fact that x_1 assigns F and x_3 plays the best response, x_2 would conclude that assigning F is better than assigning T, which is not true if x_1 switches to T. In contrast, our context-based sampling scheme explicitly maintains regret values for each context, which allows agents to adapt the multi-modal behavior of its descendants. In more detail, x_2 in our method would associate the fact "F is superior over T" with context $\{x_1 = \text{F}\}$, which does not bias the decision-making procedure under the context $\{x_1 = \text{T}\}$.

Since it contains all assignments of separators, the size of a context message is proportional to the number of agents, i.e., $O(|I|)$. Note that in sampling phase, each non-leaf agent sends a context message to each of its children via tree edges. Therefore, there are $|I| - 1$ context messages in the sampling phase, and the total information exchanged is in $O(|I|^2)$. Back-propagation phase, on the other hand, exchanges the assignments among neighbors, which induces $|F|$ messages of size



(a) The pseudo tree

x_2 \ x_1	T	F
L	2	0
R	1	3

(b) The constraint function

FIGURE 4.2: A DCOP instance with structured constraint functions

$O(1)$. Therefore, the total size of messages exchanged in each round is in $O(|I|^2 + |F|) = O(|I|^2)$.

4.3.2 Regret Rounding Scheme

Regret-matching could perform poorly in the context of DCOP solving due to insufficient exploration. Consider the simple instance in Figure 4.2. Assume that x_1 selects d_1^1 =T and x_2 selects d_2^1 =R in the first round of sampling. The following equations show the trace when x_1 updates its regret.

$$\begin{cases} S_1^1 = [1, 3] \\ s_1^1 = 1 \times 0.5 + 3 \times 0.5 = 2 \end{cases} \qquad \begin{cases} R_1^1(\emptyset, \mathrm{T}) = 2 - 1 = 1 \\ R_1^1(\emptyset, \mathrm{F}) = 2 - 3 = -1 \end{cases}$$

According to Eq. (4.4) the assignment F cannot be selected by x_1 in the subsequent rounds since its regret is negative, even though $\{x_1 = \mathrm{F}, x_2 = \mathrm{L}\}$ is the optimal solution. In other words, negative regret would limit the exploration of the algorithm. Although regret-matching$^+$ [165] mitigates the issue by resetting the negative regret to zero, selecting these assignments still depends on the positive instantaneous regret in subsequent rounds, which is highly coupled with the behavior of other agents.

We trigger the effective exploration by rounding small regret values to positive numbers. This way, all assignments have a non-zero probability in the mixed strategy and the exploration is independent of other agents' behavior. Specifically, instead of maintaining regret R_i^t for each variable x_i, we maintain the rounded regret $\bar{R}_i^t$:

$$\bar{R}_i^t(X_i, d_i) = \begin{cases} \max\left(\delta_t, \bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)\right) & X_i = X_i^t \\ \bar{R}_i^{t-1}(X_i, d_i) & \text{otherwise} \end{cases}, \tag{4.5}$$

where $\bar{R}_i^0(\cdot,\cdot) = 0$ and $\delta_t > 0$ is a non-decreasing term. In this example, if set $\delta_t = t^\alpha, \alpha > 0$, then $\bar{R}_1^1(\emptyset, \mathrm{T}) = \bar{R}_1^1(\emptyset, \mathrm{F}) = 1$ and thus x_1 still has chance to select assignment F in subsequent rounds, regardless of the strategy selected by x_2.

We theoretically show the upper bound of the rounded regret in Theorem 4.1 and provide the no-regret guarantee in Theorem 4.4.

Theorem 4.1. *For variable x_i, let $L_i = \sum_{x_j \in N_i}(f_{ij}^+ - f_{ij}^-)$ be the largest gap in its local cost, where $f_{ij}^+ = \max_{d_i}\max_{d_j} f_{ij}(d_i, d_j)$ and $f_{ij}^- = \min_{d_i}\min_{d_j} f_{ij}(d_i, d_j)$ are the upper bound and lower bound of constraint f_{ij}, respectively. After x_i finishes T rounds of sampling, the rounded regret $\bar{R}_i^T(X_i, d_i)$ is no higher than $\sqrt{|D_i|\left(TL_i^2 + \sum_{t=1}^T \delta_t^2\right)}$ for all X_i, d_i.*

Proof. Adapted from [165]. Since $\bar{R}_i^T(X_i, d_i) > 0$,

$$\left(\max_{d_i} \bar{R}_i^T(X_i, d_i)\right)^2 = \max_{d_i} \bar{R}_i^T(X_i, d_i)^2 \le \sum_{d_i} \bar{R}_i^T(X_i, d_i)^2.$$

If $X_i = X_i^T$, then

$$\begin{aligned}
\sum_{d_i} \bar{R}_i^T(X_i, d_i)^2 &= \sum_{d_i} \max\left(\bar{R}_i^{T-1}(X_i, d_i) + s_i^T - S_i^T(d_i), \delta_T\right)^2 \\
&\le \sum_{d_i} \left(\bar{R}_i^{T-1}(X_i, d_i) + s_i^T - S_i^T(d_i)\right)^2 + \delta_T^2 \\
&= |D_i|\delta_T^2 + \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2 + 2\bar{R}_i^{T-1}(X_i, d_i)\left(\sum_{d_i'} \pi_i^T(d_i') S_i^T(d_i') - S_i^T(d_i)\right) \\
&\quad + \left(\sum_{d_i'} \pi_i^T(d_i') S_i^T(d_i') - S_i^T(d_i)\right)^2.
\end{aligned}$$

Since $L_i \geq |S_i^t(d_i) - S_i^t(d_i')|$ for all $d_i \neq d_i'$, we have

$$\begin{aligned}
\sum_{d_i} \bar{R}_i^T(X_i, d_i)^2 &\leq |D_i|\delta_T^2 + |D_i|L_i^2 + \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2 + 2\bar{R}_i^{T-1}(X_i, d_i)\left(\sum_{d_i'} \frac{\bar{R}_i^{T-1}(X_i, d_i')}{\sum_{d_i''} \bar{R}_i^{T-1}(X_i, d_i'')} S_i^T(d_i') - S_i^T(d_i)\right) \\
&= |D_i|\delta_T^2 + |D_i|L_i^2 + \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2 + 2\left(\sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i) \sum_{d_i'} \frac{\bar{R}_i^{T-1}(X_i, d_i')}{\sum_{d_i''} \bar{R}_i^{T-1}(X_i, d_i'')} S_i^T(d_i')\right. \\
&\quad \left. - \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i) S_i^T(d_i)\right) \\
&= |D_i|\delta_T^2 + |D_i|L_i^2 + \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2 + 2\left(\frac{\sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)}{\sum_{d_i''} \bar{R}_i^{T-1}(X_i, d_i'')} \sum_{d_i'} \bar{R}_i^{T-1}(X_i, d_i') S_i^T(d_i')\right. \\
&\quad \left. - \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i) S_i^T(d_i)\right) \\
&= |D_i|\delta_T^2 + |D_i|L_i^2 + \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2 + 2\left(\sum_{d_i'} \bar{R}_i^{T-1}(X_i, d_i') S_i^T(d_i') - \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i) S_i^T(d_i)\right) \\
&= |D_i|\delta_T^2 + |D_i|L_i^2 + \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2.
\end{aligned}$$

If $X_i \neq X_i^T$, according to Eq. (4.5), we have

$$\sum_{d_i} \bar{R}_i^T(X_i, d_i)^2 = \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2.$$

Therefore,

$$\left(\max_{d_i} \bar{R}_i^T(X_i, d_i)\right)^2 \leq |D_i|\delta_T^2 + |D_i|L_i^2 + \sum_{d_i} \bar{R}_i^{T-1}(X_i, d_i)^2.$$

By iterating over the above formula, we get $\left(\max_{d_i} \bar{R}_i^T(X_i, d_i)\right)^2 \leq |D_i| \left(TL_i^2 + \sum_{t=1}^T \delta_t^2\right)$ and therefore $\max_{d_i} \bar{R}_i^T(X_i, d_i) \leq \sqrt{|D_i| \left(TL_i^2 + \sum_{t=1}^T \delta_t^2\right)}, \forall X_i$. □

To show our regret rounding scheme is no-regret, we first show that the total regret is bounded by the maximal regret of an individual variable. Then we show the total regret grows sublinearly.

Let $d^T = (d_1^T, \ldots, d_n^T)$ be the joint assignment of round T, $d_{-i}^T = d^T \backslash \{d_i^T\}$ be the set of assignments selected by variables other than x_i and $F(d^T) = \sum_{f_{ij} \in F} f_{ij}(d_i^T, d_j^T)$ be the sum of all constraints given joint assignment d^T.

Definition 4.1. The 1-opt regret of a DCOP is

$$R_{1-opt}^T = \max_{i \in I} \max_{d_i \in D_i} \sum_{t=1}^T \left(\sum_{d_i' \in D_i} F(d_{-i}^t, d_i') \pi_i^t(d_i') - F(d_{-i}^t, d_i) \right). \tag{4.6}$$

Lemma 4.2. *$R^T_{1-opt} \leq K \max_i R^T_i$, where $R^T_i = \max_{X_i} \max_{d_i} R^T_i(X_i, d_i)$ is the maximal local regret of variable x_i at round T and K is a constant.*

Proof. Given joint assignment d, denote the sum of the constraints which are related to x_i as $F_i(d_i, d_{N_i}) = \sum_{x_j \in N_i} f_{ij}(d_i, d_j)$ where $d_{N_i} \subseteq d$ is the set of assignments to x_i's neighbors, and the sum of those which are not related to x_i as $F_{-i}(d_{-i}) = F(d) - F_i(d_i, d_{N_i})$. Note that

$$\begin{aligned}
\max_{d_i} \sum_{t=1}^{T} \left(\sum_{d'_i} F(d^t_{-i}, d'_i)\pi^t_i(d'_i) - F(d^t_{-i}, d_i) \right) &= \max_{d_i} \sum_{t=1}^{T} \left(\sum_{d'_i} \left(F_{-i}(d^t_{-i}) + F_i(d'_i, d^t_{N_i})\right) \pi^t_i(d'_i) \right. \\
&\quad \left. - \left(F_{-i}(d^t_{-i}) + F_i(d_i, d^t_{N_i})\right) \right) \\
&= \max_{d_i} \sum_{t=1}^{T} \left(\sum_{d'_i} F_{-i}(d^t_{-i})\pi^t_i(d'_i) + \sum_{d'_i} F_i(d'_i, d^t_{N_i})\pi^t_i(d'_i) \right. \\
&\quad \left. - \left(F_{-i}(d^t_{-i}) + F_i(d_i, d^t_{N_i})\right) \right) \\
&= \max_{d_i} \sum_{t=1}^{T} \left(\sum_{d'_i} F_i(d'_i, d^t_{N_i})\pi^t_i(d'_i) - F_i(d_i, d^t_{N_i}) \right) \\
&= \max_{d_i} \sum_{t=1}^{T} \left(\sum_{d'_i} \left(\sum_{x_j \in AP(x_i)} f_{ij}(d'_i, d^t_j) + \sum_{x_k \in AC(x_i)} f_{ij}(d'_i, d^t_k) \right) \pi^t_i(d'_i) \right. \\
&\quad \left. - \left(\sum_{x_j \in AP(x_i)} f_{ij}(d_i, d^t_j) + \sum_{x_k \in AC(x_i)} f_{ij}(d_i, d^t_k) \right) \right) \\
&= \max_{d_i} \sum_{t=1}^{T} s^t_i - S^t_i(d_i).
\end{aligned}$$

Assume that x_i has received $k^T_i \leq K_i$ different contexts $X_{i,1}, \ldots, X_{i,k^T_i}$ in first T rounds, where K_i is the total number of possible contexts of x_i.

$$\begin{aligned}
\max_{d_i} \sum_{t=1}^{T} s^t_i - S^t_i(d_i) &= \max_{d_i} \sum_{j=1}^{k^T_i} R^T_i(X_{i,j}, d_i) \\
&\leq \sum_{j=1}^{k^T_i} \max_{d_i} \max_{X_i} R^T_i(X_i, d_i) = k^T_i R^T_i \leq K_i R^T_i.
\end{aligned}$$

Therefore,

$$R^T_{1-opt} \leq \max_i K_i R^T_i = K \max_i R^T_i$$

where $K = \max_i K_i$. □

Lemma 4.3. *Rounded regret are the upper bound on regret, i.e., $\bar{R}^t_i(X_i, d_i) \geq R^t_i(X_i, d_i), \forall t, X_i, d_i$.*

Proof. For any $t > 0$, if $X_i = X_i^t$, then

$$\begin{aligned}\bar{R}_i^t(X_i, d_i) - \bar{R}_i^{t-1}(X_i, d_i) &= \max\left(\bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i), \delta_t\right) - \bar{R}_i^{t-1}(X_i, d_i) \\ &\geq \bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i) - \bar{R}_i^{t-1}(X_i, d_i) \\ &= s_i^t - S_i^t(d_i) = R_i^t(X_i, d_i) - R_i^{t-1}(X_i, d_i).\end{aligned}$$

Otherwise,

$$\bar{R}_i^t(X_i, d_i) - \bar{R}_i^{t-1}(X_i, d_i) = R_i^t(X_i, d_i) - R_i^{t-1}(X_i, d_i) = 0.$$

Note that $\bar{R}_i^0(X_i, d_i) = R_i^0(X_i, d_i) = 0$. By induction, $\bar{R}_i^t(X_i, d_i) \geq R_i^t(X_i, d_i)$ holds for any t, X_i and d_i, which concludes the lemma. □

Theorem 4.4. *When* $\lim_{t\to\infty} \frac{\delta_t}{\sqrt{t}} = 0$*, the regret rounding scheme is no-regret, i.e., the total regret grows sublinearly.*

Proof. According to Lemma 4.3, we have $\bar{R}_i^T(X_i, d_i) \geq R_i^T(X_i, d_i), \forall T, X_i, d_i$. Therefore, to show the algorithm is no-regret, we only need to show the maximal rounded regret $\bar{R}_i^T = \max_{X_i} \max_{d_i} \bar{R}_i^T(X_i, d_i)$ grows sublinearly.

$$\begin{aligned}\lim_{T\to\infty} \frac{\bar{R}_i^T}{T} &\leq \lim_{T\to\infty} \sqrt{\frac{|D_i|\left(TL_i^2 + \sum_{t=1}^T \delta_t^2\right)}{T^2}} \\ &= \lim_{T\to\infty} \sqrt{\frac{|D_i|L_i^2}{T^2} + \frac{|D_i|\sum_{t=1}^t \delta_t^2}{T^2}} \\ &\leq \lim_{T\to\infty} \sqrt{\frac{|D_i|L_i^2}{T^2} + \frac{|D_i|\delta_T^2}{T}} \\ &= \lim_{T\to\infty} \sqrt{\frac{|D_i|L_i^2}{T^2} + \frac{|D_i|(\sqrt{T})^2}{T}\left(\frac{\delta_T}{\sqrt{T}}\right)^2}.\end{aligned}$$

Since $\lim_{T\to\infty} \delta_T/\sqrt{T} = 0$, we have

$$\begin{aligned}\lim_{T\to\infty} \frac{\bar{R}_i^T}{T} &\leq \lim_{T\to\infty} \sqrt{\frac{|D_i|L_i^2}{T^2} + |D_i|\left(\frac{\delta_T}{\sqrt{T}}\right)^2} \\ &= 0.\end{aligned}$$

Therefore, R_i^T grows sublinearly and according to Lemma 4.2 R_{1-opt}^T also grows sublinearly, which concludes the theorem. □

4.4 Scaling Up to Large Problems

In this section, we first address the memory challenge by using deep neural networks to approximate high-dimensional regret tables. Then we present a prioritized training scheme to speed up the proposed algorithm by reducing the number of non-concurrent training processes in each round.

4.4.1 Neural-based Sampling Scheme

A prominent issue of the sampling algorithm proposed in Section 4.3 is the high memory consumption incurred by exactly storing regret values for each context. A straightforward way would be approximating the regret tables by using linear regression. However, due to their limited capacity, linear models may not be able to capture the complex patterns when the problem is large. Alternatively, Regression Regret-Matching (RRM) [161, 162] tries to address the issue by using regression trees to estimate the accumulated regret, but it still needs to preserve all regret values during the solving process, which eliminates the most attractive advantage of reducing memory consumption. Also, since a regression tree cannot be trained incrementally, RRM needs to re-train the model on *all* data after updating regret values, which could be extremely inefficient.

In contrast, deep neural networks have been demonstrated to be a powerful function approximator and lead to great successes in a wide variety of domains in AI. Therefore, we aim to address the scalability challenge by parameterizing the regret tables via neural networks. In our neural-based sampling scheme, for each non-leaf variable x_i we maintain an estimator $V_i : \Pi_{x_j \in Sep(x_i) \cup \{x_i\}} D_j \to \mathbb{R}$ and a FIFO capacitied memory M_i to estimate regret and store cached regret, respectively. The input of V_i is the concatenation of the one-hot encoding of the assignment to $Sep(x_i)$ and x_i and the output is the estimated regret.

When receiving a context from its parent, a non-leaf variable x_i first retrieves the estimated regret $\hat{R}_i^{t-1}$ related to the context from either memory M_i or estimator V_i, depending on whether the context-assignment pair $\langle X_i^t, d_i \rangle$ presents in the memory. Then x_i computes the mixed strategy π_i^t according to $\hat{R}_i^{t-1}$ and perform sampling.

In the back-propagation phase, after collecting all the assignments of its direct descendants, x_i trains neural network V_{i,θ_i} to minimize the Mean Squared Error (MSE) between the estimated regret and cached regret for h steps. For each step, x_i samples a mini-batch $B \subseteq M_i$ of regret values and updates the parameters θ_i to minimize the MSE:

$$\mathcal{L}_i(\theta_i) = \frac{1}{|B|} \sum\nolimits_{\langle X_i, d_i \rangle \in B} \left(M_i(X_i, d_i) - V_{i,\theta_i}(X_i, d_i) \right)^2, \tag{4.7}$$

where $|B|$ is the size of the batch, $M_i(X_i, d_i)$ and $V_i(X_i, d_i)$ are the cached regret value and the estimated regret value for the context-assignment pair $\langle X_i^t, d_i \rangle$, respectively. Then x_i updates the regret according to Eq. (4.5). Particularly, a bootstrapping step is performed if context-assignment pair $\langle X_i^t, d_i \rangle$ does not present in the memory.

Next, we will show the regret bound of our neural-based sampling scheme. We begin with showing the gap between the rounded regret and the Q-regret in RM$^+$ [165] is no greater than δ_t. Then we show the regret bound by proving the upper bound of Q-regret.

Lemma 4.5. *For variable x_i, $\bar{R}_i^t(X_i, d_i) - Q_i^t(X_i, d_i) \leq \delta_t$ for all X_i, d_i, t, where $Q_i^t(X_i, d_i) = (Q_i^{t-1}(X_i, d_i) + \mathbb{I}(X_i = X_i^t)(s_i^t - S_i^t(d_i))^+$ and $Q_i^0(X_i, d_i) = 0$.*

Proof. The lemma is trivial when $t = 1$ and $X_i \neq X_i^1$, as $\bar{R}_i^1(X_i, d_i) = Q_i^1(X_i, d_i) = 0$. Consider the case that $t = 1$ and $X_i = X_i^1$

$$\bar{R}_i^1(X_i, d_i) - Q_i^1(X_i, d_i) = \max(\delta_1, s_i^1 - S_i^1(d_i)) - \max(0, s_i^1 - S_i^1(d_i)) \leq \delta_1.$$

Thus, the lemma holds for the basis. Assume the lemma holds for the $t-1$-th round, we are going to show the lemma holds for the t-th round as well.

If $X_i \neq X_i^t$, then $\bar{R}_i^t(X_i, d_i) - Q_i^t(X_i, d_i) = \bar{R}_i^{t-1}(X_i, d_i) - Q_i^{t-1}(X_i, d_i) \leq \delta_{t-1} \leq \delta_t$. Otherwise,

$$\bar{R}_i^t(X_i, d_i) - Q_i^t(X_i, d_i) = \max(\delta_t, \bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)) - \max(0, Q_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)).$$

If $s_i^t - S_i^t(d_i) \leq -Q_i^{t-1}(X_i, d_i)$, then

$$\begin{aligned} \bar{R}_i^t(X_i, d_i) - Q_i^t(X_i, d_i) &= \max(\delta_t, \bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)) - 0 \\ &\leq \max(\delta_t, \delta_{t-1} + Q_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)) \\ &\leq \max(\delta_t, \delta_{t-1}) = \delta_t. \end{aligned}$$

If $s_i^t - S_i^t(d_i) > -Q_i^{t-1}(X_i, d_i)$ and $\delta_t \leq \bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)$, then

$$\begin{aligned}\bar{R}_i^t(X_i, d_i) - Q_i^t(X_i, d_i) &= \bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i) - \left(Q_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)\right) \\ &= \bar{R}_i^{t-1}(X_i, d_i) - Q_i^{t-1}(X_i, d_i) \leq \delta_{t-1} \leq \delta_t.\end{aligned}$$

If $s_i^t - S_i^t(d_i) > -Q_i^{t-1}(X_i, d_i)$ and $\delta_t > \bar{R}_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)$, then

$$\bar{R}_i^t(X_i, d_i) - Q_i^t(X_i, d_i) = \delta_t - \left(Q_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i)\right).$$

Since $Q_i^{t-1}(X_i, d_i) + s_i^t - S_i^t(d_i) > 0$, $\bar{R}_i^t(X_i, d_i) - Q_i^t(X_i, d_i) < \delta_t$. Therefore, the lemma holds for all X_i, d_i, t. □

Theorem 4.6. *For variable x_i and round T, the regret $R^T(X_i, d_i)$ is no higher than*

$$\sqrt{L_i|D_i|\left((4\sqrt{|D_i|}\delta_T + L_i)T + 4\sum_{t=1}^{T}\sum_{d_i'}\epsilon_i^t(X_i, d')^2\right)}$$

for all X_i, d_i, where $\epsilon_i^t(X_i, d_i') = \hat{R}_i^t(X_i, d_i') - \bar{R}_i^t(X_i, d_i')$.

Proof. According to Morrill's [161] Theorem 3.0.10 and Theorem 3.0.12, the bound of Q-regrets if we use $\hat{R}_i^t(X_i, \cdot)$ to generate strategy in each round t is

$$\sqrt{L_i\sqrt{|D_i|}\left(L_iT\sqrt{|D_i|} + 4\sum_{t=1}^{T}\sum_{d_i'}|Q_i^t(X_i, d_i') - \hat{R}_i^t(X_i, d_i')|\right)}.$$

Note that

$$\begin{aligned}\sum_{t=1}^{T}\sum_{d_i'}|Q_i^t(X_i, d_i') - \hat{R}_i^t(X_i, d_i')| &= \sum_{t=1}^{T}\sum_{d_i'}|\hat{R}_i^t(X_i, d_i') - \bar{R}_i^t(X_i, d_i') + \bar{R}_i^t(X_i, d_i') - Q_i^t(X_i, d_i')| \\ &\leq \sum_{t=1}^{T}\sum_{d_i'}|\hat{R}_i^t(X_i, d_i') - \bar{R}_i^t(X_i, d_i')| + |\bar{R}_i^t(X_i, d_i') - Q_i^t(X_i, d_i')| \\ &\leq \sum_{t=1}^{T}\sum_{d_i'}|\hat{R}_i^t(X_i, d_i') - \bar{R}_i^t(X_i, d_i')| + \delta_t \\ &= \sum_{t=1}^{T}|D_i|\delta_t + \sum_{d_i'}|\hat{R}_i^t(X_i, d_i') - \bar{R}_i^t(X_i, d_i')| \\ &\leq T|D_i|\delta_T + \sum_{t=1}^{T}\sum_{d_i'}|\hat{R}_i^t(X_i, d_i') - \bar{R}_i^t(X_i, d_i')|.\end{aligned}$$

Since $\sum_{d'_i} |\hat{R}_i^t(X_i, d'_i) - \bar{R}_i^t(X_i, d'_i)| \leq \sqrt{|D_i|} \sum_{d'_i} \sqrt{(\hat{R}_i^t(X_i, d'_i) - \bar{R}_i^t(X_i, d'_i))^2}$ and $R_i^t(X_i, d_i) < Q_i^t(X_i, d_i)$ (Lemma 1 of [165]), the regret bound of neural-based sampling scheme is

$$\sqrt{L_i|D_i| \left((4\sqrt{|D_i|}\delta_T + L_i)T + 4 \sum_{t=1}^{T} \sum_{d'_i} (\hat{R}_i^t(X_i, d'_i) - \bar{R}_i^t(X_i, d'_i))^2 \right)}.$$

□

4.4.2 Prioritized Training Scheme

One potential issue of the proposed neural-based sampling scheme could be the high latency incurred by non-concurrent training processes in the back-propagation phase of each round. More specifically, agents in a chain structure sequentially train their neural network, which is not desirable when there are long chains in a pseudo tree. Unfortunately, since a pseudo tree is generated by depth-first traversal, it usually has few branches when the problem is large, which makes the training quite time-consuming.

Therefore, we propose a prioritized training scheme to reduce the training latency by selecting several *key agents* in each chain structure to perform training procedure in each round. We confine our training scheme to chain structures because agents in different branches (1) can perform in parallel in real-world scenarios and (2) need additional efforts to coordinate training process.

In more detail, in preprocessing phase, we first cluster the agents into different groups according to their position in the pseudo tree and the similarity of dimensions of their regret table. That is, starting from an empty group and the first agent of the chain, we extend the group by adding the current agent until (1) the agent has more than one child, or (2) there is an agent whose regret table dimensions differ by b variables from the union of the ones of remaining agents in the group, where b is a user-specified difference budget to control the size of each group. Then a new group is created and this procedure repeats until each agent belongs to a group.

Algorithm 5: Neural-based sampling scheme with prioritized training for variable x_i

Require: number of training steps h, rounding term δ_t and discounted factor γ

When `Initialization`**:**

call Alg. 6 to determine the group G_i variable x_i belongs to

if *x_i is not a leaf* **then**

initialize neural network V_i such that $V_i(X_i, d_i) = 0, \forall X_i, d_i$

$M_i \leftarrow \emptyset, e_i^0 \leftarrow 0, \bar{e}_i \leftarrow 0, \tilde{e}_i \leftarrow 0$

if *x_i is the root* **then** $t \leftarrow 1, X_i^t \leftarrow \emptyset$, `Sample()`

When *received CONTEXT($\{X^t, G, \bar{e}\}$) from $P(x_i)$*:

if $G_i = G$ **then** $\bar{e}_i \leftarrow \bar{e}$

$X_i^t \leftarrow X^t[Sep(x_i)]$, `Sample()`

When *received BACKPROP($\{x_k = d_k^t\}$) from $x_k \in AC(x_i)$*:

$Y_i^t \leftarrow Y_i^t \cup \{x_k = d_k^t\}$

if *x_i has received all BACKPROP from $AC(x_i)$* **then**

compute hindsight local costs S_i^t and the expected local cost s_i^t

foreach $d_i \in D_i$ **do**

if $\langle X_i^t, d_i \rangle \notin M_i$ **then** $M_i(X_i^t, d_i) \leftarrow V_i(X_i^t, d_i)$

$M_i(X_i^t, d_i) \leftarrow \max\{\delta_t, M_i(X_i^t, d_i) + s_i^t - S_i^t(d_i)\}$

$e_i^t \leftarrow \gamma e_i^{t-1} + (1-\gamma) \sum_{d_i \in D_i} |M_i(X_i^t, d_i) - V_i(X_i^t, d_i)|$

if *x_i is not the root* **then** send BACKPROP($\{x_i = \arg\min_{d_i} S_i^t(d_i)\}$) to $\forall x_j \in AP(x_i)$

When *received ACC_ERROR($\tilde{e}$) from $x_k \in C(x_i)$*:

if $|C(x_i)| > 1$ **then** $\tilde{e}_i \leftarrow e_i^t$, block until all ACC_ERROR messages from $C(x_i)$ arrive

else $\tilde{e}_i \leftarrow \tilde{e} + e_i^t$

if *random()* $< e_i^{t-1} / \bar{e}_i$ **then**

for *training step* $k = 1, \ldots, h$ **do**

uniformly sample a batch B from M_i and train V_i to minimize $\mathcal{L}_i = \frac{1}{|B|} \sum_{\langle X_i, d_i \rangle \in B} (M_i(X_i, d_i) - V_i(X_i, d_i))^2$

if *x_i is the first variable in G_i* **then**

$\bar{e}_i \leftarrow \tilde{e}_i$

if *x_i is not the root* **then** send ACC_ERROR(0) to $P(x_i)$

else $t \xleftarrow{+} 1$, `Sample()`

else send ACC_ERROR($\tilde{e}_i$) to $P(x_i)$

Function `Sample()`**:**

if *x_i is not a leaf* **then**

$\tilde{e}_i \leftarrow 0, Y_i^t \leftarrow \emptyset, \hat{R}_i^{t-1} \leftarrow []$

foreach $d_i \in D_i$ **do**

if $\langle X_i^t, d_i \rangle \in M_i$ **then** $\hat{R}_i^{t-1}(d_i) \leftarrow M_i(X_i^t, d_i)$

else $\hat{R}_i^{t-1}(d_i) \leftarrow V_i(X_i^t, d_i)$

compute the mixed strategy π_i^t according to $\hat{R}_i^{t-1}$ and perform sampling $d_i^t \sim \pi_i^t$

send CONTEXT($X_i^t \cup \{x_i = d_i^t, G_i, \bar{e}_i\}$) to $\forall x_k \in C(x_i)$

else

send BACKPROP($\{x_i = \arg\min_{d_i} \sum_{x_j \in AP(x_i)} f_{ij}(d_i, X_i^t(x_j))\}$) to $\forall x_j \in AP(x_i)$

send ACC_ERROR(0) to $P(x_i)$

In back-propagation phase, we schedule the training processes of each group G according to the prediction error of each agent. That is, each agent i maintains a discounted error e_i^t. Each time agent i performs back-propagation, it performs training with the probability $e_i^t / \sum_{j \in G} e_j^t$. The discounted error is updated by the absolute error between the predicted regret and the cached regret under current context with discount factor γ. That is,

$$e_i^t = \gamma e_i^{t-1} + (1-\gamma) \sum\nolimits_{d_i \in D_i} |M_i(X_i^t, d_i) - V_{i,\theta_i}(X_i^t, d_i)|.$$

Alg. 5 presents the sketch of our proposed neural-based sampling scheme with prioritized training. The algorithm begins with all non-leaf agents creating both neural networks and memory (line 2-4) and the root agent starts the message-driven process by sampling and propagating context (line 5). When perform sampling, a non-leaf agent retrieves the estimated regret values either from the memory or bootstrapping from the neural network, depending on whether the current context-assignment pair presents in the memory (line 27-30). Then it samples an assignment according to the derived mixed strategy and propagates the augmented context to its children (line 30-31). Leaf agents, on the other hand, do not have any subproblem and just communicates the best response via BACKPROP messages (line 33-34).

When received all BACKPROP messages from its children and pseudo children, an agent computes the hindsight local costs and updates the regret values (line 9-13). A bootstrap happens if the regret value corresponding to an assignment under current context does not appear in the memory (line 12). Finally, the agent communicates its best response to its all parents if it is not the root (line 15).

To enforce prioritized training scheme in a fully decentralized manner, each agent i calls a grouping procedure (detailed in Alg. 6) to determine the group it belongs to, and maintains current discounted error e_i^t, partial error sum $\tilde{e}_i$ and error sum $\bar{e}_i$ (line 1, 4). The error sum of a group is communicated via CONTEXT messages from the first agent in the group to the last agent in the group (line 32, 6), which is accumulated progressively via ACC_ERROR messages. In more detail, when receiving a ACC_ERROR message from a child, agent i computes $\tilde{e}_i$ by adding the partial error sum from child to its own discounted error (line 17). Then the agent continues to propagate the partial error sum if it is not the first agent of the group

(line 25). Otherwise, it assigns the error sum of the group and perform the next round of sampling if it is the root (line 22-24). It is worth noting that since our prioritized training scheme is confined to chains, an agent with more than one child must be the last one in a group. Therefore, it does not need to consider the partial error sum from its children (line 16). Finally, an agent trains its neural network with a probability proportional to its prediction error (line 18-20, 14).

Algorithm 6: Grouping procedure for variable x_i

Require: difference budget b

When `Initialization`**:**

1 **if** x_i *is the root* **then**

2 $G_i \leftarrow$ create a new group ID, mark x_i as the first variable in G_i

3 `SendGrpInfo`$(\{G_i, \{\{x_i\}\}\})$

When *received* $GROUP(\{G, DIM\})$ *from* $P(x_i)$**:**

4 **if** $G = NIL$ **then**

5 $G_i \leftarrow$ create a new group ID

6 mark x_i as the first variable in G_i

7 `SendGrpInfo`$(\{G_i, \{Sep(x_i) \cup \{x_i\}\}\})$

8 **else**

9 $DIM_i \leftarrow \{x_i\} \cup Sep(x_i), DIM' \leftarrow DIM \cup \{DIM_i\}, DIM^* \leftarrow \bigcup_{DIM_k \in DIM'} DIM_k$

10 **if** $\exists DIM_k \in DIM'$ *s.t.* $|\{x_j | x_j \in DIM_k \wedge x_j \notin DIM^*\}| > b$ **then**

11 $G_i \leftarrow$ create a new group ID, mark x_i as the first variable in G_i

12 `SendGrpInfo`$(\{G_i, \{Sep(x_i) \cup \{x_i\}\})$

13 **else**

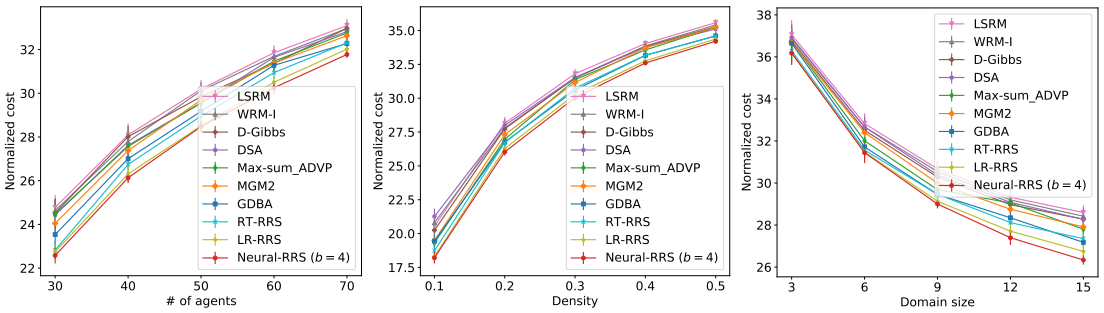
14 $G_i \leftarrow G$, `SendGrpInfo`$(\{G_i, DIM'\})$

15 **Function** `SendGrpInfo`$(\{G, DIM\})$**:**

16 **if** $|C(x_i)| > 1$ **then** send GROUP$(\{NIL, \emptyset\})$ to $\forall x_k \in C(x_i)$.

17 **else if** $|C(x_i)| = 1$ **then** send GROUP$(\{G, DIM\})$ to $x_k \in C(x_i)$.

Alg. 6 presents the sketch of the grouping procedure. The procedure begins with the root agent creating group ID and propagating the related information to its children (line 1-3). That is, if it has more than one child, then the group cannot be extended and a NIL group is communicated to its children (line 16). Otherwise it propagates the group ID and the running regret table dimensions (line 17). When an agent received a GROUP message, it creates a new group if (1) the received group ID is NIL (line 4), or (2) the difference budget is violated (line 10). The procedure ends if every non-root agent receives a GROUP message.



(a) $p_1 = 0.25$, $|D_i| = 10$ (b) 50 agents, $|D_i| = 10$ (c) 50 agents, $p_1 = 0.25$

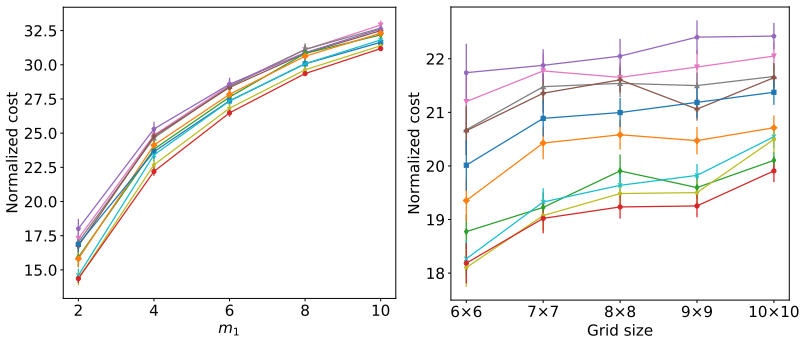
FIGURE 4.3: Performance comparison on random DCOPs

4.5 Empirical Evaluations

We empirically evaluate our proposed neural-based sampling scheme (Neural-RRS) on standard benchmark problems including random DCOPs, scale-free network problems and sensor network problems. We set $\delta_t = t^{0.45}$ and consider each estimator as a neural network with two hidden layers. Each hidden layer has 16 neurons and uses `relu` as the activation function. Each time the neural networks are trained by 2 steps of mini-batch stochastic gradient descent (SGD) with a batch size of 32. We use Adam optimizer [166] with a learning rate of 2×10^{-3} to update parameters. Finally, we set difference budget $b = 4, \gamma = 0.9$ and the capacity of the memory to 5000 regret values.

The baselines we consider include DSA-C [29], GDBA [32] as representative local search algorithms, MGM-2 [28] as a representative k-OPT algorithm, D-Gibbs [37] as a representative sampling algorithm and Max-sum_ADVP [9] as a representative belief propagation algorithm. Besides, we also include regret-based local search schemes [62] (LSRM and WRM-I) since they are context-free regret-matching algorithms for DCOPs. Finally, we also consider the variants LR-RRS and RT-RRS that use linear regression and regression tree to approximate the regret values, respectively.

We set $p = 0.8$ for DSA-C and use $\langle M, NM, T \rangle$ variant for GDBA according to [32]. Finally, all algorithms terminate after 5000 rounds and we report the anytime normalized cost (i.e., the best solution cost divided by the number of constraints) as the result. All experiments are conducted on an i7 octa-core workstation with 32 GB memory. For each experiment, we average the results over 50 random instances.



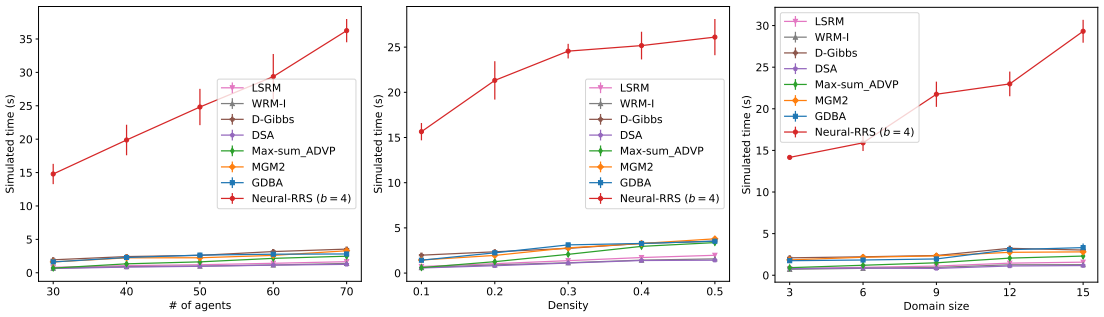
(a) Scale-free net problems (b) Sensor net problems

FIGURE 4.4: Performance comparison on structured problems

4.5.1 Performance Comparison

Results on random DCOPs. In a random DCOP, two agents randomly establish a constraint with a probability p_1, resulting a constraint graph with density p_1. For each constraint, we uniformly randomly select costs from $[0, 100]$. We vary the number of agents, density and domain size respectively and present the results in Figure 4.3. Regret-based local search algorithms explore low-quality convergences and are inferior to DSA. Similarly, D-Gibbs converges to local optima quickly and performs just like stochastic search. On the other hand, the merits of our proposed context-based regret-matching scheme are confirmed by the fact LR-RRS and Neural-RRS significantly outperform their context-free counterparts as well as MGM2 and GDBA. RT-RRS, however, fails to strictly dominate GDBA when solving the problems with large domain size. That is because regression tree cannot be trained incrementally. As a consequence, each agent in RT-RRS has to fit its model on *all* cached regret values, which is extremely inefficient and does not scale up well. In fact, RT-RRS can only finish about 500 rounds of sampling and ends up with poor solutions.

Results on scale-free network problems. In the experiment, we use Barabási-Albert model [167] to generate scale-free networks. We consider the problems with $m_0 = 10$ initially connected vertices (i.e., variables). In each iteration, a new vertex is connected to m_1 vertices with a probability that is proportional to the degree of each existing vertex. We set the number of vertices to 50 and uniformly select costs from [0,100]. Figure 4.4(a) presents the results when varying m_1. D-Gibbs performs poorly due to the inability of exploiting different contexts and gets trapped in local optima quickly. GDBA and MGM2 perform better by using



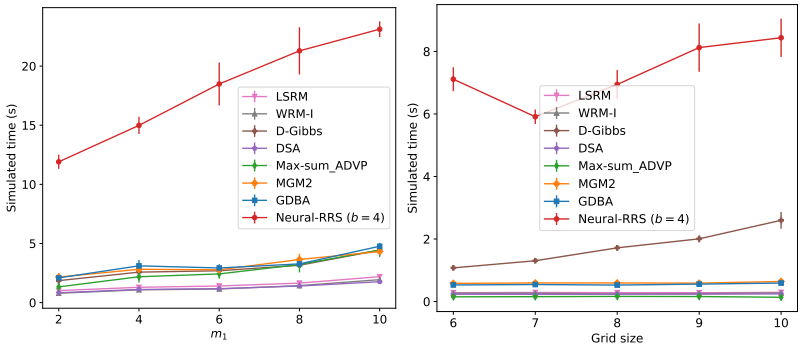
(a) $p_1 = 0.25$, $|D_i| = 10$ (b) 50 agents, $|D_i| = 10$ (c) 50 agents, $p_1 = 0.25$

FIGURE 4.5: Simulated runtime results on random DCOPs

generalized breakout mechanism and coordinated moves between two agents to escape poor convergence, respectively. On the other hand, our Neural-RRS finds significantly better solutions than all the competitors on different m_1. In fact, the average improvement of Neural-RRS over GDBA is 5.6%. That is due to the fact that our algorithm stores regret for each context, which finds significantly better solutions by allowing an agent to devise different strategies for different contexts.

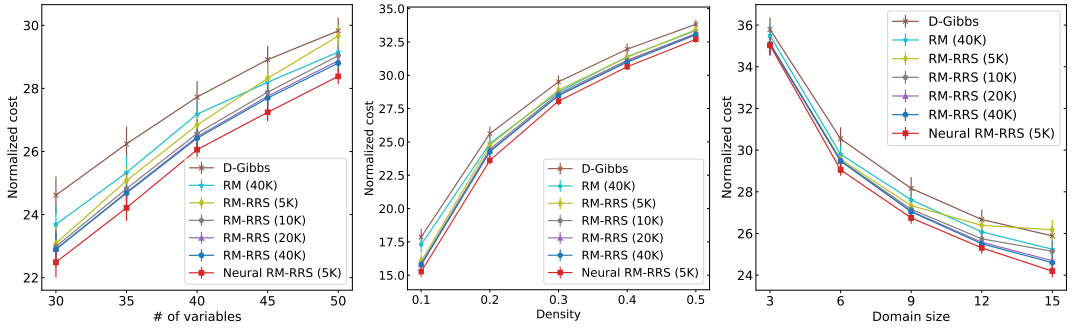
Results on sensor network problems. In a sensor network problem [37, 168], sensors are placed in a 2D grid and each sensor can move along its four cardinal directions or stay stationary (i.e., each sensor has 5 possible actions). Besides, sensors are constrained with their neighboring sensors and the costs are uniformly selected from [0,100]. We vary the grid size from 6×6 (i.e., 36 variables) to 10×10 (i.e., 100 variables) and present the results in Figure 4.4(b). Interestingly, Max-sum_ADVP is quite competitive and finds the solutions with quality much better than the ones found by GDBA and MGM2 in this set of experiments. This might be due to the high-structured topology of sensor networks. Besides, it is worth noting that the performance of LR-RRS is similar to Neural-RRS when solving small problems (i.e., the ones with grid size of 6×6), but the performance substantially degenerates w.r.t. growing grid size. That might be due to the fact that the limited representational capability of linear models cannot capture the complex patterns in large problems. In contrast, Neural-RRS leverages powerful deep neural networks to represent regret tables, leading to the best performance on all configurations.

Runtime results. We compare the simulated runtime [154] of each algorithm and present the results when solving random problems and structured problems in Figure 4.5 and Figure 4.6, respectively. It can be seen that all traditional



(a) Scale-free net problems (b) Sensor net problems

FIGURE 4.6: Simulated runtime results on structured problems



(a) $p_1 = 0.25$, $|D_i| = 10$ (b) 40 agents, $|D_i| = 10$ (c) 40 agents, $p_1 = 0.25$

FIGURE 4.7: Ablation study on random DCOPs

methods terminate very quickly. That is no surprise since they essentially perform table lookup, leading to lower overall runtime. On the other hand, our neural version of RRS require higher runtime as agents need additional efforts to train neural networks during back-propagation phase. However, it is still worth noting that combining with prioritized training scheme, the runtime of our Neural-RRS is moderate and acceptable, especially on structrued problems like sensor networks.

4.5.2 Ablation Study

To explicitly demonstrate the necessity of regret rounding scheme and function approximation, we consider the tabular counterpart of Neural RM-RRS (namely RM-RRS) and the one uses vanilla regret-matching (namely RM). We set the memory capacity to 5000 entries of regret value for Neural RM-RRS and vary the memory capacity from 5000 to 40000 for tabular counterparts to simulate different memory budgets in real-world scenarios. Figure 4.7 presents the results on random

DCOPs. Here we omit the results of RM under different memory capacities due to their similarity.

It can be clearly seen that RM is inferior to RM-RRS under the same memory capacity. In fact, given smaller memory capacity (e.g., 10000), RM-RRS can still outperform RM in the most cases. That is because the negative regret values in RM would restrict exploration and eventually lead to poor results. In contrast, RM-RRS triggers effective exploration by actively rounding the negative regret to small positive values, ensuring that all assignments have a non-zero probability.

On the other side, exploration triggered by rounded regret could also lead to a large number of different contexts. Therefore, given limited budget (e.g., 5000), RM-RRS quickly runs out of memory before finding a good solution, leading to poor performance when solving large-scale problems, while Neural RM-RRS with the same memory constraint still achieves the best performance. Therefore, the scalability of RM-RRS is severely restricted by its tabular nature, which is successfully addressed by neural function approximation.

4.6 Chapter Summary

Most of incomplete algorithms for DCOPs are context-free, which usually leads to low-quality convergence. While context-based methods tries to remedy the problem by explicitly storing information for each context, they usually suffer from low sample efficiency and high memory consumption which prohibit them from scaling up to large problems. In this chapter, we tackle the issues by proposing Neural-RRS, the first neural context-based algorithm for DCOPs, built upon regret-matching. The algorithm overcomes the pathology of low sample efficiency by accumulating regret according to the local problem of each variable. To address the scalability challenge incurred by exactly storing regret for each context, Neural-RRS approximates the regret tables by deep neural networks. Besides, we propose a prioritized training scheme to reduce the training latency. Finally, the algorithm uses a regret rounding scheme that rounds small regret values to positive numbers to ensure exploration. We theoretically show the regret bound and the extensive evaluations indicate that our algorithm can scale up to large problems and significantly outperform the state-of-the-art methods.

Chapter 5

Pretrained Cost Model for Distributed Constraint Optimization Problems

5.1 Introduction

As we have discussed in Section 2.2, the existing DCOP algorithms usually rely on handcrafted heuristics which need expertise to tune for different settings and may fail to explore the solution space efficiently. In contrast, Deep Learning-based (DL-based) techniques learn effective heuristics for existing methods automatically [49, 54, 169], achieving state-of-the-art performance in various challenging problems like MIP, CVRPs, and SATs. Unfortunately, these methods are often not generalizable to different search algorithms. Most importantly, many of these methods usually require the full knowledge about the problems to be solved, making them unsuitable for a distributed setting like DCOPs where centralization is not realistic due to geographical limitations or privacy concerns.

Therefore, in this chapter[1] we aim to develop the first general-purpose deep pre-trained model, named GAT-PCM, to generate effective heuristics for a wide range of DCOP algorithms and propose a distributed embedding schema of GAT-PCM for decentralized model inference. Specifically, we make the following key contributions:

[1] This chapter has been published in [170].

- We propose a novel directed tripartite graph representation based on microstructure [63] to encode a partially instantiated DCOP instance and use Graph Attention Networks (GATs) [64] to learn generalizable embeddings.
- Instead of generating heuristics for a particular algorithm, GAT-PCM predicts the optimal cost of a target assignment given a partial assignment, such that it can be applied to boost the performance of a wide range of DCOP algorithms where evaluating the quality of an assignment is critical. To this end, we pretrain our model on a dataset where DCOP instances are sampled from a problem distribution, partial assignments are constructed according to pseudo trees, and cost labels are generated by a complete algorithm.
- We propose a Distributed Embedding Schema (DES) to perform decentralized model inference without disclosing local constraints, where each agent exchanges only the embedded vectors via localized communication. We also theoretically show the correctness and complexity of DES.
- As a case study, we develop two efficient heuristics for DLNS [82] and backtracking search for DCOPs based on GAT-PCM, respectively. Specifically, by greedily constructing a solution, our GAT-PCM can serve as a subroutine of DLNS to repair assignments. Besides, the predicted cost of each assignment is used as a criterion for domain ordering in backtracking search.
- Extensive empirical evaluations indicate that GAT-PCM-boosted algorithms significantly outperform the state-of-the-art methods in various standard benchmarks.

The rest of this chapter is organized as follows. We review closely related works and present preliminaries in Section 5.2. In Section 5.3, we detail our pretrained cost model for DCOPs and develop DES to enable decentralized model inference. Finally, we conduct extensive empirical evaluations in Section 5.4 and conclude in Section 5.5.

5.2 Backgrounds

In this section, we review closely related works and present preliminaries including Graph Attention Networks and pretrained models.

5.2.1 Related Work

There is an increasing interest of applying neural networks to solve SAT problems in recent years. Selsam et al. [56] proposed NeuroSAT, a message passing neural network built upon LSTMs [171] to predict the satisfiability of a SAT and further decode the satisfying assignments. Yolcu and Póczos [139] proposed to use GNNs to encode a SAT and REINFORCE [114] to learn local search heuristics. Similarly, Kurin et al. [141] proposed to learn branching heuristics for a CDCL solver [142] using GNNs and DQN [44]. Beside Boolean formulas, Xu et al. [172] proposed to use CNNs [173] to predict the satisfiability of a general CSP. However, all of these methods require the total knowledge of a problem, making them unsuitable for distributed settings. Differently, *our method uses an efficient distributed embedding schema to cooperatively compute the embeddings without disclosing constraints.*

Very recently, Razeghi et al. [174] proposed to use Multi-layer Perceptrons (MLPs) to parameterize the high-dimensional data in traditional constraint reasoning techniques, e.g., Bucket Elimination [72]. Unfortunately, it follows an online learning strategy, which removes the most attractive feature of generalizing to new instances offered by neural networks. As a result, they require a significantly long runtime in order to train the MLPs. In contrast, *we aim to develop an DL model for DCOPs which is supervisely pretrained with large-scale datasets beforehand.* When applying the model to an instance, we just need several steps of model inference, which substantially reduces the overall overheads.

5.2.2 Graph Attention Network

Graph attention networks (GATs) [64] are constructed by stacking a number of graph attention layers in which nodes are able to attend over their neighborhoods' features via the self-attention mechanism.

Specifically, the attention coefficient between every pair of neighbor nodes is computed as $e_{ij} = a(\mathbf{W}h_i, \mathbf{W}h_j)$, where $h_i, h_j \in \mathbb{R}^d$ are node features, $\mathbf{W} \in \mathbb{R}^{d \times d}$ is a weight matrix, and a is single-layer feed-forward neural network. Then the attention weight α_{ij} for nodes $j \in \mathcal{N}_i$ is computed as $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$, where $\mathcal{N}_i$ is the neighborhood of node v_i in the graph (including v_i). At last, node v_i's feature

h'_i is updated as

$$h'_i = g\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} h_j\right),$$

where g is some nonlinear function such as the sigmoid. Multi-head attention [66] is also used where K independent attention mechanisms are executed and their feature vectors are averaged as

$$h'_i = g\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k h_j\right).$$

5.2.3 Pretrained Model

The idea behind pretrained models is to first pretrain the models using large-scale datasets beforehand, then apply the models in downstream tasks to achieve state-of-the-art results. Beside significantly reducing the training overhead, pretrained models also offer substantial performance improvement over learning from scratch, leading to great successes in natural language processing [175, 176] and computer vision [41, 177, 178].

In this chapter, we aim to develop the first effective and general-purpose deep pretrained model for DCOPs. In particular, we are interested in training a cost model $\mathcal{M}_\theta$ to predict the optimal cost of a partially instantiated DCOP instance, which is a core task in many DCOP algorithms:

$$\mathcal{M}_\theta(P, x_m = d_m; \Gamma) \mapsto \mathbb{R}, \tag{5.1}$$

where $P \equiv \langle I, X, D, F \rangle$ is a DCOP instance[2], Γ is a partial assignment, $x_m = d_m$ is the target assignment, and variable x_m does not appear in Γ. This way, our cost model can be applied to a wide range of DCOP algorithms where evaluating the quality of an assignment is critical.

[2]In this chapter, we assume that all constraint functions are binary, i.e., $|scp(f_i)| = 2, \forall f_i \in F$. Therefore, a constraint function f_k with $scp(f_k) = \{x_i, x_j\}$ is also written as f_{ij}.

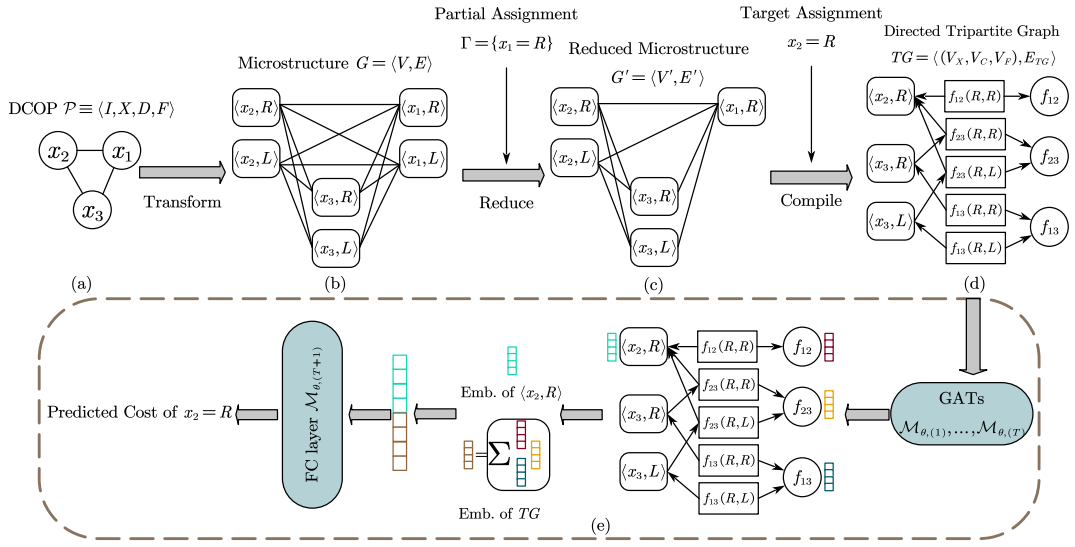


FIGURE 5.1: An illustration of the architecture of GAT-PCM with a small DCOP instance. A DCOP instance in (a) is first transformed into an equivalent microstructure G in (b), and then G is instantiated with a partial assignment Γ by removing some nodes and edges in (c) and then further compiled to a directed tripartite graph with a given target assignment in (d) (cf. Section 5.3.1) in (e). Finally, we use GATs to learn an embedding with supervised training (cf. Section 5.3.2 and Section 5.3.3).

5.3 Pretrained Cost Model for DCOPs

In this section, we elaborate our pretrained cost model GAT-PCM. We begin with illustrating the architecture of the model in Figure 5.1. We then outline the centralized pretraining procedure for the model in Alg. 7 to learn generalizable cost heuristics. We further propose a distributed embedding schema for decentralized model inference in Alg. 8. Finally, we show how to use GAT-PCM to construct effective heuristics to boost DCOP algorithms.

5.3.1 Graph Representations

The architecture of our GAT-PCM is illustrated in Figure 5.1. Recall that we aim to train a model to predict the optimal cost of a partially instantiated DCOP instance (cf. Eq. (5.1)) and thus, we first need to embed a partially instantiated DCOP instance and the key is to build a suitable graph representation for a partially instantiated DCOP instance.

Since DCOPs can be naturally represented by graphs with arbitrary sizes, we resort to GATs to learn generalizable and permutation-invariant representation of DCOPs. To this end, we first transform a DCOP instance $P \equiv \langle I, X, D, F \rangle$ to a microstructure representation [63] where each variable assignment corresponds to a vertex and the constraint cost between a pair of vertices is represented by a weighted edge (cf. Figure 5.1(b)). After that, for each assignment $x_i = d_i$ in the partial assignment Γ, we remove all the other variable-assignment vertices of x_i, except $\langle x_i, d_i \rangle$, and their related edges from the microstructure (cf. Figure 5.1(c)). Then the reduced microstructure represents the partially instantiated DCOP instance w.r.t. Γ.

The reduced microstructure is further compiled into a directed tripartite graph $TG = \langle (V_X, V_C, V_F), E_{TG} \rangle$ which serves as the input of our GAT-PCM model (cf. Figure 5.1(d)). Specifically, for each edge in the microstructure, we insert a constraint-cost node $v_c \in V_C$ which corresponds to the constraint cost between the related pair of variable assignments. For each constraint function $f \in F$, we also create a function node $v_f \in V_F$ in the graph, and each related constraint-cost node will be directed to v_f. Note that v_f can be regarded as the *proxy* of all related constraint-cost nodes. Besides, variable-assignment nodes related to Γ will also be removed from the tripartite graph since they are *subsumed* by their related constraint-cost nodes.

Finally, we note that the loopy nature of undirected microstructure may lead to missense propagation and potentially cause an oversmoothing problem [179]. For example, $\langle x_3, R \rangle$ should be independent of $\langle x_3, L \rangle$ since they are two different assignments of the same variable. However, $\langle x_3, L \rangle$ could indirectly influence $\langle x_3, R \rangle$ through multiple paths (e.g., $\langle x_3, L \rangle - \langle x_2, R \rangle - \langle x_3, R \rangle$) when applying GATs. Therefore, we require the tripartite graph to be directed and acyclic such that each constraint-cost node or variable-assignment node has a path to the target variable-assignment node. Specifically, we determine the directions between constraint-cost nodes and variable-assignment nodes through a two-phase procedure. First, we build a Directed Acyclic Graph (DAG) for a constraint graph induced by the set of unassigned variables such that every unassigned variable has a path to the target variable. To this end, we build a pseudo tree PT [67] with the target variable as its root and use PT as the DAG where each node of PT will be directed to its parent or pseudo parents. Second, for any pair of constrained variables x_i and x_j in the DAG,

where x_i is the precursor of x_j, and any related pair of variable assignments $\langle x_i, d_i \rangle$ and $\langle x_j, d_j \rangle$, we set the node of $\langle x_i, d_i \rangle$ to be the precursor of the constraint-cost node of $f_{ij}(d_i, d_j)$ and set the constraint-cost node of $f_{ij}(d_i, d_j)$ to be the precursor of the node of $\langle x_j, d_j \rangle$ in the tripartite graph. Note that constraint-cost nodes related to a unary function will be set to be the precursor of their corresponding variable-assignment nodes.

For space complexity, given an instance with $|I|$ variables and maximum domain size of $D_{\max} = \max_{x_i \in X} |D_i|$, the directed acyclic tripartite graph has $O(D_{\max}|I|)$ variable-assignment nodes, $O(|I|^2)$ function nodes and $O(D_{\max}^2|I|^2)$ constraint-cost nodes.

5.3.2 Graph Embeddings

Given a directed tripartite graph representation, we use GATs to learn an embedding with supervised training (cf. Figure 5.1(e)). Each node v_i has a four-dimensional initial feature vector $h_i^{(0)} \in H^{(0)}$ where the first three elements denote the one-hot encoding of node types (i.e., variable-assignment nodes, constraint-cost nodes, and function nodes) and the last element is set to be the constraint cost of v_i if it is a constraint-cost node and otherwise, 0. The initial feature matrix $H^{(0)}$ is then embedded through T layers of the GAT. Formally,

$$H^{(t)} = \mathcal{M}_{\theta,(t)}(H^{(t-1)}), \quad t = 1, \ldots, T, \tag{5.2}$$

where $H^{(t)}$ is the embedding in the t-th timestep and $\mathcal{M}_{\theta,(t)}$ is the t-th layer of the GAT. Finally, given a target variable-assignment $x_m = d_m$ and a partial assignment Γ, we predict the optimal cost of the partially instantiated problem instance induced by $\Gamma \cup \{x_m = d_m\}$ based on the embedding of the node of $v_m = \langle x_m, d_m \rangle \in V_X$ and the accumulated embedding of all function nodes of the tripartite graph as follows:

$$\hat{c}_m = \mathcal{M}_{\theta,(T+1)}(h_m^{(T)} \oplus \sum_{v_i \in V_F} h_i^{(T)}), \tag{5.3}$$

where $\mathcal{M}_{\theta,(T+1)}$ is a fully-connected layer and $\oplus$ is the concatenation operation.

Note that, by our construction of the tripartite graph, function nodes are the proxies of all constraint-cost nodes and all the other variable-assignment nodes

have been directed to the target variable-assignment node. Therefore, we do not need to include the embeddings of constraint-cost nodes and variable-assignment nodes except the target variable-assignment node in Eq. (5.3).

Algorithm 7: Offline pretraining procedure

Input: number of training epochs N, number of training iterations K, problem distribution $\mathcal{P}$, optimal DCOP algorithm $\mathcal{A}$, capacitated FIFO buffer $\mathcal{B}$

1 **for** $n = 1, \dots, N$ **do**
  Phase I: generating labelled data
2 $P \equiv \langle I, X, D, F \rangle \sim \mathcal{P}$, $PT \leftarrow$ build a pseudo tree for P
3 **forall** $x_i \in X$ **do**
4 $Sep(x_i) \leftarrow$ ancestors connecting x_i and its descents in PT
5 **forall** *context* $\Gamma_i \in \Pi_{x_j \in Sep(x_i)} D_j$ **do**
6 **forall** $d_i \in D_i$ **do**
7 $P' \leftarrow \textsc{Reduce}(P, \Gamma_i, x_i = d_i)$
8 $c^* \leftarrow \mathcal{A}(P')$, $\mathcal{B} \leftarrow \mathcal{B} \cup \{\langle P, \Gamma_i, x_i = d_i, c^* \rangle\}$
  Phase II: training the model
9 **for** $k = 1, \dots, K$ **do**
10 $B \leftarrow$ sample a batch of data from $\mathcal{B}$
11 train the model $\mathcal{M}_\theta$ to minimize Eq. (5.4)
12 **return** $\mathcal{M}_\theta$

5.3.3 Pretraining

Alg. 7 sketches the training procedure. For each epoch, we first generate labelled data (i.e., partial assignments, target assignments and corresponding optimal costs) in phase I and then train our model in phase II.

Specifically, we first sample a DCOP instance P from the problem distribution $\mathcal{P}$. For each target variable x_i, instead of randomly generating partial assignments, we build a pseudo tree PT and use its contexts w.r.t. PT as partial assignments (line 2-5). In this way, we avoid redundant partial assignments by focusing only on the variables that are constrained with x_i or its descendants. After conditioning on $\Gamma_i \cup \{x_i = d_i\}$ and obtaining the subproblem rooted at x_i (cf. procedure Reduce), we apply any off-the-shelf optimal DCOP algorithm $\mathcal{A}$ to solve P' to get the optimal cost c^* (line 6-8).

Each tuple of partial assignment, target assignment, optimal cost and problem instance will be stored in a capacitated FIFO buffer $\mathcal{B}$. In phase II, we uniformly

sample a batch B of data from the buffer to train our model using the mean squared error loss:

$$\mathcal{L}(\theta) = \frac{1}{|B|} \sum_{\langle P,\Gamma,x_i=d_i,c^*\rangle \in B} (\mathcal{M}_\theta(P, x_i = d_i; \Gamma) - c^*)^2. \tag{5.4}$$

5.3.4 Distributed Embedding Schema

Different from pretraining stage where the model has access to all the knowledge (e.g., variables, domains, constraints, etc.) about the instance to be solved, an agent in real-world scenarios usually can only be aware of its local problem due to privacy concern and/or geographical limitation, posing a significant challenge when applying our model to solve DCOPs. Also, centralized model inference could overwhelm a single agent. Therefore, we aim to develop a distributed schema for model inference in which each agent only uses its local knowledge to cooperatively compute Eq. (5.2) and Eq. (5.3).

We exploit the directed and acyclic nature of our tripartite graph and propose an efficient Distributed Embedding Schema (DES) in Alg. 8. The general idea is that each agent maintains the embeddings w.r.t. its local problem. Specifically, an agent i maintains the following components: (1) its own variable-assignment nodes and (induced) unary constraint-cost and function nodes; (2) all function nodes f_{ij} where x_j is a successor of x_i; and (3) all constraint-cost nodes $f_{ij}(d_i, d_j)$ where x_j is a successor of x_i. Each time the agent updates the local embeddings via a single step of model inference after receiving the embeddings from its precursors. Taking the tripartite graph in Figure 5.1(d) as an example, x_2 maintains embeddings for $\langle x_2, R\rangle$, $f_{12}(R, R)$, f_{12}, $f_{23}(R, R)$ and $f_{23}(R, L)$. To update its local embeddings for $\langle x_2, R\rangle$, $f_{12}(R, R)$ and f_{12}, x_2 only needs one step of model inference after receiving the latest embedding of constraint-cost nodes $f_{23}(R, R)$ and $f_{23}(R, L)$ from its precursor x_3.

Next, we give details about the schema. First, we use primitive STACK to concatenate the initial features of local nodes[3] to construct the initial embeddings $H_i^{(0)}$ (line 3-12). After that, agent i computes its first round embeddings and sends the updated embeddings of constraint-cost nodes to each of its successors (line 13-14). If agent i is a source node, i.e., $P_i = \emptyset$, it directly updates the subsequent

[3]We omit unary functions for simplicity.

Algorithm 8: Distributed embedding schema for agent i

Input: trained model $\mathcal{M}_\theta$, precursors P_i, successors S_i, target assignment $x_m = d_m$, initial variable-assignment node feature $h_X^{(0)}$, initial function node feature $h_F^{(0)}$, one-hot encoding for constraint-cost node $h_C^{(0)}$

When `Initialization`**:**

$Cache_i \leftarrow []$, $H_i^{(0)} \leftarrow$ empty tensor
for $t = 1, \dots, T$ **do** $Cache_i[t] \leftarrow$ empty map
if $x_i = x_m$ **then** $H_i^{(0)} \leftarrow \textsc{Stack}(H_i^{(0)}, h_X^{(0)})$
else
  forall $d_i \in D_i$ **do** $H_i^{(0)} \leftarrow \textsc{Stack}(H_i^{(0)}, h_X^{(0)})$
forall $j \in S_i$ **do**
  $H_i^{(0)} \leftarrow \textsc{Stack}(H_i^{(0)}, h_F^{(0)})$
  forall *cost value* $c_{ij} \in f_{ij}(\cdot, \cdot)$ **do**
    $H_i^{(0)} \leftarrow \textsc{Stack}(H_i^{(0)}, h_C^{(0)} \oplus c_{ij})$
forall $j \in P_i$ **do**
  forall *cost value* $c_{ji} \in f_{ji}(\cdot, \cdot)$ **do**
    $H_i^{(0)} \leftarrow \textsc{Stack}(H_i^{(0)}, h_C^{(0)} \oplus c_{ji})$
$\ell_i \leftarrow$ zero vector, $t_i \leftarrow 1$, $H_i^{(t_i)} \leftarrow \mathcal{M}_{\theta,(t_i)}(H_i^{(t_i-1)})$
send $H_i^{(t_i)}[f_{ij}(\cdot, \cdot)]$ to j, $\forall j \in S_i$
if $P_i = \emptyset$ **then**
  for $t_i = 2, \dots, T$ **do**
    $H^{(t_i)} \leftarrow \mathcal{M}_{\theta,(t_i)}(H^{(t_i-1)})$
    if $t_i < T$ **then**
      send $H_i^{(t_i)}[f_{ij}(\cdot, \cdot)]$ to j, $\forall j \in S_i$
  $\ell_i \leftarrow \sum_{j \in S_i} H_i^{(T)}[f_{ij}]$, send ℓ_i to $j' \in S_i$

When *Receive embedding* $\bar{H}^{(t_j)}$ *from* $j \in P_i$**:**

$Cache_i[t_j][j] \leftarrow \bar{H}^{(t_j)}$
if $|Cache_i[t_i]| = |P_i|$ **then**
  forall $j' \in P_i$ **do**
    $H_i^{(t_i)}[f_{ij'}(\cdot, \cdot)] \leftarrow Cache[t_i][j']$
  $t_i \leftarrow t_i + 1$
  $H^{(t_i)} \leftarrow \mathcal{M}_{\theta,(t_i)}(H^{(t_i-1)})$
  if $t_i < T$ **then**
    send $H_i^{(t_i)}[f_{ij'}(\cdot, \cdot)]$ to j', $\forall j' \in S_i$
  else
    $\ell_i \leftarrow \sum_{j'' \in S_i} H_i^{(T)}[f_{ij''}]$, send ℓ_i to $j' \in S_i$

When *Receive accumulated embedding* ℓ_j *from* $j \in P_i$**:**

if $x_i \neq x_m$ **then**
  send ℓ_j to $j' \in S_i$
else
  $\ell_i \leftarrow \textsc{Add}(\ell_i, \ell_j)$
  if *all accumulated embeddings have arrived* **then**
    computes Eq. (5.5)

embeddings and sends the constraint-cost node embeddings at each timestep to its successors (line 15-19) since it does not need to wait for the embeddings from its precursors. Besides, the agent also sends the local accumulated function node embedding ℓ_i to one of its successors (line 20).

After receiving the constraint-cost node embeddings from its precursor j, agent i temporarily stores the embeddings to $Cache_i$ according to the timestamp t_j (line 21). If all the precursors' constraint-cost node embeddings for the t_i-th layer have arrived, agent i updates the local embedding $H_i^{(t_i)}$ with those embeddings stored in $Cache_i[t_i]$ (line 22-24). Then the agent computes the embeddings $H_i^{(t_i+1)}$ and sends the up-to-date embeddings to its successors (line 25-28). If all GAT layers are exhausted, the agent computes the local accumulated function node embedding ℓ_i and sends it to one of its successors (line 29-30). After receiving an accumulated function-node embedding, agent i either directly forwards the embedding to one of its successors or adds to its own accumulated function-node embedding, depending on whether it is the target agent (line 31-34). After received the accumulated embedding messages of all the other agents, the target agent m outputs the predicted optimal cost by

$$\hat{c}_m = \mathcal{M}_{\theta,(T+1)}(H_m^{(T)}[\langle x_m, d_m\rangle] \oplus \ell_m), \tag{5.5}$$

where $H_m^{(T)}[\langle x_m, d_m\rangle]$ is the embedding for variable-assignment node $\langle x_m, d_m\rangle$ in $H_m^{(T)}$ (line 35-36).

We now show the soundness and complexity of DES. We first show that DES results in the same embeddings as its centralized counterpart.

Lemma 5.1. *In DES, each agent i with $P_i \neq \emptyset$ receives exactly $T-1$ constraint-cost node embedding messages from $j, \forall j \in P_i$, one for each timestep $t_j = 1, \ldots, T-1$.*

Proof. Consider the base case where all the precursors are source i.e., $P_j = \emptyset, \forall j \in P_i$. Since it cannot receive a embedding from other agent, each precursor j sends exactly $T-1$ constraint-cost node embeddings to i, one for each timestep $t_j = 1, \ldots, T-1$ according to line 14, 18-19.

Assume that the lemma holds for all $j \in P_i$ with $P_j \neq \emptyset$. By assumption, the condition of line 22 holds for $t_j = 1, \ldots, T-1$ and hence precursor j sends embedding to i for $t_j = 2, \ldots, T-1$ (line 25-28). Together with the embedding sent in line

13, each precursor j sends $T-1$ constraint-cost node embedding messages to i in total, one for each timestep $t_j = 1, \ldots, T-1$, which concludes the lemma. □

Lemma 5.2. *For any agent i and timestep $t = 1, \ldots, T$, after performing DES, its local embeddings are the same as the ones in $H^{(t)}$. I.e., $H_i^{(t)}[\langle x_i, d_i\rangle] = H^{(t)}[\langle x_i, d_i\rangle]$, $H_i^{(t)}[f_{ij}(d_i, d_j)] = H^{(t)}[f_{ij}(d_i, d_j)]$, $H_i^{(t)}[f_{ij}] = H^{(t)}[f_{ij}]$, $\forall d_i \in D_i, x_j \in S_i, d_j \in D_j$.*

Proof. We only show the proof for variable-assignment nodes. Similar argument can be applied to constraint-cost nodes and function nodes.

In the first timestep, i.e., $t = 1$, for each node $\langle x_i, d_i\rangle$, Eq. (5.2) computes $H^{(1)}[\langle x_i, d_i\rangle]$ based on the initial feature $H^{(0)}[f_{ij}(d_i, d_j)]$, $\forall j \in P_i, d_j \in D_j$, which is the same as in DES, i.e., line 10-13.

Assume that the lemma holds for $t > 1$. Before computing the embeddings for $(t+1)$-th timestep, agent i must have received the embedding $H_j^{(t)}[f_{ij}(d_i, d_j)]$, which equals to $H^{(t)}[f_{ij}(d_i, d_j)]$ according to the assumption, from $j, \forall j \in P_i, d_i \in D_i, d_j \in D_j$ (line 19, 28, 21-24, Lemma 5.1). Therefore, agent i computes $H_i^{(t+1)}[\langle x_i, d_i\rangle]$ according to $H^{(t)}[f_{ij}(d_i, d_j)], \forall j \in P_i, d_j \in D_j$, which is equivalent to Eq. (5.2). Consequently, $H_i^{(t+1)}[\langle x_i, d_i\rangle] = H^{(t+1)}[\langle x_i, d_i\rangle]$ and the lemma holds by induction. □

Lemma 5.3. *For target agent m, after performing DES, $\ell_m = \sum_{v_i \in V_F} h_i^{(T)}$.*

Proof. We prove the lemma by showing each agent sends exactly one accumulated embedding message w.r.t. its local function nodes to one of its successors (i.e., line 20 and 30). It is trivial for the agents without precursor since they do not receive any message (line 28) and only send one accumulated embedding message by the end of procedure `Initialization` (line 20).

Consider an agent i with $P_i \neq \emptyset$. According to Lemma 5.1, i executes line 25-30 for $T-1$ times. Given the initial value of 1 (line 13), t_i will eventually equal to T, implying line 30 will be executed only once. Since it does not perform line 20, i sends exactly one accumulated embedding message w.r.t. its local function nodes.

Since by construction each agent in the DAG has a path to the target agent m, all the accumulated embeddings will be forwarded to m (line 31-34). Therefore, by Lemma 5.2,

$$\ell_m = \sum_{i \in I} \sum_{j \in S_i} H_i^{(T)}[f_{ij}] = \sum_{i \in I} \sum_{j \in S_i} H^{(T)}[f_{ij}].$$

Note that $\forall f_{ij} \in F$, it must be either the case $j \in S_i$ if $i \prec j$ or the case $i \in S_j$ if $j \prec i$ in the DAG. Hence,

$$\ell_m = \sum_{i \in I} \sum_{j \in S_i} H^{(T)}[f_{ij}] = \sum_{f_{ij} \in F} H^{(T)}[f_{ij}] = \sum_{v_i \in V_F} h_i^{(T)}.$$

□

Then we show the soundness of our DES as follows:

Proposition 5.1. *DES is sound, i.e., Eq. (5.5) returns the same result as Eq. (5.3).*

Proof. According to the Lemma 5.2 and Lemma 5.3, by the end of DES, the target agent has the same variable-assignment embedding and accumulated function node embedding as the ones computed by Eq. (5.2). Therefore, Eq. (5.5) is equivalent to Eq. (5.3). □

Finally, we show the complexity of our DES as follows:

Proposition 5.2. *Each agent in DES requires T steps of model inference, $O(|I|D_{\max}^2)$ spaces, and communicates $O(T|I|D_{\max}^2)$ information.*

Proof. By line 13, 25-26 and Lemma 5.1, each agent performs T times of model inference. Each agent i needs to maintain embedding for $O(D_{\max})$ assignment-variable nodes (line 3-5), $O(|S_i|D_{\max}^2) + |P_i|D_{\max}^2)$ constraint-cost nodes (line 6, 8-12), and $O(|S_i|)$ function nodes (line 7). Since in the worst case, the agent is constrained with all the other $|I| - 1$ agents, i's space complexity is $O(|I|D_{\max}^2)$. Finally, since for each timestep $t_i = 1, \ldots, T-1$ agent i sends the constraint-cost node embeddings to its successors, its communication overhead is $O(T|S_i|D_{\max}^2) = O(T|I|D_{\max}^2)$. □

5.3.5 GAT-PCM as Heuristics

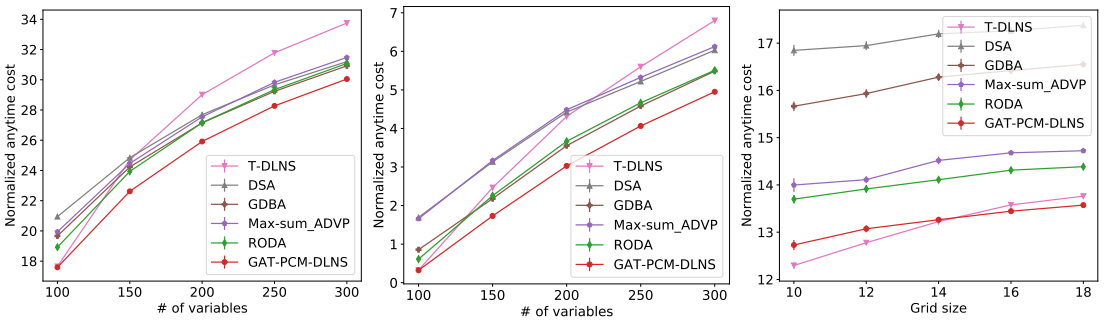
Since our model GAT-PCM predicts the optimal cost of a target assignment given a partial assignment, it can serve as a general heuristic to boost the performance of a wide range of DCOP algorithms where the core operation is to evaluate the quality of an assignment. We consider two kinds of well-known algorithms and show how our model can boost them as follows:

- Local search. A key task in local search is to find good assignments for a set of variables given the other variables' assignments. For example, in Distributed Large Neighborhood Search (DLNS) [82] , each round a subroutine is called to solve a subproblem induced by the destroyed variables (also called repair phase). Currently, DPOP [25] is used to solve a tree-structured relaxation of the subproblem, which ignores a large proportion of constraints and thus leads to poor performance on general problems. Instead, we use our GAT-PCM to solve the subproblem without relaxation (i.e., all constraints between all pairs of destroyed variables are included) since the overhead is polynomial in the number of agents (cf. Proposition 5.2). Specifically, for each connected subproblem, we assume a variable ordering (e.g., lexicographical ordering, pseudo tree). Then we greedily assign each variable according to the costs predicted by GAT-PCM, i.e., we select an assignment with the smallest predicted cost for each variable.

- Backtracking search. Domain ordering is another important task in backtracking search for DCOPs. Previously, domain ordering utilizes local information only, e.g., prioritizing the assignment with minimum conflicts w.r.t. each unassigned variable [180] or querying a lower bound lookup table. On the other hand, our GAT-PCM offers a more general and systematic way for domain ordering. Specifically, for an unassigned variable, we could query GAT-PCM for the optimal cost of each assignment under the current partial assignment and give the priority to the one with minimum predicted cost.

5.4 Empirical Evaluation

In this section, we perform extensive empirical studies. We begin with introducing the details of experiments and pretraining stage. Then we analyze the results and demonstrate the capability of our GAT-PCM to boost DCOP algorithms.



(a) $p_1 = 0.05$, $D_{\max} = 10$ (b) $p_1 = 0.05$, $D_{\max} = 3$ (c) $D_{\max} = 10$

FIGURE 5.2: Solution quality comparisons: (a) random DCOPs; (b) weighted graph coloring problems; (c) grid networks

5.4.1 Benchmarks

We consider four types of benchmarks in our experiments, i.e., random DCOPs, scale-free networks, grid networks, and weighted graph coloring problems. For random DCOPs and weighted graph coloring problems, given density of $p_1 \in (0, 1)$, we randomly create a constraint for a pair of variables with probability p_1. For scale-free networks, we use the BA model [167] with parameter m_0 and m_1 to generate constraint relations: starting from a connected graph with m_0 vertices, a new vertex is connected to m_1 vertices with a probability which is proportional to the degree of each existing vertex in each iteration. Besides, variables in a grid network are arranged into a 2D grid, where each variable is constrained with four neighboring variables excepts the ones located at the boundary. Finally, for each constraint in random DCOPs, scale-free networks and grid networks, we uniformly sample a cost from $[0, 100]$ for each pair of variable-assignments. Differently, constraints of the weighted graph coloring problems incur a cost which is also uniformly sampled from $[0, 100]$ if two constrained variables have the same assignment.

5.4.2 Baselines

We consider four types of baselines: local search, belief propagation, region optimal method, and large neighborhood search. We use DSA [29] with $p = 0.8$ and GDBA [32] with $\langle M, NM, T \rangle$ as two representative local search methods, Max-sum_ADVP [9] as a representative belief propagation method, RODA [83] with $t = 2, k = 3$ as a representative region optimal method, and T-DLNS [82] with destroy probability $p = 0.5$ as a representative large neighborhood search method. All experiments are

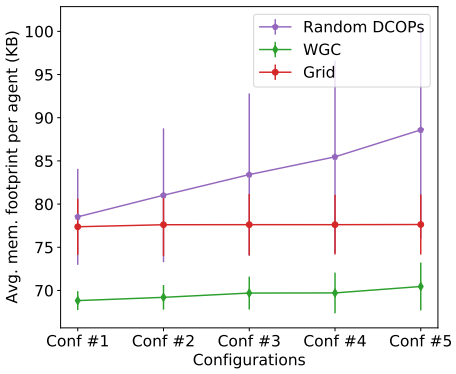


FIGURE 5.3: Memory footprint of GAT-PCM-DLNS

conducted on an Intel i9-9820X workstation with GeForce RTX 3090 GPUs. For each data point, we average the results over 50 instances and report standard error of the mean (SEM) as confidence intervals.

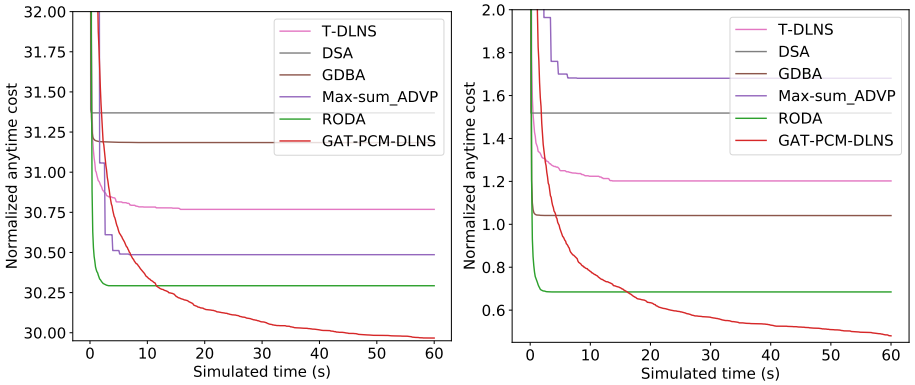
5.4.3 Implementation and Hyperparameters

Our GAT-PCM model has four GAT layers (i.e., $T = 4$). Each layer in the first three layers has 8 output channels and 8 heads of attention, while the last layer has 16 output channels and 4 heads of attention. Each GAT layer uses `ELU` [181] as the activation function. In the pretraining stage, we consider a random DCOP distribution with $|I| \in [15, 30]$, $D_{\max} \in [3, 15]$ and $p_1 \in [0.1, 0.4]$. Finally, we use DPOP [25] to generate the optimal cost labels.

For hyperparameters, we set the batch size and the number of training epochs to be 64 and 5000, respectively. Our model was implemented with the PyTorch Geometric framework [182] and the model was trained with the Adam optimizer [166] using the learning rate of 0.0001 and a 5×10^{-5} weight decay ratio.

5.4.4 Results

In the first set of experiments, we evaluate the performance of our GAT-PCM when combined with the DLNS framework, which we name it GAT-PCM-DLNS, in solving large-scale DCOPs. We run GAT-PCM-DLNS with destroy probability of 0.2 for 1000 iterations and report the normalized anytime cost (i.e., the best solution cost divided by the number of constraints) as the result. Figure 5.2 presents the

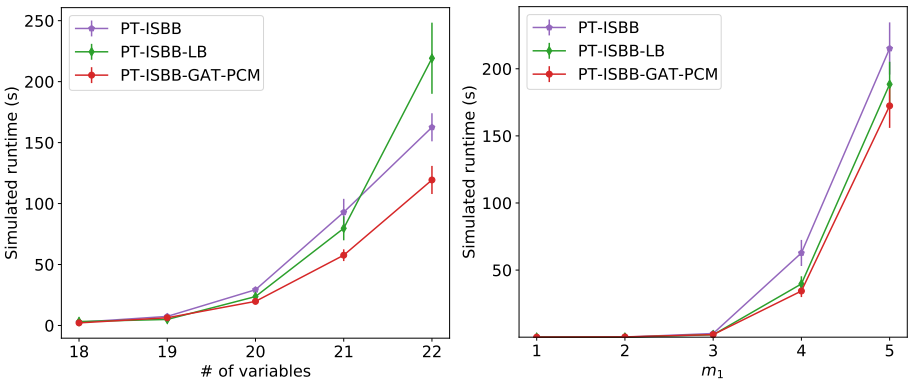


(a) random DCOPs (b) weighted graph coloring

FIGURE 5.4: Convergence analysis

results of solution quality where all baselines run for the same simulated runtime as GAT-PCM-DLNS. It can be seen that DSA explores low-quality solutions since it iteratively approaches a Nash equilibrium, resulting in 1-opt solutions similar to Max-sum_ADVP. GDBA improves by increasing the weights when agents get trapped in quasi-local minima. RODA finds solutions better than 1-opt by coordinating the variables in a coalition of size 3. T-DLNS, on the other hand, tries to optimize by optimally solving a tree-structured relaxation of the subproblem induced by the destroyed variables in each round. However, T-DLNS could ignore a large proportion of constraints and therefore perform poorly when solving complex problems (e.g., the problems with more than 200 variables). Differently, our GAT-PCM-DLNS solves the induced subproblem without relaxation, leading to a significant improvement over the state-of-the-arts when solving unstructured problems (i.e., Figure 5.2(a-b)). Interestingly, T-DLNS achieves the best performance when solving small grid networks. That is because the variables in the problem are under-constrained and T-DLNS only needs to drop few edges to obtain a tree-structured problem. In fact, the average degree in these problems is less than 3.8. However, our GAT-PCM-DLNS still outperforms T-DLNS when the grid size is higher than 14.

We display the average memory footprint per agent of GAT-PCM-DLNS in the first set of experiments in Figure 5.3, where "Conf #1" to "Conf #5" refer the growing complexity of each experiment. Specifically, the memory overhead of each agent consists of two parts, i.e., storing the pretrained model and local embeddings. The former consumes about 60KB memory, while the latter requires space proportional to the number of agents and the size of each constraint matrix (cf. Proposition 5.2).



(a) random DCOPs (b) scale-free networks

FIGURE 5.5: Runtime comparison on the problems with $D_{\max} = 5$

It can be concluded that our method has a modest memory requirement and scales up to large instances well in various settings. In particular, our method has a (nearly) constant memory footprint when solving grid network problems since each agent is constrained with at most four other agents regardless of the grid size.

To investigate how fast our GAT-PCM-DLNS finds a good solution, we conduct a convergence analysis which measures the performance in terms of simulated time [154] on the problems with $|I| = 1000, p_1 = 0.005$ and $D_{\max} = 3$ and present the results in Figure 5.4. It can be seen that local search algorithms including DSA and GDBA quickly converge to a poor local optimum, while RODA finds a better solution in the first three seconds. T-DLNS slowly improves the solution but is strictly dominated by RODA. In contrast, our GAT-PCM-DLNS improves much steadily, outperforming all baselines after 18 seconds.

Finally, we demonstrate the merit of our GAT-PCM in accelerating backtracking search for random DCOPs with $p_1 = 0.25$ and scale-free networks with $|I| = 18, m_0 = 5$ by conducting a case study on the symmetric version of PT-ISABB [79] (referred as PT-ISBB) and present the results in Figure 5.5. Specifically, we set the memory budget $k = 2$ and compare the simulated runtime of PT-ISBB using three domain ordering generation techniques: alphabetically, lower bound lookup tables (PT-ISBB-LB), and our GAT-PCM (PT-ISBB-GAT-PCM). For PT-ISBB-GAT-PCM, we only perform domain ordering for the variables in the first three levels in a pseudo tree. It can be observed that the backtracking search with alphabetic domain ordering performs poorly and is dominated by the one with the lower bound induced domain ordering in the most cases. Notably, when solving the problems with 22 variables, PT-ISBB-LB exhibits the worst performance, because the lower

bounds generated by approximated inference are not tight in complex problems, and hence the induced domain ordering may not prioritize promising assignments properly. On the other hand, our GAT-PCM powered backtracking search uses the predicted total cost of a subproblem as the criterion, resulting in a more efficient domain ordering and thus achieving the best results in solving complex problems.

5.5 Chapter Summary

In this chapter, we present GAT-PCM, the first effective and general purpose deep pretrained model for DCOPs. We propose a novel directed acyclic graph representation schema for DCOPs and leverage the Graph Attention Networks (GATs) to embed our graph representations. Instead of generating heuristics for a particular algorithm, we train the model with optimally labelled data to predict the optimal cost of a target assignment given a partial assignment, such that GAT-PCM can be applied to boost the performance of a wide range of DCOP algorithms where evaluating the quality of an assignment is critical. To enable efficient graph embedding in a distributed environment, we propose DES to perform decentralized model inference without disclosing local constraints, where each agent exchanges only the embedded vectors via localized communication. Finally, we develop several heuristics based on GAT-PCM to improve local search and backtracking search algorithms. Extensive empirical evaluations confirm the superiority of GAT-PCM based algorithms over the state-of-the-arts.

For future work, we plan to extend GAT-PCM to deal with the problems with higher-arity constraints and hard constraints. Besides, since agents in DES exchange the embedded vectors instead of constraint costs, it is promising to extend our methods to an asymmetric setting [80].

Chapter 6

Deep Attentive Belief Propagation: Integrating Reasoning and Learning for Solving Constraint Optimization Problems

6.1 Introduction

Belief Propagation (BP) [68, 89] is an important message-passing algorithm for various reasoning tasks over graphical models, e.g., computing partition function of a Markov random field [91], estimating the marginal distribution of a given set of random variables [88], and decoding LPDC codes [90]. When applied into solving COPs[1], BP, also known as Min-sum message passing [33], seeks to find a cost-optimal solution by propagating cost information over the corresponding factor graph.

It is known that vanilla BP does not guarantee convergence on a factor graph containing loops and thus, BP often explores low-quality solutions on COPs with cyclic factor graphs due to excessive loopy propagation. Therefore, considerable

[1] In this chapter, we focus on centralized COPs. The proposed method could be further adapted to solve DCOPs with minor adaptations. See Section 6.4 for details.

research efforts [9, 34–36, 92, 93] have been devoted to tackling the convergence issue of loopy BP. Among them, Damped BP (DBP)[2] [36] has drawn significant attention recently. By mixing the new message composed in each iteration with the old message composed in the previous iteration, i.e., damping, it has been shown that DBP with a suitable damping factor will often converge and achieve state-of-the-art performance in practice. In fact, damping can be considered as a degree of freedom to balance exploration and exploitation in BP [36].

Due to the significant impact of damping on BP [155], a damping factor is often treated as a hyperparameter and requires laborious case-by-case tuning. Besides, existing works often use a homogeneous and static damping factor for all variable nodes which limits DBP's capability. Moreover, DBP simply treats each variable node's neighbors equally when composing a new message, which fails to explore optimal composition strategy. On the other hand, Deep Neural Networks (DNNs) have been applied to boost the performance of BP [183–185]. However, existing DNN-based BP variants often require supervised learning and expensive training labels and thus, they cannot be directly applied to solve COPs when training labels are not available. They also manipulate or simulate the BP messages with some black-box DNN components, which destroys the semantic of a BP message being the weighted sum of cost functions' marginalizations and thus, the variable decision-making rule of BP for COPs is no longer applied (i.e., selecting the assignments with the minimum beliefs).

To address the above issues, in this chapter[3] we propose the *first self-supervised* DNN-based BP for solving COPs by seamlessly integrating BP, Gated Recurrent Units (GRUs) [65], and Graph Attention Networks (GATs) [64] within the message-passing framework to reason about *dynamic* weights and damping factors for composing new BP messages. Specifically, we make the following key contributions:

- We extend DBP by allowing dynamic damping factors and different neighbor weights for each variable node. Consequently, we have more fine-grained control for composing new messages to trigger effective exploration without altering the semantic of BP messages.

[2]The method is referred as "Damped Max-sum" in [36]. We use Damped BP for a coherent presentation.

[3]This chapter has been published in [186].

- We infer the optimal damping factors and neighbor weights for each iteration automatically by seamlessly integrating BP and DNNs. Our model, Deep Attentive Belief Propagation (DABP), firstly embeds respectively the factor graph and BP messages with GATs and GRUs, and infers damping factors and neighbor weights through a multi-head attention [66, 187] layer; Then, BP runs with those updated factors and weights to compose and propagate new messages, and so on. Note that integrating GRUs can capture the dynamics of BP messages and avoid the gradient vanishing/exploding issue in long BP reasoning sequences.

- We further equip our DABP with a novel self-supervised learning loss which is a smoothed surrogate for the cost under BP decision-making rule. Therefore, unlike existing DNN-based BP variants, we do not require expensive training labels which are often not available in practice, and also our model can avoid out-of-distribution issue with efficient online learning.

- We conduct extensive experimental evaluations on four standard benchmarks. The results show that our DABP achieves significantly higher convergence rate, where it successfully converges on average 96.25% instances given 1000 iterations, and it also outperforms the state-of-the-art baselines by considerable margins.

The rest of this chapter is organized as follows. Section 6.2 reviews related work and necessary preliminaries about BP for COP solving. In Section 6.3, we elaborate our DABP model. We include short discussion on how to extend our DABP into distributed settings in Section 6.4. Finally, Section 6.5 presents experimental results and Section 6.6 concludes.

6.2 Backgrounds

In this section, we review the closely related work and the preliminaries of Min-sum BP and Damped BP.

6.2.1 Related Work

Min-sum BP is an effective approximate algorithm for solving COPs and has been successfully applied to many real-world scenarios [13, 33, 146]. However, vanilla BP offers no convergence guarantee and usually returns low-quality solutions on problems with cyclic factor graphs. Therefore, considerable research efforts [9, 34–36, 92, 93] have been devoted to improve the performance of Min-sum BP on loopy problems. Among them, Damped BP (DBP) [36] has drawn significant attention recently, which increases the chance of convergence by mixing the new messages composed in each iteration with the old messages in previous iterations with a suitable damping factor. However, selecting a good damping factor often requires expertise and laborious case-by-case tuning [155], and DBP uses a static damping factor and treats each node's neighbors equally when composing a new message, which limits its exploration. Our DABP overcomes the aforementioned issues by learning to attend neighbors with dynamic damping factors and neighbor weights when composing new BP messages. Obviously, DABP generalizes DBP since DBP can be considered as a static version of DABP. Moreover, equipped with a self-supervised learning algorithm, our DABP is able to infer the optimal damping factors and weights to enhance the performance of DBP.

On the other hand, LP-based methods solve a COP by leveraging message-passing to solve a Linear Program (LP) relaxation of the combinatorial problem. MPLP [188] is a simple BP variant that solves the convex-dual of the LP relaxation via block gradient descent. Unfortunately, the global convergence is not always guaranteed since coordinate descent may get stuck in suboptimal points. Tree-reweighted belief propagation (TRBP) [189] and Fractional Belief Propagation [190] consider respectively a linear combination of free energies defined on spanning trees of the factor graph and a fractional free energy that generalizes the well-known Bethe free energy. Finally, Norm-Product [191] provides a uniform framework that generalizes Max-product, Sum-product and their TRBP counterpart. Unlike these methods, our DABP directly solve the problem by learning optimal hyperparameters for DBP, and does not require any form of LP relaxation which may either induce intractable number of constraints or only offer a lower bound [192].

Recently, there has been increasing interest in applying DNNs to boost the performance of BP. The most relevant works including: NEBP [183] learns to refine

BP messages with some black-box DNN components in each iteration and is applied to solve the error correction decoding task; In the same vein, BPNN [184] uses damping layers to modify factor-to-variable messages and is applied to estimate the partition function of Probabilistic Graphical Models (PGMs); FGNN [185] generalizes Graph Neural Networks (GNNs) to represent Max-Product BP and is applied to find approximate Maximum A Posteriori (MAP) assignment of a PGM. However, all these methods require supervised learning and cannot be directly applied to solve COPs when training labels are not available; They also destroy the semantic of a BP message being the weighted sum of cost functions' marginalizations. Contrarily, we only learn optimal dynamic factors and neighbor weights with DNNs to generalize DBP, which preserves the semantic of the BP messages and also enables us using a self-supervised manner to train our model. Therefore, our model does not require expensive training labels which are often not available in practice due to the heavy computation of exact solvers; and also avoids out-of-distribution issue and enjoys a nice anytime property [84] through efficient online learning.

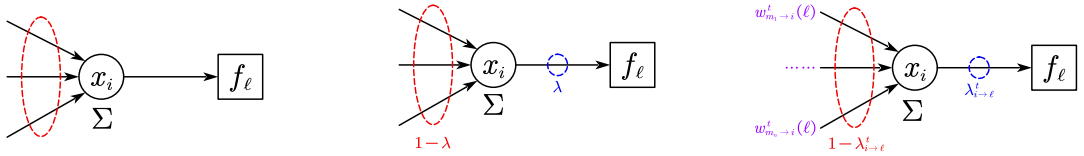
6.2.2 Min-sum Belief Propagation

Min-sum Belief Propagation (Min-sum BP) [33] is an important algorithm for COPs which performs message-passing on factor graphs. More specifically, the message sent from a variable node x_i to its neighbor f_ℓ in iteration t is a function $\mu^t_{x_i \to f_\ell} : D_i \to \mathbb{R}$ computed by

$$\mu^t_{x_i \to f_\ell} = \sum_{f_m \in \mathcal{N}_i \setminus \{f_\ell\}} \mu^{t-1}_{f_m \to x_i}, \tag{6.1}$$

where $\mathcal{N}_i$ is the neighbors of x_i in the factor graph, and $\mu^{t-1}_{f_m \to x_i} : D_i \to \mathbb{R}$ is the message sent from f_m to x_i in the previous iteration. Similarly, f_ℓ computes a message for its neighbor x_i by

$$\mu^t_{f_\ell \to x_i} = \min_{\mathcal{N}_\ell \setminus \{x_i\}} \left(f_\ell + \sum_{x_j \in \mathcal{N}_\ell \setminus \{x_i\}} \mu^{t-1}_{x_j \to f_\ell} \right). \tag{6.2}$$



(a) Vanilla BP (b) DBP (c) DABP

FIGURE 6.1: Comparison of BP variants. Obviously, our DABP generalizes both vanilla BP and DBP.

Finally, variable node x_i makes a decision by choosing a value with minimum belief cost:

$$d_i^t = \arg\min_{d_i \in D_i} \sum_{f_\ell \in \mathcal{N}_i} \mu^t_{f_\ell \to x_i}(d_i), \tag{6.3}$$

where $\mu^t_{f_\ell \to x_i}(d_i)$ is the belief cost of assignment d_i in the message $\mu^t_{f_\ell \to x_i}$.

However, vanilla Min-sum BP usually suffers from non-convergence and explores low-quality solutions on cyclic COPs due to excessive loopy propagation. Damped BP (DBP) [36] attempts to alleviate the issues by damping the messages sent from variable nodes to function nodes. Formally,

$$\mu^t_{x_i \to f_\ell} = \lambda \mu^{t-1}_{x_i \to f_\ell} + (1-\lambda) \sum_{f_m \in \mathcal{N}_i \setminus \{f_\ell\}} \mu^{t-1}_{f_m \to x_i}. \tag{6.4}$$

By choosing a suitable damping factor $\lambda \in (0, 1]$, DBP can drastically increase the chance of convergence as well as the performance of vanilla Min-sum BP.

6.3 Deep Attentive Belief Propagation

Although message damping has proved to be an effective technique to improve the performance of BP on COPs [36], it still suffers from various issues. First, selecting a damping factor is largely accomplished by empirical tuning, which could be laborious and the number of trials is limited by computational resources and/or runtime. Second, using a homogeneous and static damping factor for all variable nodes may limit DBP's capability. Finally, DBP simply treats each variable node's neighbors equally when composing a new message, which may fail to explore optimal message composition strategy.

We address the above limitations by allowing dynamic damping factors and different neighbor weights for each variable node. More specifically, in our framework a variable node x_i computes a message to function node f_ℓ at iteration t by

$$\mu^t_{x_i \to f_\ell} = \lambda^t_{i\to\ell}\mu^{t-1}_{x_i\to f_\ell} + (1-\lambda^t_{i\to\ell})(|\mathcal{N}_i|-1)\sum_{f_m\in\mathcal{N}_i\setminus\{f_\ell\}} w^t_{m\to i}(\ell)\mu^{t-1}_{f_m\to x_i}, \tag{6.5}$$

where $\lambda^t_{i\to\ell} \in [0,1]$ and $w^t_{m\to i}(\ell) \in [0,1]$ are learnable damping factor and neighbor weights for the message from x_i to f_ℓ at iteration t, respectively. Particularly, we require $\sum_{f_m\in\mathcal{N}_i\setminus\{f_\ell\}} w^t_{m\to i}(\ell) = 1$ and use $|\mathcal{N}_i|-1$ to rescale the weighted sum of the messages from neighbors. Consequently, we have more fine-grained control for composing new messages to trigger effective exploration without altering the semantic of BP messages. Figure 6.1 compares three different BP variants.

A key challenge of our framework is to determine the values of a large number of *time-varying* damping factors and neighbor weights (cf. Eq. (6.5)). Given a factor graph with $|X|$ variables and maximum degree of v, running for T iterations would require to select $O(T|X|v^2)$ hyperparameters from the continuous range of $[0,1]$, which obviously cannot be done by the traditional trial-based hyperparameter tuning methods (e.g., grid search, evolutionary optimization).

Therefore, we propose to automatically infer the optimal hyperparameters for Eq. (6.5) in each iteration by seamlessly integrating BP and DNNs within the parallel message-passing framework. Our model, Deep Attentive Belief Propagation (DABP), directly outputs the optimal damping factors and weights by learning to attend neighbors given the factor graph and the BP messages in each iteration:

$$\boldsymbol{w}^{\boldsymbol{t}}, \boldsymbol{\lambda}^{\boldsymbol{t}} = \mathcal{M}_\theta\left(FG, \boldsymbol{\mu}^{\mathbf{1:t-1}}_{\boldsymbol{x}\to\boldsymbol{f}}, \boldsymbol{\mu}^{\mathbf{1:t-1}}_{\boldsymbol{f}\to\boldsymbol{x}}\right), \tag{6.6}$$

where FG is the factor graph, $\boldsymbol{\mu}^{\mathbf{1:t-1}}_{\boldsymbol{x}\to\boldsymbol{f}}$ and $\boldsymbol{\mu}^{\mathbf{1:t-1}}_{\boldsymbol{f}\to\boldsymbol{x}}$ are the BP messages from variable nodes to function nodes and from function nodes to variable nodes in the previous iterations, respectively.

However, implementing such DNN-based model requires to perform inference along with BP message-passing iterations (cf. Eq. (6.6)), which could incur a severe gradient vanishing or gradient exploding problem in a long reasoning sequence. Moreover, obtaining optimal training labels can be extremely expensive for COPs due to the heavy computation of exact solvers, making it impracticable to perform

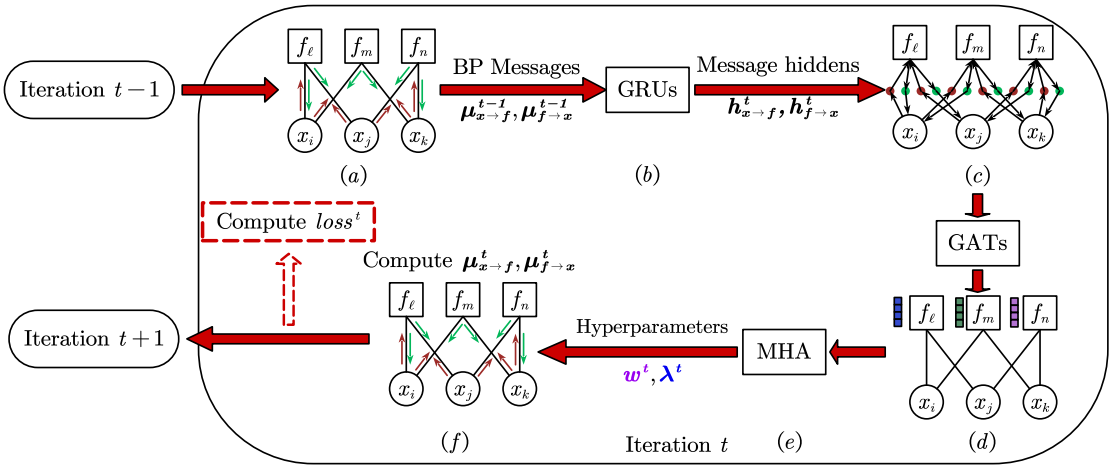


FIGURE 6.2: The overall architecture of DABP: (a) The BP messages from iteration $t-1$; (b) To update the message hidden vectors with the messages from iteration $t-1$. Note that these message hidden vectors capture the dynamics of a sequence of BP messages; (c) To insert message nodes with the hidden vectors into the factor graph; (d) To embed the augmented factor graph; (e) To infer optimal hyperparameters by using a multi-head attention layer (MHA); and (f) To compose and propagate new messages with updated hyperparameters and compute training loss in iteration t.

supervised model training on large instances. In Section 6.3.1, we address the first problem by introducing GRUs as a part of our encoder; while we design a novel self-supervised loss function and an efficient online learning framework in Section 6.3.2 to address the second issue.

6.3.1 Model Architecture

Figure 6.2 gives the overall architecture of our DABP. It consists of an encoder to embed the factor graph and the BP messages, and an attention module to infer the dynamic optimal hyperparameters.

Encoder. The purpose of our encoder is to embed the factor graph given a sequence of BP messages (cf. Figure 6.2(a-d)). Therefore, to capture the BP message dynamics and avoid gradient vanishing/exploding issue, we first use two GRUs to embed the BP messages from variable nodes to function nodes and the ones from function nodes to variable nodes into q-dimensional vectors, respectively (cf. Figure 6.2(b)):

$$\boldsymbol{h}^{\boldsymbol{t}}_{\boldsymbol{x}\to\boldsymbol{f}} = GRU_{\phi_1}(\boldsymbol{h}^{\boldsymbol{t-1}}_{\boldsymbol{x}\to\boldsymbol{f}}, \boldsymbol{\mu}^{\boldsymbol{t-1}}_{\boldsymbol{x}\to\boldsymbol{f}}), \quad \boldsymbol{h}^{\boldsymbol{t}}_{\boldsymbol{f}\to\boldsymbol{x}} = GRU_{\phi_2}(\boldsymbol{h}^{\boldsymbol{t-1}}_{\boldsymbol{f}\to\boldsymbol{x}}, \boldsymbol{\mu}^{\boldsymbol{t-1}}_{\boldsymbol{f}\to\boldsymbol{x}}), \tag{6.7}$$

where $\boldsymbol{h}^{\boldsymbol{t}}_{\boldsymbol{x}\to\boldsymbol{f}}, \boldsymbol{h}^{\boldsymbol{t}}_{\boldsymbol{f}\to\boldsymbol{x}} \in \mathbb{R}^{r\times q}$ are the message hiddens for iteration t and r is the number of edges in the factor graph.

To consider both the topology of a factor graph and the message dynamics, we insert a set of *message nodes* that carry the message hiddens (i.e., the brown and green dots in Figure 6.2(c)) and make the graph directed to reflect the message-passing directions. Then the augmented factor graph is embedded through G layers of GAT (cf. Figure 6.2(d)). Formally, in the $g \in \{1, \ldots, G\}$-th layer GAT_{ψ_g}, we compute the embedding $e_i^{t,(g)}$ for node i according to

$$e_i^{t,(g)} = \text{LeakyReLU}\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(g)} \boldsymbol{W}^{(g)} e_j^{t,(g-1)}\right), \quad \alpha_{ij}^{(g)} = \frac{\exp(s_{ij}^{(g)})}{\sum_{j'\in\mathcal{N}_i}\exp(s_{ij'}^{(g)})}, \tag{6.8}$$

where $\boldsymbol{W}^{(g)} \in \psi_g$ is a learnable matrix, $s_{ij}^{(g)} = a^{(g)}(\boldsymbol{W}^{(g)} e_i^{t,(g-1)}, \boldsymbol{W}^{(g)} e_j^{t,(g-1)})$ is the attention score between nodes i and j, and $a^{(g)}$ is a single-layer feed-forward neural network.

Attention Module. Given the embedding $e_\ell^{t,(G)}$ for each $f_\ell \in F$, we update the damping factors and neighbor weights by using a multi-head attention (MHA) layer [66] (cf. Figure 6.2(e)), where queries and keys are the embeddings of the target and corresponding neighboring function nodes, respectively. Specifically, for each variable node x_i and a target function node $f_\ell \in \mathcal{N}_i$, we first compute an attention score for each neighboring function node $f_m \in \mathcal{N}_i$ in $k \in \{1, \ldots, K\}$-th head Att_{ζ_k} by:

$$a_m^{t,(k)}(\ell) = \sigma\left(\boldsymbol{W_1^{(k)}}\left[\boldsymbol{W_2^{(k)}} e_\ell^{t,(G)} || \boldsymbol{W_3^{(k)}} e_m^{t,(G)}\right]\right), \tag{6.9}$$

where $\boldsymbol{W_1^{(k)}}, \boldsymbol{W_2^{(k)}}, \boldsymbol{W_3^{(k)}} \in \zeta_k$ are learnable matrices, σ is the Sigmoid function and $||$ is the concatenation operation, respectively. Then the attention scores are normalized to compute neighbor weights respect to target function node f_ℓ:

$$w_{m\to i}^{t,(k)}(\ell) = \frac{\exp\left(a_m^{t,(k)}(\ell)\right)}{\sum_{f_{m'}\in\mathcal{N}_i\setminus\{f_\ell\}}\exp\left(a_{m'}^{t,(k)}(\ell)\right)}, \quad \forall f_m \in \mathcal{N}_i\setminus\{f_\ell\}. \tag{6.10}$$

Similarly, the damping factor is computed by normalizing the attention score of f_ℓ w.r.t. the mean attention score of the remaining neighboring function nodes:

$$\lambda_{i\to\ell}^{t,(k)} = \frac{\exp\left(a_\ell^{t,(k)}(\ell)\right)}{\exp\left(a_\ell^{t,(k)}(\ell)\right) + \exp\left(\frac{1}{|\mathcal{N}_i|-1}\sum_{f_{m'}\in\mathcal{N}_i\backslash\{f_\ell\}} a_{m'}^{t,(k)}(\ell)\right)} \tag{6.11}$$

Finally, both the neighbor weights and damping factors are normalized across different heads, i.e.,

$$\lambda_{i\to\ell}^{t} = \frac{\sum_{k=1}^{K}\lambda_{i\to\ell}^{t,(k)}}{K}, \quad w_{m\to i}^{t}(\ell) = \frac{\sum_{k=1}^{K} w_{m\to i}^{t,(k)}(\ell)}{K}, \ \forall f_m \in \mathcal{N}_i\backslash\{f_\ell\}, \tag{6.12}$$

and are plugged into Eq. (6.5) to compute new BP messages for iteration t (cf. Figure 6.2(f)).

6.3.2 A Novel Self-supervised Learning Algorithm for Solving COPs

Recall that our objective is to find a cost-optimal solution (cf. Eq. (2.1)), a nature choice of the loss function would be the cost of the solution induced by BP messages in each iteration. Unfortunately, such loss function is not differentiable since variables are greedily assigned via `argmin` (cf. Eq. (6.3)). Instead, we propose to use a smoothed cost as the surrogate objective. Formally, for each iteration t, we consider the following self-supervised objective:

$$loss^t = \sum_{f_\ell\in F}\left(\sum_{\langle d_{i_1},\ldots,d_{i_n}\rangle\in\Pi_{x_i\in scp(f_\ell)}D_i} f_\ell(d_{i_1},\ldots,d_{i_n})\prod_{j=1:n} p_{i_j}^t(d_{i_j})\right), \tag{6.13}$$

where $p_i^t(d_i)$ is the probability of $x_i = d_i$ under the current belief, i.e.,

$$p_i^t(d_i) = \frac{\exp(-b_i^t(d_i))}{\sum_{d_i'\in D_i}\exp(-b_i^t(d_i'))}, \quad b_i^t(d_i) = \sum_{f_\ell\in\mathcal{N}_i}\mu_{f_\ell\to x_i}^t(d_i). \tag{6.14}$$

Intuitively, an assignment d_i is associated with a high probability if it has a low belief cost $b_i^t(d_i)$, which simulates the behavior of `argmin` in Eq. (6.3) and is in alignment with the objective in Eq. (2.1).

We now theoretically show the error bound of the smoothed cost.

Theorem 6.1. *Let* $\Delta_\ell = \max_{\langle d_{i_1},\dots,d_{i_n}\rangle} f_\ell(d_{i_1},\dots,d_{i_n}) - \min_{\langle d_{i_1},\dots,d_{i_n}\rangle} f_\ell(d_{i_1},\dots,d_{i_n})$ *be the maximum difference between cost values of each constraint function* f_ℓ *and* $\tau^t = \langle d_1^t,\dots,d_{|X|}^t\rangle$ *be the assignment induced by Eq. (6.3). We have*

$$\left| loss^t - \sum_{f_\ell \in F} f_\ell(d^t|_{scp(f_\ell)}) \right| \leq \sum_{f_\ell \in F} \Delta_\ell \left(1 - \prod_{x_i \in scp(f_\ell)} \frac{1}{|D_i|} \right).$$

Proof.

$$\begin{aligned}
\left| loss^t - \sum_{f_\ell \in F} f_\ell(\tau^t|_{scp(f_\ell)}) \right| &= \sum_{f_\ell \in F} \left| f_\ell(\tau^t|_{scp(f_\ell)}) - \sum_{\langle d_{i_1},\dots,d_{i_n}\rangle \in \Pi_{x_i \in scp(f_\ell)} D_i} f_\ell(d_{i_1},\dots,d_{i_n}) \prod_{j=1:n} p_{i_j}^t(d_{i_j}) \right| \\
&= \sum_{f_\ell \in F} \left| \sum_{\langle d_{i_1},\dots,d_{i_n}\rangle \in \Pi_{x_i \in scp(f_\ell)} D_i} \left(f_\ell(\tau^t|_{scp(f_\ell)}) - f_\ell(d_{i_1},\dots,d_{i_n}) \right) \prod_{j=1:n} p_{i_j}^t(d_{i_j}) \right| \\
&\leq \sum_{f_\ell \in F} \sum_{\substack{\langle d_{i_1},\dots,d_{i_n}\rangle \in \Pi_{x_i \in scp(f_\ell)} D_i \\ \langle d_{i_1},\dots,d_{i_n}\rangle \neq \tau^t|_{scp(f_\ell)}}} \Delta_\ell \prod_{j=1:n} p_{i_j}^t(d_{i_j}) \\
&= \sum_{f_\ell \in F} \Delta_\ell \left(1 - \prod_{x_i \in scp(f_\ell)} p_i(d_i^t) \right)
\end{aligned}$$

According to Eq. (6.3) and Eq. (6.14), for each $x_i \in X$ it must be the case that $d_i^t = \arg\min_{d_i \in D_i} b_i^t(d_i)$ and hence $d_i^t = \arg\max_{d_i \in D_i} p_i^t(d_i)$. We now show that $p_i^t(d_i^t) \geq \frac{1}{|D_i|}$. Assume by contradiction that $p_i^t(d_i^t) < \frac{1}{|D_i|}$. Since d_i^t has the highest probability,

$$\sum_{d_i \in D_i} p_i^t(d_i) \leq \sum_{d_i \in D_i} p_i^t(d_i^t) = |D_i| p_i^t(d_i^t) < 1,$$

which contradicts to the fact that p_i^t is a probability distribution over D_i. Therefore,

$$\left| loss^t - \sum_{f_\ell \in F} f_\ell(\tau^t|_{scp(f_\ell)}) \right| \leq \sum_{f_\ell \in F} \Delta_\ell \left(1 - \prod_{x_i \in scp(f_\ell)} p_i(d_i^t) \right) \leq \sum_{f_\ell \in F} \Delta_\ell \left(1 - \prod_{x_i \in scp(f_\ell)} \frac{1}{|D_i|} \right).$$

□

Given the smoothed objective for each iteration, we define the following loss function:

$$\mathcal{L}(\theta) = \frac{1}{|T^*|} \sum_{t \in T^*} loss^t, \tag{6.15}$$

Algorithm 9: A self-supervised online learning algorithm for the DABP model $\mathcal{M}_\theta$

Input: A COP instance $\mathcal{P} = \langle X, D, F \rangle$, the number of restart R, the maximal iteration limit $T_{\max}$, the update interval $T_{\text{upd}} \leq T_{\max}$, and the number of effective iterations $T_{\text{eff}} \in \{1, \dots, T_{\text{upd}}\}$

1 $\tau^* \leftarrow$ nil; $FG \leftarrow$ factor graph of $\mathcal{P}$
2 **for** $rs = 1, \dots, R$ **do**
3 $\quad \mu^0_{i\to\ell} \leftarrow \mathbf{0}, \mu^0_{\ell\to i} \leftarrow \mathbf{0}, \forall x_i \in X, f_\ell \in \mathcal{N}_i$
4 $\quad$ **for** $t = 1, \dots, T_{\max}$ **do**
5 $\quad\quad \boldsymbol{w^t}, \boldsymbol{\lambda^t} \leftarrow \mathcal{M}_\theta \left(FG, \boldsymbol{\mu^{t-1}_{x\to f}}, \boldsymbol{\mu^{t-1}_{f\to x}} \right)$
6 $\quad\quad$ compute $\mu^t_{i\to\ell}$ according to Eq. (6.5) with hyperparameters $\boldsymbol{w^t}, \boldsymbol{\lambda^t}, \forall x_i \in X, f_\ell \in \mathcal{N}_i$
7 $\quad\quad$ compute $\mu^t_{\ell\to i}$ according to Eq. (6.2) $\forall x_i \in X, f_\ell \in \mathcal{N}_i$
8 $\quad\quad$ compute solution $\tau^t = \langle d^t_1, \dots, d^t_{|X|} \rangle$ according to Eq. (6.3)
9 $\quad\quad$ **if** *cost*(τ^*)>*cost*(τ^t) **then** $\tau^* \leftarrow \tau^t$
10 $\quad\quad$ **if** $t \,\%\, T_{upd} = 0$ **then**
11 $\quad\quad\quad T^* \leftarrow$ select the best T_{eff} iterations according to cost($\tau^{t'}$), $t - T_{\text{upd}} < t' \leq t$
12 $\quad\quad\quad$ compute $\mathcal{L}(\theta)$ according to Eq. (6.15), $\theta \leftarrow$ADAM($\theta, \nabla\mathcal{L}(\theta)$)
13 $\quad\quad$ **if** *BP messages converge* **then break**
14 **return** τ^*

where $\theta = \{\phi_1, \phi_2, [\psi_1, \dots, \psi_G], [\zeta_1, \dots, \zeta_K]\}$ is the model parameters and T^* is a selected subset of iterations used for optimizing, respectively. Particularly, we rank each iteration t according to the quality of solution τ^t and select the best T_{eff} iterations as effective iterations T^*. This way, we encourage exploration by considering only good-enough solutions.

Since our model is self-supervised by the smoothed cost, there is no need for costly label generation for each instance and thus, DABP can be directly applied to solve COPs via online learning without any pre-training procedure. That is, from an initial DABP model $\mathcal{M}_\theta$, we continuously improve it by training on the instance multiple rounds (i.e., *restart*) on the COP to be solved and return the best solution as the result. Therefore, our DABP avoids the out-of-distribution issue and has a nice anytime property [84]. The procedure is detailed in Alg. 9.

Specifically, each round of restart begins with resetting BP messages to zero vectors (line 3), followed by a sequence of message-passing iterations (line 4-13). In each iteration, we query our DABP for the damping factors and neighbor weights (line 5), perform BP message passing according to the updated hyperparameters (line 6-7), and finally update the current best solution (line 8-9). Particularly, we perform

model update every T_{upd} iterations (line 10-12) and thus, we further alleviate the gradient vanishing/exploding issue by restricting the maximum length of reasoning sequence to T_{upd}. Lastly, we early stop each round whenever BP messages converge (line 13) and return the best solution.

Finally, it is noteworthy that our DABP can be easily applied to Max-sum BP with two slight modifications. First, the probability of x_i taking assignment $d_i \in D_i$ under the current belief is modified as

$$p_i^t(d_i) = \frac{\exp(b_i^t(d_i))}{\sum_{d_i' \in D_i} \exp(b_i^t(d_i'))}, \quad b_i^t(d_i) = \sum_{f_\ell \in \mathcal{N}_i} \mu_{f_\ell \to x_i}^t(d_i).$$

Intuitively, an assignment $x_i = d_i$ has a higher probability if it has a higher belief $b_i^t(d_i)$, which simulates the greedily assigning strategy (i.e., taking `argmax` w.r.t. beliefs) in Max-sum BP. Second, the self-supervised learning objective for each iteration is modified as

$$loss^t = -\sum_{f_\ell \in F} \left(\sum_{\langle d_{i_1}, \ldots, d_{i_n} \rangle \in \Pi_{x_i \in scp(f_\ell)} D_i} f_\ell(d_{i_1}, \ldots, d_{i_n}) \prod_{j=1:n} p_{i_j}^t(d_{i_j}) \right),$$

to accommodate the maximization objective of Max-sum BP.

6.4 Discussions on Extending DABP to DCOPs

Our DABP can be adapted to solve DCOPs through minor modifications on graph embedding (line 5) and self-supervised learning (line 11), which are respectively discussed as follows.

Graph embedding. To learn representation for each function node in a fully-decentralized manner, agents just need to perform graph embedding cooperatively, because updating the message hidden does not involve any interaction among agents and can be done locally. Note that GAT layers are applied sequentially when performing graph embedding, i.e., the embedding of node i in $j > 1$ step is computed by applying the j-th layer of GAT on the embedding of the neighbors of node i in $j - 1$ step. Therefore, we could implement fully-decentralized graph embedding with synchronous message-passing like BP. That is, after applying j-th layer of GAT, each node in the factor graph sends its embedding vector to its

neighbors to facilitate the computation of the next step of graph embedding, until all the GAT layers are exhausted.

Self-supervised learning. Recall that the loss function (cf. Eq. (6.15)) is the average of smoothed cost over iterations T^*. Therefore, by the linearity of differentiation,

$$\begin{aligned}\nabla\mathcal{L}(\theta) &= \frac{1}{|T^*|}\sum_{t\in T^*}\nabla loss^t \\ &= \frac{1}{|T^*|}\sum_{t\in T^*}\sum_{f_\ell\in F}\Bigg(\underbrace{\nabla \sum_{\langle d_{i_1},\ldots,d_{i_n}\rangle\in\Pi_{x_i\in scp(f_\ell)}D_i} f_\ell(d_{i_1},\ldots,d_{i_n})\prod_{j=1:n}p_{i_j}^t(d_{i_j})}_{\text{gradients of the smoothed local cost of } f_\ell}\Bigg).\end{aligned}$$

Note that the gradients of the smoothed local cost can be computed by each function node f_ℓ locally. As a result, the computation of overall gradient $\nabla\mathcal{L}(\theta)$ can be efficiently implemented in a fully-decentralized way, by exploiting the anytime local search framework [39].

In more detail, each agent keeps track of the gradients of its owned function node, and communicates the local cost to the root agent through a BFS spanning tree. When perform model updating, the root agent selects the set of iterations T^* and notifies all other agents through the BFS tree. After that, each agent sends the sum of local gradients over selected iterations to the root agent. After collecting the gradient information from all the other agents, the root agent is able to compute the overall gradient $\nabla\mathcal{L}(\theta)$ and broadcasts it to all agents through BFS tree. Finally, all agents apply gradient by using ADAM optimizer [166].

6.5 Empirical Evaluations

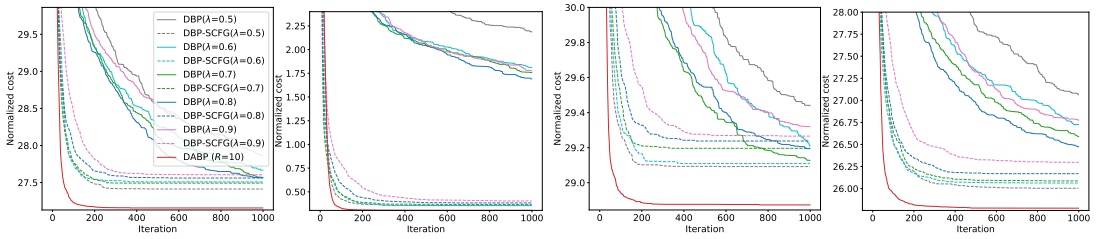
In this section, we perform extensive empirical studies. We begin with introducing the details of experiments and implementation. Then we analyze the impact of hyperparameters of DBP and our DABP. Finally, we show the great superiority of our DABP over the state-of-the-arts.

6.5.1 Experimental Setups

Benchmarks. We consider four types of benchmarks in our experiments, i.e., random COPs, scale-free networks, small-world networks, and Weighted Graph Coloring Problems (WGCPs). For random COPs and WGCPs, given $|X|$ variables and density of $p_1 \in (0, 1)$, we randomly create a constraint for a pair of variables with probability p_1. For scale-free networks, we use the BA model [167] with parameter m_0 and m_1 to generate constraints: starting from a connected graph with m_0 vertices, a new vertex is connected to m_1 vertices with a probability which is proportional to the degree of each existing vertex in each iteration. For small-world networks, we generate problem topology according to Newman-Watts-Strogatz model [193]: starting from a ring of $|X|$ vertices, each vertex is connected to its k nearest neighbors, and for each edge underlying the ring with k nearest neighbors, we create a new shortcut by randomly selecting another vertex with a probability p. Finally, for each variable and constraint in the benchmarks except WGCPs, we set domain size to 15 and uniformly sample a cost from $[0, 100]$ for each possible assignment combination, respectively. Differently, for WGCPs each variable has a domain of 5 values, and for any pair of constrained variables, we uniformly sample a cost from $[1, 100]$ for an unanimous variable assignment and otherwise, a zero cost.

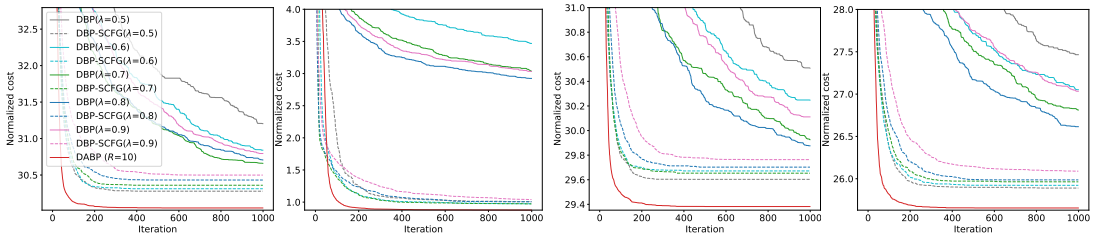
Baselines. We compare our DABP with the following state-of-the-art COP solvers: (1) DBP with a damping factor of 0.9 and its splitting constraint factor graph version (DBP-SCFG) with a splitting ratio of 0.95 [36]; (2) GAT-PCM-LNS with a destroy probability of 0.2 [170], which is a local search method combining the LNS framework [82, 122] with neural-learned repair heuristics; (3) Mini-bucket Elimination (MBE) with an i-bound of 9 [194], which is a memory-bounded inference algorithm; (4) Toulbar2 with timeout of 1200s [195], which is a highly optimized exact solver written in C++. The hyperparameters for DBP and GAT-PCM-LNS are set according to the original papers, while the memory budget for MBE and timeout for Toulbar2 are set based on our computational resources.

All experiments are conducted on an Intel i9-9820X workstation with GeForce RTX 3090 GPUs and 384GB memory. We terminate DBP(-SCFG) and GAT-PCM-LNS whenever convergence or the maximum iteration limit ($T_{\max} = 1000$) reaches. Finally, we report the best solution cost for each run, and for each experiment we average the results over 100 random problem instances.



(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.3: Solution quality comparison with different λ ($|X| = 60$)



(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

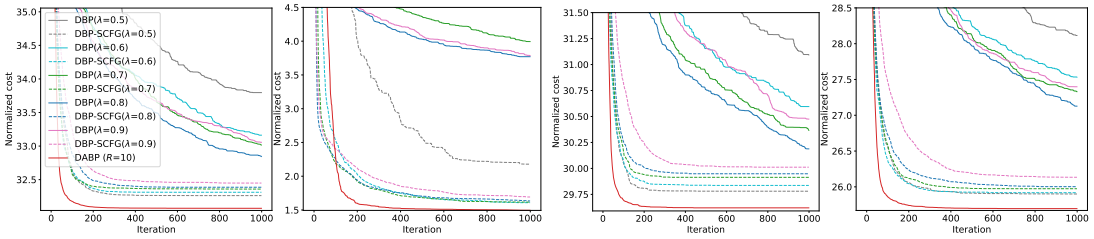
FIGURE 6.4: Solution quality comparison with different λ ($|X| = 80$)

Implementation Details. Our DABP consists of two GRUs which embeds BP messages into 8-dimensional hidden vectors, followed by $G = 4$ layers of GAT, each of them has 8 output channels and 4 attention-heads. We then infer the optimal damping factors and neighbor weights by using a multi-head attention layer with $K = 4$ attention-heads. Finally, we also adopt the SCFG scheme with the splitting ratio of 0.95 [36]. Our model was implemented with the PyTorch Geometric framework [182] and the model was trained with the Adam optimizer [166] using a learning rate of 10^{-4} and a weight decay ratio of 5×10^{-5}. For each instance, we perform $R = 20$ restarts with iteration limit $T_{\max} = 1000$ and update the model every $T_{\text{upd}} = 20$ iterations.

6.5.2 Hyperparameter Tuning

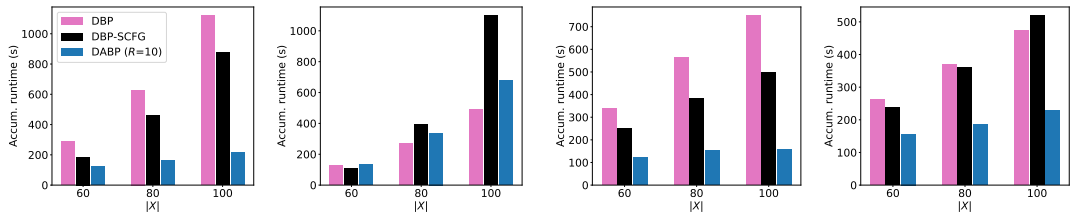
In this subsection, we empirically study the impact of damping factor λ on the performance of DBP and DBP-SCFG, and the impact of the number of effective iterations on the performance of our DABP, respectively.

Impact of Damping Factor λ. We vary λ from 0.5 to 0.9 with a step size of 0.1, and Figure 6.3-6.5 present the solution quality performance of DBP and DBP-SCFG under different damping factors. It can be observed that the performance



(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.5: Solution quality comparison with different λ ($|X| = 100$)

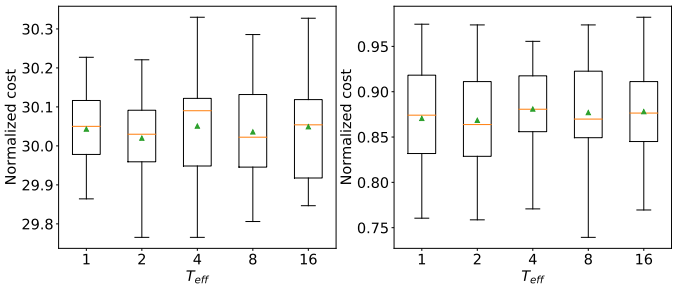


(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.6: Accumulated runtime comparison

of DBP varies a lot with different λ while DBP-SCFG is relatively less sensitive to the damping factor. In more detail, DBP with a small λ (e.g., $\lambda = 0.5$ or $\lambda = 0.6$) trends to produce low-quality solutions, which is due to the fact that a small damping factor usually cannot eliminate the effect of the costs accumulated before the final periodical of the BP process [155]. On the other hand, a large damping factor (e.g., $\lambda = 0.9$) can also lead to poor results, since it requires significantly more iterations to update the beliefs before finding a high-quality solution.

However, it can be extremely tedious and time-consuming to tune the damping factor. Figure 6.6 presents the accumulated runtime of tuning damping factor in DBP and DBP-SCFG, and the one of DABP with 10 times of restart. It can be seen that both DBP and DBP-SCFG require significantly higher runtime if we tune the damping factor. In fact, the runtime is generally proportional to the number of damping factors we have attempted. On the other hand, to obtain high-quality solutions one needs to perform extensive tuning (e.g., by using a smaller step size or a wider range), which substantially increases the overall runtime overhead. In contrast, our DABP automatically infers the optimal hyperparameters from the BP messages in the previous iterations, eliminating the need of notoriously laborious tuning procedure and finding better solutions in all test cases with much smaller runtime.



(a) rnd COPs (b) WGCPs

FIGURE 6.7: Solution quality when varying T_{eff} (i.e., the number of effective iterations in Alg. 9).

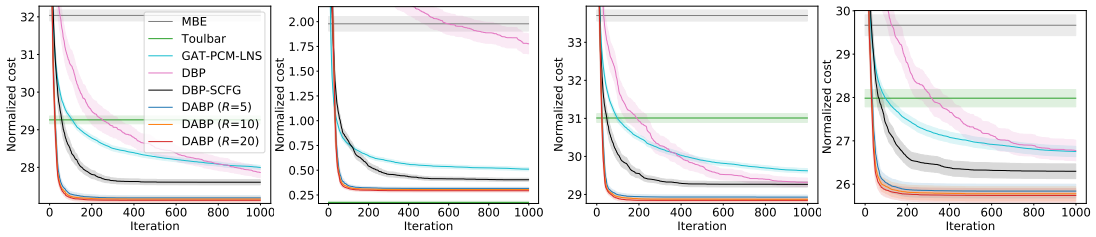
Impact of the Number of Effective Iterations T_{eff}. We train our DABP with different T_{eff} on the Random COPs and WGCPs with 80 variables and $p_1 = 0.25$. Figure 6.7 presents the solution quality when varying T_{eff}. It can be seen that a large T_{eff} usually produces inferior solutions. In such scenario, T^* may contain many suboptimal iterations and their solutions are relatively easy to be improved. Consequently, DABP is more exploitative and thus, it is prone to get trapped in local optima. In contrast, a small T_{eff} forces DABP to improve only good-enough solutions and encourages exploration. In our experiments, we use $T_{\text{eff}} = 2$ due to its better solution quality.

6.5.3 Performance Comparison

Table 6.1 compares the performance of different methods on the four standard benchmarks. We observe that, although given a relatively high timeout (i.e., 20 minutes), Toulbar2 still performs poorly on the most test cases. This is not surprising because it systematically explores the whole solution space, which is often infeasible for large problem instances. Also, it highlights the extreme difficulty of obtaining optimal labels with an exact solver for the existing DNN-based BP variants. MBE, on the other hand, performs memory-bounded inference and thus runs much faster than Toulbar. However, MBE fails to find good solutions and is strictly dominated by the other algorithms. GAT-PCM-LNS performs iterative large neighborhood search with a machine-learned repair heuristic and often outperforms DBP. However, GAT-PCM-LNS requires significant longer runtime than DBP since it needs to perform multiple times of model inference in each iteration to re-assign the destroyed variables. With a small modification in a factor graph,

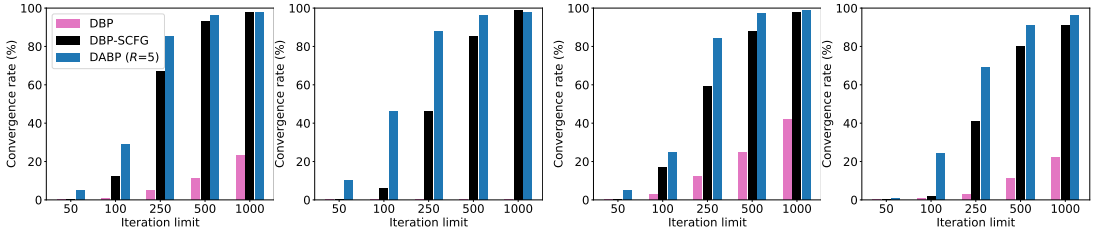
TABLE 6.1: The performance of different methods. Costs are normalized by the number of constraints $|F|$. Gap is the ratio of cost difference w.r.t. the best result. The best results are shown in **bold**.

Random COPs ($p_1 = 0.25$)									
	$\|X\| = 60$			$\|X\| = 80$			$\|X\| = 100$		
Methods	Cost	Gap	Time	Cost	Gap	Time	Cost	Gap	Time
Toulbar2	29.26	7.88%	20m	32.49	8.23%	20m	34.36	7.23%	20m
MBE	32.04	18.11%	4m2s	34.85	16.10%	8m38s	36.76	14.72%	11m58s
GAT-PCM-LNS	28.00	3.21%	5m27s	30.80	2.59%	12m55s	32.78	2.31%	24m47s
DBP	27.86	2.72%	1m11s	30.79	2.58%	2m25s	33.06	3.17%	4m18s
DBP-SCFG	27.60	1.77%	**52s**	30.50	1.60%	2m6s	32.45	1.28%	3m59s
DABP ($R = 5$)	27.19	0.24%	1m4s	30.09	0.23%	**1m20s**	32.12	0.26%	**1m49s**
DABP ($R = 10$)	27.16	0.12%	2m2s	30.05	0.10%	2m41s	32.07	0.10%	3m37s
DABP ($R = 20$)	**27.12**	**0.00%**	4m	**30.02**	**0.00%**	5m20s	**32.04**	**0.00%**	7m20s
WGCPs ($p_1 = 0.25$)									
Toulbar2	**0.18**	**0.00%**	20m	1.23	41.06%	20m	2.11	42.09%	20m
MBE	1.98	1028.69%	**0s**	2.81	223.13%	**0s**	3.43	131.05%	**1s**
GAT-PCM-LNS	0.51	191.35%	57s	1.16	33.98%	2m40s	1.79	20.47%	5m56s
DBP	1.78	913.53%	29s	3.03	249.36%	1m4s	3.79	154.99%	1m55s
DBP-SCFG	0.40	130.39%	30s	1.04	19.77%	1m59s	1.69	13.97%	4m29s
DABP ($R = 5$)	0.32	80.52%	1m9s	0.89	2.39%	2m53s	1.57	5.50%	5m47s
DABP ($R = 10$)	0.30	73.80%	2m15s	0.88	0.92%	5m37s	1.50	0.69%	11m19s
DABP ($R = 20$)	0.29	67.12%	4m26s	**0.87**	**0.00%**	11m10s	**1.49**	**0.00%**	22m35s
Scale-free networks ($m_0 = m_1 = 10$)									
Toulbar2	31.01	7.51%	20m	31.70	8.07%	20m	32.69	10.51%	20m
MBE	33.70	16.85%	4m8s	34.26	16.82%	5m43s	34.58	16.91%	7m9s
GAT-PCM-LNS	29.62	2.70%	6m25s	30.41	3.66%	10m55s	31.10	5.16%	17m7s
DBP	29.32	1.65%	1m16s	30.11	2.66%	2m8s	30.47	3.04%	2m57s
DBP-SCFG	29.27	1.46%	1m18s	29.76	1.47%	1m51s	30.01	1.48%	2m27s
DABP ($R = 5$)	28.93	0.30%	**1m2s**	29.43	0.32%	**1m15s**	29.67	0.33%	**1m18s**
DABP ($R = 10$)	28.87	0.10%	2m2s	29.38	0.17%	2m33s	29.62	0.15%	2m37s
DABP ($R = 20$)	**28.84**	**0.00%**	4m3s	**29.33**	**0.00%**	5m9s	**29.58**	**0.00%**	5m12s
Small-world networks ($k = 10, p = 0.3$)									
Toulbar2	27.98	8.72%	20m	28.28	10.37%	20m	28.36	10.60%	20m
MBE	29.67	15.26%	2m46s	29.39	14.71%	3m39s	29.54	15.17%	4m50s
GAT-PCM-LNS	26.75	3.93%	4m	26.65	4.04%	6m19s	26.68	4.02%	9m12s
DBP	26.77	4.02%	**1m**	27.03	5.52%	**1m23s**	27.40	6.83%	**1m50s**
DBP-SCFG	26.30	2.17%	1m4s	26.09	1.83%	1m31s	26.13	1.90%	2m10s
DABP ($R = 5$)	25.84	0.39%	1m15s	25.70	0.33%	1m31s	25.73	0.34%	1m56s
DABP ($R = 10$)	25.78	0.15%	2m36s	25.65	0.14%	3m6s	25.70	0.20%	3m49s
DABP ($R = 20$)	**25.74**	**0.00%**	5m8s	**25.62**	**0.00%**	6m20s	**25.65**	**0.00%**	7m31s



(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.8: Solution quality comparison ($|X| = 60$)

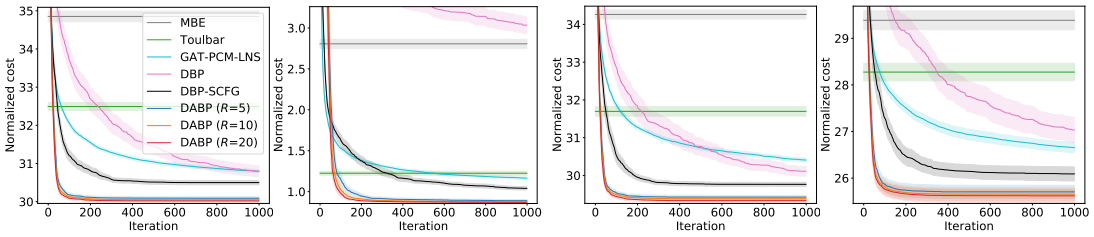


(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.9: Convergence rates under different iteration limits ($|X| = 60$)

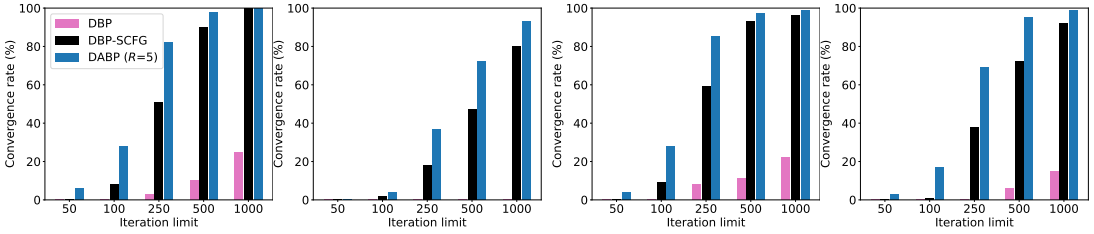
DBP-SCFG drastically improves the performance of DBP but it is computationally heavier. That is because the number of function-nodes in DBP-SCFG is doubled due to constraint function splitting.

Our DABP exhibits great superiority in various benchmarks. Specifically, given 5 times of restart, DABP substantially outperforms DBP-SCFG, which is currently the strongest approximate solver for COPs, on all test cases and the gap is widened with the increasing R. This demonstrates the merits of our learned dynamic damping factors and neighbor weights over the static and homogeneous counterpart. Importantly, although our DABP requires to perform online learning when solve each instance, our DABP's runtime results are still comparable to DBP-SCFG and also scales up well to large problem instances, thanks to our seamlessly integration of BP and DNNs within the message-passing framework which enables efficient parallel computation on GPUs. In fact, when applied to scale-free networks, DABP ($R = 5$) requires less runtime for all configurations, and has a much lower growing rate ($\sim$23.81%) than DBP-SCFG ($\sim$37.37%). Besides, equipped with the self-supervised loss function for minimizing the smoothed cost, our DABP is able to infer the optimal hyperparameters that foster fast convergence, which also improves our model's efficiency.



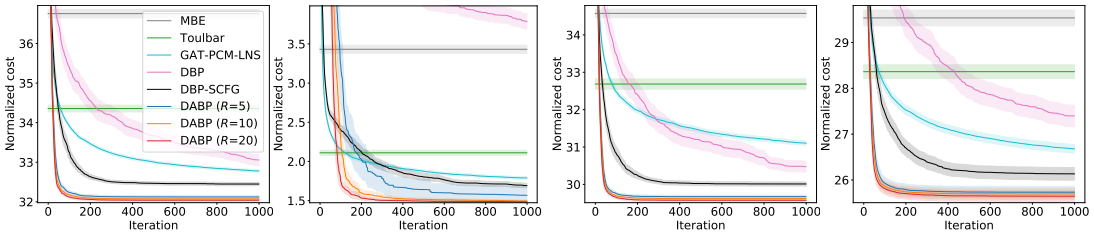
(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.10: Solution quality comparison ($|X| = 80$)



(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.11: Convergence rates under different iteration limits ($|X| = 80$)

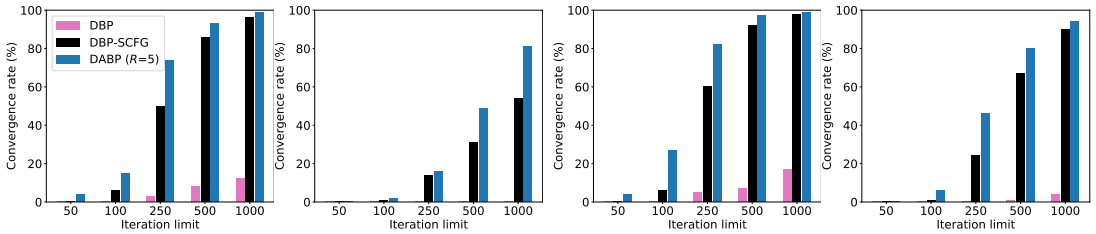


(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.12: Solution quality comparison ($|X| = 100$)

6.5.4 Convergence Analysis

We further compare the solution quality of different methods in each iteration and analyze the convergence rate of BP variants in Figure 6.8-6.13. We omit the cases of DABP with $R = 10$ and $R = 20$ in convergence rate results since they are similar to the one of DABP ($R = 5$). It can be seen that DBP improves slowly and fails to find good solutions within 1000 iterations, especially on WGCPs. DBP-SCFG substantially improves the convergence as well as solution quality of DBP by introducing another degree of freedom through asymmetric function splitting [36]. On the other hand, our DABP allows more flexibility in composing BP messages by reasoning optimal dynamic hyperparameters through seamlessly integrating BP and DNNs, and it also significantly outperforms and converges much faster than both DBP and DBP-SCFG.



(a) rnd COPs (b) WGCPs (c) SF nets (d) SW nets

FIGURE 6.13: Convergence rates under different iteration limits ($|X| = 100$)

6.6 Chapter Summary

In this chapter, we propose DABP, the first self-supervised DNN-based BP for solving COPs. By allowing dynamic damping factors and different neighbor weights for each variable-node, our DABP strictly generalizes the state-of-the-art DBP and has more fine-grained control for composing new messages to trigger effective exploration without altering the semantic of BP messages. Moreover, by seamlessly integrating BP, GRUs and GATs within the message-passing framework, DABP is able to infer the optimal hyperparameters automatically according to the BP message dynamics. Finally, equipped with a novel self-supervised loss function, our DABP does not require expensive training labels and can avoid out-of-distribution issue with efficient online learning. Extensive experiments confirm the great superiority of our DABP over the state-of-the-art baselines.

Since it relies smoothed costs as the self-supervised loss function, our DABP is dedicated to solve COPs and is not trivial to be adapted for other scenarios. Therefore, in future we will explore more training paradigms to extend our model to solve other important reasoning tasks over graphical models, e.g., computing partition function [91], and finding constrained most probable explanation [196]. It is also interesting to further boost the performance of our model by incorporating popular deep learning techniques such as contrastive learning [197] and transformer [66].

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

DCOPs are an important formalism for multi-agent coordination and optimization and have been successfully applied to various real-world applications including radio channel allocation, vessel navigation, mobile sensor team and smart grids. In this thesis, we aim to develop *scalable* and the *efficient* techniques for solving large-scale DCOPs by leveraging the power of heuristic search and deep learning to efficiently explore the high-dimensional solution space.

In Chapter 3, we improve the scalability of Max-sum BP by combining B&B and domain pruning technique. We first propose GD^{2}P to accelerate BP on the problems with densely distributed utility functions by continuously tightening a running lower bound. We further enhance GD^{2}P by merging the entries with the same utility value into a search tree to enable efficient B&B on tied entries, resulting in a new algorithm called ST-GD^{2}P. To balance the reconstruction overhead and the domain pruning efficiency, we propose a discretization mechanism for ST-GD^{2}P which merges small search trees into a bigger one. Finally, we introduce a Partial Tree-sorting Scheme (PTS) with different sorting criteria to reduce the memory consumption of ST-GD^{2}P. We theoretically show the soundness and the superiority in terms of pruning capability of our algorithms. Extensive empirical evaluations

indicate that the proposed algorithms significantly outperform the state-of-the-art acceleration techniques in various benchmarks.

In Chapter 4, we develop a novel scalable context-based sampling algorithm for solving DCOPs by leveraging DNNs to approximate the high-dimensional tables generated during the solving process. We first introduce a new context-based sampling algorithm built upon regret-matching, with a novel regret rounding scheme for incentivizing exploration. Compared to the existing regret-based local search algorithms, our method performs sampling on a pseudo tree and stores regret values for each context separately, which allows an agent to devise different strategies for different contexts. Then we present a neural-based scheme to improve the scalability of the proposed sampling algorithm by using DNNs to approximate the regret values, which is the first attempt to leverage DNNs to solve DCOPs. Finally, we theoretically analyze the regret bound of our algorithms, and demonstrate the superiority over the state-of-the-art baselines through extensive empirical evaluations.

In Chapter 5, we develop the first general-purpose deep model for DCOPs, called GAT-PCM, to generate efficient heuristics for a wide range of DCOP algorithms by learning to estimate the optimal cost of a target assignment given a partially-instantiated DCOP instance. Specifically, by introducing a novel directed tripartite graph representation based on microstructure, we effectively encode a partially instantiated DCOP instance and use GATs to learn generalizable embeddings. We then pretrain the model with the optimally labelled data generated by an exact DCOP solver. To enable decentralized model inference without disclosing local constraint costs, we propose a Distributed Embedding Schema (DES) where each agent exchanges only the embedded vectors via localized communication. As a case study, we develop two efficient heuristics for local search and backtracking search for DCOPs based on GAT-PCM, respectively. Finally, extensive empirical evaluations indicate that GAT-PCM-boosted algorithms significantly outperform the state-of-the-art methods in various standard benchmarks.

In Chapter 6, we investigate using DNNs to boost the performance of BP. Our model, DABP, seamlessly integrates BP, GRUs, and GATs within the message-passing framework to reason about dynamic neighbor weights and damping factors for composing new BP messages. Specifically, given a factor graph and the BP

messages in previous iterations, we infer the optimal hyperparameters for current iteration through GRUs and GATs, followed by a multi-head attention layer. Furthermore, unlike existing neural-based BP variants, we propose a novel self-supervised learning algorithm for DABP with a smoothed cost, which does not require expensive training labels and also avoids the common out-of-distribution issue through efficient online learning. Extensive experiments show that our model significantly outperforms state-of-the-art baselines.

7.2 Future Directions

In this section, we discuss several future directions.

First, the existing acceleration methods for Max-sum BP are based on purely static knowledge and fail to exploit the characteristics of GDL algorithms or runtime information. For example, query messages in Damped Max-sum [36] could change slowly when damping factor is high. Therefore, one can leverage such similarity between two consecutive iterations and give priority to search the neighborhood of optimal assignment found in the last iteration. As a result, a possible research direction is to investigate how to utilize the algorithmic characteristics or running history to further improve BP's scalability.

Second, beside context-based regret-matching, there are also DCOP algorithms like DPOP [25] and DUCT [4] that require to store exponential size of information during the solving process, which severely limits their scalability. Unfortunately, these methods usually involve highly nonlinear operations when computing such information (e.g., taking min w.r.t. a cost table in DPOP), which makes it nontrival to parameterize them by DNNs due to the compounding approximation errors. Therefore, an interesting yet challenging direction would be developing effective neural scheme for these algorithms to reduce the memory consumption while keeping the approximation errors acceptable.

Third, although our GAT-PCM show superior performance on generating efficient heuristics for DCOP algorithms, it cannot deal with the instances with hard constraints. That is primarily due to the fact that hard constraints are typically denoted by infinity cost, making it difficult to train directly due to the numerical

instability when regressing the optimal cost values. Consequently, training the omnipotent cost model for general DCOPs that support hard constraints could be a promising research direction. Besides, privacy is a critical concern when deploying DCOP algorithms. Therefore, another important research direction is to develop secure mechanisms that prevent agents from discovering sensitive information (e.g., constraint costs) by reverse-engineering the embedded vectors exchanged in DES.

Finally, our DABP is dedicated to solve COPs and is not trivial to be adapted for other scenarios since it relies smoothed costs as the self-supervised loss function. Therefore, it is interesting to explore more training paradigms to extend our model to solve other important reasoning tasks over graphical models, e.g., computing partition function [91], and finding constrained most probable explanation [196]. Besides, our DABP currently solves each COP instance from scratch, i.e., starting from randomly initialized model weights and continuously improving through multiple restarts, which is simple but less efficient when the problem instances have a similar pattern. Therefore, another promising direction for future work could be leveraging meta-learning [198] to learn efficient initial weights, so as to adapt new COP instances fast and reduce the number of restarts needed.

List of Publications

International Conferences

- **Yanchen Deng** and Bo An. Speeding up incomplete GDL-based algorithms for multi-agent optimization with dense local utilities. Proceedings of the 29th International Joint Conferences on Artificial Intelligence (**IJCAI'20**), pp. 31-38, 2020. [147]
- **Yanchen Deng**, Runsheng Yu, Xinrun Wang and Bo An. Neural regret matching for distributed constraint optimization problems. Proceedings of the 30th International Joint Conference on Artificial Intelligence (**IJCAI'21**), pp.146-153, 2021. [159]
- **Yanchen Deng**, Shufeng Kong and Bo An. Pretrained cost model for distributed constraint optimization problems. Proceedings of the 36th AAAI Conference on Artificial Intelligence (**AAAI'22**), pp.9331-9340, 2022. [170]
- **Yanchen Deng**, Shufeng Kong, Caihua Liu and Bo An. Deep attentive belief propagation: Integrating reasoning and learning for solving constraint optimization problems. Proceedings of 36th Conference on Neural Information Processing Systems (**NeurIPS'22**), pp.25436-25449, 2022. [186]

Journal Article

- **Yanchen Deng** and Bo An. Utility distribution matters: Enabling fast belief propagation for multi-agent optimization with dense local utility function. Autonomous Agents and Multi-Agent Systems, Vol.35, No.2, Article 24, 2021. [148]

Bibliography

[1] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018. 1, 7

[2] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005. 1, 7, 10

[3] Tânia L Monteiro, Guy Pujolle, Marcelo E Pellenz, Manoel C Penna, and Richard Demo Souza. A multi-agent approach to optimal channel assignment in WLANs. In *WCNC*, pages 2637–2642, 2012. 1

[4] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology*, 8(5):69:1–69:27, 2017. 1, 2, 15, 52, 63, 131

[5] Shijie Li, Rudy R. Negenborn, and Gabriël Lodewijks. Distributed constraint optimization for addressing vessel rotation planning problems. *Engineering Applications of Artificial Intelligence*, 48:159–172, 2016. 1

[6] K Hirayama, K Miyake, T Shiotani, and T Okimoto. DSSA+: Distributed collision avoidance algorithm in an environment where both course and speed changes are allowed. *International Journal on Marine Navigation and Safety of Sea Transportation*, 13(1):117–123, 2019. 1, 2

[7] RT Maheswaran, M Tambe, E Bowring, JP Pearce, and P Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS*, pages 310–317, 2004. 1

[8] Rajiv T Maheswaran, Jonathan P Pearce, Emma Bowring, Pradeep Varakantham, and Milind Tambe. Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems*, 13:27–60, 2006.

[9] Roie Zivan, Tomer Parash, Liel Cohen, Hilla Peled, and Steven Okamoto. Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems*, 31(5):1165–1207, 2017. 1, 13, 21, 42, 81, 101, 108, 110

[10] Roie Zivan, Harel Yedidsion, Steven Okamoto, Robin Glinton, and Katia Sycara. Distributed constraint optimization for teams of mobile sensing agents. *Autonomous Agents and Multi-Agent Systems*, 29:495–536, 2015. 1

[11] Arseni Pertzovskiy, Roie Zivan, and Noa Agmon. CAMS: Collision avoiding max-sum for mobile sensor teams. In *AAMAS*, pages 104–112, 2023. 1

[12] Sarvapali D Ramchurn, Alessandro Farinelli, Kathryn S Macarthur, and Nicholas R Jennings. Decentralized coordination in robocup rescue. *The Computer Journal*, 53(9):1447–1461, 2010. 1

[13] Kathryn Macarthur, Ruben Stranders, Sarvapali Ramchurn, and Nicholas Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *AAAI*, pages 701–706, 2011. 3, 14, 21, 110

[14] Sofia Amador, Steven Okamoto, and Roie Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In *AAAI*, pages 1384–1390, 2014. 1

[15] Pierre Rust, Gauthier Picard, and Fano Ramparany. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *IJCAI*, pages 468–474, 2016. 1

[16] Ferdinando Fioretto, William Yeoh, and Enrico Pontelli. A multiagent system approach to scheduling devices in smart homes. In *AAMAS*, pages 981–989, 2017.

[17] William Kluegel, Muhammad A Iqbal, Ferdinando Fioretto, William Yeoh, and Enrico Pontelli. A realistic dataset for the smart home device scheduling problem for DCOPs. In *AAMAS*, pages 125–142, 2017. 1

[18] Pritee Agrawal, Akshat Kumar, and Pradeep Varakantham. Near-optimal decentralized power supply restoration in smart grids. In *AAMAS*, pages 1275–1283, 2015. 1

[19] Ferdinando Fioretto, William Yeoh, Enrico Pontelli, Ye Ma, and Satishkumar J Ranade. A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response. In *AAMAS*, pages 999–1007, 2017. 1

[20] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *CP*, pages 222–236, 1997. 1, 10

[21] Amir Gershman, Amnon Meisels, and Roie Zivan. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, 34:61–88, 2009. 10

[22] William Yeoh, Ariel Felner, and Sven Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, 38:85–133, 2010. 10

[23] Arnon Netzer, Alon Grubshtein, and Amnon Meisels. Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence*, 193:186–216, 2012. 10, 53

[24] Omer Litov and Amnon Meisels. Forward bounding on pseudo-trees for DCOPs and ADCOPs. *Artificial Intelligence*, 252:83–99, 2017. 1, 10

[25] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005. 1, 10, 100, 102, 131

[26] Adrian Petcu and Boi Faltings. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In *IJCAI*, pages 1452–1457, 2007. 11

[27] Ziyu Chen, Wenxin Zhang, Yanchen Deng, Dingding Chen, and Qiang Li. RMB-DPOP: Refining MB-DPOP by reducing redundant inference. In *AAMAS*, pages 249–257, 2020. 1, 11

[28] Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *ISCA PDCS*, pages 432–439, 2004. 1, 11, 81

[29] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, 2005. 11, 81, 101

[30] Hideaki Katagishi and Jonathan P Pearce. KOPT: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In *DCR Workshop*, 2007. 12

[31] Jonathan P. Pearce and Milind Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI*, pages 1446–1451, 2007. 12

[32] Steven Okamoto, Roie Zivan, and Aviv Nahon. Distributed breakout: Beyond satisfaction. In *IJCAI*, pages 447–453, 2016. 1, 12, 81, 101

[33] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, pages 639–646, 2008. 1, 12, 21, 24, 43, 63, 107, 110, 111

[34] Alex Rogers, Alessandro Farinelli, Ruben Stranders, and Nicholas R Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759, 2011. 13, 42, 108, 110

[35] Ziyu Chen, Yanchen Deng, Tengfei Wu, and Zhongshi He. A class of iterative refined max-sum algorithms via non-consecutive value propagation strategies. *Autonomous Agents and Multi-Agent Systems*, 32(6):822–860, 2018. 13, 21, 42

[36] Liel Cohen, Rotem Galiki, and Roie Zivan. Governing convergence of max-sum on DCOPs through damping and splitting. *Artificial Intelligence*, 279: 103212, 2020. 1, 2, 13, 42, 108, 110, 112, 121, 122, 127, 131

[37] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. Distributed Gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research*, 64:705–748, 2019. 2, 15, 81, 83

[38] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, 6(6):721–741, 1984. 2, 15

[39] Roie Zivan, Steven Okamoto, and Hilla Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212: 1–26, 2014. 2, 12, 13, 120

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1106–1114, 2012. 2

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 2, 90

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019. 3

[43] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020. 3

[44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 3, 16, 63, 89

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3

[46] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020. 3

[47] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020. 3

[48] Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. In *IJCAI*, pages 4348–4355, 2021. 3

[49] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. 3, 87

[50] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, pages 6348–6358, 2017. 3

[51] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR*, 2018. 3, 16

[52] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *NeurIPS*, pages 537–546, 2018. 3, 16

[53] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *ICML*, pages 134–144, 2020. 3

[54] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *NeurIPS*, pages 15554–15566, 2019. 3, 87

[55] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020. 3, 16, 18

[56] Daniel Selsam, Matthew Lamm, B Benedikt, Percy Liang, Leonardo de Moura, David L Dill, et al. Learning a SAT solver from single-bit supervision. In *ICLR*, 2018. 3, 16, 89

[57] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for Boolean satisfiability. In *NeurIPS*, pages 7990–8001, 2019. 3

[58] Ruben Stranders, Alessandro Farinelli, Alex Rogers, and Nicholas R. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *IJCAI*, pages 299–304, 2009. 3, 14, 21

[59] Ziyu Chen, Xingqiong Jiang, Yanchen Deng, Dingding Chen, and Zhongshi He. A generic approach to accelerating belief propagation based incomplete algorithms for DCOPs via a branch-and-bound technique. In *AAAI*, pages 6038–6045, 2019. 3, 14, 21, 26, 46, 52

[60] Md Mosaddek Khan, Long Tran-Thanh, and Nicholas R Jennings. A generic domain pruning technique for GDL-based DCOP algorithms in cooperative multi-agent systems. In *AAMAS*, pages 1595–1603, 2018. 3, 14, 21, 25, 53

[61] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000. 4, 64, 65

[62] Archie C. Chapman, Alex Rogers, Nicholas R. Jennings, and David S. Leslie. A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems. *Knowledge Engineering Review*, 26(4):411–444, 2011. 4, 11, 65, 81

[63] Philippe Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *AAAI*, pages 731–736, 1993. 4, 88, 92

[64] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 4, 88, 89, 108

[65] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014. 4, 108

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. 5, 16, 90, 109, 115, 128

[67] Eugene C Freuder and Michael J Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *IJCAI*, pages 1076–1078, 1985. 8, 92

[68] Frank R Kschischang, Brendan J Frey, Hans-Andrea Loeliger, et al. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001. 8, 12, 21, 107

[69] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003. ISBN 978-1-55860-890-0. 9

[70] Bozhena Bidyuk and Rina Dechter. On finding minimal w-cutset. In *UAI*, pages 43–50, 2004. 9

[71] Anton Chechetka and Katia Sycara. No-commitment branch and bound search for distributed constraint optimization. In *AAMAS*, pages 1427–1429, 2006. 10

[72] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Learning in Graphical Models*, volume 89 of *NATO ASI Series*, pages 75–104. Springer, 1998. 10, 89

[73] Adrian Petcu and Boi Faltings. ODPOP: An algorithm for open/distributed constraint optimization. In *AAAI*, pages 703–708, 2006. 11

[74] Meritxell Vinyals, Juan A Rodriguez-Aguilar, and Jesús Cerquides. Generalizing DPOP: Action-GDL, a new complete algorithm for DCOPs. In *AAMAS*, pages 1239–1240, 2009. 11

[75] James Atlas, Matt Warner, and Keith Decker. A memory bounded hybrid approach to distributed constraint optimization. In *DCR*, pages 37–51, 2008. 11

[76] Adrian Petcu and Boi Faltings. Approximations in distributed optimization. In *CP*, pages 802–806, 2005. 11

[77] Yoonheui Kim and Victor Lesser. DJAO: A communication-constrained DCOP algorithm that combines features of ADOPT and Action-GDL. In *AAAI*, pages 2680–2687, 2014. 11

[78] Mark Paskin, Carlos Guestrin, and Jim McFadden. A robust architecture for distributed inference in sensor networks. In *IPSN*, pages 55–62, 2005. 11

[79] Yanchen Deng, Ziyu Chen, Dingding Chen, Xingqiong Jiang, and Qiang Li. PT-ISABB: A hybrid tree-based complete algorithm to solve asymmetric distributed constraint optimization problems. In *AAMAS*, pages 1506–1514, 2019. 11, 104

[80] Tal Grinshpoun, Alon Grubshtein, Roie Zivan, Arnon Netzer, and Amnon Meisels. Asymmetric distributed constraint optimization problems. *Journal of Artificial Intelligence Research*, 47:613–647, 2013. 11, 105

[81] Dingding Chen, Yanchen Deng, Ziyu Chen, Wenxin Zhang, and Zhongshi He. HS-CAI: A hybrid DCOP algorithm via combining search with context-based inference. In *AAAI*, pages 7087–7094, 2020. 11

[82] Khoi D Hoang, Ferdinando Fioretto, William Yeoh, Enrico Pontelli, and Roie Zivan. A large neighboring search schema for multi-agent optimization. In *CP*, pages 688–706, 2018. 12, 88, 100, 101, 121

[83] Tal Grinshpoun, Tamir Tassa, Vadim Levit, and Roie Zivan. Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. *Artificial Intelligence*, 266:27–50, 2019. 12, 101

[84] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–73, 1996. 12, 111, 118

[85] Katsutoshi Hirayama and Makoto Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161(1-2):89–115, 2005. 12

[86] Makoto Yokoo, Edmund H Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering*, 10(5):673–685, 1998. 12

[87] Zhepeng Yu, Ziyu Chen, Jingyuan He, and Yanchen Deng. A partial decision scheme for local search algorithms for distributed constraint optimization problems. In *AAMAS*, pages 187–194, 2017. 12

[88] Srinivas M Aji and Robert J McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000. 12, 21, 107

[89] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan kaufmann, 1988. 12, 107

[90] David JC MacKay and David JC Mac Kay. *Information theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. 12, 107

[91] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005. 12, 107, 128, 132

[92] Emma Rollon and Javier Larrosa. Improved bounded max-sum for distributed constraint optimization. In *CP*, pages 624–632, 2012. 13, 108, 110

[93] Emma Rollon and Javier Larrosa. Decomposing utility functions in bounded max-sum for distributed constraint optimization. In *CP*, pages 646–654, 2014. 13, 108, 110

[94] Yoonheui Kim and Victor Lesser. Improved max-sum algorithm for DCOP with n-ary constraints. In *AAMAS*, pages 191–198, 2013. 14

[95] Daniel Tarlow, Inmar Givoni, and Richard Zemel. HOP-MAP: Efficient message passing with high order potentials. In *AISTAT*, pages 812–819, 2010. 14, 53

[96] Inmar E Givoni and Brendan J Frey. A binary variable model for affinity propagation. *Neural computation*, 21(6):1589–1600, 2009. 14

[97] Nikos Komodakis and Nikos Paragios. Beyond pairwise energies: Efficient optimization for higher-order mrfs. In *CVPR*, pages 2985–2992, 2009.

[98] Marc Pujol-Gonzalez, Jesus Cerquides, Pedro Meseguer, Juan Antonio Rodríguez-Aguilar, and Milind Tambe. Engineering the decentralized coordination of UAVs with limited communication range. In *CAEPIA*, pages 199–208, 2013. 14

[99] Ziyu Chen, Tengfei Wu, Yanchen Deng, and Cheng Zhang. An ant-based algorithm to solve distributed constraint optimization problems. In *AAAI*, pages 4654–4661, 2018. 15

[100] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006. 15

[101] Saaduddin Mahmud, Moumita Choudhury, Md Mosaddek Khan, Long Tran-Thanh, and Nicholas R Jennings. AED: An anytime evolutionary DCOP algorithm. In *AAMAS*, pages 825–833, 2020. 15

[102] Ziyu Chen, Lizhen Liu, Jingyuan He, and Zhepeng Yu. A genetic algorithm based framework for local search algorithms for distributed constraint optimization problems. *Autonomous Agents and Multi-Agent Systems*, 34(2): 1–31, 2020. 15

[103] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. 15

[104] Saaduddin Mahmud, Md Mosaddek Khan, Moumita Choudhury, Long Tran-Thanh, and Nicholas R Jennings. Learning optimal temperature region for solving mixed integer functional DCOPs. In *IJCAI*, pages 268–275, 2021. 15

[105] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., 1989. 15

[106] Dirk P Kroese, Thomas Taimre, and Zdravko I Botev. *Handbook of Monte Carlo methods*. John Wiley & Sons, 2013. 15

[107] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NeurIPS*, pages 2692–2700, 2015. 16

[108] William J Cook, David L Applegate, Robert E Bixby, and Vasek Chvatal. *The traveling salesman problem: a computational study*. Princeton university press, 2011. 16

[109] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016. 16

[110] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 16

[111] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, pages 6348–6358, 2017. 16

[112] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, pages 2702–2711, 2016. 16

[113] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pages 3837–3845, 2016. 16

[114] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. 16, 89

[115] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *NeurIPS*, pages 21188–21198, 2020. 16

[116] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, pages 25531–25546, 2022. 16

[117] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *ICLR*, 2019. 16

[118] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *NeurIPS*, pages 6278–6289, 2019. 17

[119] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *ICLR*, 2019. 17

[120] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *NeurIPS*, pages 26198–26211, 2021. 17

[121] Hanni Cheng, Haosi Zheng, Ya Cong, Weihao Jiang, and Shiliang Pu. Select and optimize: Learning to aolve large-scale TSP instances. In *AISTATS*, pages 1219–1231, 2023. 17

[122] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP*, pages 417–431, 1998. 17, 121

[123] André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *ICLR*, 2021. 17

[124] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *NeurIPS*, pages 3483–3491, 2015. 17

[125] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *ECAI*, volume 325, pages 443–450, 2020. 17

[126] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *AAAI*, pages 7474–7482, 2021. 17

[127] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012. 17

[128] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9):5057–5069, 2021. 17

[129] Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. In *ICLR*, 2022. 17

[130] Chris Voudouris and Edward Tsang. Partial constraint satisfaction problems and guided local search. In *PACT*, pages 337–356, 1996. 17

[131] Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009. 18

[132] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *AAAI*, pages 724–731, 2016. 18

[133] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Finding cuts in the TSP (A preliminary report). Technical report, AT&T Bell Labs, 1995. 18

[134] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *NeurIPS*, pages 15554–15566, 2019. 18

[135] Prateek Gupta, Maxime Gasse, Elias B. Khalil, Pawan Kumar Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. In *NeurIPS*, 2020. 18

[136] Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *ICLR*, 2023. 18

[137] Defeng Liu, Matteo Fischetti, and Andrea Lodi. Learning to search in local branching. In *AAAI*, pages 3796–3803, 2022.

[138] Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. In *Advances in Neural Information Processing Systems*, pages 30075–30087, 2021. 18

[139] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. In *NeurIPS*, pages 7990–8001, 2019. 18, 89

[140] Bart Selman, Henry A Kautz, Bram Cohen, et al. Noise strategies for improving local search. In *AAAI*, pages 337–343, 1994. 18

[141] Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can Q-Learning with graph networks learn a generalizable branching heuristic for a SAT solver? *NeurIPS*, pages 9608–9621, 2020. 18, 89

[142] Joao P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. 18, 89

[143] Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. NLocalSAT: Boosting local search with solution prediction. In *IJCAI*, pages 1177–1183, 2021. 18

[144] Wen Song, Zhiguang Cao, Jie Zhang, Chi Xu, and Andrew Lim. Learning variable ordering heuristics for solving constraint satisfaction problems. *Engineering Applications of Artificial Intelligence*, 109:104603, 2022. 18

[145] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. ISBN 978-0-444-52726-4. 18

[146] Yoonheui Kim, Michael Krainin, and Victor Lesser. Effective variants of the max-sum algorithm for radar coordination and scheduling. In *WI/IAT*, pages 357–364, 2011. 21, 22, 52, 110

[147] Yanchen Deng and Bo An. Speeding up incomplete GDL-based algorithms for multi-agent optimization with dense local utilities. In *IJCAI*, pages 31–38, 2020. 22, 133

[148] Yanchen Deng and Bo An. Utility distribution matters: Enabling fast belief propagation for multi-agent optimization with dense local utility function. *Autonomous Agents and Multi-Agent Systems*, 35(2):24, 2021. 22, 133

[149] Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artificial intelligence*, 171(2-3):73–106, 2007. 27

[150] Radu Marinescu and Rina Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173 (16-17):1457–1491, 2009. 27

[151] Yair Weiss and William T Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, 2001. 41

[152] Jorge E Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National academy of Sciences*, 102(46):16569–16572, 2005. 47

[153] D Pepyne, D Westbrook, B Philips, E Lyons, M Zink, and J Kurose. A design for distributed collaborative adaptive sensing of the atmosphere. Technical report, University of Massachusetts, 2007. 52

[154] Evan A Sultanik, Robert N Lass, and William C Regli. DCOPolis: A framework for simulating and deploying distributed constraint reasoning algorithms. In *AAMAS*, pages 1667–1668, 2008. 53, 83, 104

[155] Roie Zivan, Omer Lev, and Rotem Galiki. Beyond trees: Analysis and convergence of belief propagation in graphs with multiple cycles. In *AAAI*, pages 7333–7340, 2020. 56, 108, 110, 123

[156] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018. 63

[157] OpenAI. OpenAI Five. `https://blog.openai.com/openai-five/`, 2018. 63

[158] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. 63

[159] Yanchen Deng, Runsheng Yu, Xinrun Wang, and Bo An. Neural regret-matching for distributed constraint optimization problems. In *IJCAI*, pages 146–153, 2021. 64, 133

[160] Gürdal Arslan, Jason R Marden, and Jeff S Shamma. Autonomous vehicle-target assignment: A game-theoretical formulation. *Journal of Dynamic Systems, Measurement and Control*, 129(5):584–596, 2007. 65

[161] Dustin Morrill. Using regret estimation to solve games compactly. Master's thesis, University of Alberta, 2016. 65, 74, 76

[162] Kevin Waugh, Dustin Morrill, James Andrew Bagnell, and Michael H. Bowling. Solving games with functional regret estimation. In *AAAI*, pages 2138–2145, 2015. 65, 74

[163] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *ICML*, pages 793–802, 2019. 65

[164] David Blackwell et al. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956. 66

[165] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit Texas hold'em. In *IJCAI*, pages 645–652, 2015. 69, 70, 75, 77

[166] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 81, 102, 120, 122

[167] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. 82, 101, 121

[168] Duc Thien Nguyen, William Yeoh, and Hoong Chuin Lau. Stochastic dominance in stochastic DCOPs for risk-sensitive applications. In *AAMAS*, pages 257–264, 2012. 83

[169] Gil Lederman, Markus Rabe, Sanjit Seshia, and Edward A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *ICLR*, 2020. 87

[170] Yanchen Deng, Shufeng Kong, and Bo An. Pretrained cost model for distributed constraint optimization problems. In *AAAI*, pages 9331–9340, 2022. 87, 121, 133

[171] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 89

[172] Hong Xu, Sven Koenig, and TK Satish Kumar. Towards effective deep learning for constraint satisfaction problems. In *CP*, pages 588–597, 2018. 89

[173] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In *NeurIPS*, pages 396–404, 1989. 89

[174] Yasaman Razeghi, Kalev Kask, Yadong Lu, Pierre Baldi, Sakshi Agarwal, and Rina Dechter. Deep bucket elimination. In *IJCAI*, pages 4235–4242, 2021. 89

[175] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 90

[176] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 90

[177] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 90

[178] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 90

[179] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *ICCV*, pages 9267–9276, 2019. 92

[180] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *IJCAI*, pages 572–578, 1995. 100

[181] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *ICLR*, 2016. 102

[182] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 102, 122

[183] Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *AISTATS*, pages 685–693, 2021. 108, 110

[184] Jonathan Kuck, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon. Belief propagation neural networks. In *NeurIPS*, pages 667–678, 2020. 111

[185] Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. In *NeurIPS*, pages 8577–8587, 2020. 108, 111

[186] Yanchen Deng, Shufeng Kong, Caihua Liu, and Bo An. Deep attentive belief propagation: Integrating reasoning and learning for solving constraint optimization problems. In *NeurIPS*, pages 25436–25449, 2022. 108, 133

[187] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015. 109

[188] Amir Globerson and Tommi S. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *NIPS*, pages 553–560, 2007. 110

[189] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Tree-reweighted belief propagation algorithms and approximate ML estimation by pseudo-moment matching. In *AISTATS*, pages 308–315, 2003. 110

[190] Wim Wiegerinck and Tom Heskes. Fractional belief propagation. In *NIPS*, pages 438–445, 2002. 110

[191] Tamir Hazan and Amnon Shashua. Norm-product belief propagation: Primal-dual message-passing for approximate inference. *IEEE Transactions on Information Theory*, 56(12):6294–6316, 2010. 110

[192] Akshat Kumar and Shlomo Zilberstein. MAP estimation for graphical models by likelihood maximization. In *NIPS*, pages 1180–1188, 2010. 110

[193] Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, 1999. 121

[194] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, 2003. 121

[195] S. de Givry D. Allouche and T. Schiex. Toulbar2, an open source exact cost function network solver. Technical report, INRA, 2010. 121

[196] Sara Rouhani, Tahrima Rahman, and Vibhav Gogate. A novel approach for constrained optimization in graphical models. In *NeurIPS*, pages 11949–11960, 2020. 128, 132

[197] Junwen Bai, Shufeng Kong, and Carla P Gomes. Gaussian mixture variational autoencoder with contrastive learning for multi-label classification. In *ICML*, pages 1383–1398, 2022. 128

[198] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017. 132