

基于推理的分布式约束优化问题完备求解 算法研究



重庆大学博士学位论文 (学术学位)

学生姓名：陈定定

指导教师：何中市 教授

学科门类：工 学

学科名称：计算机科学与技术

研究方向：分布式人工智能

答辩委员会主席：李传东 教授

授位时间：2022 年 9 月

The Study on Inference Based Complete Algorithms for Distributed Constraint Optimization Problems



A Thesis Submitted to Chongqing University
In Partial Fulfillment of the Requirement for the
Doctor's Degree of Engineering
By

Dingding Chen
Supervised by Prof. Zhongshi He

September, 2022

摘要

分布式约束优化问题（Distributed Constraint Optimization Problem, DCOP）是多智能体系统协作优化的基本框架，已被成功应用于分布式资源调度、传感器网络等领域。非对称分布式约束优化问题（Asymmetric Distributed Constraint Optimization Problem, ADCOP）在 DCOP 的基础上增加了智能体的私有偏好，具有更强的建模能力和更广阔的应用前景。完备求解算法保障能找到问题最优解且具有较强的理论性质，是目前 DCOP/ADCOP 求解算法研究的热点。主流的 DCOP 完备求解策略包含搜索和推理。其中，完备推理算法基于动态规划的求解思路，其仅需要线性的消息数但其内存消耗却是指数级的。因此，近似推理和上文推理分别提出通过牺牲解质量和更多的消息数以降低算法的内存消耗。完备搜索算法采用分枝定界的求解策略，其消息大小是线性级的但消息数却是指数级的。此外，这类算法的求解效率依赖于界的紧度。但现有完备搜索算法依赖于本地知识或近似推理来计算界，导致内存受限时，界的紧度较低。此外，由于 ADCOP 对私有偏好保障的要求，完备推理的 DCOP 算法无法直接用于求解 ADCOP。而现有 ADCOP 完备算法仅采用搜索策略，其求解问题规模有限且私有偏好保障性能较差。针对上述问题，本文致力于利用推理策略的优势，提升 DCOP/ADCOP 完备算法的求解性能。本文主要创新点如下：

① 提出上文推理与搜索混合的 DCOP 完备算法

率先提出将上文推理与搜索混合的求解思路应用于 DCOP 的完备求解。在内存受限情况下，上文推理依然可以提供较紧的下界，提出基于上文推理与搜索混合的 DCOP 完备算法(Hybrid Search and Context-based Inference, HS-CAI)。在 HS-CAI 中，上文推理与搜索互相辅助、并行执行。其中，搜索为上文推理提供推理所需的上文以降低迭代推理所产生的巨大计算量；上文推理为搜索提供紧的下界来加速剪枝，并减少上文推理的次数。此外，本文提出上文评估机制为上文推理筛选出上文以进一步降低迭代推理带来的计算量开销。理论和实验证明 HS-CAI 提供下界紧度优于目前性能最好的 DCOP 完备搜索算法。

② 提出基于非本地消元的推理与搜索混合的 ADCOP 算法

率先提出将推理策略用于辅助 ADCOP 的完备求解。针对 ADCOP 的非对称性质，研究如何采用推理策略辅助求解 ADCOP，提出基于推理与搜索混合的 ADCOP 完备算法（Pseudo Tree-Inference/SyncABB-1ph, PT-ISABB）。PT-ISABB 包含推理和搜索两个阶段。其中，推理阶段预先求出求解问题单面约束，为搜索阶段提供下界用于剪枝。利用推理阶段提供的下界，搜索阶段在伪树不同分支上并行执

行分支定界搜索。为提升算法提供下界的紧度，本文提出针对于非对称环境的非本地消元机制。同时，提出预估汇报机制以避免智能体私有偏好的直接泄露。此外，通过结合绝对误差机制和相对误差机制，提出两种非完备扩展版本。理论证明所提出算法的正确性，且在内存空间维度为伪树诱导宽度时，算法提供下界的紧度优于现有性能最好的 ADCOP 完备搜索算法所提供的下界。实验结果表明 PT-ISABB 在多种测试问题上均优于现有的完备搜索算法，其非完备版本可在指定误差内快速地找到一组可行解。

③ 提出基于推理的 ADCOP 完备算法

率先提出仅采用推理策略实现 ADCOP 完备求解。DCOP 完备推理算法直接求解 ADCOP 会导致私有偏好的直接泄露。受启发于研究点 2 提出的非本地消元机制，本文提出广义非本地消元机制，以确保私有约束代价函数不直接泄露的同时保证变量消元的完备性。基于该机制，提出仅采用推理策略的 ADCOP 完备算法（Asymmetric DPOP，AsymDPOP）。为解决 AsymDPOP 中推迟消元带来的内存占用以及计算量增加的问题，本文提出桶集合的传播策略和小批次的消元策略分别用于降低算法的内存占用和巨大的计算量。为使得提出的推理算法适用于内存有界的环境，本文将内存有界推理思想适配到所提出的算法中，并引入分支独立枚举机制和两阶段的效用集合传播机制来降低算法中的冗余推理。理论分析所提出算法的内存占用。实验结果表明所提出算法在多种测试问题上均优于现有的性能最好的 ADCOP 完备算法。

关键词：分布式约束优化问题；非对称分布式约束优化问题；完备算法；完备搜索；完备推理

Abstract

Distributed Constraint Optimization Problem (DCOP) is a primary framework for multi-agent system where agents aim to find assignments cooperatively to optimize a global objective. It has been successfully applied to model many real-world applications including distributed resource scheduling and sensor network. Asymmetric Distributed Constrained Optimization Problem (ADCOP) extends DCOP by attaching private preferences for agents, and thus has stronger modeling ability and broader application prospects. Complete algorithms for DCOP/ADCOP can guarantee to find the optimal solution and has strong theoretical properties, and thus has been the focus of research in this field. For complete algorithms for DCOP, search and inference are the two main strategies. The complete inference algorithms solve the problem by using dynamic programming paradigm, which only need a linear number of messages of exponential size. Therefore, the approximate inference and context-based inference proposed to reduce the message size with sacrificing the solution quality and increasing the number of messages, respectively. The complete search algorithm adopts branch and bound strategy, which has the advantage of linear memory consumption but the number of messages is exponential. In addition, the efficiency of such algorithms depends on the tightness of the provided bounds. However, the existing complete search algorithms rely on the local knowledge or use the approximate inference to compute the bounds, resulting in low efficiency when the memory is bounded. Due to the privacy concerns, the inference-based algorithms for optimally solving DCOP are not suitable for ADCOP. On the other hand, the existing complete algorithms for ADCOP only exploit search, and thus cannot solve large scale problems. Therefore, this thesis aims to use the inference strategy to improve the performance of the DCOP/ADCOP algorithm. The main contributions of this thesis are listed as follows:

- ① A complete algorithms for DCOP that combines context-based inference and search.

Taking the advantage that the context-based inference can still provide tight lower bounds under memory-bounded conditions, this thesis studies how to effectively hybrid context-based inference and search, and presents a complete algorithm for DCOP named HS-CAI (Hybrid Search and Context-based Inference). In HS-CAI, the context-based inference and search benefit from each other and are carried out

concurrently, in which the context-based inference utilizes the contexts derived from the search process to establish tight lower bounds while the search uses such bounds for efficient pruning and thereby reduces contexts for the inference. Furthermore, this thesis introduces a context evaluation mechanism to select the context patterns for the inference to further reduce the overheads incurred by iterative context-based inferences. This thesis theoretically proves that HS-CAI is correct, and the lower bounds provided by the context-based inference are tighter than the ones produced by the approximate inference under the memory-bounded conditions. The experimental results demonstrate its superiority over the state-of-the-art.

② A complete algorithm for ADCOP that hybrids inference and search and its two suboptimal variants

In this thesis, we consider the possibility of combining both inference and search to efficiently solve ADCOP at an acceptable loss of privacy. Specifically, this thesis proposes a novel hybrid scheme to solve ADCOP, which consists an inference phase and a search phase. Here, the inference phase performs bottom-up utility propagation with respect to a subset of constraints to establish lower bounds and upper bounds for the search process. While, the search phase uses a tree-based complete search algorithm for ADCOP to exhaust the search space. Further, this thesis introduces non-local Elimination mechanism for a more complete utility table and an estimation reporting mechanism to avoid the direct exposure of private constraint costs. In addition, this thesis presents two suboptimal variants of PT-ISABB based on bounded-error approximation mechanisms to enable trade-off between theoretically guaranteed solutions and coordination overheads. In this thesis, we prove that PT-ISABB and its suboptimal variants are correct, and the lower bounds in PT-ISABB are tighter than the ones in AsymPT-FB. The experimental results demonstrate that PT-ISABB exhibits great superiorities over the state-of-the-art search-based complete algorithms and its suboptimal variants can quickly find a solution within the user-specified bounded-error.

③ A class of inference-based complete algorithms for ADCOP

Inspired by the non-local elimination mechanism, this thesis proposes a generalized non-local elimination mechanism (GLNE) to void their private functions being exposed directly, so as to protect the privacy through the solving process. Based on GNLE, this thesis presents the first inference-based complete algorithm for ADCOP, named AsymDPOP (Asymmetric DPOP). To solve the severe scalability problems incurred by delayed eliminations, we propose a table-set propagation scheme to reduce the memory

consumption, and a mini-batch elimination scheme to reduce the computation efforts. Furthermore, to ensure the proposed algorithms can scale up to large-scale problems within the limited memory budget, we adapt the memory-bounded inference to AsymDPOP where a branch-independent distributed enumeration mechanism and a two-phase utility set propagation scheme are introduced to reduce the redundancy in memory-bounded inference. Finally, this thesis theoretically shows the complexity of the proposed algorithms, and the empirical evaluation indicates that our proposed methods significantly outperform the state-of-the-art.

Keywords: Distributed constrained optimization problem; Asymmetric distributed constraint optimization problem; Complete algorithms; Search-based complete algorithms; Inference-based complete algorithms

目 录

中文摘要	I
英文摘要	III
插图清单	XI
附表清单	XIII
算法清单	XV
术语和符号的注释	XVII
1 絮论	1
1.1 研究背景及意义	1
1.2 国内外研究现状	3
1.2.1 DCOP 算法研究现状	3
1.2.2 ADCOP 算法研究现状	7
1.2.3 现有 DCOP 和 ADCOP 完备算法研究面临的问题与挑战	8
1.3 论文主要研究内容和创新之处	9
1.3.1 论文主要研究内容	9
1.3.2 论文创新之处	10
1.4 论文组织结构	11
2 分布式约束优化问题研究基础	13
2.1 分布式约束优化问题	13
2.1.1 问题定义	13
2.1.2 实际问题建模	14
2.2 非对称分布式约束优化问题	17
2.3 算法的通信结构	19
2.4 分布式约束优化问题实验	20
2.4.1 实验测试问题	20
2.4.2 算法评价指标	21
2.5 本章小结	22
3 基于上文推理与搜索混合的 DCOP 完备算法	23
3.1 引言	23
3.2 算法背景	23
3.2.1 分布式伪树优化过程及其近似版本	24
3.2.2 同步分支定界算法	25

3.3 基于上文推理与搜索混合的 DCOP 完备算法	26
3.3.1 预处理阶段	26
3.3.2 上文推理与搜索混合阶段	28
3.4 理论分析	32
3.4.1 提供下界紧度分析	32
3.4.2 完备性证明	33
3.4.3 复杂度分析	34
3.5 实验评估	35
3.5.1 参数调优	36
3.5.2 实验结果与分析	36
3.6 本章小结	43
4 基于非本地消元的推理与搜索混合的 ADCOP 算法	45
4.1 引言	45
4.2 基于非本地消元的推理与搜索混合的 ADCOP 完备算法	46
4.2.1 基于非本地消元的推理阶段	46
4.2.2 搜索阶段	48
4.2.3 私有偏好泄露分析	55
4.3 理论分析	55
4.3.1 提供下界紧度分析	55
4.3.2 完备性证明	56
4.3.3 复杂度分析	59
4.4 基于非本地消元的推理与搜索混合的 ADCOP 非完备算法	60
4.4.1 绝对误差机制	61
4.4.2 相对误差机制	62
4.5 实验评估	62
4.5.1 参数调优	63
4.5.2 完备算法的性能比较	65
4.5.3 非完备算法的性能比较	72
4.6 本章小结	74
5 基于广义非本地消元的 ADCOP 完备推理算法	77
5.1 引言	77
5.2 算法背景	78
5.3 非对称伪树优化过程	79
5.3.1 基于广义非本地消元的效用传播阶段	79

5.3.2 值传播阶段	81
5.3.3 复杂度分析	82
5.4 TSPS 与 MBES 两种权衡机制	84
5.4.1 桶集合传播机制	84
5.4.2 小批次消元机制	89
5.5 两种内存有界的 ADCOP 完备算法	91
5.5.1 内存有界的 AsymDPOP	91
5.5.2 带桶集合传播机制的内存有界 AsymDPOP	96
5.6 私有偏好泄露分析	103
5.7 实验评估	104
5.7.1 参数调优	104
5.7.2 实验结果与分析	106
5.8 本章小结	113
6 总结与展望	115
6.1 本文工作总结	115
6.2 未来研究展望	116
参考文献	117
附 录	131
A. 作者在攻读学位期间发表的论文目录	131
B. 作者在攻读学位期间取得的科研成果目录	131
C. 学位论文数据集	132
致 谢	133

插图清单

序号	图号	图题	页码
1	图 1.1	DCOP 算法分类	3
2	图 1.2	ADCOP 算法分类	7
3	图 1.3	论文框架结构图	11
4	图 2.1	DCOP 实例	13
5	图 2.2	目标跟踪实例	14
6	图 2.3	目标跟踪问题的 DCOP 建模	15
7	图 2.4	WLAN 网络部署实例	16
8	图 2.5	非对称约束的实例	18
9	图 2.6	ADCOP 实例	18
10	图 2.7	伪树结构实例	19
11	图 3.1	说明 HS-CAI 算法运行的 DCOP 实例	27
12	图 3.2	HS-CAI 混合阶段的流程图	29
13	图 3.3	不同密度下变化参数 ρ , HS-CAI 算法产生的网络负载对比	36
14	图 3.4	不同智能体数(稀疏配置)下各个算法性能对比	37
15	图 3.5	不同智能体数(稠密配置)下各个算法性能对比	38
16	图 3.6	不同密度下各个算法性能对比	39
17	图 3.7	不同值域下各个算法性能对比	40
22	图 3.8	各个算法在求解无尺度网络问题时的性能对比	42
23	图 3.9	各个算法在求解分布式信道分配问题时的性能对比	43
24	图 4.1	说明 PT-ISABB 算法运行的 ADCOP 实例	45
25	图 4.2	PT-ISABB 搜索阶段的流程图	51
26	图 4.3	不同智能体数下各个算法性能对比	65
27	图 4.4	不同密度下各个算法性能对比	66
28	图 4.5	各个算法在求解无尺度网络问题时的性能对比	67
29	图 4.6	各个算法在求解会议调度问题时的性能对比	68
30	图 4.7	各个算法在求解不同紧度下非对称 MaxDCSP 问题时的性能对比	69
31	图 4.8	各个算法在求解不同紧度下非对称 MaxDCSP 问题时的私有偏好	70

泄露

32	图 4.9 不同相对误差下, 各个算法在求解稀疏和稠密问题时解的归一化 代价值	71
33	图 4.10 不同相对误差下, 各个算法在求解稀疏和稠密问题时的性能对比	73
34	图 5.1 说明 ADCOP 完备推理算法运行的 ADCOP 实例	75
35	图 5.2 链式结构的伪树	83
36	图 5.3 带 TSPS 的 AsymDPOP 流程图	84
37	图 5.4 效用表集合	87
38	图 5.5 RMB-AsymDPOP 的流程图	91
39	图 5.6 带 TSPS 的 RMB-AsymDPOP 预处理阶段流程图	95
40	图 5.7 带 TSPS 的 RMB-AsymDPOP 效用集合传播阶段的流程图	97
41	图 5.8 不同智能体数下 AsymDPOP 以及在不同 k_p 和 k_e 下带 TSPS 和 MBES 的 AsymDPOP 性能对比	103
42	图 5.9 不同智能体数下在不同 k_{mb} 下 RMB-AsymDPOP 以及在不同 k_{mb} 和 k_p 下带 TSPS 的 RMB-AsymDPOP 性能对比	104
43	图 5.10 不同智能体数下各个算法的性能对比	105
44	图 5.11 不同密度下各个算法的性能对比	106
45	图 5.12 不同值域下各个算法的性能对比	107
46	图 5.13 各个算法在求解无尺度网络问题时的性能对比	108
47	图 5.14 各个算法在求解会议调度问题时的性能对比	109
48	图 5.15 各个算法在求解不同紧度下非对称 MaxDCSP 问题时的性能对比	111

附表清单

序号	表号	表题	页码
1	表 2.1	信道分配的代价函数	17
2	表 4.1	四种版本 PT-ISABB 的具体配置	67
3	表 4.2	PT-ISABB 四种变体在 $k_{mb} = 4$ 的性能比较	67
4	表 4.3	PT-ISABB 四种变体在 $k_{mb} = \infty$ 的性能比较	68

算法清单

序号	图号	算法题目	页码
1	算法 3.1	HS-CAI 预处理阶段	26-27
2	算法 3.2	HS-CAI 混合阶段	30-31
3	算法 4.1	PT-ISABB 推理阶段	47-48
4	算法 4.2	PT-ISABB 搜索阶段	51-54
5	算法 5.1	AsymDPOP 效用传播阶段	79
6	算法 5.2	AsymDPOP 值传播阶段	80
7	算法 5.3	带 TSPS 的 AsymDPOP	84-85
8	算法 5.4	MBES	88
9	算法 5.5	RMB-AsymDPOP	92-94
10	算法 5.6	带 TSPS 的 RMB-AsymDPOP 预处理阶段	95-96
11	算法 5.7	带 TSPS 的 RMB-AsymDPOP 效用表集合传播阶段	98-101

术语和符号的注释

1. 术语

缩写	全称
DCOP	分布式约束优化问题 (Distributed Constraint Optimization Problem)
ADCOP	非对称分布式约束优化问题 (Asymmetric Distributed Constraint Optimization Problem)
DPOP	分布式伪树优化过程 (Distributed Pseudo tree Optimization Procedure)
ADPOP	近似分布式伪树优化过程 (Approximate DPOP)
MB-DPOP	内存有界的分布式伪树优化过程 (Memory-bounded DPOP)
RMB-DPOP	优化的内存有界分布式伪树优化过程 (Refined MB-DPOP)
SyncBB	同步分支定界算法 (Synchronous Branch and Bound)
TreeBB	基于树型结构的同步分支定界算法(Tree-based SyncBB)
SyncABB-1	一阶段的非对称同步分支定界算法 (Asymmetric version of SyncBB with phase)
	One Phase Check)
NCLOs	非并发约束检查 (Non-concurrent Constraint Checks)
TSPS	桶集合传播机制 (Table Set Propagation Scheme)
MBES	小批次消元机制 (Mini-Batch Elimination Scheme)
ISM	迭代选择机制 (Iterative Selection Mechanism)
DEM	分布式枚举机制 (Distributed Enumeration Mechanism)

2. 符号

符号名	符号全称
a_i	标号为 i 的智能体
x_i	a_i 所控制的变量
d_i	x_i 对应的赋值
$N(a_i)$	a_i 的邻居节点
$P(a_i)$	a_i 的父节点
$PP(a_i)$	a_i 的伪父节点
$C(a_i)$	a_i 的孩子节点
$PC(a_i)$	a_i 的伪孩子节点
$AP(a_i)$	a_i 的所有父节点
$AC(a_i)$	a_i 的所有孩子节点

符号名	符号全称
$Anc(a_i)$	a_i 的祖先节点
$Desc(a_i)$	a_i 的子孙节点：
$Sep(a_i)$	a_i 的分割点集
f_{ij}	由 a_i 所持有的 a_i 与其邻居 a_j 的约束代价函数
w^*	伪树的诱导宽度
$util_{c \rightarrow i}$	孩子节点 $a_c \in C(a_i)$ 向 a_i 发送的效用表
$util_{i \rightarrow p}$	a_i 向其父节点发送的效用表
$utilSet_{c \rightarrow i}$	孩子节点 $a_c \in C(a_i)$ 向 a_i 发送的效用表集合
$utilSet_{i \rightarrow p}$	a_i 向其父节点发送的效用表集合
Cpa_i	a_i 的分割点集的赋值组合
\min	效用表的最小化消元操作
\max	效用表的最大化消元操作
\perp	效用表的最小、最大化消元操作
argmin	求变量的最优（最小化）赋值操作
\otimes	效用表或约束代价函数的联合操作

1 绪论

1.1 研究背景及意义

智能体（Agent）（例如软件程序或机器人）是一种自治的计算实体，它可以通过传感器感知环境、效应器作用于环境，并且在某种程度上能够自主行为来达到其设计的目标^[1]。即：智能体具有感知性、适应性、自治性、主动性和协作性。多智能体系统（Multi-Agent Systems, MAS）^[2-5]是由多个可以互相交互的自治智能体所组成的计算系统。其目的在于解决大规模、复杂、实时和有不确定信息的现实问题，而这类问题一般是单个智能体所不能解决的。多智能体系统通常具有自主性、分布性、协调性等特征，并具有自组织能力、学习能力和推理能力。在多智能体系统中，智能体可以相互协作去达成一个共同目标，也可以相互竞争来使得个体利益最大化。由于可以广泛用于建模那些信息和控制分布在多个智能体间的环境，多智能体系统已成为分布式人工智能的重要分支。多智能体系统体现人类的社会智能，其本身也是决策理论、博弈论及约束规划等领域的研究热点^[6]。

分布式约束优化问题（Distributed Constraint Optimization Problem, DCOP）^[7,8]是多智能体系统协作优化问题的重要框架，是解决分布式智能系统建模和协同优化的有效技术，具有重要的研究意义和实用价值。DCOP 利用约束规划思想，建立多智能体之间的决策约束关系并以联合决策产生的代价（即约束函数）之和为优化目标协调各智能体的决策行为。DCOP 可以看作是约束满足问题（Constraint Satisfaction Problems, CSP）在分布式环境下引入软约束的优化模型，其强调利用本地局部交互获得全局最优。不同于集中式的优化模型（例如 CSP），其需要智能体将其本地子问题传达给一个中心节点，中心节点应用集中式的求解算法找到问题的一组解。在 DCOP 中，智能体保有对本地子问题的控制权和自主决策权，不需要将自身的子问题都集中在某一个节点上。具体地，相比集中式的优化模型，DCOP 建模的优势体现在如下几方面：

① 问题的无界性：由于智能体本地问题规模或应用场景限制，无法将整个问题信息集中在某个中心计算节点实现问题求解。将智能体本地私有信息集中到某个中心节点所花费的成本远大于集中式求解所带来的收益。例如，在供应链场景下，零件供应商需要预先计算并发送其提供的不同类型产品的交货日期、价格和数量的所有组合。这种子问题信息往往是巨大的，且存在实时性和动态性。

② 智能体私有信息的保障：传统集中式求解要求智能体将其私有本地信息集中到求解器，从而在求解过程中容易受到攻击、贿赂等问题。例如，在会议调度问题中，智能体（参会人）不希望自己的参会行程透漏给其他人。

③ 时延性：在动态环境中，智能体可能随时进入或离开、改变其本地问题信息等。集中式求解该问题时，每次问题局部的变化都会使得中心计算节点重新计算，然后将计算结果转发给每个智能体。若变化发生的比较频繁时，这一过程将引入巨大的时延开销。在分布式环境下，智能体之间可以局部地调整、处理小的局部变化，从而能快速地针对环境的改变做出决策。

④ 性能瓶颈：采用集中式求解问题时，所有智能体都处于空闲状态，等待中心计算节点的计算结果。中心节点必须拥有所有的计算和存储资源（例如 CPU、内存）来求解问题，这使得中心计算节点的计算和存储能力成为系统性能的瓶颈。在分布式环境下，每个智能体都参与计算，因此能更充分地利用多智能体系统的并行计算优势。

⑤ 鲁棒性：集中式计算过程中采用单个计算节点进行计算时，该计算节点可能由于各种原因（例如电源、处理器故障、网络连接问题、DOS 攻击等）而脱机。而在分布式场景下，单个智能体的故障只会影响其附近的少数智能体。

DCOP 由五元组定义：智能体集合、变量集合、值域集合以及约束关系集合。其中，每个智能体控制一个或多个变量；每个变量仅由一个智能体控制且对应一个有限且离散的值域，变量的值域中每一个赋值代表其可能所处的状态；约束关系描述一组相关联变量之间的关系，即：这组变量不同赋值组合下对应的约束代价值。一般而言，每个智能体仅能知道关于自己所控制变量上的约束关系，且智能体需要通过互相协作为其所控制的变量赋值从而使得全局函数最优（如使得所有约束代价函数之和最小）。因此，DCOP 具有更强的容错性和更高的并行度，能够对多智能体系统领域中很多实际问题进行建模。目前已经广泛应用于微电网控制^[9,10]、资源分配^[11,12]、分布式调度^[13,14]和传感器网络^[15,16]等领域。

然而，DCOP 的对称性限制其在工程中的实际应用，尤其是无法对含有个体偏好的问题建模。这里，DCOP 的对称性指智能体与其有约束关系智能体的特征、值域空间及约束代价函数有完全且准确的信息。也就是说，约束关系在值域空间上的收益（代价）对于被约束各智能体是“共享知识”，每个智能体没有个体偏好特征并且约束双方之间无法具有私有信息。因此，对于非对称或具有偏好特征的问题，采用 DCOP 建模和求解将无法满足实际需求^[17,18]。例如，在护士排班问题中，护士对于不同日程安排有自身的偏好；在飞机时间槽调度问题中，不同航空公司对于同一时间槽有不同偏好。这种偏好信息表明公司计划飞行的航线，是一种敏感的战略信息，因此不希望将其泄露给竞争对手。

非对称分布式约束优化问题（Asymmetric Distributed Constraint Optimization Problem, ADCOP）^[19]是在 DCOP 的基础上增加非对称特性的新模型。具体地说，ADCOP 中同一个约束代价函数为约束各方所给出的代价不同。因此，ADCOP 可

以较好地表征智能体的个体偏好，能更好地适应实际的工程需要。但是，由于在 ADCOP 中智能体的偏好是私有的，给求解算法带来挑战，例如：更复杂的搜索空间、求解时的私密性保障等。

1.2 国内外研究现状

1.2.1 DCOP 算法研究现状

依据是否保证获得问题最优解，DCOP 算法可为完备算法和非完备算法，如图 1.1 所示。完备算法保证获得问题最优解，其依据算法控制的角度又可分为部分集中和完全分散。基于部分集中的 DCOP 算法包括 Opt-APO^[20,21] 和 PC-DPOP^[22]，其求解思路是：部分智能体将其约束信息传播到一些中心节点，随后中心节点采用集中式的方式实现问题的求解。由于中心节点需要收集其他智能体的约束代价函数信息，且在求解过程中仅有中心节点在执行计算。这样将导致部分智能体的局部信息泄露，且无法充分利用多智能体系统并行计算的优势。因此，完全分散式的完备算法是研究的重点。依据求解策略的不同，完全分散式的完备算法可分为完备搜索算法和完备推理算法。依据算法求解过程中采用通信结构的不同，完备搜索算法又可分为基于链式结构的求解算法和基于伪树结构的求解算法。

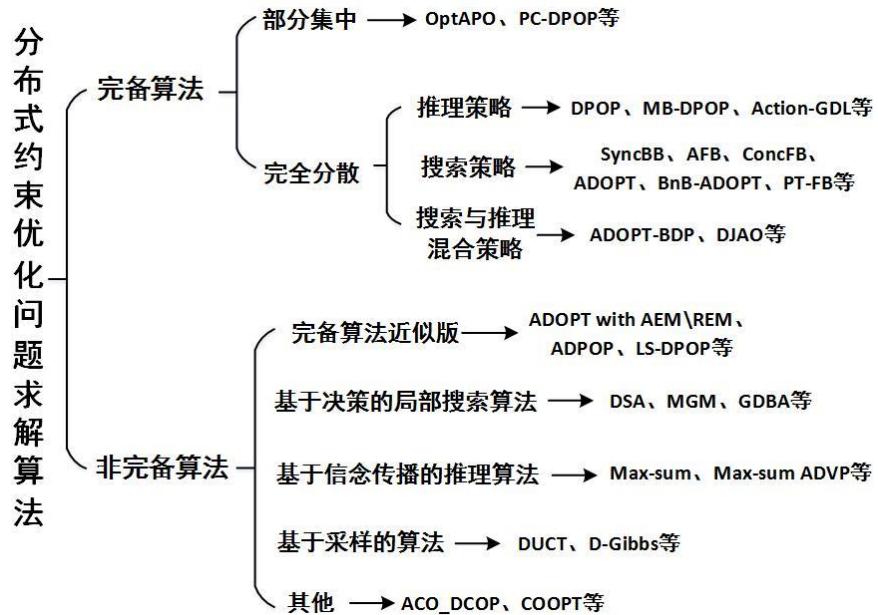


图 1.1 DCOP 算法分类

Fig.1.1 The taxonomy of DCOP algorithms

同步分支定界算法（Synchronous Branch and Bound, SyncBB）^[23-25]是早期采用链式结构的完备搜索算法，其是分支定界策略的分布式实现，包含分支和定界

两部分。其中，分支部分通过智能体之间的消息传递执行部分解的扩展和部分解对应代价值的累积，最终实现对子问题的划分；定界部分为：通过比较搜索上界和预估下界实现解空间的剪枝。异步前向定界算法（Asynchronous Forward Bounding, AFB）^[26,27]和并发前向定界算法（Concurrent Forward Bounding, ConcFB）^[28]在 SyncBB 的基础上引入前向定界过程以提高算法的剪枝效率。在前向定界过程中，当前智能体传递部分解给未赋值的智能体，请求它们发送在当前部分解下自身的代价预估。随后，当前智能体累积收到代价预估来计算下界，从而提高算法剪枝效率。然而在链式结构中，智能体仅能依预定顺序串行地执行计算任务，无法利用多智能体系统并行计算的优势。此外，链式通信结构还会使不具有约束关系的智能体之间互相通信，这违背分布式环境下不相连智能体间不能通信的限制。相比之下，基于树型的通信结构具有更高的并发性，且不具有约束关系的智能体之间无法直接通信。

异步分布式优化（Asynchronous Distributed Optimization, ADOPT）^[29,30]算法是基于伪树结构的完备搜索算法，其求解思路包含异步搜索、减少解重构和终止条件检测三个部分。其中，异步搜索通过采用最佳优先搜索和分布式回溯策略实现；减少解重构利用搜索阈值更新和分配机制实现；终止条件检测通过智能体为其搜索过程维护搜索上、下界实现。在 ADOPT 中，由于采用最佳优先搜索策略，所有智能体都依据局部信息进行异步快速响应。其异步性虽然能提高算法的并行性，但却产生巨大的消息数。随后，一系列基于 ADOPT 求解框架的改进算法^[31-38]被提出。其中 ADOPT+^[34]采用时间戳检测机制判断并丢弃冗余的消息。IDB-ADOPT^[35]，ADOPT-ng^[36]和 ADOPT-ing^[37,38]通过改进搜索策略和消息结构来提升算法的求解效率。BnB-ADOPT^[39]提出通过采用深度优先搜索来进一步降低 ADOPT 中解重构的问题。之后，学者提出许多基于 BnB-ADOPT 的改进算法^[40-43]。通过结合 ADOPT 和 BnB-ADOPT 两种算法的解决思路，ADOPT(k)^[44]引入参数 k 动态地控制算法的搜索策略，BD-ADOPT^[45]采用分层的思想结合两种搜索策略。对于上述基于树型结构的异步完备搜索算法而言，由于智能体仅依赖于本地累积知识进行独立地决策，因此这类算法通常具有很高的实时响应速度和对消息丢失的鲁棒性，但由于每个智能体决策依据的上文不一致导致大量冗余消息的产生。而基于树型结构的同步完备搜索算法规定智能体的决策顺序，从而避免异步算法中智能体之间上文不一致的情况。NCBB^[46]和 PT-FB^[47]是典型基于树型结构的同步完备搜索算法，其在树型结构上执行深度优先搜索和分支定界策略实现解空间的遍历。为提高算法的剪枝效率，NCBB 采用局部代价预估及时传播策略，而 PT-FB 则是在树型结构的不同独立分支上执行前向定界策略。此外，有学者引入软弧一致性技术^[48-53]以及 Anyspace 框架^[54-56]来提高完备搜索类算法的求解效率。其中，

算法通过在搜索过程中保持软弧一致性从而实现尽早地检测次优解，达到缩小搜索空间的目的。Anyspace 框架通过利用更多的内存空间缓存历史探索结果，减少算法对搜索子空间的重复探索。

不同于基于搜索的完备算法需要显式地遍历解空间来寻找最优解，基于推理的完备算法使用动态规划思想实现问题的求解。分布式伪树优化过程（Distributed Pseudo-tree Optimization Procedure, DPOP）^[57]是最具代表性的完备推理算法，它是桶消元^[58,59]在伪树结构上的分布式实现。DPOP 的求解过程可分为效用传播阶段和值传播阶段。在效用传播阶段，智能体计算并自下而上地传播上层节点赋值组合对应的效用表。这里，智能体传播的效用表由如下两步计算得到。智能体首先联合本地效用表和所有孩子节点发送的效用表（即子问题的推理结果），然后对联合后的效用表执行本地消元。在值传播阶段，智能体计算并自上而下地传播最优赋值。DPOP 所需的消息数线性于智能体个数，但其消息大小却指数于伪树的诱导宽度。之后，一些基于 DPOP 的改进算法被提出。其中，Léauté 等人^[60]提出一系列的 P-DPOP 来实现算法在问题求解过程中不同层度的私有信息的保证。MB-DPOP^[61]、RMB-DPOP^[62]和 O-DPOP^[63,64]提出通过传递更多的消息数来为降低算法的内存消耗。为提升算法的求解效率，有学者提出在非传统的树型结构上执行桶消元过程，例如：DCPOP^[65]采用最佳优先树，DCTREE^[66]采用簇树结构，Action_GDL^[67,68]采用分布式联合树^[69]，BFS-DPOP^[70]采用广度优先树。此外，还有学者提出通过利用问题中的硬约束^[71-73]来降低算法在效用传播阶段的消息大小，或通过函数过滤技术^[74-78]来降低算法的通信负载。

尽管基于搜索的完备算法产生的消息大小与智能体个数线性相关，但由于其需要系统地探索整个解空间，算法所需的消息数是指数级的。相反，基于推理的完备算法仅需要线性级的消息数，但由于其需要传播整个子问题对应的推理结果，算法产生的最大消息大小是指数级的。因此，ADOPT-BDP^[79]、DJAO^[80]提出结合两种求解策略的优势，即：在预处理阶段采用近似的推理策略获得初始下界，随后采用完备搜索策略实现问题的求解。

非完备算法可分为完备算法的近似版本、基于决策的局部搜索算法、基于信念传播的推理算法、采样算法和其他算法。其中，完备算法的近似版本，例如：ADOPT-AEM/REM^[81]和 A-DPOP^[82]和 LS-DPOP^[83]通过简化完备算法的优化条件并加入容错机制，在牺牲解的质量的同时也降低计算所需的成本。

分布式随机算法（Distributed Stochastic Algorithm, DSA）^[84]和最大增益消息算法（Maximal Gain Message, MGM）^[85,86]是典型的基于决策的局部搜索算法。该类算法中，智能体不断地将自身状态信息（如自身赋值、增益等）广播给邻居智能体，并基于自身状态与接收到的邻居状态确定否将自身赋值替换为最佳响应。

不同算法之间的主要区别在于自身赋值的替换条件。例如：在 DSA 中，智能体依概率确定是否替换赋值。而在 MGM 中，邻居节点中增益最大的节点才替换赋值。泛化分布式打破算法（Generalized Distributed Breakout Algorithm, GDBA）^[87]提出将用于求解分布式约束满足问题^[88]的分布式打破算法（Distributed Breakout Algorithm, DBA）^[89,90]扩展到 DCOP 求解。近期，Khoi D. Hoang 等人提出将大邻域搜索^[91,92]思想用于分布式求解 DCOP。由于基于局部搜索算法具有类似的结构，有学者提出一种统一的框架^[93]来分析和比较该类中各种算法。由于基于决策的局部搜索算法多采用贪心策略探索解空间，算法极易陷入局部最优。为此，有学者将 k -优^[94-99]思想引入到基于决策的局部搜索算法中。这里， k -优是指的是：任意一组数量不大于 k 的智能体赋值变化，都无法提升解的质量。MGM-2、DSCA-2、KOPT^[100,101]就是基于 k -优思想提出的算法。针对于 k -优算法的局部信息保障问题，有学者提出带信息保障的区域最优算法^[102,103]来保障算法在问题求解过程中的约束和部分决策等局部私有信息。不同于 k -优思想，有学者发现智能体可以通过忽略部分的邻居赋值来扩大决策空间，从而打破局部最优状态来提升算法的求解质量^[104]。此外，很多局部搜索算法无法记录算法求解过程中所获得的最优解（即不具备 Anytime 性质）^[105]。因此，有学者提出基于 Anytime 的局部搜索算法框架^[106]。该框架通过构建一棵广度优先树来累积全局代价，并记录算法在当前搜索到的最优解以及对应的代价值。

最大和（Max-sum）^[107]是基于信念传播的推理算法中最重要的一种算法，该算法通过在因子图^[108,109]上迭代地传播和累积信念来实现全局目标函数的边际化。但面对有环问题时，Max-sum 通常无法收敛。针对该问题，有学者提出通过去除对解质量影响较小的边，将有环因子图转换成为无环因子图^[110-112]，然后采用 Max-sum 进行求解。也有学者提出通过预先规定消息传递的方向将有环因子图转换成为有向无环图，进而保证算法收敛^[113-116]。近期，有学者通过研究 Max-sum 消息传递中的衰减算法求解质量的影响，提出受衰减的 Max-sum^[117,118]。为达到平衡探索和利用的目的，他们还将标准的因子图转化为等价的分裂约束因子图来控制每个约束的不对称程度。此外，他们还采用回溯代价树对 Max-sum 算法在带多个环的树下收敛性进行理论分析^[119]。为保护算法在求解过程中的局部私有信息，P-Max-sum^[120,121]提出通过使用密码学的技术来保障算法的约束、拓扑以及决策等局部信息。此外，Max-sum 类算法在计算信念消息时结合本地约束代价函数与收到的消息以及执行最大化操作所需的计算量是指数级的，这使得这类算法在可扩展性方面仍面临着巨大的挑战。为解决这一问题，学者提出采用分支定界的算法^[122-124]和基于排序的算法^[125-128]。

在基于采样的算法中，智能体通过对搜索空间不断地进行采样，然后利用统计

推理等方法将采样获得的数据计算为自身变量赋值的概率分布以引导采样。分布式 UCT (Distributed Upper Confidence Tree, DUCT) [129,130]是该类算法的典型代表，其也是 UCT^[131,132]算法在分布式环境下的扩展。在 DUCT 中，智能体为自身变量的每一个赋值构建一个置信界作为该赋值所产生代价的乐观估计，并在采样时选择最小置信界对应的赋值。但 DUCT 的内存消耗却指数于智能体的个数，因此 Nguyen 等人将 Gibbs^[134]扩展到 DCOP 求解中，提出分布式吉布斯采样 (Distributed Gibbs, D-Gibbs) 算法^[135]，一种仅需线性内存的算法。在 D-Gibbs 中，DCOP 被映射成极大似然估计问题^[136,137]，因此可采用 Gibbs 算法进行求解。

此外，学者还提出采用分治协调策略实现求解 DCOP 的求解^[138-141]。其中，DCOP 被划分为多个独立的子问题，之后通过协调每个子问题寻优过程来获得问题的解。最近有者将蚁群优化^[142]、遗传算法^[143,144]、耦合振子的同步^[145]以及深度神经网络^[146,147]等技术应用于 DCOP 的求解。

1.2.2 ADCOP 算法研究现状

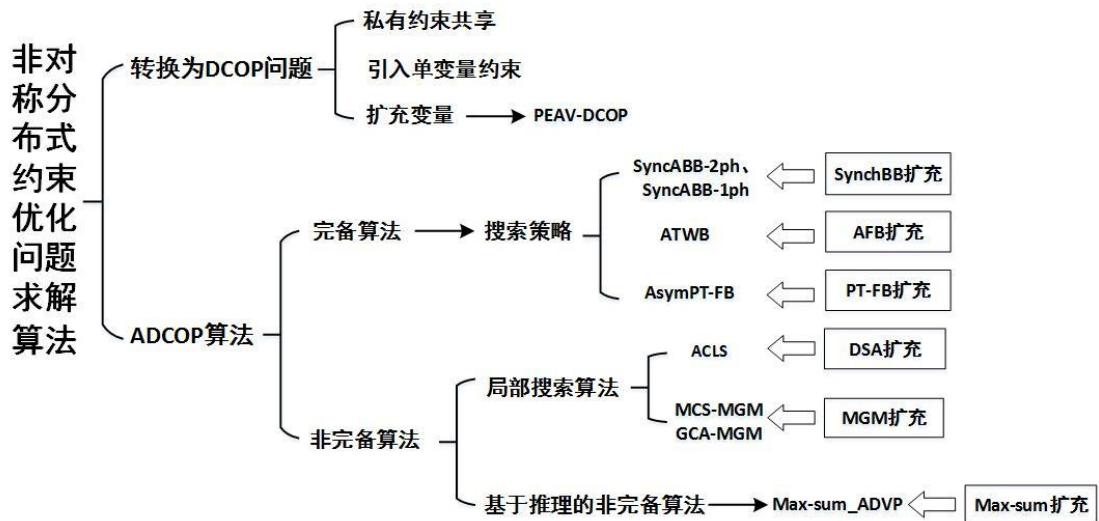


图 1.2 ADCOP 算法分类

Fig. 1.2 The taxonomy of ADCOP algorithms

ADCOP 求解思路可分为模型转换和直接求解两类，如图 1.2 所示。其中，模型转换是将 ADCOP 转换为 DCOP，随后采用 DCOP 算法进行求解。私有约束共享是一种最直接的模型转换方式，其通过泄露约束各方的私有约束代价函数实现问题的转换。但该方式会极大地泄露智能体的私有偏好^[148,149]，因此不适用于 ADCOP 求解。引入额外的一元约束是另外一种模型转化方式，其思想是：采用一个对称约束代价函数和若干一元约束代价函数来表征一个非对称约束代价函数。

但可以证明：至少存在一种 ADCOP 无法通过该方法转换为 DCOP。私有事件作为变量（Private Events As Variables, PEAV）^[150]是目前比较成熟的模型转换方式。PEAV 通过在智能体中增加虚拟变量来映射其邻居智能体所控制的变量，并通过引入硬约束来保证虚拟变量和其所对应变量之间赋值的一致性。由于虚拟变量的引入将极大地增加问题的求解难度，限制其可求解问题的规模。因此，直接对 ADCOP 进行求解是 ADCOP 算法研究的主流。

依据是否保证获得最优解，ADCOP 算法可分为完备算法和非完备算法。现有的 ADCOP 完备算法大多是 DCOP 完备搜索算法在非对称环境下的扩充，通过与一阶段策略或两阶段策略结合来实现双边约束代价的累积。具体地，带一阶段策略的算法在部分解的扩展中不断实现双面约束代价的累积。当算法达成一组完整解时，该完整解对应的双边代价也累积完毕。而带两阶段的算法分为如下两阶段：第一阶段完成部分解扩展以及单面约束代价的累积；在达成一组完整解后，第二阶段开始累积另一面的约束代价。两阶段非对称 SyncBB 算法（SyncABB-2ph）^[19]是 SyncBB 结合两阶段策略后在非对称环境下的扩展。而一阶段非对称 SyncBB 算法（SyncABB-1ph）^[19]与双向定界算法（Asymmetric Two Ways Bounding, ATWB）^[19]则是 SyncBB 与 AFB 算法结合一阶段策略在非对称环境下的扩展。为实现一阶段策略，SyncABB-1ph 使用反向检测的回溯搜索过程，而 ATWB 使用异步反向回溯机制。近期，Omer 等人^[47]通过上层约束代价直接泄露的方式将 PT-FB 算法扩展到 ADCOP 求解中，提出基于树型结构的 ADCOP 完备算法 AsymPT-FB。在 AsymPT-FB 中，智能体执行前向定界过程实现搜索下界的计算，并同时执行回溯过程实现双面代价的累积。在前向定界的过程中，智能体将当前部分解发送给其上层节点去请求它们的代价预估，然后累积其收到的预估实现下界的计算。在反向定界的过程中，智能体将当前部分解发送给其下层节点请求它们关于另一面的约束代价。

现有的 ADCOP 非完备算法大多是 DCOP 非完备算法在非对称上的扩展。非对称协调局部搜索（Asymmetric Coordinated Local Search, ACLS）^[18,19]算法，最小约束共享 MGM（Minimum Constraint Sharing, MCS-MGM）^[18,19]算法和保证收敛的非对称 MGM（Guaranteed Convergence Asymmetric MGM, GC-MGM）^[18,19]算法是 DSA 和 MGM 算法在非对称上的扩展。在这些算法中，智能体通过交换最佳响应所对应的单面约束代价值实现协调决策。最近，有学者^[151,152]提出在非对称因子图上执行使用 Max-sum 及其变体实现 ADCOP 的求解。

1.2.3 现有 DCOP 和 ADCOP 完备算法研究面临的问题与挑战

虽然在 DCOP 与 ADCOP 完备算法研究方面已经有一些研究进展，但仍存在较大的发展空间，尤其对于 ADCOP 完备算法而言，其在求解思路方面仍存在很多值

得探索的问题。总结起来有如下三点：

① 现有 DCOP 完备算法大多采用单一的求解策略，即：仅采用搜索策略或仅采用推理策略。如 1.2.1 节所述：仅采用搜索策略的求解算法所需的消息大小线性于智能体个数，但产生消息的数量却指数于智能体个数；仅采用推理策略的求解算法所需的消息数量是线性于智能体个数，但消息的大小却指数于问题的规模。因此，有学者提出采用推理与搜索混合的方式实现 DCOP 的求解，实现两种求解策略的优势互补。但这种混合仅仅只是低层次的，即：推理阶段仅作为预处理为搜索提供初始下界，之后问题的求解依然依赖于搜索阶段。也就是说，现有的推理与搜索混合 DCOP 完备算法性能依赖于预处理阶段提供初始下界的紧度。当在内存受限时，预处理阶段往往无法提供紧的下界，这使得算法的求解效率依然很低。

② 现有 ADCOP 完备算法采用搜索策略实现问题的求解。而基于搜索策略的完备算法通过下界实现剪枝以提升算法的求解性能，但现有的 ADCOP 完备算法仅采用智能体所拥有的本地知识来建立下界，这使得算法获得下界的紧度较低，进而算法的求解性能低且无法应用于较大规模问题的求解。

③ 现有 ADCOP 完备算法采用搜索策略来系统地遍历解空间获得问题的最优解，这使得算法在求解过程会产生指数级的消息数，且能求解的问题规模有限。另一方面，传统的 DCOP 完备推理算法仅需要线性的消息数即可实现问题的求解，且该类算法的问题求解规模大于基于搜索策略的完备算法，但由于私有偏好保障的问题，现有推理策略无法用于 ADCOP 的完备求解。

1.3 论文主要研究内容和创新之处

1.3.1 论文主要研究内容

本文从 DCOP/ADCOP 完备算法入手，研究以下关键科学问题：1) 在内存空间有限的情况下，如何利用上文的推理策略提升推理与搜索混合 DCOP 完备算法的求解性能；2) 如何利用推理策略提升 ADCOP 完备搜索算法的求解性能；3) 如何利用推理策略实现 ADCOP 的完备求解。本文的具体研究内容如下：

① 针对于 1.2.3 节问题 1：本文利用在内存受限情况下，上文推理依然可以提供紧的下界的优势，提出基于上文推理与搜索混合的 DCOP 完备算法。在该算法中，上文推理与搜索互相辅助、并行执行。其中，搜索为上文推理提供推理所需的上文以降低迭代的上文推理所带来的巨大网络负载消耗，同时上文推理提供紧的下界可以提升搜索的剪枝效率，从而减少上文推理的次数。这样，算法整体的求解性能获得提升。此外，本文提出上文评估机制选择合适的上文进行推理来进一步降低上文推理带来的网络负载消耗。

② 针对于 1.2.3 节问题 2：本文考虑 ADCOP 的非对称特性，研究如何利用推理策略求解 ADCOP，提出基于推理与搜索混合的 ADCOP 完备算法。该算法首先在推理阶段求出问题单面约束为搜索阶段提供下界以实现剪枝。利用推理阶段获得的下界，搜索阶段在伪树的不同分枝并行执行分布式回溯搜索。为进一步提升推理阶段提供下界的紧度，本文提出非本地消元机制，即：变量消元位置推迟到其父节点位置。此外，通过结合绝对误差机制和相对误差机制，本文提出两种非完备扩展版本。这两种非完备适用于在用户分别指定绝对误差界和相对误差界的前提下，以较小的计算量和消息数来快速地找到满足条件的一组近似解。

③ 针对于 1.2.3 节问题 3：本文研究如何通过广义非本地消元机制将推理策略用于 ADCOP 的完备求解，率先提出仅采用推理策略的 ADCOP 完备算法。不同于本地消元机制，变量在本地执行的消元操作；也不同于非本地消元机制，变量消元位置推迟到其父亲节点；在广义非本地消元机制中，变量的消元位置推迟到伪树上位置最高的（伪）父节点，从而降低推理过程中智能体私有约束的信息损失。为解决由于广义非本地消元机制带来的内存占用以及计算量增加，本文提出基于桶集合的传播策略来降低算法的内存占用以及基于分批次的消元策略来减少算法所需的计算量。同时为使得提出的算法适用于内存有限的环境，本文将内存有界推理思想适配到提出的算法来提升算法性能。此外，提出分支独立枚举机制和两阶段的效用集合传播机制来降低内存有界推理中的冗余推理。

1.3.2 论文创新之处

① 本文率先提出将上文推理与搜索混合应用于 DCOP 的完备求解。尽管已有将推理与搜索策略混合来求解 DCOP，但现有的混合仅利用近似推理作为预处理过程，为搜索的剪枝过程提供可信赖的下界。随后，算法利用下界实现次优解的提前检测，提高剪枝的位置，从而提升剪枝效率。在内存有限的情况下，预处理阶段中推理无法获得较紧的下界，而实验发现上文推理依然可以获得较紧的下界。基于上述分析，提出上文推理与搜索混合算法。研究在该算法中，上文推理与搜索分工与协作关系。理论证明在内存空间限制的条件下，上文推理过程相比于现有的预处理推理过程提供更紧的下界，且实验结果证明该算法在多种测试问题上均优于目前性能最好的 DCOP 完备算法。

② 本文率先提出将推理策略应用于 ADCOP 的完备求解。由于 ADCOP 对私有偏好保障的要求，基于 DCOP 的完备推理算法不适用于 ADCOP。本文创新性地提出推理与搜索混合的 ADCOP 完备算法。与现有基于完备搜索的 ADCOP 算法不同，本文提出的算法利用推理策略预先求解问题单面约束为搜索提供更紧的下界。进一步，提出非本地消元机制来提升算法所提供下界的紧度，从而提升算法的求解效率。本文理论证明在内存空间维度等于问题诱导宽度的前提下，算法提供下

界的精度优于现有性能最好的 ADCOP 完备搜索算法,且实验结果表明该算法在多种测试问题上均优于目前最好的 ADCOP 完备算法。

③ 本文率先提出一种仅需线性级消息数的 ADCOP 完备求解算法。现有 ADCOP 完备算法仅依赖于系统地搜索解空间来获得问题最优解,导致算法需要指指数级的消息数,从而限制 ADCOP 完备算法的问题求解规模。由于 ADCOP 对私有偏好保障的要求,DCOP 完备推理算法无法直接求解 ADCOP。因此本文创新性地提出基于广义非本地消元机制的 ADCOP 完备推理算法,引入两种权衡算法性能的加速机制,并结合内存有界推理进一步提升算法的问题求解规模。理论分析所提出算法的内存占用,实验结果表明该算法在多种测试问题上均优于目前最好的 ADCOP 完备算法,在问题的求解规模上远超现有的 ADCOP 完备算法。

1.4 论文组织结构

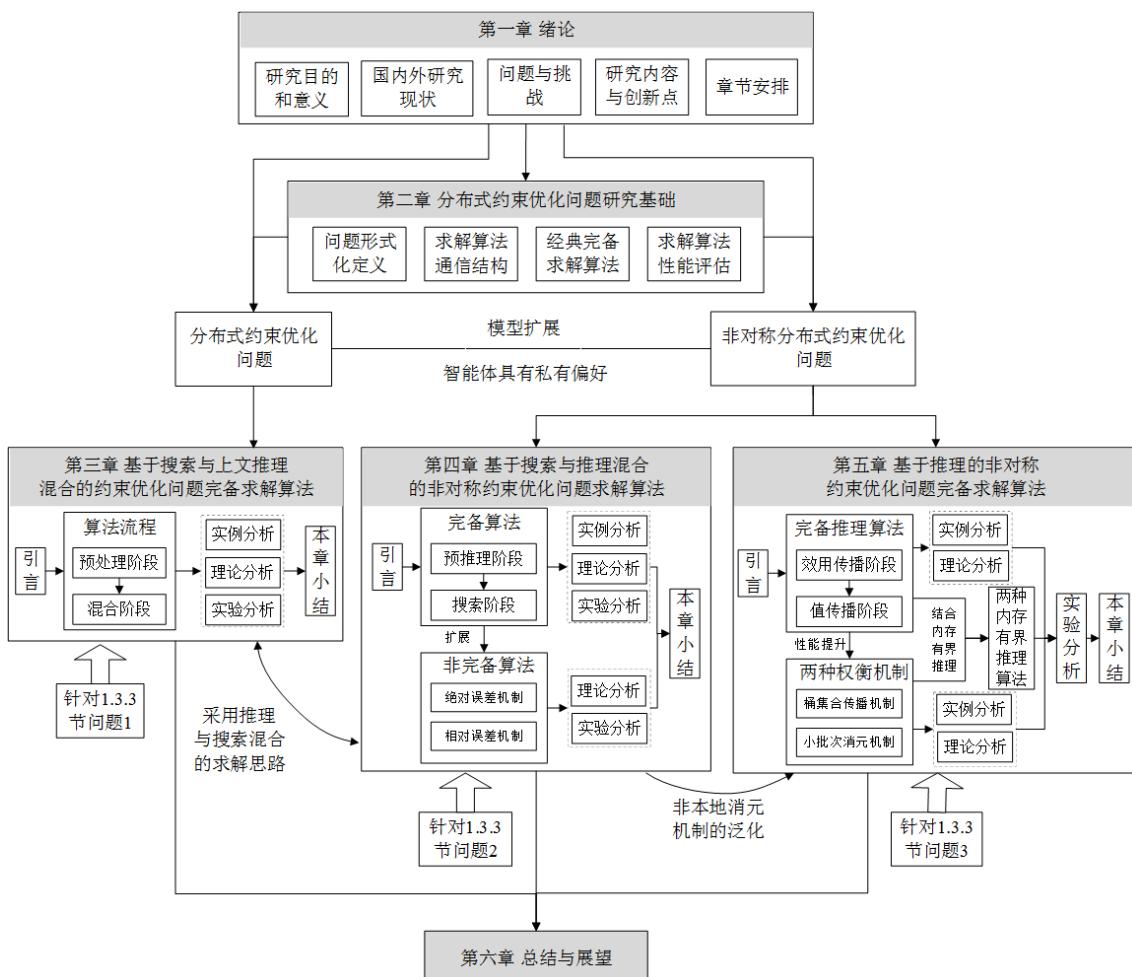


图 1.3 论文框架结构图

Fig. 1.3 The chart of the organization of this dissertation

本文共分为六章，论文组织结构图如图 1.3 所示，具体内容安排如下：

第一章，绪论。首先，介绍分布式约束优化问题（DCOP）以及非对称分布式约束优化问题（ADCOP）的研究背景与研究意义；然后，综述 DCOP/ADCOP 算法的国内外研究现状，分析当前研究不足之处；最后，概括本文的研究内容、创新点和组织结构。

第二章，分布式约束优化问题研究基础。给出 DCOP 和 ADCOP 的定义，并介绍算法采用的通信结构；最后，介绍实验采用的测试问题和性能评价指标。

第三章，基于上文推理与搜索混合的分布式约束优化问题完备算法。首先，分析现有推理与搜索混合的 DCOP 完备算法在内存有限的条件下，无法获得较好的剪枝效率，从而导致算法的求解性能低。基于上述分析，提出上文推理与搜索混合的 DCOP 完备算法 HS-CAI。随后，理论分析 HS-CAI 算法的性质和时间、空间复杂度。最后，实验结果表明 HS-CAI 在性能和问题求解规模上都优于目前性能最好的完备算法。

第四章，基于推理与搜索混合的非对称分布式约束优化问题算法。首先，分析现有 ADCOP 完备算法仅利用本地知识建立下界，导致智能体私有偏好泄露严重。基于上述分析，提出混合推理与搜索的 ADCOP 完备算法 PT-ISABB。此外，提出两种 PT-ISABB 的非完备版本。随后，理论分析分析 PT-ISABB 算法的性质和时间、空间复杂度。最后，实验结果表明 PT-ISABB 在性能上优于现有的完备搜索算法，且 PT-ISABB 的非完备版本能在较小的计算量和通信负载下获得在用户指定误差下的近似解。

第五章，基于推理的非对称分布式约束优化问题完备算法。首先分析现有 ADCOP 的完备算法仅采用搜索策略会产生指数级的消息数且问题求解的规模有限。此外，现有的 DCOP 完备推理算法由于存在私有偏好完全泄露的问题无法直接用于 ADCOP 的求解。基于上述分析，提出基于推理的 ADCOP 完备算法 AsymDPOP。为提升 AsymDPOP 算法性能，提出两种算法性能指标的权衡机制，并将所提出算法与内存有界推理进行结合，使得算法适用于内存受限的环境。理论分析所提出算法的内存消耗，实验结果表明所提出算法优于现有的完备算法。

第六章，总结与展望。对本文的研究内容进行总结，并对后续研究做出展望。

2 分布式约束优化问题研究基础

2.1 分布式约束优化问题

2.1.1 问题定义

分布式约束优化问题(Distributed Constraint Optimization Problem, DCOP)^[23,31]由四元组 $\langle A, X, D, F \rangle$ 组成，其中：

- ① $A = \{a_1, \dots, a_n\}$ 是智能体集合，一个智能体为一个或多个变量取值；
- ② $X = \{x_1, \dots, x_m\}$ 是变量的集合，一个变量仅由一个智能体控制；
- ③ $D = \{D_1, \dots, D_m\}$ 是值域的集合，其中变量 x_i 从值域 D_i 中取值；
- ④ $F = \{f_1, \dots, f_q\}$ 是一组约束代价函数的集合，每个约束代价函数 $f_i: D_{i_1} \times \dots \times D_{i_k} \rightarrow \mathbb{R}_{\geq 0}^+$ 是与该函数相关联的 k 个变量组成的集合 $S_i = \{x_{i_1}, \dots, x_{i_k}\}$ 的每个赋值组合到一个非负代价的映射。

不失一般性地，DCOP的一个完整解是包含所有变量的赋值组合，而最优解是使得所有约束代价函数之和最小的完整解，即：

$$X^* = \operatorname{argmin}_X \sum_{f_i \in F} f_i \quad (2.1)$$

为便于理解，假设一个智能体仅控制一个变量（即 $n = m$ ）且所有约束代价函数都是二元函数。因此，智能体、节点和变量可以被认为是同一个概念，且可以相互替换。二元代价约束代价函数指仅涉及到两个变量的约束代价函数，即 $f_{ij}: D_i \times D_j \rightarrow \mathbb{R}_{\geq 0}^+$ 。因此，DCOP的最优解可以被定义为：

$$X^* = \operatorname{argmin}_{d_i \in D_i, d_j \in D_j} \sum_{f_{ij} \in F} f_{ij}(x_i = d_i, x_j = d_j) \quad (2.2)$$

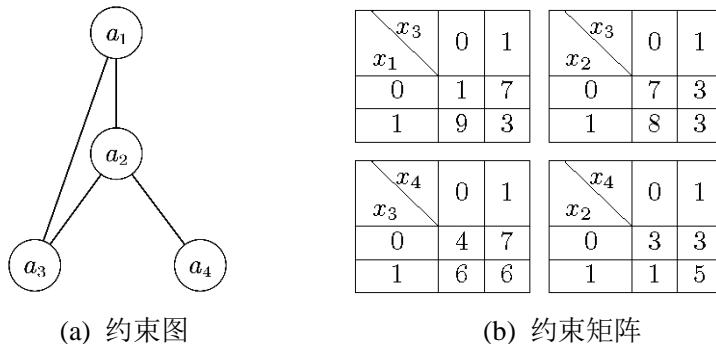


图 2.1 DCOP 实例

Fig. 2.1 A DCOP instance

DCOP可以用约束图来直观地表示。在约束图中，一个节点代表一个智能体，

一条边则代表一个约束关系。图 2.1 给出一个有四个智能体和四个二元约束关系的 DCOP 的实例，其中左边是其约束图，右边是相对应的约束代价函数。

2.1.2 实际问题建模

本节将列举两个实际应用问题（即传感器网络目标跟踪问题和分布式信道分配问题）来展示 DCOP 对实际问题的建模过程。

① 传感器网络目标跟踪问题

传感器网络目标跟踪问题^[153,154]由多个静止的传感器 $S = \{s_1, s_2, \dots, s_n\}$ 和多个移动的目标 $T = \{t_1, t_2, \dots, t_m\}$ 组成，其通过将目标分配给对应的传感器以实现对目标的精准跟踪。一个传感器某一时刻仅能跟踪一个目标，且其感知范围有限，因此只能跟踪附近的目标。假设必须将三个传感器分配给一个目标才能实现对其准确地跟踪。传感器能够使用低带宽射频通信，但其通信范围也有限，因此传感器仅能与附近的代理进行通信。图 2.2 显示了网格配置中的 6 个传感器和 2 个目标，每个目标具有一定的权重用于描述目标的重要性。每个目标只能被最近的 4 个传感器跟踪。如图 2.2 (a) 所示，目标 t_1 可以被传感器 s_1, s_2, s_4, s_5 跟踪；每个传感器仅能和其邻居节点进行通信。传感器 s_1 仅能与传感器 s_2, s_4, s_5 进行消息传播，如图 2.2 (b) 所示。因此，传感器必须通过消息传递来协调其需要跟踪的目标，使得被跟踪的目标权重之和最大。如图 2.2 (b) 所示，传感器 s_1, s_2, s_4 跟踪目标 t_1 ，传感器 s_3, s_5, s_6 跟踪目

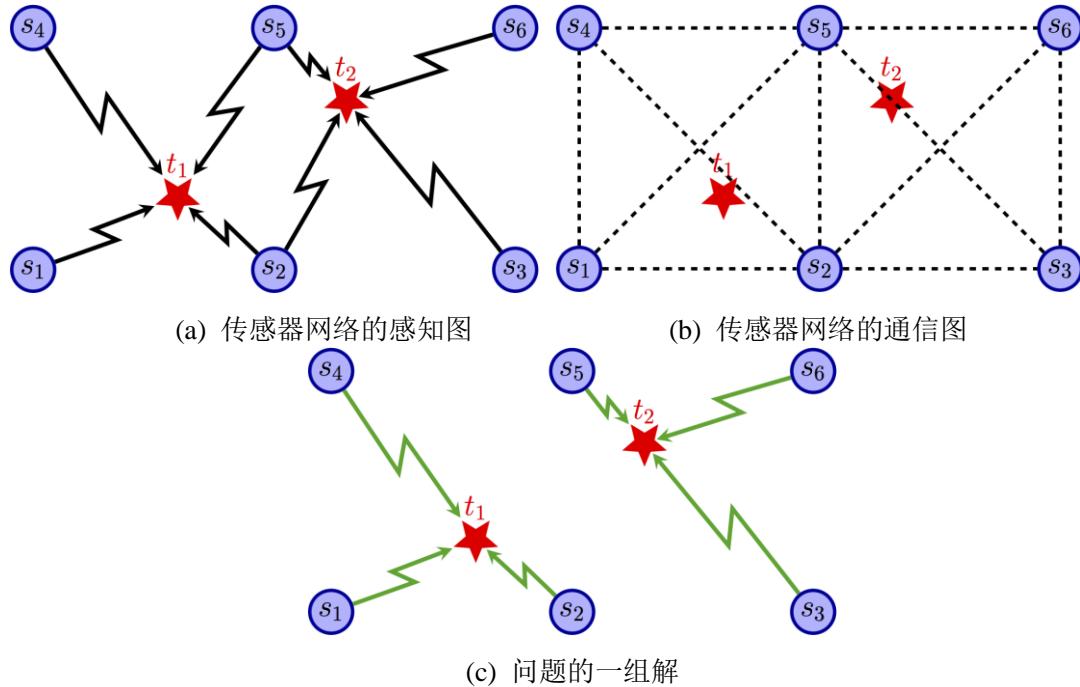


图 2.2 目标跟踪实例

Fig. 2.2 A target tracking instance

标 t_2 是该问题的一组解。该问题具有如下特性：

1) 内在的分布式特性：传感器网络中，没有具有全局问题信息或全局控制能力的中心节点。每个传感器必须根据本地信息以及从其邻居传感器接收到的信息，自主决定应该跟踪哪个目标。

2) 通信的限制：传感器网络中，通信受到通信范围和带宽的限制，无法在单个传感器上收集问题的所有信息。此外，由于射频通信不可靠，消息可能会丢失。在一些领域，信息保障要求的限制以及信息转换的高昂成本也限制了通信。

3) 有限的决策时间：目标跟踪问题中，待跟踪的目标是移动的，因此，传感器确定目标分配的时间是有限。对于特定位置的一组目标，传感器必须能快速地确定全局分配，在目标移动到其他位置之前对其进行跟踪。

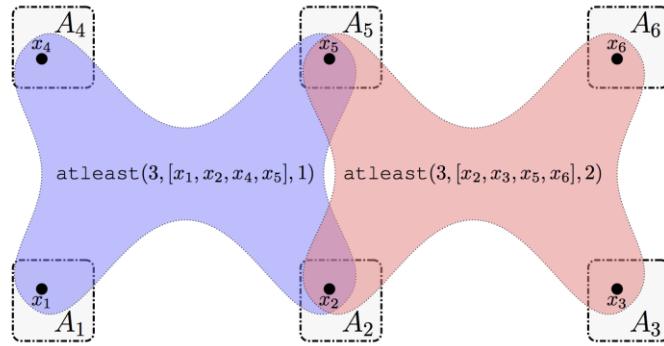


图 2.3 目标跟踪问题的 DCOP 建模

Fig. 2.3 DCOP formulation of target tracking

上述目标跟踪问题可以被 DCOP 四元组 $\langle A, X, D, F \rangle$ 建模，其中：

- a. $A = \{a_1, a_2, \dots, a_n\}$ 为 n 个智能体，每个智能体代表一个传感器；
- b. $X = \{x_1, x_2, \dots, x_n\}$ 表示控制传感器状态的变量。 x_i 表示传感器 s_i 跟踪目标的状态；
- c. $D = \{D_1, D_2, \dots, D_n\}$ 表示传感器的状态空间。其中 D_i 为离散的常数集合，不同的常数表示传感器 s_i 可跟踪的不同目标。以图 2.2 为例，传感器 s_1 对应变量的值域为 {1}，传感器 s_2 对应变量的值域为 {1, 2}。
- d. $F = \{f_1, f_2, \dots, f_q\}$ 表示传感器之间的效用函数集合。效用函数由每个目标是否被其附近的传感器跟踪确定。以图 2.2 为例，对于传感器 s_1 而言，它有一个可跟踪的目标，因此有一个约束 $\text{atleast}(3, [x_1, x_2, x_4, x_5], 1)$ ，表示 s_1 及其邻居 s_2, s_4, s_5 中至少有 3 个传感器跟踪目标 1，才会获得跟踪目标 1 对应的效用。否则，其值为 0。

上述问题的 DCOP 建模如图 2.3 所示。问题的目标被定义为找到一组包含所有

变量的赋值组合 X^* ，使得系统效用之和最大化^①，即：

$$X^* = \operatorname{argmin}_X \sum_{f_i \in F} f_i$$

② 分布式信道分配问题

无线局域网^[155]（WLAN, Wireless local Area Network）是一种无线计算机网络，使用无线信道代替有线传输介质连接两个或多个 AP（Access Point）设备形成一个局域网（LAN, Local Area Network），典型的部署场景如家庭、学校、校园或企业办公楼等。一个 WLAN 网络通常由大量的 AP 设备构成，相连的 AP 设备之间出现同信道干扰和相邻信道干扰会显著降低网络性能。每个 AP 设备会有可供分配的信道区间。WLAN 部署的目标为：AP 设备从其可供选择的信道区间为自己分配信道，使得 WLAN 能为其覆盖区域提供良好的无线网络连接服务。AP 设备信道分配问题可以看作加权图着色问题的一种特例，其约束代价矩阵由 AP 设备在受到干扰时的信号接收性能指标，即信噪比（SINR, Signal to Interference and Noise Ratio），对应的计算公式获得^[156]。在特定的部署环境（例如人口密集的城市），AP 设备可能属于不同的管理区域，对 AP 设备的管理控制权限需要下放给不同的管理单元。这种应用场景下，没有一个中心节点进行集中式的计算和控制，需要一种分布式的信道分配方案。

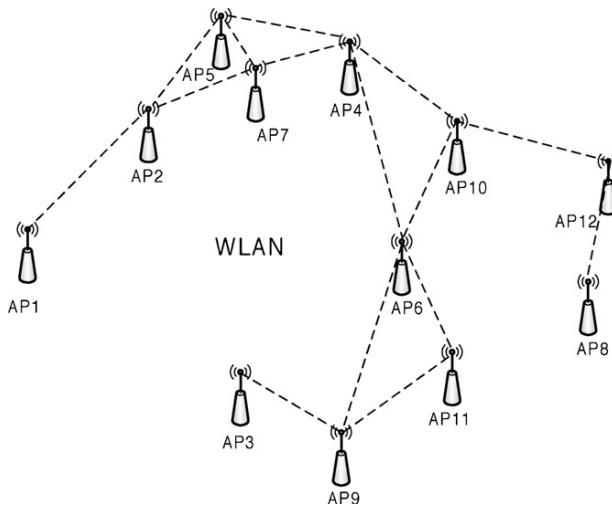


图 2.4 WLAN 网络部署实例

Fig. 2.4 An example of WLAN deployment

^①为了与 2.1.1 节问题定义一致，效用函数之和最大化问题可以被转换为代价函数之和最小的问题

表 2.1 信道分配的代价函数

Table 2.1 A cost function of channel assignment

$D_i \setminus D_j$	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0.7272	0.2714	0.0375	0.0054	0.0008	0.0002	0	0	0	0	0
1	0.7272	1	0.7272	0.2714	0.0375	0.0054	0.0008	0.0002	0	0	0	0
2	0.2714	0.7272	1	0.7272	0.2714	0.0375	0.0054	0.0008	0.0002	0	0	0
3	0.0375	0.2714	0.7272	1	0.7272	0.2714	0.0375	0.0054	0.0008	0.0002	0	0
4	0.0054	0.0375	0.2714	0.7272	1	0.7272	0.2714	0.0375	0.0054	0.0008	0.0002	0
5	0.0008	0.0054	0.0375	0.2714	0.7272	1	0.7272	0.2714	0.0375	0.0054	0.0008	0.0002
6	0.0002	0.0008	0.0054	0.0375	0.2714	0.7272	1	0.7272	0.2714	0.0375	0.0054	0.0008
7	0	0.0002	0.0008	0.0054	0.0375	0.2714	0.7272	1	0.7272	0.2714	0.0375	0.0054
8	0	0	0.0002	0.0008	0.0054	0.0375	0.2714	0.7272	1	0.7272	0.2714	0.0375
9	0	0	0	0.0002	0.0008	0.0054	0.0375	0.2714	0.7272	1	0.7272	0.2714
10	0	0	0	0	0.0002	0.0008	0.0054	0.0375	0.2714	0.7272	1	0.7272
11	0	0	0	0	0	0.0002	0.0008	0.0054	0.0375	0.2714	0.7272	1

图 2.4 给出了一个由 12 个 AP 节点组成的 WLAN 网络。假设每个 AP 节点可供选择的信道范围的 $\{0,1,2,3,4,5,6,7,8,9,10,11\}$ ，相连 AP 节点之间（例如 AP9 和 AP11）的约束代价函数由 SINR 公式计算得到（如表 2.1 所示）。问题的目标：为 12 个 AP 节点分配信道，使得系统整体的代价之和最小。

上述分布式信道分配问题可以被 DCOP 四元组 $\langle A, X, D, F \rangle$ 建模，其中：

- 1) $A = \{a_1, a_2, \dots, a_n\}$ 为 n 个智能体，每个智能体代表一个 AP 设备；
- 2) $X = \{x_1, x_2, \dots, x_n\}$ 表示 AP 设备信道的变量。 x_i 表示 AP 设备 AP_i 的信道分配状态；
- 3) $D = \{D_1, D_2, \dots, D_n\}$ 表示表示 AP 设备可分配的信道空间。其中 D_i 为离散的常数集合，不同的常数表示不同的信道。

4) $F = \{f_{ij} | \forall a_i \in A, a_j \in N(a_i)\}$ 表示传感器之间的效用函数集合。这里， $N(a_i)$ 代表 AP 设备 AP_i 的邻居。代价函数 f_{ij} 的取值由 SINR 公式计算得到，如表 2.1 所示。

因此，问题的目标为找到一组包含所有变量的赋值组合 X^* ，使得系统代价之和最小化，即：

$$X^* = \underset{d_i \in D_i, d_j \in D_j}{\operatorname{argmin}} \sum_{f_{ij} \in F} f_{ij}(x_i = d_i, x_j = d_j)$$

2.2 非对称分布式约束优化问题

非对称约束优化问题（Asymmetric Distributed Constraint Optimization Problem, ADCOP）^[19]在 DCOP 的基础上增加智能体私有偏好，即：对于每一个约束关系，每个智能体都有自己的私有约束代价函数，且不希望自身的私有函数被其他智能体所知。假想如下场景，一对好朋友爱丽丝和鲍勃讨论周末是否看电影。鲍勃喜欢爱丽丝，但不太关心是否去看电影；爱丽丝，想看电影，但不喜欢鲍勃的陪伴。这里，用字母 A 代表爱丽丝，B 代表鲍勃；T 代表去看电影，F 代表不去看电影。

爱丽丝的收益矩阵如图 2.5 (a) 所示：如果她一个人去看电影，收益最高；若鲍勃和她一起，收益略低；如果爱丽丝呆在家里，鲍勃是否去看电影对她都没有任何影响。鲍勃的收益矩阵如图 2.5 (b) 所示：鲍勃想和爱丽丝一起去看电影；如果得知爱丽丝去看电影，而自己留在家里，他会觉得非常失望；如果爱丽丝不去，则鲍勃略微倾向于去看电影。

\diagdown	B	T	F
A			
T	5	6	
F	2	2	

\diagdown	B	T	F
A			
T	6	1	
F	4	3	

(a) 爱丽丝的收益矩阵 (b) 鲍勃的收益矩阵

图 2.5 非对称约束的实例

Fig. 2.5 An asymmetric constraint instance

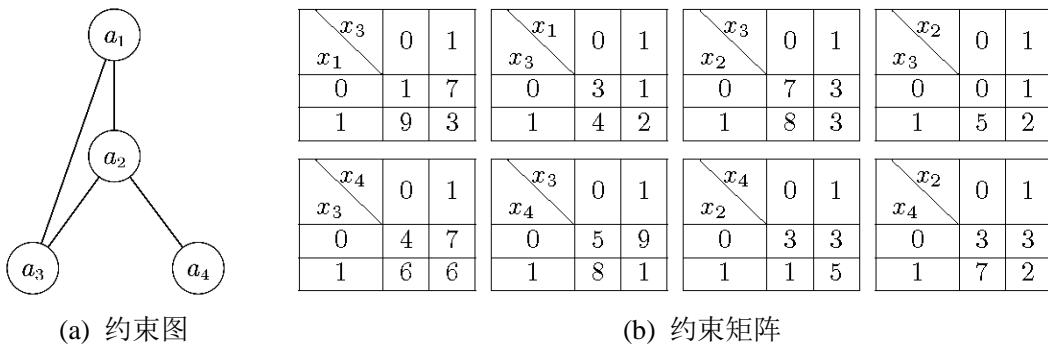
上述的例子，ADCOP 可以显示地通过定义智能体的私有代价函数进行定义。具体地，在 ADCOP 中一个约束关系是与该函数相关联的 k 个变量 $(x_{i_1}, \dots, x_{i_k})$ 的每个赋值组合到一个非负代价向量的映射，即：

$$f_i: D_{i_1} \times \dots \times D_{i_k} \rightarrow \{\mathbb{R}_{\geq 0}^+\}^k \quad (2.3)$$

也就是说，同一个约束关系为约束各方给出的代价值不同。对于 ADCOP，本文也假设每个智能体仅控制一个变量，且每一个约束代价函数都是二元函数。对于变量 x_i 和 x_j 而言，它们之间的私有约束代价函数为 f_{ij} 和 f_{ji} 。ADCOP 的求解目标是找到一个使得所有双边约束代价函数之和最小的完整解，即：

$$X^* = \operatorname{argmin}_{d_i \in D_i, d_j \in D_j, f_{ij}, f_{ji} \in F} f_{ij}(x_i = d_i, x_j = d_j) + f_{ji}(x_j = d_j, x_i = d_i) \quad (2.4)$$

图 2.6 给出一个有四个智能体和四个二元约束关系的 ADCOP 的实例，其中左边是其约束图，右边是对应的私有约束代价函数。



(a) 约束图

(b) 约束矩阵

图 2.6 ADCOP 实例

Fig. 2.6 An ADCOP instance

2.3 算法的通信结构

完备算法求解中，各个智能体按照通信结构规定的顺序执行消息的收发。链式结构和伪树结构是完备算法中最常用的两种通信结构。其中，在链式结构中，各个智能体按照某个优先级顺序排成一条链。链式结构虽然结构简单，但其确有如下两个缺点：①一个时刻仅有一个智能体执行消息处理，这极大地浪费多智能系统的计算资源优势；②一个智能体可能会与非邻居智能体通信，这打破 DCOP 中智能体通过局部交互获得全局最优的理念，增加智能体局部信息泄露的风险。

基于伪树的通信结构则可避免上述问题。伪树结构可通过对问题的约束图进行深度优先遍历获得，并将约束边分为树边和伪边（即非树边）。在一棵伪树上不同分支相互独立（即不同分支下的智能体不存在约束关系），因此基于伪树结构的算法可以利用这一特性实现对问题并行地求解。接下来，对本文所用到的伪树相关概念给出形式化的定义。

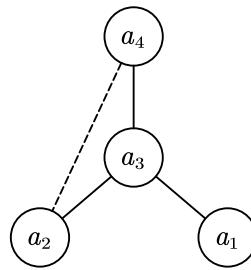


图 2.7 伪树结构实例

Fig. 2.7 A pseudo tree instance

对于智能体 a_i 而言，其邻居节点 $N(a_i)$ 根据其在伪树中的位置以及相互间所连接边的类型分为父节点 $P(a_i)$ ，伪父节点 $PP(a_i)$ ，孩子节点 $C(a_i)$ 以及伪孩子节点 $PC(a_i)$ 。具体地，上述变量可以做如下形式化的定义：

- 1) $N(a_i)$ 表示与所有与 a_i 具有约束关系的节点集合。
- 2) $P(a_i)$ 表示通过树边与 a_i 连接且在伪树上位置比 a_i 高的 $N(a_i)$ （这里，规定在伪树上根节点位置最高，越靠近根节点的节点位置越高）。
- 3) $PP(a_i)$ 是通过伪边与 a_i 连接，在伪树上位置比 a_i 高的 $N(a_i)$ 。
- 4) $C(a_i)$ 是通过树边与 a_i 连接，在伪树上位置比 a_i 低的 $N(a_i)$ 。
- 5) $PC(a_i)$ 是通过伪边与 a_i 连接，在伪树上位置比 a_i 低的 $N(a_i)$ 。

为简单起见，我们也采用如下的变量：

- 1) $AP(a_i)$ 是 a_i 的所有父节点，即： $AP(a_i) = \{P(a_i)\} \cup PP(a_i)$ 。
- 2) $AC(a_i)$ 是 a_i 的所有孩子节点，即： $AC(a_i) = C(a_i) \cup PC(a_i)$ 。
- 3) $Anc(a_i)$ 是 a_i 所有的祖先节点集合，可通过递归的方式进行定义为

$$Anc(a_i) = AP(a_i) \cup (\bigcup_{a_c \in C(a_i)} Anc(a_c) \setminus \{a_i\})。$$

4) $Desc(a_i)$ 是 a_i 所有的子孙节点集合, 可通过递归的方式进行定义为
 $Desc(a_i) = AC(a_i) \cup (\bigcup_{a_c \in C(a_i)} Desc(a_c))$ 。

5) $Sep(a_i)$ 是 a_i 的分割点集 (Separator), 其包含所有与 $Desc(a_i) \cup \{a_i\}$ 存在约束关系的 $Anc(a_i)$ 。可被形式化定义为
 $Sep(a_i) = \{a_j \in Anc(a_i) \mid a_k \in Desc(a_i) \cup \{a_i\}, a_j \in AP(a_k)\}$ 。

此外, 我们还定义节点 a_i 的诱导宽度为 $|Sep(a_i)|$, 伪树的诱导宽度为 $w^* = \max_{a_j \in A} |Sep(a_j)|$ 。

图 2.7 给出图 2.6 中约束图的一种可能的伪树结构, 其中实线代表树边, 虚线代表伪边。以图中智能体 a_3 为例, 其邻居节点集合 $N(a_3) = \{a_1, a_2, a_4\}$ 被分为父节点 $P(a_3) = a_4$, 伪父节点集合 $PP(a_3) = \emptyset$, 孩子节点集合 $C(a_3) = \{a_2, a_1\}$, 伪孩子节点集合 $PC(a_3) = \emptyset$, 所有父节点集合 $AP(a_3) = \{a_4\}$ 和所有孩子节点集合 $AC(a_3) = \{a_2, a_1\}$ 。此外, 我们有祖先节点集合 $Anc(a_3) = \{a_4\}$ 和子孙节点集合 $Desc(a_3) = \{a_2, a_1\}$ 。由于祖先节点 a_4 与自己和孩子节点 a_2 都具有约束关系, 因此, 其分割点集合 $Sep(a_3) = \{a_4\}$ 。

2.4 分布式约束优化问题实验

2.4.1 实验测试问题

本文实验中所使用的测试问题包括随机 DCOP/ADCOP、随机非对称 MaxDCSP、无尺度网络、分布式信道分配问题以及会议调度问题, 具体描述如下:

① 随机 DCOP/ADCOP (Random DCOP/ADCOP)。其代表一种通用的分布式约束优化问题形式。在随机 DCOP/ADCOP 中, 两个智能体随机建立约束关系, 直到达到给定的约束图密度。对于约束代价函数, 每个赋值组合对应的代价都是随机地从给定的区间范围内选取。具体地, 该问题由如下参数定义:

- 1) 智能体个数 $|A|$, 定义问题中智能体的个数;
- 2) 约束图密度 p_1 , 定义问题中约束的密度;
- 3) 值域大小 $|D_i|$, 定义问题中每个变量的值域大小;
- 4) 代价范围 $[minCost, maxCost]$, 定义约束代价函数中代价取值范围。

② 随机非对称 MaxDCSP。其代表一般化的无结构问题, 是随机 ADCOP 的一个子集。其与随机 ADCOP 的区别在于: 每一个赋值组合对应的代价值是依据紧度从 0 或 1 中选取。其中, 1 代表违反该约束, 0 代表满足该约束。问题的目标是找到一组使得违反约束数最少的完整解。在该类问题中, 智能体个数、约束图密度、值域大小和代价范围的定义与随机 DCOP/ADCOP 相同。此外, 随机非对称 MaxDCSP 包含紧度 p_2 , 其定义每个赋值组合的代价值为 0 的概率。

③ 无尺度网络。该类问题是结构化的 DCOP，其约束图对应拓扑结构中节点度的分布为幂率分布。本文使用 BA 模型^[157]来生成无尺度网络问题。具体地，模型首先生成一个由 m_1 个节点组成的连通图，并迭代地向图中加入一个新的节点，该新的节点与图中的 m_2 个节点相连。这样一直迭代，直至所有节点都加图中。该类问题的智能体个数、值域大小和代价范围的定义与随机 DCOP/ADCOP 相同。

④ 分布式信道分配问题（Distributed Channel Assignment）。如 2.1.2 节所述，分布式信道分配问题是为一个无线局域网（WLAN，Wireless local Area Network）中的多个 AP（Access Point）设备分配信道，使得 WLAN 的网络性能最优（或受同信道干扰和相邻信道干扰的影响最小）。该问题可以看作加权图着色问题的一种特例，其约束代价函数值由相连 AP 设备的可分配信道决定，可由信噪比（SINR，Signal to Interference and Noise Ratio）公式计算获得。

⑤ 随机会议调度问题（Random Meeting Scheduling Problem）^[117,143]。会议调度问题由参会人集合和会议集合构成，其中参会人从会议集合中选择若干个会议参与，随后参会人通过协调确定每一个会议安排的时间。在该问题中，随机选择两个会议，并定义两个会议之间的时间间隔。参会人需要参加这两个会议，那么这两个会议的间隔时间应该小于参会人的行走时间（即参会人到达会场参会所需的时间）。若参会人无法按照时间要求参加会议，则会产生一定的约束代价，其代价值为无法正常参会的人数。

2.4.2 算法评价指标

在实验对比中，本文采用的评价指标如下：

① 消息数（Message number）指算法求解过程中所交换的全部消息的数量，该指标反映算法所需的网络通信负载。

② 网络负载（Network Load）是指算法在运行过程中所交换的全部消息大小的之和。同消息数一样，该指标反映算法所需的网络通信负载。

③ 非并发逻辑检查（Non-Concurrent Logical Checks，NCLOs）是非并发约束检查扩展。该指标是用于统计算法消息的处理时间，反映算法的计算负载。为计算该指标，每个智能体内部维护一个 NCLOs 变量。当一个智能体执行一次约束检查时，该变量就增加 1。

④ 运行时间（Run time）算法从开始到终止时在实验平台上模拟的运行时间。

⑤ 隐私损失（Privacy loss）。该指标反映算法在求解过程中所造成智能体私有约束代价函数信息的泄露程度。在 ADCOP 中，智能体持有私有约束代价函数信息用信息熵来量化，即：

$$H_i = - \sum_{a_j \in N(a_i)} \sum_{k \in S_{ij}} p_k \log_2 p_k \quad (2.5)$$

其中, S_{ij} 是 a_i 与其邻居智能体 a_j 之间所有可能的状态集合, p_k 是状态 $k \in S_{ij}$ 的概率。整个系统对应的信息量为所有智能体的信息熵之和。因此, 算法的隐私损失为算法开始前整个系统对应的信息量和算法运行结束后的信息量之差。

⑥ 最终解对应的归一化代价 (Normalized Costs)。该指标表示算法终止时获得解的代价与问题最优解对应的代价比例值, 用于评价非完备算法获得解的质量。

2.5 本章小结

本章首先给出 DCOP 和 ADCOP 的形式化定义。其次, 介绍必要的基础知识。最后, 介绍 DCOP 和 ADCOP 算法的测试问题和性能评价指标。

3 基于上文推理与搜索混合的 DCOP 完备算法^①

3.1 引言

基于搜索的 DCOP 完备算法（例如 SyncBB, PT-FB, ADOPT 等）通过执行分布式回溯搜索穷举解空间。这类算法可以利用上、下界来实现剪枝，从而减少需要探索的解空间，但剪枝效率与算法提供的上、下界紧度密切相关。但大部分基于搜索的 DCOP 完备算法仅利用本地信息来计算初始下界，导致算法剪枝效率低下，无法求解智能体数较多的问题。另一方面，基于推理的 DCOP 完备算法（例如 DPOP）可以快速地累积全局信息，但其内存消耗指数级于伪树的诱导宽度，从而使得这类算法无法求解密度较高的问题。

为克服上述缺陷，将搜索和推理两者优势结合实现 DCOP 求解是很自然的。即，首先通过执行内存有界的推理来构造有效的搜索下界，然后搜索利用推理提供的下界进行有效的剪枝。但现有的推理与搜索混合算法（例如 ADPOT-BDP 和 DJAO）仅在预处理阶段执行近似推理以构造下界。具体地，ADOPT-BDP 使用 ADPOP 算法来建立初始下界，而 DJAO 使用函数过滤技术来获得下界。这导致在内存受限的条件下，这些算法在预处理阶段中传播的效用表完整度不够，从而使得算法的剪枝效率依然很低。

基于上述分析，我们考虑在内存受限的条件下采用上文推理来为搜索提供更紧的下界。换句话说，在计算效用表的过程中，不是消除所有的“近似维度”使得传播的效用表维度小于预定的内存维度，而是对某些“近似维度”通过搜索提供的上文进行赋值以获得更完整的效用表。也就是说，我们旨在通过结合上文推理与搜索的优势来求解 DCOP。与现有的推理与搜索混合 DCOP 算法不同，本章提出的算法中，搜索为上文推理提供上文，随后上文推理为搜索提供紧的下界。

本章内容安排如下：首先，3.2 节本章所需的算法背景；随后，3.3 节详细地描述本章提出的上文推理与搜索混合 DCOP 完备算法 HS-CAI；接着，3.4 节从完备性、下界的紧度以及复杂度三个方面对提出算法进行理论分析，并在 3.5 节通过实验验证本章提出的 DCOP 完备算法的优势；最后，在 3.6 节对本章工作进行小结。

3.2 算法背景

本节将介绍与本章提出算法相关基础算法背景。在介绍具体算法之前，首先介绍算法会用到的一些定义。

^① 本章内容已被会议《AAAI Conference on Artificial Intelligence》录用

定义 2.1 (维度) 令 u 为一个效用表 (或约束代价函数), 则 $u.dims$ 为效用表 u 的维度, 即: 该效用表所涉及的变量集合。

定义 2.2 (切片) 令 P 为一组键值对, 则 $P_{[K]}$ 为 P 在变量集合 K 上的切片, 即:

$$P_{[K]} = \{(k, v) \in P \mid \forall k \in K\}$$

定义 2.3 (联合) 令 u 和 u' 为两个效用表, $D_u = \prod_{x_i \in u.dims} D_i$ 和 $D_{u'} = \prod_{x_i \in u'.dims} D_i$ 为它们的值域空间, 则 $u \otimes u'$ 为效用表 u 和 u' 在值域空间 $D_{u \otimes u'} = \prod_{x_i \in u.dims \cup u'.dims} D_i$ 上的联合, 即:

$$(u \otimes u')(p) = u(p_{[u.dims]}) + u'(p_{[u'.dims]}), \forall p \in \{((u \otimes u').dims, V) \mid \forall V \in D_{u \otimes u'}\}$$

定义 2.4 (最大、最小消元) 令 u 为一个效用表, S 为 u 维度的一个子集 (即 $S \subset u.dims$)。则 $u \perp_S$ 为 u 在维度集合 S 上的最大、最小消元后的结果, 即:

$$u \perp_S = \{u \perp_S^+, u \perp_S^-\}$$

其中, $u \perp_S^+$ 为 u 在维度集合 S 上的执行最大操作后的结果, $u \perp_S^-$ 为 u 在维度集合 S 上的执行最小操作后的结果, 即:

$$u \perp_S^+(p) = \max_{p_s \in P_{[S]}} u(p \cup p_s), \forall p \in P(u.dims \setminus S)$$

$$u \perp_S^-(p) = \min_{p_s \in P_{[S]}} u(p \cup p_s), \forall p \in P(u.dims \setminus S)$$

这里, $P = \{(u.dims, V) \mid \forall V \in \prod_{x_i \in u.dims} D_i\}$ 。为便于描述, 将公式 $u^+ = u'^+ \otimes u''^+$ 和 $u^- = u'^- \otimes u''^-$ 简写为 $u^\pm = u'^\pm \otimes u''^\pm$ 。为与现有算法统一, 将 $u \perp_S^-$ 写为 $\min_S u$ 。

3.2.1 分布式伪树优化过程及其近似版本

分布式伪树优化过程 (Distributed Pseudo tree Optimization Procedure, DPOP) 是桶消元算法的分布式实现。在 DPOP 中, 智能体在伪树上执行一个自下而上的效用传播阶段和一个自上而下的值传播阶段。

在效用传播阶段, 智能体 a_i 联合本地效用表 $localUtil_i$ 与收到的孩子节点效用表 $util_{c \rightarrow i}, \forall a_c \in C(a_i)$, 消除自身维度以获得传播给父节点的效用表 $util_{i \rightarrow p}$, 即:

$$util_{i \rightarrow p} = \min_{x_i} \left(localUtil_i \otimes \left(\bigotimes_{a_c \in C(a_i)} util_{c \rightarrow i} \right) \right)$$

这里, 我们将 a_i 消除自身维度的过程称为“本地消元”。随后, a_i 向上传播效用表 $util_{i \rightarrow p}$ 。这里, 本地效用表 $localUtil_i$ 是 a_i 与 $AP(a_i)$ 节点约束代价函数的累积, 即:

$$localUtil_i = \bigotimes_{a_j \in AP(a_i)} f_{ij}$$

在值传播阶段, a_i 根据父节点给出的最优部分解 PA^* 以及效用传播阶段收到的效用表 $util_{c \rightarrow i}, \forall a_c \in C(a_i)$ 确定自己的最优取值 d_i^* , 即:

$$d_i^* = \operatorname{argmin}_{x_i} \left(localUtil_i(PA^*) \otimes \left(\bigotimes_{a_c \in C(a_i)} util_{c \rightarrow i}(PA^*) \right) \right)$$

之后, a_i 用自己的最优取值扩展 PA^* , 并将扩展后的部分解传播给孩子节点。

尽管 DPOP 在问题求解过程中仅需要线性数量的消息数, 其最大的消息大小指大于伪树的诱导宽度, 这限制其求解较大规模问题的能力。因此, ADPOP (Approximate DPOP) 提出通过牺牲解的质量来降低算法计算传播效用表所需的内存消耗。具体而言, 在效用传播阶段, ADPOP 限制最大传播效用表中的维度大小为 k_{mb} 。当发送的效用表维度超过 k_{mb} 时, a_i 通过近似消元消除维度集合 S 以确保传播的效用表维度不大于 k_{mb} 。这里, a_i 传播的效用表 $util_{i \rightarrow p}^\pm$ 由如下公式计算。

$$util_{i \rightarrow p}^\pm = \left(\min_{\{x_i\}} \left(localUtil_i \otimes \left(\bigotimes_{a_c \in C(a_i)} util_{c \rightarrow i}^\pm \right) \right) \right) \perp_S$$

其中, $util_{c \rightarrow i}^\pm$ 为从孩子节点 $a_c \in C(a_i)$ 收到的效用表。

ADPOP 的值传播阶段与 DPOP 中的值传播阶段类似, 每个智能体 a_i 根据从父亲节点处接收到部分解 PA^* , 以及在效用传播阶段收到孩子节点的最大或最小效用表 $u_c \in \{util_{c \rightarrow i}^-, util_{c \rightarrow i}^+\}, \forall a_c \in C(a_i)$ 为自己选择最优赋值 d_i^* 。随后, a_i 用自己的最优赋值扩展 PA^* , 然后将该扩展后的部分解传播给孩子节点。

3.2.2 同步分支定界算法

同步分支定界算法 (Synchronous Branch and Bound, SyncBB) 是经典的 DCOP 完备搜索算法。该算法是在链式通信结构上分布式实现分支定界策略, 对解空间进行有序地展开, 并利用搜索上界对解空间进行剪枝, 直至探索完整个解空间。算法主要包含如下三部分。

① 智能体链式排序部分: 基于预定义的顺序 (例如按照智能体的下标顺序) 将智能体排列成一条链。除第一个节点外, 其他节点有且仅有一个前驱节点; 除最后一个节点外, 其他节点有且仅有一个后继节点。

② 分支定界部分: 节点接收其前驱节点的部分解以及对应的代价; 依据自己的值域对问题划分为多个子问题, 并取出其中一个赋值用于扩展部分解来实现分支部分; 同时, 计算该扩展部分解对应的预估下界, 并与当前的搜索上界进行比较来实现定界部分。若搜索上界大于智能体的预估下界, 则最优解不可能出现该分支对应的子问题中。否则, 通过将扩展后的部分解集对应的代价发送给后继节点, 直到最后一个节点。在最后一个节点时, 其可获得一个完整解, 因此用该完整解对应的代价更新搜索上界, 并开启回溯部分。

③ 回溯部分: 当节点探索完所有分支后, 其向前驱节点发送回溯消息, 请求前驱节点调用分布界定部分查找下一个可行的赋值。若下一个可行的赋值存在, 则该赋值用于扩展部分解, 发送给节点让其探索新的子问题。否则, 前驱节点也开启回溯部分, 直到第一个节点。但第一个节点的值域中不存在下一个可行的赋

值时时，算法结束。

基于树型结构的同步分支定界算法（Tree-based SyncBB，TreeBB）是 SyncBB 在伪树上的扩展版本。与 SyncBB 不同，TreeBB 采用伪树作为通信结构，在伪树的各个分支上并行地执行分支定界策略。

3.3 基于上文推理与搜索混合的 DCOP 完备算法

针对 3.1 节所述的问题，本章提出基于上文推理与搜索混合的 DCOP 完备算法（Hybrid Search Context-based Inference，HS-CAI）。该算法在伪树上依次执行预处理阶段和上文推理与搜索混合阶段。在预处理阶段，智能体执行定制版本的近似推理来传播近似消元维度和不完整的效用表。其中，近似消元维度为混合阶段的上文推理部分确定上文模式，不完整的效用表为混合阶段的搜索部分建立初始下界。在混合阶段，上文推理部分与搜索部分并行执行。其中，搜索部分采用 TreeBB 算法对解空间不重不漏地枚举并为推理部分提供上文；依据搜索部分提供的上文，上文推理部分迭代地传播基于上文的效用表，进而为搜索部分提供紧的下界。此外，为降低上文推理部分巨大的计算和通信消耗，本章引入上文评估机制为上文推理选择上文模式。

3.3.1 预处理阶段

预处理阶段采用内存维度限制为 k_{mb} 的 ADPOP 定制版本实现近似维度和不完整效用表在伪树上自下而上的累积与传播。具体地，当收到所有孩子节点的推理消息后，智能体 a_i 近似消除最高的祖先维度 S_i 使其内存消耗不超过 k_{mb} 。然后，通过由公式 (3.1) 和公式 (3.2) 计算从 a_i 发送到父节点的近似维度 $SList_i$ （即由 a_i 与其所有子孙节点近似的维度）和效用表 $preUtil_{P(a_i)}^i$ ，即：

$$SList_i = S_i \cup \left(\bigcup_{a_c \in C(a_i)} SList_i^c \right) \quad (3.1)$$

$$preUtil_{P(a_i)}^i = \min_{S_i \cup \{x_i\}} \left(localUtil_i \otimes \left(\bigotimes_{a_c \in C(a_i)} preUtil_i^c \right) \right) \quad (3.2)$$

这里， $SList_i^c$ 和 $preUtil_i^c$ 分别为孩子节点 $a_c \in C(a_i)$ 发送的近似维度和效用表。

$localUtil_i$ 表示 a_i 与（伪）父节点 $AP(a_i)$ 之间的约束代价函数联合，即：

$$localUtil_i = \bigotimes_{a_i \in AP(a_i)} f_{ij} \quad (3.3)$$

算法 3.1 HS-CAI 预处理阶段的伪代码

Algorithm 3.1 Sketch of preprocessing phase in HS-CAI

Preprocessing phase in HS-CAI for each agent a_i

输入：智能体 a_i 的约束函数集合

 Preprocessing phase in HS-CAI for each agent a_i

 输出：智能体 a_i 的效用表

When Initialization():

 1. **if** a_i is a leaf agent **then**

 2. $join_i \leftarrow copy(localUtil_i), SList \leftarrow \emptyset$

 3. **SendUtil()**
When received UTIL($preUtil_c, SList_c$) from $a_c \in C(a_i)$: //处理接收到的效用表

 4. $preUtil_i^c \leftarrow preUtil_c, join_i \leftarrow join_i \otimes preUtil_i^c$

 5. $SList_i^c \leftarrow SList_c, SList_i \leftarrow SList_i \cup SList_i^c$

 6. **if** a_i has received all UTIL from $C(a_i)$ **then**

 7. **if** a_i is the root **then**

 8. start **Hybird** phase

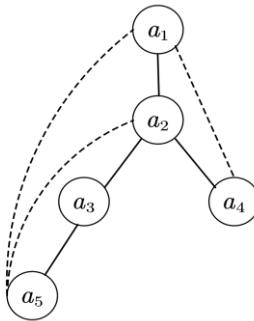
 9. **else**

 10. **SendUtil()**
Function SendUtil(): //效用表以及近似维度的计算与传播

 11. **if** $|dims(join_i)| > k$ **then**

 12. select $S_i \subset dims, s.t. |S_i| = |join_i| - k$

 13. $SList_i \leftarrow SList_i \cup S_i, join_i \leftarrow \min_{S_i} join_i$

 14. send UTIL($\min join_i, SList_i$) to $P(a_i)$
 x_i


(a) 伪树

$x_i \backslash x_j$	0	1	2	3
0	1	7	4	8
1	2	3	1	3
2	6	2	8	6
3	1	4	4	5

(b) 约束矩阵

图 3.1 说明 HS-CAI 算法运行的 DCOP 实例

Fig. 3.1 An example of a DCOP for HS-CAI

算法 3.1 给出 HS-CAI 算法在预处理阶段的伪代码。该阶段开始于叶子节点通过 UTIL 消息将本地效用表以及自身近似维度发送给父节点（1-3 行， 11-14 行）。

若发送的效用表维度超过内存维度限制（11 行），则采用近似消元消除超出的维度、合并被近似的维度（12-13 行）。当收到孩子节点发送的 UTIL 消息后， a_i 保存收到的效用表以及近似维度（4-5 行）。当收到所有孩子的 UTIL 消息后，若 a_i 为非根节点，则根据公式（3.1）和（3.2）计算并传播效用表 $preUtil_{P(a_i)}^i$ 以及近似维度 $SList_i$ 给父节点（11-14 行）。否则，开启上文推理与搜索混合阶段（7-8 行）。

以图 3.1 (a) 为例，假设 $k_{mb} = 1$ ，则在预处理阶段，智能体 a_4 消除的维度集合为 $\{x_1, x_2\}$ 。因此，从 a_4 发送到 a_3 的近似维度和效用表分别为 $SList_4^3 = \{x_1, x_2\}$ 和 $preUtil_4^3 = \min_{\{x_1, x_2, x_4\}}(f_{41} \otimes f_{42} \otimes f_{43})$ 。

3.3.2 上文推理与搜索混合阶段

上文推理与搜索混合阶段由搜索部分和上文推理部分组成。搜索部分执行 TreeBB 扩展可行的部分解并为推理部分提供上文。依据搜索部分提供的上文，推理部分迭代地传播基于上文的效用表，为搜索部分提供紧的下界。

一般而言，上文推理需要对所有近似维度（对于智能体 a_i 而言，所有近似维度为预处理阶段获得的 $SList_i$ ）的赋值执行推理，即：执行上文推理所对应的上文维度等于近似维度。例如，在 MB-DPOP 算法中，簇内节点需要对该簇 CC 节点的每个赋值组合迭代地执行上文推理。然而，这种策略不适用于本文提出的算法，具体原因如下：首先，CC 节点赋值组合的数量为指数级，若对所有的 CC 赋值执行上文推理将导致巨大的计算量和通信量开销；此外，在迭代推理中传播带一个特定赋值组合的效用表，该效用表很快就会失效。这是因为只要赋值组合中的任一变量赋值发生变化，就需要针对变化后的赋值组合执行一个上文推理，而且在搜索过程中这种变量赋值变化很频繁的现象很常见。

因此，我们通过让传播的效用表兼容更多的上文来减少上文推理数量。具体地，智能体 a_i 需要执行上文推理的赋值维度为 $PList_i \subseteq SList_i$ 。即，采用 $PList_i$ 特定的赋值作为上文模式，并且传播的效用表将覆盖所有匹配该模式的上文。这里， a_i 的上文模式 $ctxt_i$ 可由公式（3.4）计算获得。

$$ctxt_i = \{(x_j, Cpa_i(x_j)) | \forall x_j \in PList_i\} \quad (3.4)$$

其中， Cpa_i 是 a_i 收到的部分解， $Cpa_i(x_j)$ 是 x_j 的赋值。

以图 3.1 中的 a_3 为例，给定内存限制 $k_{mb} = 1$ ，有 $SList_3 = \{x_1, x_2\}$ 。假设 $PList_3 = \{x_1\}$ ，则上文推理部分只考虑维度 x_1 对应的赋值，而丢弃维度 x_2 （即 $SList_3 \setminus PList_3$ ）对应的赋值。进一步假设 $ctxt_3 = \{(x_1, 0)\}$ ，由于 $SList_3 = \{x_1, x_2\}$ ，可知上文模式 $ctxt_3$ 将覆盖如下四个上文： $\{(x_1, 0), (x_2, 0)\}$ ， $\{(x_1, 0), (x_2, 1)\}$ ， $\{(x_1, 0), (x_2, 2)\}$ 和 $\{(x_1, 0), (x_2, 3)\}$ 。因此， a_3 仅需要对 $ctxt_3$ 执行推理，而不是上面的四个上文。

选择上文推理模式需要权衡提供下界的紧度和兼容上文的数量。因此，确定一

一个好的上文推理模式具有极大挑战性。通过上述分析，一个好的上文模式应符合以下两方面的要求。首先，为减少不必要的上文推理，好的上文模式应该覆盖尽可能多的上文，即：好的上文模式包含的赋值在短期内不会发生变化；此外，好的上文模式还应确保上文推理部分能够提供更紧的下界。因此，本节提出上文评估机制，其出发点是：上文模式的选择依据为近似维度中每个维度对应赋值变化的频次。也就是说，上文模式由赋值变化频次大于指定阈值 t 的赋值组成，即：

$$PList_i = \{x_j | Cnt_i(\langle x_j, Cpa_i(x_j) \rangle) > t, \forall x_j \in SList_i\} \quad (3.5)$$

其中， $Cnt_i(\langle x_j, Cpa_i(x_j) \rangle)$ 是 x_j 赋值 $Cpa_i(x_j)$ 的变化频次。给定一个适当的 t ，上文模式中的赋值在短期内不会发生改变。另一方面，若该赋值难以被剪枝，则上文模式中将包含更多这样的赋值，这样就保证上文推理所提供下界的紧度。另外，引入布尔变量 $eval_i$ 来确保 a_i 的后代仅对一个上文模式进行推理。一旦 a_i 找到或收到一个上文模式，则 $eval_i$ 设置为 `false` 以停止上文评估机制。接下来，我们将介绍如何实现上文评估机制和上文推理部分。

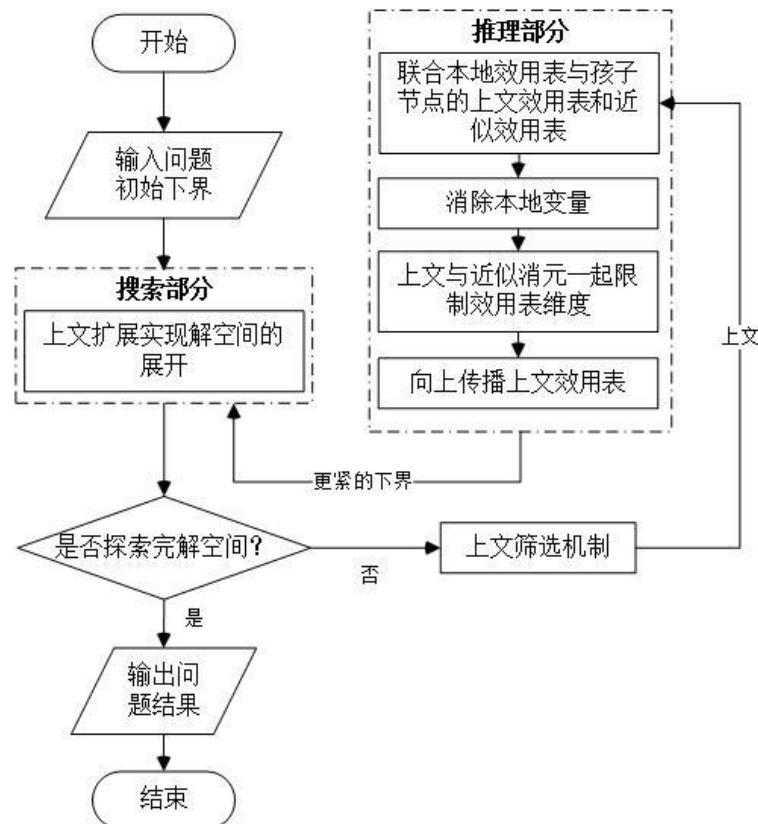


图 3.2 HS-CAI 混合阶段的流程图

Fig. 3.2 Flow chart of Hybrid phase in HS-CAI

算法 3.2 HS-CAI 混合阶段的伪代码

Algorithm 3.2 Sketch of hybrid phase in HS-CAI

Hybrid phase in HS-CAI for each agent a_i

输入：智能体 a_i 的值域、约束函数集合以及预处理阶段的效用表与被近似维度输出：智能体 a_i 最终的赋值**When Initialize():**

1. perform ADPOP as a preprocessing phase
2. **if** a_i is the root **then**
3. $eval_i \leftarrow true$
4. $d_i \leftarrow$ the first element in D_i , $Cpa \leftarrow (x_i, d_i)$
5. send CPA($Cpa, eval$) to $a_c \in C(a_i)$

When received CPA($Cpa, eval$) from $P(a_i)$:

6. $preCpa_i \leftarrow Cpa_i, Cpa_i \leftarrow Cpa, eval_i \leftarrow eval$
7. **InitBounds()**
8. **UpdateSCounter()**
9. **if** $eval_i$ **then** //开启推理上文的评估
10. $ctxt_i \leftarrow \{(x_j, Cpa_i(x_j)) | Cnt_i(\langle x_j, Cpa_i(x_j) \rangle) > t, \forall x_j \in SList_i\}$
11. **if** $|ctxt_i| > 0$ **then** //分配推理上文
12. **AllocateContext()**
13. $eval_i \leftarrow false$
14. $ctxt_i \leftarrow \emptyset$
15. //mark a_i as the starter of the context-base inference part
16. //original CPA message handler

When received CTXT($ctxt_i$) from $P(a_i)$:

17. **AllocateContext()**
18. **if** $|inferChild_i| = 0$ **then**
19. **SendCtxtUtil()**

When received CTXTUTIL($ctxtUtil_c$) from $a_c \in inferChild$:

20. $ctxtUtil_i^c \leftarrow ctxtUtil_c$
21. **if** received all CTXTUTIL from $inferChild_i$ and $|ctxt_i| > 0$ **then**
22. **SendCtxtUtil()**

Function Initbounds(): //初始化搜索下界

23. **foreach** $a_c \in C(a_i)$ **do**
-

Hybrid phase in HS-CAI for each agent a_i

```

24.      if received a CTXTUTIL that is compatible with  $Cpa_i$  from  $a_c$  then
25.           $lb_i^c(d_i) \leftarrow ctxtUtil_i^c(x_i = d_i, Cpa_i(Sep(a_c)))$ 
26.      else
27.           $lb_i^c(d_i) \leftarrow preUtil_i^c(x_i = d_i, Cpa_i(Sep(a_c)))$ 

Function UpdateSCounter(): //统计每个赋值的频数
28.      foreach  $x_j \in SList \wedge (x_i, d_i) \in Cpa_i$  do
29.          if  $preCpa_i(x_j) \neq Cpa_i(x_j)$  then
30.               $Cnt_i(\langle x_j, Cpa_i(x_j) \rangle) \leftarrow 1$ 
31.          else
32.               $Cnt_i(\langle x_j, Cpa_i(x_j) \rangle) \leftarrow Cnt_i(\langle x_j, Cpa_i(x_j) \rangle) + 1$ 

Function AllocateContext(): //将推理上下文分配给孩子节点
33.       $ctxt_i^c \leftarrow \{(x_j, d_j) | (x_j, d_j) \in ctxt_i, \forall x_j \in SList_i^c\}, \forall a_c \in C(a_i)$ 
34.       $inferChild_i \leftarrow \{a_c | |ctxt_i^c| > 0, \forall a_c \in C(a_i)\}$ 
35.      send CTXT( $ctxt_i^c$ ) to  $a_c, \forall a_c \in inferChild_i$ 

Function SendCtxtUtil(): //计算并发送上文推理结果
36.       $F \leftarrow localUtil_i \cup \{ctxtUtil_i^c | \forall a_c \in inferChild_i\}$ 
37.       $\cup \{preUtil_i^c | \forall a_c \in C(a_i) \setminus inferChild_i\}$ 
38.       $S'_i \leftarrow \{x_j | \forall x_j \in S_i \wedge (x_j, d_j) \notin ctxt_i\}$ 
39.       $join \leftarrow \bigotimes_{f \in F} f(ctxt_i)$ 
40.       $ctxtUtil_i \leftarrow \min_{S'_i \cup \{x_i\}} join$ 
41.      send CTXTUTIL( $ctxtUtil_i$ ) to  $P(a_i)$ 

```

图 3.2 和算法 3.2 分别给出上述过程的流程图和伪代码。这里，我们忽略搜索部分，因为它与 TreeBB 算法的求解过程类似。在预处理阶段完成之后（1 行），根节点初始化 $eval_i$ 为 true 来启动上文评估机制（3 行）。此外，它还通过 CPA 消息将其第一个可行的赋值发送孩子节点来开启搜索部分（4-5 行）。接收到 CPA 消息后， a_i 首先保存先前的 Cpa_i ，并存储当前收到的 Cpa_i 和 $eval_i$ （6 行），然后根据接收到的效用表来初始化每个孩子节点对应的下界（7 行，23-27 行）。具体地，若存在与 Cpa_i 兼容的基于上文的效用表， a_c 的下界则由该效用表计算获得；否则， a_c 的下界是由 a_c 在预处理阶段收到的效用表计算获得。接下来， a_i 根据收到的 Cpa_i 和前一个 Cpa_i 更新 Cnt_i （8 行，28-32 行）。具体地，对于 $SList_i$ 中的每个维度，如果其赋值与其先前的赋值不同，则清除其计数器（29-30 行）；否则，增加该计数器（31-32 行）。随后，若用于上文推理的上文模式尚未确定， a_i 通过查找 Cnt_i 确

定上文模式 $ctxt_i$ 。找到一个上文模式 $ctxt_i$ 后, a_i 将 $eval_i$ 设置为 true 并通过 CTXT 消息将 $ctxt_i$ 发送给需要执行上文推理的孩子节点 (10-15 行, 33-35 行)。这里, 发送给 a_c 的上文模式是基于 a_c 的近似维度 $SList^c$ 对 $ctxt_i$ 的切片。

当接收到 CTXT 消息后, 若存在需要执行上文推理的孩子节点, a_i 将收到的上文模式 $ctxt_i$ 分配给对应的孩子节点 (17 行)。否则, 它将基于上文的效用表向上传播 (18-19 行, 36-40 行)。这里, 基于上文的效用表由如下三个步骤进行计算: 首先, 联合本地效用表、 $inferChild_i$ 的基于上文效用表与其他孩子节点的效用表 (36 行); 然后, 采用 $ctxt_i$ 中的赋值来固定 S_i 中某些维度的值, 以提高基于上文效用表的完整性 (37 行); 最后, 消除剩余维度降低效用表的大小, 以确保效用表维度在 k_{mb} 之内 (38 行)。在收到所有 $inferChild_i$ 基于上文效用表之后, 若 a_i 不是上文推理的启动者, 则向上传播带上文的效用表 (21-22 行)。

考虑图 3.1 (c) 中的智能体 a_3 , 假定目前尚未确定上文模式并且有 $Cnt_3 = \{(\langle x_1, 0 \rangle, 1), (\langle x_2, 0 \rangle, 1)\}$, 给定 $t=1$, a_3 收到包含 $\{(x_1, 0), (x_2, 1)\}$ 的 CPA 信息之后, 我们有 $Cnt_3 = \{(\langle x_1, 0 \rangle, 2), (\langle x_2, 0 \rangle, 1)\}$ 和 $ctxt_3 = \{(x_1, 0)\}$ 。由于孩子节点 a_4 的近似维度为 $\{x_1, x_2\}$, 因此 a_3 将带有上文模式 $\{(x_1, 0)\}$ 的 CTXT 消息发送给 a_4 。当接收到 $\{(x_1, 0)\}$ 后, a_4 将基于上文的效用表 $ctxtUtil_3^4 = \min_{\{x_2, x_4\}} (f_{41}(x_1 = 0) \otimes f_{42} \otimes f_{43})$ 发送给 a_3 。 a_3 在接收到 $\{(x_1, 0), (x_2, 2)\}$ 或 $\{(x_1, 0), (x_2, 3)\}$ 的 CPA 消息后, 使用 $ctxtUtil_3^4$ 计算 a_4 的下界。

3.4 理论分析

本节首先证明上文推理获得的下界紧度不小于预处理阶段获得的下界紧度, 以此说明 HS-CAI 中上文推理部分的有效性; 随后, 证明 HS-CAI 的完备性; 最后, 给出 HS-CAI 算法的复杂度分析。

3.4.1 提供下界紧度分析

引理 3.1: 给定 Cpa_i , 基于上文的效用表 $ctxtUtil_i^c$ 生成的下界至少与预处理阶段获得的效用表 $preUtil_i^c$ 生成的下界一样紧。即, $ctxtUtil_i^c(PA) \geq preUtil_i^c(PA)$, 其中 $PA = Cpa_{i[Sep(a_c)]} \cup \{(x_i, d_i)\}$ 。

证明: 由算法 3.2 的第 23 行可知, 上文推理部分中消除的维度 S'_c 可被定义为:

$$S'_c = \{x_j | (x_j, d_j) \notin ctxt_c, \forall x_j \in S_c\}$$

其中, $ctxt_c$ 是 a_c 用于上文推理的上文模式, S_c 是 a_c 在预处理阶段消除的维度。由于 a_i 从 a_c 收到 $ctxtUtil_i^c$, 从算法 3.2 的 20-21 行和 6-7 行可知 $|ctxt_c| > 0$ 。因此, 有 $S'_c \subset S_c$ 。

接下来, 将通过归纳法证明引理 3.1。

归纳基础: a_i 的孩子是叶子节点, 对于所有孩子节点 $a_c \in C(a_i)$, 有:

$$\begin{aligned}
 ctxtUtil_i^c(PA) &= \left(\min_{S'_c \cup \{x_c\}} localUtil_c \right) (PA) \\
 &= \min_{x_c} \left(\sum_{x_j \in AP(a_c) \setminus S'_c} f_{cj}(x_c, d_j) + \sum_{x_j \in S'_c} \min_{x_j} f_{cj}(x_c, x_j) \right) \\
 &= \min_{x_c} \left(\sum_{x_j \in AP(a_c) \setminus S_c} f_{cj}(x_c, d_j) + \sum_{x_j \in S_c} \min_{x_j} f_{cj}(x_c, x_j) \right. \\
 &\quad \left. + \sum_{x_j \in S_c \setminus S'_c} \left(f_{cj}(x_c, d_j) - \min_{x_j} f_{cj}(x_c, x_j) \right) \right) \\
 &\geq \min_{x_c} \left(\sum_{x_j \in AP(a_c) \setminus S_c} f_{cj}(x_c, d_j) + \sum_{x_j \in S_c} \min_{x_j} f_{cj}(x_c, x_j) \right) \\
 &= \left(\min_{S_c \cup \{x_c\}} localUtil_c \right) (PA) \\
 &= preUtil_i^c(PA)
 \end{aligned}$$

其中， d_j 是 PA 中 x_j 的赋值。由于 $S'_c \subset S_c$ ，上式第三步到第四步成立。因此，归纳基础成立。

现假设上述命题对于 a_i 的所有孩子 $a_c \in C(a_i)$ 成立。接下来，将证明该命题对于 a_i 也成立。对于每个 $a_c \in C(a_i)$ ，有：

$$\begin{aligned}
 ctxtUtil_i^c(PA) &= \left(\min_{S'_c \cup \{x_c\}} \left(\begin{array}{l} localUtil_c + \sum_{a_{c'} \in inferChild_c} ctxtUtil_{c'}^{c'} \\ + \sum_{a_{c'} \in C(a_c) \setminus inferChild_c} preUtil_{c'}^{c'} \end{array} \right) \right) (PA) \\
 &\geq \left(\min_{S_c \cup \{x_c\}} \left(\begin{array}{l} localUtil_c + \sum_{a_{c'} \in inferChild_c} ctxtUtil_{c'}^{c'} \\ + \sum_{a_{c'} \in C(a_c) \setminus inferChild_c} preUtil_{c'}^{c'} \end{array} \right) \right) (PA) \\
 &\geq \left(\min_{S_c \cup \{x_c\}} \left(localUtil_c + \sum_{a_{c'} \in C(a_c)} preUtil_{c'}^{c'} \right) \right) (PA) \\
 &= preUtil_i^c(PA)
 \end{aligned}$$

其中， $inferChild_c$ 是 a_c 需要执行上文推理的孩子节点。据此，定理证明完毕。

3.4.2 完备性证明

引理 3.2: HS-CAI 算法在有限次数的迭代后会终止

证明：从伪代码可知，预处理阶段只需要线性数量的消息便终止。此外，由于基于上文的迭代推理次数是严格小于节点收到的 Cpa 个数。因此，HS-CAI 算法的终止取决于搜索部分（采用 TreeBB 算法求解思路）。为证明算法能终止，只需要证明相同的部分解并不会在搜索阶段被探索两次即可，即一个智能体不会收到两个相同的 Cpa 。由于根节点不会收到任何 Cpa ，因此该命题在根节点处成立。当一个节点 a_i 收到来自父节点 $P(a_i)$ 发送的 Cpa ，它将发送多个 Cpa 给其孩子节点 $a_c \in C(a_i)$ 。由于每一个 Cpa 所包含 a_i 的赋值均不同，因此孩子节点 $a_c \in C(a_i)$ 不会收到两个相同的 Cpa 。因为根节点不会收到 Cpa ，且对于一个部分解，任意一个节

点发送给孩子节点的 Cpa 都是不同的。因此，算法终止性得以保证。

引理 3.3： 给定最优解 X^* ，以 $a_c \in C(a_i)$ 为根的子问题代价 $cost_c(X^*)$ 不小于下界 $lb_i^c(X^*(x_i))$ ，即 $cost_c(X^*) \geq lb_i^c(X^*(x_i))$ 。

证明：由于在引理 3.1 中，已经证明上文推理部分的效用表构造的下界至少与预处理阶段的效用表构造的下界一样紧。因此，为证明此引理，只需证明 $cost_c(X^*) \geq ctxtUtil_i^c(X^*)$ 。接下来，将通过数学归纳法证明引理 3.3。

归纳基础：当 a_i 的孩子都是叶子节点时，对于每一个孩子节点 $a_c \in C(a_i)$ ，有：

$$\begin{aligned} cost_c(X^*) &= \sum_{a_j \in AP(a_c)} f_{cj}(d_c^*, d_j^*) \\ &= localUtil_c(X^*) \\ &\geq \left(\min_{S'_c \cup \{x_c\}} localUtil_c \right) (X^*) \\ &= ctxtUtil_i^c(X^*) \end{aligned}$$

其中， d_l^* 是 X^* 在 x_l 中的赋值，而 S'_c 是在上文推理部分中 a_c 消除的维度。因此，归纳基础成立。

现假设上述命题对所有 $a_c \in C(a_i)$ 成立。接下来，将证明该命题对于 a_i 也成立。对于每个孩子 $a_c \in C(a_i)$ ，有：

$$\begin{aligned} cost_c(X^*) &= \sum_{a_j \in AP(a_c)} f_{cj}(d_c^*, d_j^*) + \sum_{a_{c'} \in C(a_c)} cost_{c'}(X^*) \\ &= localUtil_c(X^*) + \sum_{a_{c'} \in C(a_c)} cost_{c'}(X^*) \\ &\geq localUtil_c(X^*) + \sum_{a_{c'} \in C(a_c)} ctxtUtil_{c'}^{c'}(X^*) \\ &= \left(localUtil_c + \sum_{a_{c'} \in C(a_c)} ctxtUtil_{c'}^{c'} \right) (X^*) \\ &\geq \left(\min_{S'_c \cup \{x_c\}} \left(localUtil_c + \sum_{a_{c'} \in C(a_c)} ctxtUtil_{c'}^{c'} \right) \right) (X^*) \\ &\geq \left(\min_{S'_c \cup \{x_c\}} \left(localUtil_c + \sum_{a_{c'} \in inferChild_c} ctxtUtil_{c'}^{c'} + \sum_{a_{c'} \in C(a_c) \setminus inferChild_c} preUtil_{c'}^{c'} \right) \right) (X^*) \\ &= ctxtUtil_i^c(X^*) \end{aligned}$$

由此，上述命题成立。

定理 3.1： HS-CAI 是完备算法。

证明：从引理 3.2 中可知，HS-CAI 算法在有限次数的迭代后会终止。此外，从引理 3.3 中可知，在 HS-CAI 中最优解将不会被修剪。因此，HS-CAI 算法是完备的。

3.4.3 复杂度分析

算法的复杂度主要体现在四个方面：1) 智能体的存储复杂度；2) 智能体的计

算复杂度；3) 消息大小；4) 消息数量。

在智能体的存储复杂度方面：智能体 a_i 需要存储在预处理阶段获得的推理结果 $preUtil_i^c, \forall a_c \in C(a_i)$ 与上文推理部分获得的推理结果 $ctxtUtil_i^c, \forall a_c \in C(a_i)$ ，而这些推理结果的最大内存占用与子树的诱导宽度 w^* 和内存限制 k_{mb} 成指数级关系。此外，在搜索部分， a_i 还需保存已探索的搜索结果，如： $lb_i^c(d_i)$ 、 $lb_i(d_i)$ 、 Cpa_i 等。因此， a_i 的整体存储复杂度为 $O(|C(a_i)| * d_{\max}^{\min(k_{mb}, |Sep(a_i)| + 1)} + (|C(a_i)| + 1) * |D_i|)$ 。这里， $d_{\max} = \max_{x_i \in X} |D_i|$ 。

在智能体的计算复杂度方面：在预处理阶段/上文推理部分，智能体处理一条 UTIL/CTXUTIL 消息的计算复杂度为 $O(d_{\max}^{\min(k_{mb}, |Sep(a_i)| + 1)})$ 。在搜索阶段，当智能体收到一条 CPA 消息后会去找自己第一个可行的赋值。因此， a_i 处理一条 CPA 消息的计算复杂度为 $O(|D_i| * |AP(a_i)|)$ 。当智能体收到来自子节点的 BACKTRACK 回溯消息后， a_i 会更新搜索上界，并为孩子节点找到下一个可行的赋值，因此其处理回溯消息的整体计算复杂度为 $O(|D_i| * |AP(a_i)|)$ 。

对于算法产生的消息大小而言：由于每个 UTIL / CTXTUTIL 消息的维度并不会超过 k_{mb} ，因此其大小为 $O(d_{\max}^{\min(k_{mb}, |Sep(a_i)| + 1)})$ 。对于 CPA 消息，其包含对每个智能体的赋值和搜索上界，因此其大小为 $O(|A|)$ 。其它消息包括 BACKTRACK 和 TERMINATE 都只携带一些标量，因此只需要 $O(1)$ 的空间。

对于算法所需的消息数而言：不同于 DPOP/ADPOP 算法，由于没有值传播阶段，因此 HS-CAI 算法在预处理阶段仅需要 $|A| - 1$ 的消息数。与其它基于搜索的完备算法一样，HS-CAI 算法在搜索部分产生 $O(d_{\max}^{|A|})$ 数量的消息。由于上文推理部分中迭代推理的次数依赖于搜索部分的 CPA 消息数且与近似的维度大小相关（即 $w^* - k_{mb}$ ），因此 HS-CAI 在上文推理部分所产生的消息数量是 $O(d_{\max}^{w^* - k_{mb}})$ 。

3.5 实验评估

本节首先实验观察上文评估机制中参数 t 对 HS-CAI 算法的影响。然后，在随机 DCOP、无尺度网络问题上对 HS-CAI 算法与目前最好的 DCOP 完备算法（包括 PT-FB，DPOP 和 RMB-DPOP）进行性能比较。为证明上文推理能提供紧的下界，与不带上文推理部分的 HS-CAI（即 HS-AI）进行对比；为证明上文评估机制的优势，也与不带上文评估机制的 HS-CAI（即 HS-CAI(-M)）进行试验对比。此外，考虑 HS-CAI 算法中参数 t 与 h 和 d_{\max} 相关， h 是伪树的高度。因此，设置 $t = (d_{\max})^{\rho^* h}$ 。并且，对于 RMB-DPOP, HS-AI, HS-CAI(-M) 和 HS-CAI，选择 $k_{mb} = 6$ 和 $k_{mb} = 10$ 作为低和高内存配置。

3.5.1 参数调优

实验一：不同密度下，对 HS-CAI 算法上文评估机制中参数 ρ 的调参实验。该实验的目的是从增加约束图密度的角度增加问题的复杂性，测试不同参数下 HS-CAI 算法的性能比较，最终为算法选择性能最好的一组参数，同时验证上文评估机制的有效性。具体地，我们选择值域大小为 3 的随机 DCOP 问题，设置代价选取范围为 $[0, 100]$ ，固定智能体个数为 22，并将密度按步长 0.1，从 0.2 变到 0.6。设置 ρ 以步长 0.2 从 0.05 变化到 0.65。由于 ρ 大于 0.65 获得的实验结果与 $\rho = 0.65$ 的结果完全相同，因此没有显示 ρ 大于 0.65 的实验结果。

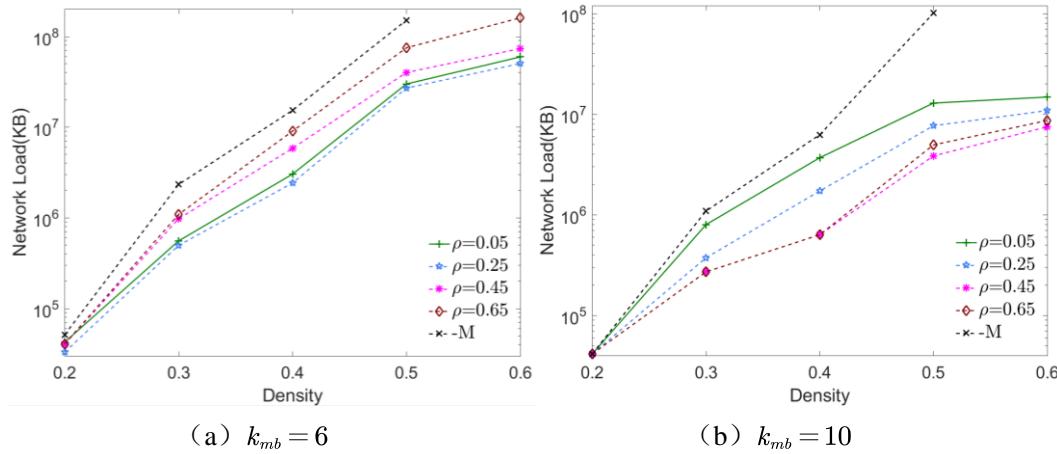


图 3.3 不同密度下变化参数 ρ ，HS-CAI 算法产生的网络负载对比

Fig.3.3 Network load of varying ρ on different densities

图 3.3 给出实验一下，HS-CAI (-M) 和 HS-CAI 的网络负载。从图中可以看出，HS-CAI 所需的网络负载比 HS-CAI (-M) 小得多。这是因为 HS-CAI 仅对上文评估机制选择的上文模式执行推理，而 HS-CAI (-M) 需要对所有上文执行推理。此外，在给定内存限制 k_{mb} 的情况下，可以观察到 HS-CAI 产生的网络负载不会一直随着 ρ 的增加而减少。这是由于增大 ρ 会导致较大的 t ，从而会减少上文推理的数量，但也会在一定程度上降低上文推理提供下界的紧度。如上文 3.2.2 节所述，本文提出的上文评估机制实际上是在提供下界的紧度和兼容的上文数量之间进行权衡。此外，从图中可以看出对于 HS-CAI ($k_{mb} = 6$)， ρ 的最优值接近 0.25，而对于 HS-CAI ($k_{mb} = 10$)， ρ 的最优值接近 0.45。因此，在 3.4.2 节的算法性能对比实验中，对于 HS-CAI ($k_{mb} = 6$)，选择 ρ 为 0.25；对于 HS-CAI ($k_{mb} = 10$)，选择 ρ 为 0.45。

3.5.2 实验结果与分析

实验二：稀疏配置中不同智能体数下，DCOP 完备算法的性能比较实验。具体

地, 我们选择值域大为 3 的随机 DCOP 问题, 代价选取范围为[0,100], 密度为 0.25, 并将智能体个数按步长 2, 从 22 变化到 32。

图 3.4 给出实验二的实验结果。从图中可以看出, 尽管采用搜索与推理混合的 DCOP 完备算法(例如 HS-AI 和 HS-CAI)和 PT-FB 都使用搜索策略来寻找最优解, HS-AI 和 HS-CAI 在网络负载和消息数量方面均优于 PT-FB。这是因为 PT-FB 在计算下界的过程时仅考虑已赋值智能体相关的约束, 导致其无法提供紧的下界进而剪枝效率较低。同样, 在固定的 k_{mb} 的情况下, HS-CAI 比 HS-AI 所需的消息更少, 这是因为上文推理所产生的下界比近似推理所获得的下界更紧。此外, 从图中还可以看出 HS-CAI ($k_{mb} = 6$) 能解决比 HS-AI ($k_{mb} = 6$) 和基于推理的完备算法(例如 DPOP 和 RMB-DPOP) 较大规模的问题, 这也说明当内存预算相对较低时, 搜

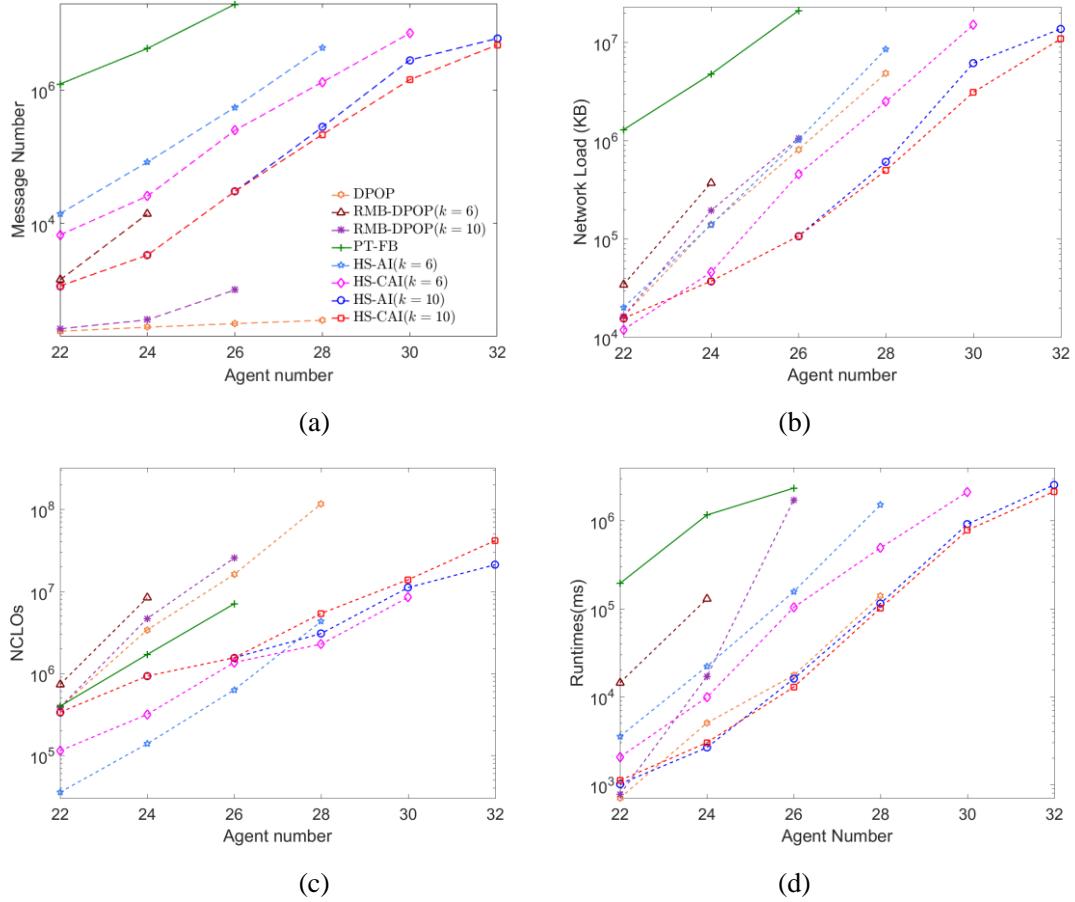


图 3.4 不同智能体数 (稀疏配置) 下各个算法性能对比

Fig.3.4 Performance comparison under different agent numbers on spare configuration

索与上文推理混合求解 DCOP 的必要性。此外, 尽管基于推理的完备算法会产生比基于搜索的完备算法更大的消息, 但从图 3.4 (b) 可以看出, HS-CAI 的网络负

载比 HS-AI 少得多。这表明在 HS-CAI 中，有效的剪枝和上文评估机制可以降低消息数从而快速地找到最优解。尽管迭代推理产生指数级的计算开销，但从图 3.4 (c) 可以发现当解决问题的智能体数为 28 时，HS-CAI ($k_{mb} = 6$) 所需的 NCLOs 比 HS-AI ($k_{mb} = 6$) 少。这是因为 HS-CAI ($k_{mb} = 6$) 可以提供紧的下界以加快搜索速度，从而在很大程度上降低求解较大规模问题时算法所产生的约束检查。

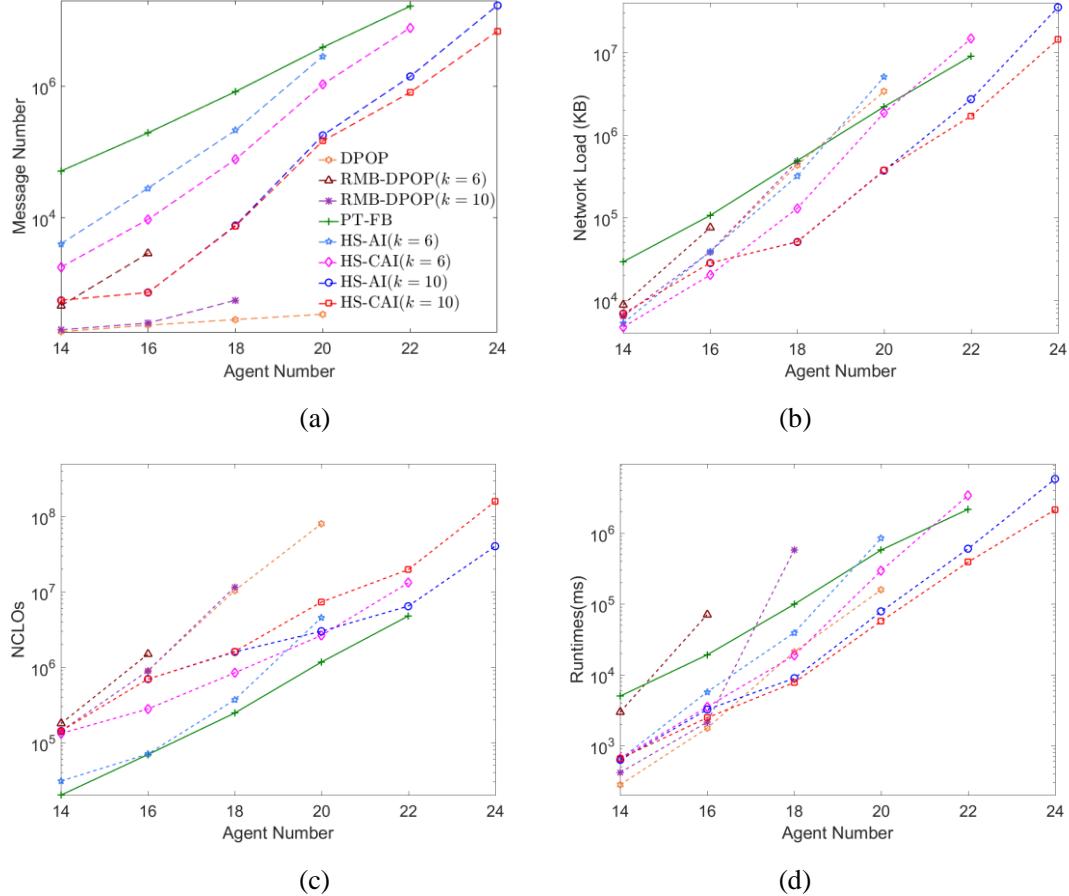


图 3.5 不同智能体数（稠密配置）下各个算法性能对比

Fig.3.5 Performance comparison under different agent numbers on dense configuration

实验三：稠密配置中不同智能体数下，DCOP 完备算法的性能比较实验。该项实验的目的是测试稠密配置的不同智能体数下，各个算法的性能。该实验中，我们设置密度为 0.6，并将智能体个数按步长 2，从 22 变化到 32，其他的实验配置与实验二中的配置相同。

图 3.5 给出实验三的实验结果。实验中，伪树的平均诱导宽度为 8~18。由于问题生成的伪树诱导宽度较大，DPOP 和 MB-DPOP 无法求解智能体数大于 20 的问题。此外，由于基于推理的完备算法必须探索整个解空间，这些算法比其他基于

搜索的完备算法需要更多的 NCLOs。尽管 HS-CAI 和 MB-DPOP 都需要执行上文推理，但从图中可以看出在网络负载和 NCLOs 等评价指标上，HS-CAI 相对于 RMB-DPOP 具有较大优势。这是因为 HS-CAI 仅对上文评估机制提取的上文模式进行推理，而 RMB-DPOP 需要针对 CC 节点的所有上文进行迭代地推理。对于具有不同 k_{mb} 值的 HS-CAI 而言，HS-CAI ($k_{mb} = 10$) 比 HS-CAI ($k_{mb} = 6$) 需要更少的消息数，但需要更多的 NCLOs。这是因为 HS-CAI ($k_{mb} = 10$) 比 HS-CAI ($k_{mb} = 6$) 能提供更紧的下界，但同时也会造成更大的计算开销。

实验四：不同密度下，各个 DCOP 完备算法的性能比较实验。该实验的具体配置与实验一中的配置相同。图 3.6 展示实验四的实验结果，其中实验所建立伪树的平均诱导宽度为 8 到 16。从图中可以看出，现有的单纯采用搜索或单纯采用推理

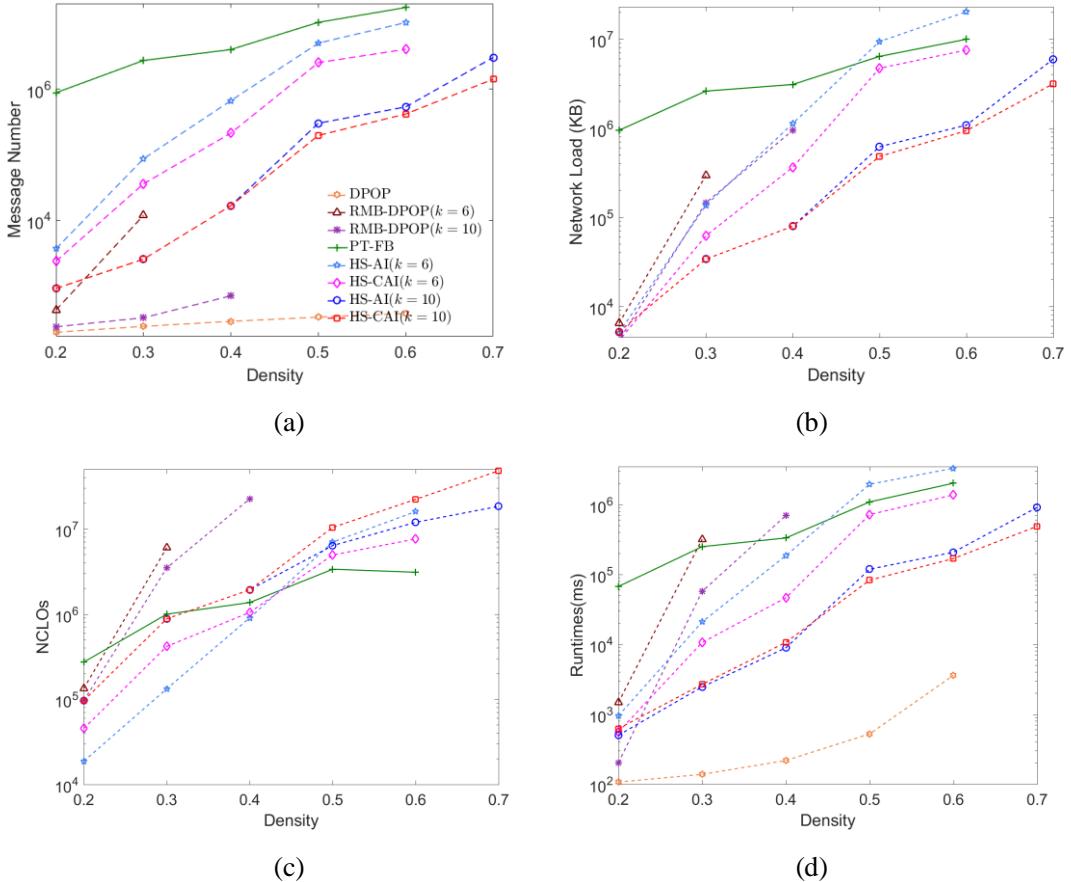


图 3.6 不同密度下各个算法性能对比

Fig.3.6 Performance comparison under different densities

策略 DCOP 完备算法无法求解密度大于 0.5 的随机 DCOP。其原因如下：1) 对于采用纯推理策略的算法（例如 DPOP）而言，其最大的消息大小与所建立伪树的诱导宽度成指数级关系。而在求解高密度问题时，其所建立伪树的诱导宽度往往较

大，如在求解密度为 0.6 的问题时，所建立伪树的平均诱导宽度为 14。因此，算法需要巨大的内存消耗来计算推理的效用表。2) 对于采用纯搜索策略的算法（例如 PT-FB），虽然其消息大小仅与智能体数线性相关，其求解问题所需的消息数是指数于智能体个数。但在求解密度变化的情况下，其智能体个数未发生改变。随着密度的增加，纯采用搜索策略的 DCOP 完备算法（PT-FB 中的前向定界过程）获得下界的紧度也在降低。从而，在求解高密度的问题时，算法的剪枝效率低，需要探索更多的解空间。虽然，采用推理与搜索混合求解思路的 DCOP 完备算法也采用搜索策略来穷举解空间，但得益于推理（尤其是在内存限制较大时，如 $k_{mb} = 10$ ）提供紧的下界，这些算法的剪枝效率实际上要比纯搜索策略的完备算法要高。这种现象也说明搜索与推理混合求解的优势。

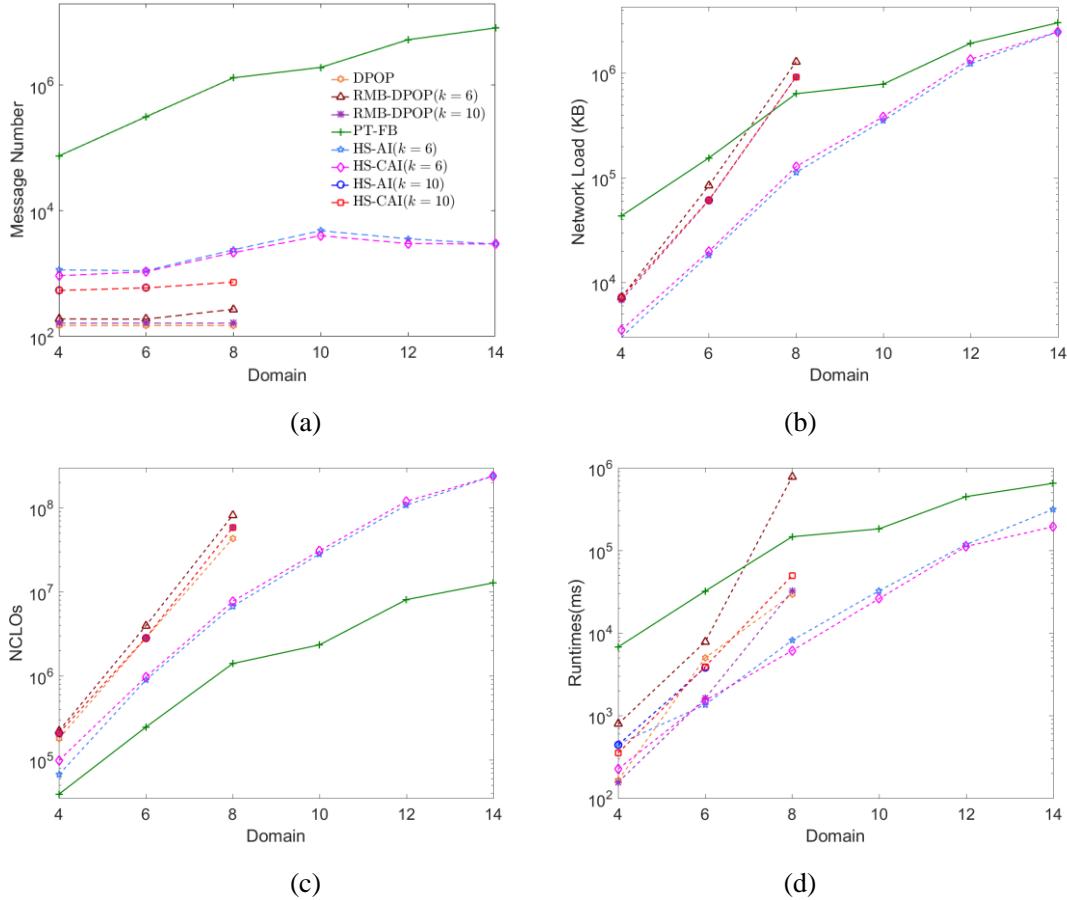


图 3.7 不同值域下各个算法性能对比

Fig.3.7 Performance comparison under different domain sizes

实验五：不同值域下，各个 DCOP 完备算法的性能比较实验。具体地，我们选择密度为 0.4 和智能体数为 14 的随机 DCOP 问题，代价选取范围为 [0, 100]，并将

值域大小按步长 2, 从 4 变化到 14。

图 3.7 给出在实验五的实验结果。从图中可以看出, 随着值域大小的增加, 所有算法的通信和计算开销都呈指数增长。其中, 采用纯搜索策略的 DCOP 完备算法 PT-FB 与采用(上文)推理与搜索混合的 DCOP 完备算法 HS-AI ($k_{mb} = 6$) 与 HS-CAI ($k_{mb} = 6$) 能求解该配置下的所有问题, 其他算法仅能求解值域大小最大为 8 的问题。这是由于基于推理完备算法产生的最大消息是指数级的, 其中基数是值域大小。因此, 它们在求解值域较大的问题时会耗尽运行内存。尽管在 RMB-DPOP ($k_{mb} = 6$) 中, 最大消息的维度也仅仅只是 6, 但由于该算法需要对 CC 节点的每一个赋值组合进行迭代的上文的推理, 因此同 DPOP 算法一样需要指数级的通信和计算负载。在图中可以看出, HS-CAI ($k_{mb} = 6$) 在消息数和算法的运行时间上略优于 HS-AI ($k_{mb} = 6$), 但其在 NCLOs 和网络负载上表现要差于 HS-AI ($k_{mb} = 6$)。这种现象说明相比于预处理中的近似推理提供的下界, HS-CAI 算法中的上文推理能够提供更紧的下界, 同时上文推理结果的计算也需要巨大的网络负载消耗和计算量。图中也可以看出, 基于(上文)推理与搜索混合的 DCOP 完备算法 HS-CAI ($k_{mb} = 6$) 与 HS-AI ($k_{mb} = 6$) 在消息数、网络负载和运行时间等指标上明显优于 PT-FB, 但它们这些算法在 NCLOs 上却远差于 PT-FB。这是因为基于搜索与推理混合的算法中, 推理阶段需要大量的计算量和通信负载计算效用表为搜索提供下界。而 PT-FB 仅需依靠前向定界过程来实现下界的计算, 而执行一次前向定界过程所需的计算量却是线性的。

实验六: 不同算法在求解无尺度网络问题时的性能对比。该实验的目的是对比各个 DCOP 完备算法在求解结构化问题时的性能。具体地, 我们选择值域大小均为 3 的无尺度网络问题, 代价选取范围为 [0, 100], 固定智能体个数为 26, m_0 为 10, 并将 m_1 按步长 2, 从 2 变到 10。

图 3.8 给出实验六的实验结果。从图中可以看出采用纯搜索策略的 DCOP 完备算法(例如 PT-FB)能求解 $m_1 \leq 4$ 的无尺度网络问题。这主要是因为无尺度网络问题配置下的智能体个数为 26, 而现有的纯搜索策略的 DCOP 完备求解能求解的问题规模都较小。如图 3.4 和图 3.5 所示, 在低密度下 PT-FB 最大能求解的问题的智能体个数为 26, 在高密度下 PT-FB 最大能求解的问题的智能体个数为 22。此外, 随着 m_1 的增加, 问题的密度也逐渐增加。但除本章提出的 HS-CAI ($k_{mb} = 10$) 算法外, 其他算法最大仅能求解 $m_1 = 6$ 的无尺度网络问题。这是因为上文推理获得的下界要基于在预处理阶段采用近似推理获得的下界, 也说明上文推理与搜索混合的优势和必要性。

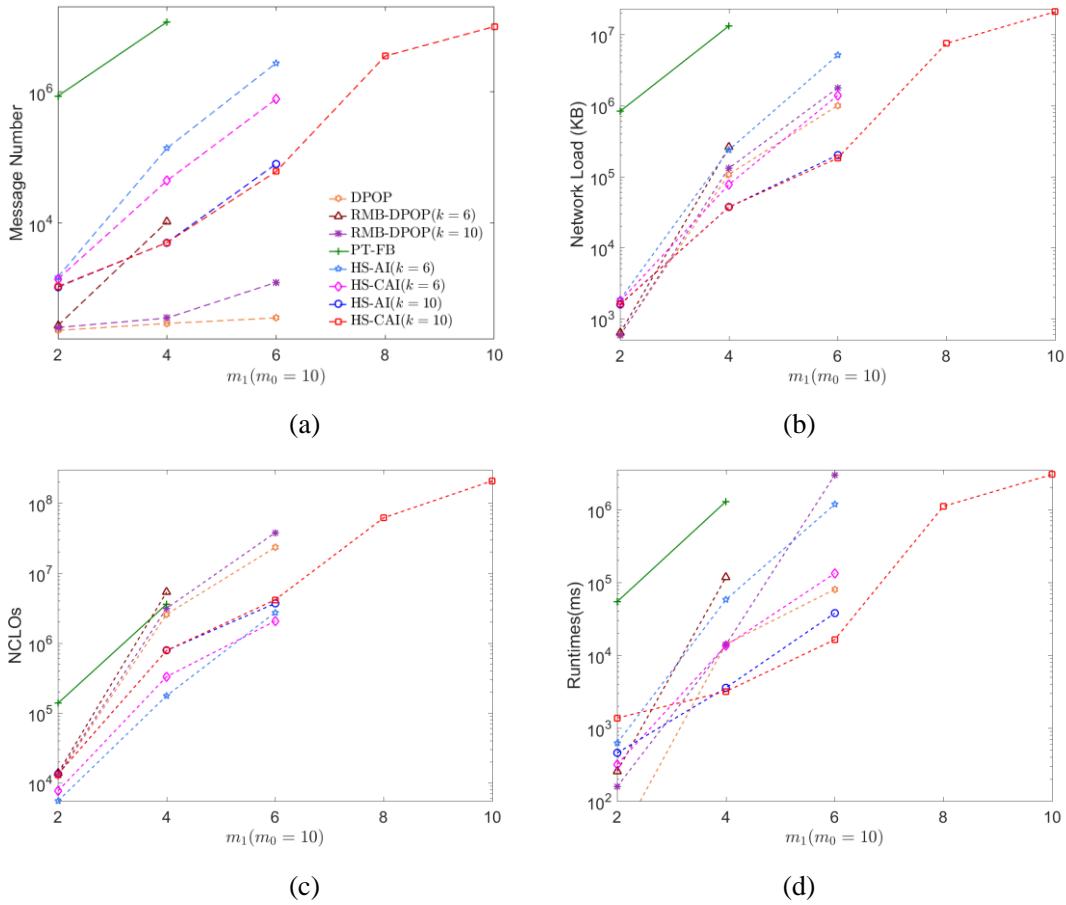


图 3.8 各个算法在求解无尺度网络问题时的性能对比

Fig. 3.8 Performance comparison on scale-free networks

实验七：不同算法在求解分布式信道分配问题时的性能对比。该实验的目的是对比各个 DCOP 完备算法在求解实际应用问题时的性能。具体地，我们设置智能体数按步长为 2，从 20 变化到 30；固定每个 AP 设备可分配的信道区间（或每个变量的值域）为 {1,6,11}；网络拓扑结构的密度为 0.2。

图 3.9 给出在不同智能体数下各种算法在信道分配问题上的运行结果。从图 3.9 (a) 中可以看出，尽管我们提出的算法 HS-CAI 与 PT-FB 算法都采用搜索策略获得问题的最优解，但 HS-CAI 所需的消息数近似于线性级的（与完备推理算法 DPOP 的增长趋势类似），而 PT-FB 算法所需的消息数却随着智能体的增加呈指数级增长。这是由于信道分配问题中约束矩阵的特殊结构造成的。基于 2.1.2 节可知，由于本实验设置变量的值域为 {1,6,11}，问题中的约束矩阵由 0、0.008、1 这三个数构成，且该矩阵具有对称性。因此，推理阶段的效用表能提供较紧的下界，进而极大地加速搜索过程。正如图 3.9 (b) - (d) 所示，即使在内存预算较低（即 $k=6$ ）时，推理阶段的效用表能提供下界的紧度对于搜索阶段已经足够，所以内存预算

的提升（即 $k=10$ ）反而降低了 HS-CAI 算法的求解性能。因此，在对该问题的求解中，也需要针对问题规模对内存预算进行调优，进而选择出最优的算法参数。

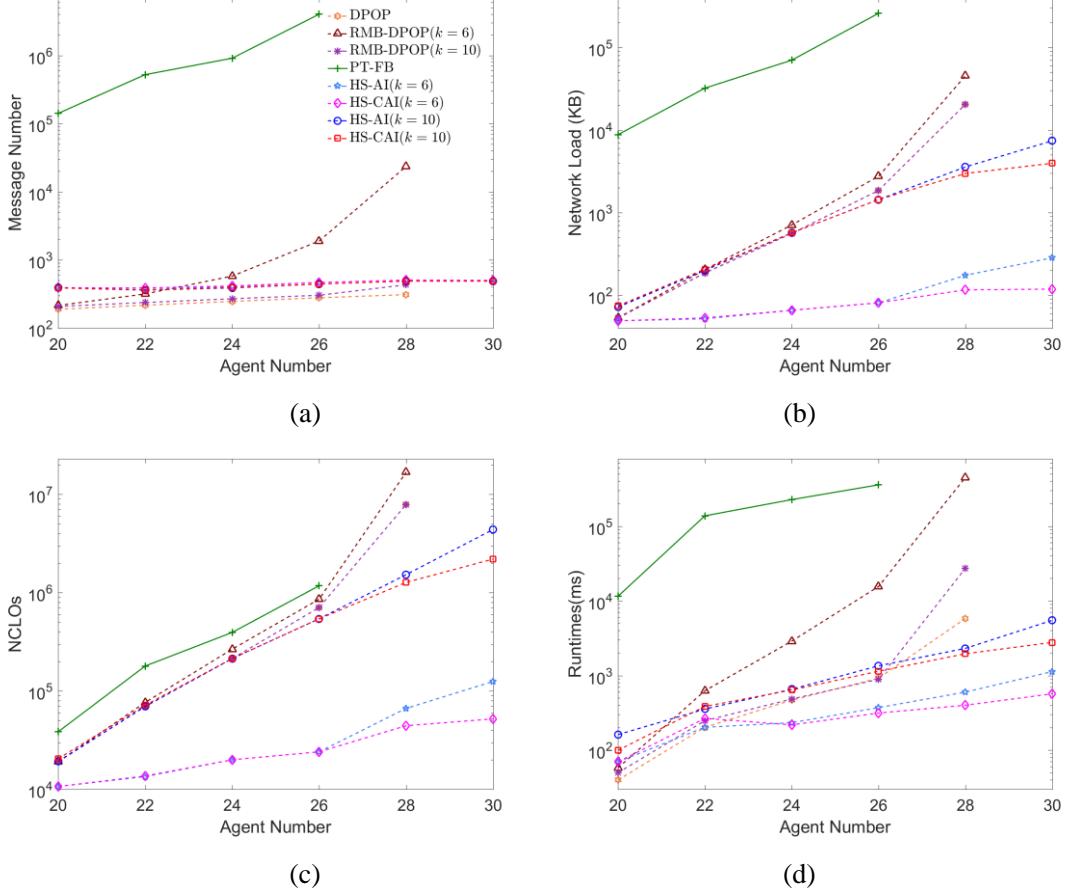


图 3.9 各个算法在求解分布式信道分配问题时的性能对比

Fig. 3.9 Performance comparison on sensor networks

3.6 本章小结

通过分析搜索和推理混合的可行性，提出一种 DCOP 完备算法 HS-CAI。该算法通过将搜索与上文推理相结合实现 DCOP 的求解。与现有的搜索与推理混合完备算法不同，HS-CAI 通过迭代执行上文推理构造紧的下界从而达到加快搜索过程的目的。借助于上文评估机制，它仅需要对搜索过程中部分上文执行推理，从而减少上文推理所产生的巨大通信量开销。从理论上证明，在相同的内存预算下，上文推理可比近似推理产生更紧的下界。此外，实验结果表明，与目前最好的 DCOP 完备算法相比，HS-CAI 不仅可以更快地找到最优解，并且通信量开销更少。

4 基于非本地消元的推理与搜索混合的 ADCOP 算法^①

4.1 引言

ADCOP 在 DCOP 的基础之上引入智能体的私有偏好（即非对称特性，约束智能体双方对于同一约束具有不同的私有约束代价函数），因此在求解 ADCOP 时，还应将私有偏好保障作为一个重要的评价指标。如 1.2.2 节所述，现有 ADCOP 完备算法通常采用私有约束代价函数值直接泄露的方式实现双面约束累积，且采用分支定界搜索策略实现问题的求解。因此，智能体的私有偏好保障性能完全依赖于算法的剪枝。此外，对于采用分支定界策略的完备搜索算法而言，其剪枝性能严重依赖于算法提供上、下界的紧度。但现有的 ADCOP 完备算法仅利用本地知识（即与智能体自身相关的约束代价函数）构建下界，因此剪枝性能较差。在求解复杂问题时，算法会使得整个系统近一半的私有偏好泄漏。接下来，以目前性能最好的 ADCOP 完备算法 AsymPT-FB 为例来阐述这一问题。

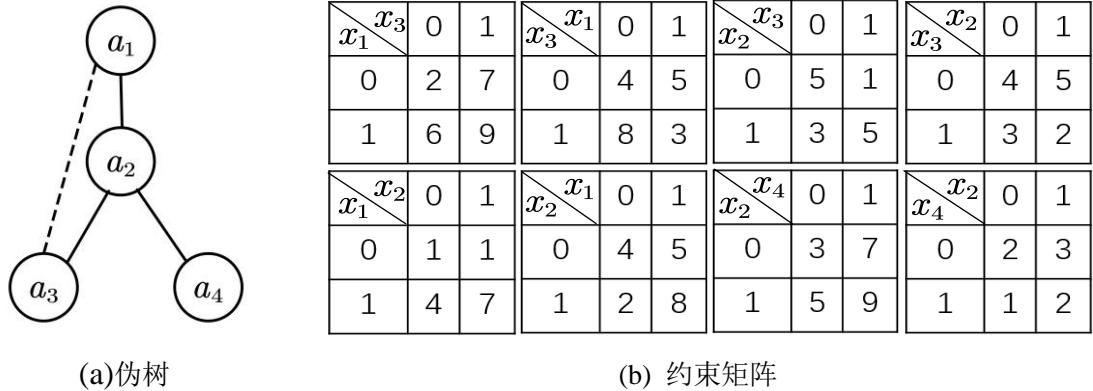


图 4.1 说明 PT-ISABB 算法的 ADCOP 实例

Fig. 4.1 An ADCOP instance for PT-ISABB

在 AsymPT-FB 中，下界计算通过前向定界过程实现。在前向定界过程中，智能体向其邻居节点请求在当前部分解下本地最佳的累积代价。例如在图 4.1 (a) 中， a_2 通过发送 LB_Report 消息请求父节点 a_1 泄露私有约束 f_{12} 在当前部分解下对应的代价（即 f_{12} 在给定 x_2 与 x_1 赋值下对应的代价值），向其孩子节点 a_3 和 a_4 请求它们私有约束 f_{32} 和 f_{42} 对应下界（即 f_{32} 和 f_{42} 在给定 x_2 赋值情况下的最小代价值）。因此， a_2 在计算下界时仅能依赖自己的本地知识（即与 a_2 相关的约束 f_{12} ， f_{32} 和

^① 本章内容已被期刊《Journal of Autonomous Agents and Multi-Agent Systems》录用

f_{42} ），无法获得其对应子问题下的其他约束（例如 a_1 和 a_3 之间的约束 f_{13} 和 f_{31} ）。总的来说，研究如何提供更紧的上、下界是提高基于搜索的 ADCOP 完备算法性能的必经之路，也是私有偏好保障的重要手段。

另一方面，现有基于推理的 DCOP 完备算法（例如 DPOP）能通过本地消元与效用传播实现全局知识的快速累积，但无法直接应用于 ADCOP 求解。这是因为在非对称环境下，智能体必须获得与本地变量相关的双面约束来保证本地消元的完备性。也就是说，所有的（伪）父节点需要通过将它们的私有约束转移到本地节点来实现本地变量的消元，这将导致系统一半以上的私有偏好泄露。以图 4.1(a) 为例， a_2 需要父节点 a_1 直接泄露私有约束 f_{12} 来实现本地消元，这使得节点 a_1 的私有约束全部被泄露。换句话说，现有 DCOP 的推理完备算法无法用于求解 ADCOP。

基于上述分析，我们考虑在尽量避免私有偏好泄露的前提下，通过结合推理与搜索来提升 ADCOP 完备算法的求解性能。具体地，本章提出一种推理与搜索混合完备算法，该算法包含推理和搜索两个阶段。其中，推理阶段通过执行针对非对称特性定制的 ADPOP 算法快速地累积部分约束，并存储每个分支对应的推理结果。搜索阶段通过在伪树上实现 SyncABB-1ph，同时利用保存的推理结果建立紧的上、下界，进而求解出问题的最优解。

本章内容安排如下：首先，4.2 节详细地描述本章提出的推理与搜索混合的完备算法，并分析该算法的私有偏好泄露情况；随后，4.3 节从提供下界的紧度、完备性以及复杂度三个方面对提出算法进行理论分析；接着，4.4 节通过结合绝对误差机制和相对误差机制，将提出的算法扩展为非完备算法，并证明所提出非完备算法的正确性；最后，4.5 节通过实验验证本章提出的 ADCOP 完备算法以及其非完备版本的优势，并在 4.6 节对本章工作进行小结。

4.2 基于非本地消元的推理与搜索混合的 ADCOP 完备算法

针对 4.1 节所述的问题，本节提出基于非本地消元的推理与搜索混合 ADCOP 完备算法（Pseudo Tree-Inference/SyncABB-1ph, PT-ISABB）。PT-ISABB 包含推理和搜索两个阶段。在推理阶段，智能体通过执行仅考虑一部分约束的效用传播实现全局信息累积的同时避免私有偏好完全泄露，并保存传播的效用表为搜索阶段提供紧的上、下界。在搜索阶段将传统基于链式结构的 SyncSABB-1ph 算法扩展到伪树上实现对解空间不重不漏地枚举。此外，还引入预估汇报机制在搜索阶段避免私有约束代价直接泄露以及获得完整上界。

4.2.1 基于非本地消元的推理阶段

如上文 4.1 节所述，利用本地消元策略完备求解 ADCOP 将导致系统一半以上的私有偏好泄露。因此在推理阶段，不要求（伪）父节点通过泄露其私有约束代

价函数来执行消元。另一方面，在执行消元时忽略（伪）父节点的私有约束代价函数将导致推理阶段传播的效用表不完整，无法为搜索阶段提供紧的界。也就是说，算法需要权衡界的紧度和私有偏好保障性能。据此，本节提出非本地消元机制来缓解这个问题，该机制具体描述如下：

当智能体 a_i 收到孩子节点 a_c 效用表 $util_c^\pm$ 时， a_i 将其私有约束代价函数 f_{ic} 通过联合的方式加入效用表 $util_c^\pm$ 之后再消除 x_c ，以此来提高 $util_c^\pm$ 的完整度。为进一步提高 $util_c^\pm$ 的完整度，达到为搜索阶段提供更紧的目的，将 a_c 分支中剩余的约束代价函数也通过联合的方式加入到效用表 $util_c^\pm$ 中，即：

$$util_c^\pm = (util_c^\pm \otimes f_{ic}) \perp_{x_c} \otimes \left(\bigotimes_{a_j \in PC(a_i) \cap Desc(a_c)} f_{ij} \perp_{x_j} \right) \quad (4.1)$$

不同于基于本地消元的推理算法（例如 DPOP 和 ADPOP），对变量的消除实际上推迟到父节点，而非在本地执行消元操作。

当收到所有孩子的效用消息后， a_i 根据公式 (4.1) 处理收到的效用表，并联合本地效用表 $localUtil_i$ 。当联合后效用表的维度超出内存预算 k_{mb} 时， a_i 消除超出维度集合 S 来保证效用表维度不超过 k_{mb} ，随后向上传播消元后的结果 $util_i^\pm$ ，即：

$$util_i^\pm = \left(\left(\bigotimes_{a_c \in C(a_i)} util_i^{c\pm} \right) \otimes localUtil_i \right) \perp_S \quad (4.2)$$

其中， a_i 的本地效用表 $localUtil_i$ 为 a_i 和 $AP(a_i)$ 在 a_i 这一面约束代价函数的累积，即：

$$localUtil_i = \bigotimes_{a_j \in AP(a_i)} f_{ij} \quad (4.3)$$

从公式 (4.2) 可以看出，执行变量消元时， a_i 仅考虑本地私有约束代价函数，不要求 $AP(a_i)$ 节点泄露它们的私有函数。因此， $AP(a_i)$ 的私有偏好没有直接泄露。

算法 4.1 PT-ISABB 推理阶段的伪代码

Algorithm 4.1 Sketch of inference phase in PT-ISABB

Inference phase in PT-ISABB for each agent a_i

输入：智能体 a_i 的约束函数集合

输出：智能体 a_i 的效用表

When Initialization():

1. $util_i^\pm \leftarrow copy(local_util_i)$
 2. **if** a_i is a leaf **then**
-

 Inference phase in PT-ISABB for each agent a_i

3. **SendUtil()**

When received **UTIL**($util_c^\pm$) from $a_c \in C(a_i)$: //处理接收到的效用表

4. compute $util_i^{c\pm}$ according to Eq.(4.1)

5. $util_i^\pm \leftarrow util_i^\pm \otimes util_i^{c\pm}$

6. **if** a_i has received all UTIL $C(a_i)$ **then**

7. **if** a_i is the root **then**

8. start **Search** phase

9. **else**

10. **SendUtil()**

Function SendUtil(): //效用表的计算与传播

11. **if** $|dims(util_i^-)| > k$ **then**

12. select $S \subset dims(util_i^-)$, s.t. $|S| = |dims(util_i^-)| - k$

13. $util_i^\pm \leftarrow util_i^\pm \perp_S$

14. send UTIL($util_i^\pm$) to $P(a_i)$

算法 4.1 给出 PT-ISABB 算法在推理阶段的伪代码。该阶段开始于叶子节点将本地效用表通过 UTIL 消息发送给父节点（1-3 行，11-14 行）。若发送的效用表维度超过内存维度限制 k_{mb} （11 行），则 a_i 通过近似消元消除超出的维度（12-13 行）。当收到孩子节点 a_c 发送的 UTIL 消息后， a_i 根据公式（4.1）处理收到的效用表（4 行）。当收到所有孩子的 UTIL 消息后，若 a_i 是非根节点，则采用公式（4.2）计算效用表 $util_i^\pm$ ，并该效用表传播给父节点（9-10 行，11-14 行）。否则，开启搜索阶段（7-8 行）。

4.2.2 搜索阶段

搜索阶段通过在伪树上执行带一阶段的分支定界搜索策略，以实现对解空间不重不漏地枚举。具体地，在伪树上，分支节点将问题分解成为多个子问题，随后让其孩子节点并行地求解这些子问题。在求解过程中，为快速地发现最优解并丢弃次优解，每个智能体 a_i 需要为其子问题的搜索代价维护搜索下界 LB_i 和搜索上界 UB_i 。其中，为计算完整 UB_i ， a_i 需要获取与其子问题相关的所有双边约束信息（即还需考虑在 $Sep(a_i)$ 一侧的私有约束代价）。据此，本节提出预估汇报机制来解决这一问题。该机制中， $Sep(a_i)$ 节点主动汇报涉及 a_i 子问题中私有约束的乐观和悲观代价预估。这里，给定 (x_j, d_j) ，二元函数 $f_{ji}(x_j, x_i)$ 的乐观和悲观代价预估分别为一元函数 $f_{ji}(d_j, x_i)$ 的下界 $\min_{d_j \in D_j} f_{ji}(d_j, x_i)$ 和上界 $\max_{d_j \in D_j} f_{ji}(d_j, x_i)$ 。进一步，对于 f_{ji} ，给定 $\{(x_j, d_j), (x_i, d_i)\}$ ，称真实约束代价值与乐观预估代价之间的差值为

剩余约束代价（即 $f_{ji}(d_j, d_i) - \min_{d_j \in D_j} f_{ji}(d_j, x_i)$ ）。因此，在积累双边的约束代价时，（伪）父节点仅需要传递它们的剩余约束代价，而非直接泄露它们的私有约束代价。具体地，每个智能体 a_i 需要维护如下数据结构：

- ① Cpa_i 表示 a_i 探索的当前部分解，其包含 $Sep(a_i)$ 的全部赋值信息。
- ② $Srch_val^c$ 记录当前正在被以分支 $a_c \in C(a_i)$ 所探索的 a_i 赋值。该数据结构是不可缺少的，因为不同分支在伪树上并发探索不同 a_i 的赋值。
- ③ $EstRep_i$ 包含对 a_i 及其后代的所有乐观和悲观的累积代价预估。其中， $EstRep_i(x_j) \in EstRep_i$ 是 $a_j \in \{a_i\} \cup Desc(a_i)$ 的累积代价预估值，即：

$$EstRep_i(x_j) = \bigotimes_{a_l \in Sep(a_i) \cap AP(a_j)} f_{lj}(Cpa_i(x_l), x_j) \perp_{x_j} \quad (4.4)$$

这里， $EstRep_i(x_j) = \{EstRep_i(x_j)^-, EstRep_i(x_j)^+\}$ ，其中 $EstRep_i(x_j)^-$ 和 $EstRep_i(x_j)^+$ 分别 x_j 的乐观和悲观累积代价预估值。

- ④ $high_cost_i(d_i)$ 表示在 $Cpa_i \cup \{(x_i, d_i)\}$ 下， a_i 与 $AP(a_i)$ 相关的双面约束代价总和。其初始化为 a_i 一侧的约束代价与 a_i 的乐观累积代价预估之和，即：

$$high_cost_i(d_i) = \sum_{a_j \in AP(a_i)} f_{ij}(d_i, Cpa_i(x_j)) + EstRep_i(x_j)^- \quad (4.5)$$

在公式 (4.5) 中引入 $EstRep_i(x_j)^-$ 是为实现在获得完整的 $high_cost_i(d_i)$ 的过程中避免直接泄露 $AP(a_i)$ 节点的私有约束代价。因此，为获得完整的 $high_cost_i(d_i)$ ， a_i 仅需要获取 $AP(a_i)$ 节点关于 d_i 的剩余约束代价，而非实际约束代价。当收到所有 $AP(a_i)$ 节点的剩余约束代价后， $high_cost_i(d_i)$ 则为 a_i 和 $AP(a_i)$ 关于 d_i 的完整双边约束代价，即：

$$\sum_{a_j \in AP(a_i)} f_{ij}(d_i, Cpa_i(x_j)) + f_{ji}(d_i, Cpa_i(x_j))$$

- ⑤ $lb_i^c(d_i)$ 表示在 Cpa_i 下， $a_c \in C(a_i)$ 子问题在赋值 $d_i \in D_i$ 的下界。其初始值设为在 $Cpa_i \cup \{(x_i, d_i)\}$ 下 $util_i^{c-}$ 的值加上在 $Desc(a_c) \cup \{a_c\}$ 中所有智能体的乐观累积代价预估之和，即：

$$lb_i^c(d_i) = util_i^{c-}(Cpa_i[Sep(a_c)], d_i) + \sum_{a_j \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_j)^- \quad (4.6)$$

- ⑥ $ub_i^c(d_i)$ 表示在 Cpa_i 下， $a_c \in C(a_i)$ 子问题在赋值 $d_i \in D_i$ 的上界。其初始值设为在 $Cpa_i \cup \{(x_i, d_i)\}$ 下 $util_i^{c+}$ 的值加上在 $Desc(a_c) \cup \{a_c\}$ 中所有智能体的悲观累积代价预估之和，即：

$$ub_i^c(d_i) = util_i^{c+}(Cpa_i[Sep(a_c)], d_i) + \sum_{a_j \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_j)^+ \quad (4.7)$$

- ⑦ $lb_i(d_i)$ 表示在 Cpa_i 下， a_i 子问题在赋值 $d_i \in D_i$ 的下界，即：

$$lb_i(d_i) = high_cost(d_i) + \sum_{a_c \in C(a_i)} lb_i^c(d_i) \quad (4.8)$$

⑧ $ub_i(d_i)$ 是在 Cpa_i 下, a_i 子问题在赋值 $d_i \in D_i$ 的上界, 初始化为:

$$ub_i(d_i) = high_cost(d_i) - EstRep_i(x_i)^- + EstRep_i(x_i)^+ + \sum_{a_c \in C(a_i)} ub_i^c(d_i) \quad (4.9)$$

当 a_i 收到所有的来自 $AP(a_i)$ 关于 d_i 的剩余约束代价后, $ub_i(d_i)$ 被替换为:

$$high_cost(d_i) + \sum_{a_c \in C(a_i)} ub_i^c(d_i)$$

⑨ LB_i 表示在 Cpa_i 下, a_i 子问题的上界, 即:

$$LB_i = \min_{d_i \in D_i} lb_i(d_i) \quad (4.10)$$

⑩ UB_i 表示在 Cpa_i 下, a_i 子问题的下界, 即:

$$UB_i = \min_{d_i \in D_i} ub_i(d_i) \quad (4.11)$$

为在伪树上实现深度优先分支定界和一阶段策略, 在搜索阶段使用如下四种类型的消息。

⑪ CPA 消息由 a_i 送给其孩子节点 a_c , 其包括部分解 Cpa_c 、阈值 TH_c 以及累积代价预估 $EstRep_c$, 即:

$$a_i \rightarrow a_c : CPA(Cpa_c, TH_c, EstRep_c), a_c \in C(a_i)$$

其中 Cpa_c 是 Cpa_i 扩展 (x_i, d_i) 的结果 (即 $Cpa_c = Cpa_{i[Sep(a_c)]} \cup \{(x_i, d_i)\}$)。

TH_i 为在 Cpa_i 下, a_i 子问题最优代价的阈值, 由公式 (4.12) 计算获得。

$$TH_c = \min(UB_i, TH_i) - high_cost(d_i) - \sum_{a_j \in C(a_i) \wedge j \neq c} lb_i^j(d_i) \quad (4.12)$$

特别的, 如果 a_i 是根节点, 那么 $TH_i = \infty$ 。

⑫ BACKTRACK 消息由 a_i 发送到父节点 $P(a_i)$, 其包含 a_i 汇报的已探索到的子问题最优代价 opt_i 以及相应的部分赋值 $Spa_i(d_i)$, 即:

$$a_i \rightarrow P(a_i) : BACKTRACK(opt_i, Spa_i(d_i^*))$$

其中 opt_i 是 a_i 在部分解 Cpa_i 下子问题的最优代价 (即 $opt_i \leq TH_i$) 或 ∞ (即 $opt_i > TH_i$, 证明 Cpa_i 不可能是最优解的一部分)。 $Spa_i(d_i)$ 表示其子问题在 $Cpa_i \cup \{(x_i, d_i^*)\}$ 下的最优赋值, 其中 $d_i^* = \operatorname{argmin}_{d_i \in D_i} ub_i(d_i)$ 。这里, $Spa_i(d_i)$ 初始化为 $\{(x_i, d_i^*)\}$ 。若 a_i 是根节点, 那么 $Spa_i(d_i^*)$ 为问题的最优解。

⑬ RESD_REQ 消息由 a_i 发送给 (伪) 父节点 $a_j \in AP(a_i)$ 。用于请求获得在赋值为 $\{(x_i, d_i), (x_j, d_j)\}$ 时 f_{ji} 的剩余约束代价, 即:

$$a_i \rightarrow a_j : RESD(r_{ij}, d_j), a_j \in AC(a_i)$$

这里, 剩余约束代价 r_{ij} 由公式 (4.13) 计算获得。

$$r_{ij} = f_{ij}(d_i, d_j) - \min_{d'_j \in D_j} f_{ij}(d_i, d'_j) \quad (4.13)$$

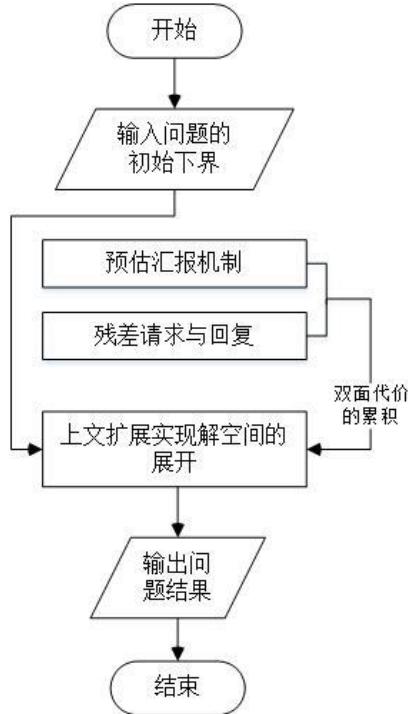


图 4.2 PT-ISABB 搜索阶段的流程图

Fig 4.2 Flow chart of Search phase in PT-ISABB

算法 4.2 PT-ISABB 搜索阶段的伪代码

Algorithm 4.2 Sketch of search phase in PT-ISABB

Search phase in PT-ISABB for each agent a_i 输入：智能体 a_i 的值域、约束函数集合以及预处理阶段的效用表输出：智能体 a_i 最终的赋值**When** Initialization()

1. **if** a_i is the root **then**
2. **InitializeVariables()**
3. $d_i \leftarrow \text{NextFeasibleAssignment}(null)$
4. **SendCpa**(d_i, a_c), $\forall a_c \in C(a_i)$

When received **CPA**($Cpa_i, TH_i, EstRep_i$) from $P(a_i)$: //处理 CPA 消息

5. stores $\{Cpa_i, TH_i, EstRep_i\}$
6. **InitializeVariables()**
7. **if** $LB_i \geq \min(UB_i, TH_i)$ **then**

Search phase in PT-ISABB for each agent a_i

8. send BACKTRACK(∞, \emptyset) to $P(a_i)$
 9. **else**
 10. $d_i \leftarrow \text{NextFeasibleAssignment}(null)$
 11. $Srch_val_i^c \leftarrow d_i, \forall a_c \in C(a_i)$
 12. send RESD_REQ($Cpa_i(x_j), d_i$) to $a_j, \forall a_j \in AP(a_i)$
 - When received RESD_REQ(d_i, d_c) from $a_c \in C(a_i) \cup PC(a_i)$:**
 13. send $\text{RESD}\left(f_{ic}(d_i, d_c) - \min_{d'_c \in D_c} f_{ic}(d_i, d'_c), d_c\right)$ to a_i
 - When received RESD (r_{ij}, d_i) from $a_j \in AP(a_i)$: //处理代价值的残差消息**
 14. $high_cost_i(d_i) \leftarrow high_cost_i(d_i) + r_{ji}$
 15. **if** a_i has received all RESD from $AP(a_i)$ for d_i **then**
 16. $ub_i(d_i) \leftarrow \sum_{a_c \in C(a_i)} ub_i^c(d_i) + high_cost_i(d_i)$
 17. **if** $LB_i \geq \min(UB_i, TH_i)$ **then**
 18. **SendBackTrack()**
 19. **else**
 20. **if** a_i is a leaf **then**
 21. $d'_i \leftarrow \text{NextFeasibleAssignment}(d_i)$
 22. Send RESD_REQ($Cpa_i(x_j), d'_i$) to $a_j, \forall a_j \in AP(a_i)$
 23. **else** //解空间的展开
 24. **if** $lb_i(d_i) \geq \min(UB_i, TH_i)$ **then**
 25. $d'_i \leftarrow \text{NextFeasibleAssignment}(d_i)$
 26. **if** $d'_i \neq null$ **then**
 27. $Srch_val_i^c \leftarrow d'_i, \forall a_c \in C(a_i) \wedge Srch_val_i^c = d_i$
 28. send RESD_REQ($Cpa_i(x_j), d'_i$) to $a_j, \forall a_j \in AP(a_i)$
 29. **else**
 30. **SendCpa**(d_i, a_c), $\forall a_c \in C(a_i) \wedge Srch_val_i^c = d_i$
 - When received BACKTRACK (opt_c, Spa_c) from $a_c \in C(a_i)$: //处理回溯消息**
 31. $d_i \leftarrow Srch_val_i^c, Spa_i(d_i) \leftarrow Spa_i(d_i) \cup Spa_c,$
 32. $lb_i^c(d_i) \leftarrow \max(lb_i^c(d_i), opt_c), ub_i^c(d_i) \leftarrow \min(ub_i^c(d_i), opt_c)$
 33. **if** a_i is the root **then**
 34. send TERMINATE to $a_c, \forall a_c \in C(a_i)$
 35. terminate
 36. **else**
-

Search phase in PT-ISABB for each agent a_i

37. **SendBackTrack()**
 38. **else**
 39. $d'_i \leftarrow \text{NextFeasibleAssignment}(d_i), Srch_val_i^c \leftarrow d'_i$
 40. **if** $d'_i \neq null$ **then**
 41. **if** a_i has received all RESD from $AP(a_i)$ for d'_i **then**
 42. **SendCpa**(d'_i, a_c)
 43. **else if** a_i has not request RESD for d'_i **then**
 44. send RESD_REQ($Cpa_i(x_j), d'_i$) to $a_j, \forall a_j \in AP(a_i)$

When received TERMINATE from $P(a_i)$:

45. send TERMATE to $a_c, \forall a_c \in C(a_i)$

46. terminate

When received STOPEXPLOR from $P(a_i)$:

47. send STOPEXPLOR to $\forall a_c \in C(a_i) \wedge srch_val_i^c \neq null$
 48. sleep and wait for a CPA message to wake it up

Function InitializeVariables:

49. initialize $Srch_val_i^c, \forall a_c \in C(a_i)$
 50. initialize $lb_i^c(d_i), ub_i^c(d_i), \forall a_c \in C(a_i), d_i \in D_i$
 51. initialize $high_cost_i(d_i), Spa_i(d_i), lb_i(d_i), ub_i(d_i), \forall d_i \in D_i$
 52. initialize LB_i, UB_i

Function NextFeasibleAssignment(d_i) //查找可探索的赋值

53. $d'_i \leftarrow$ the element next to d_i in D_i
 54. **while** $d'_i \neq null \wedge lb_i(d'_i) \geq \min(UB_i, TH_i)$ **do**
 55. $d'_i \leftarrow$ the next element in D_i
 56. return d'_i

Function SendCpa(d_i, a_c) //CPA 消息的计算与传播

57. $Cpa_c \leftarrow Cpa_{i[Sep(a_c)]} \cup \{(x_i, d_i)\}, Srch_val_i^c \leftarrow d_i$
 58. $TH_c \leftarrow \min(UB_i, TH_i) - high_cost_i(d_i) - \sum_{a_j \in C(a_i) \wedge j \neq c} lb_i^j(d_i)$
 59. $EstRep_c \leftarrow \text{GetEstReports}(d_i, a_c)$
 60. send CPA($Cpa_c, TH_c, EstRep_c$) to a_c

Function SendBackTrack(): //回溯消息的计算与传播

61. **if** $LB_i = UB_i \wedge LB_i < TH_i$ **then**
 62. $d_i^* \leftarrow \operatorname{argmin}_{d_i \in D_i} ub_i(d_i)$
 63. send BACKTRACK($ub_i(d_i^*), Spa_i(d_i^*)$) to $P(a_i)$

 Search phase in PT-ISABB for each agent a_i

```

64.   else
65.       send BACKTRACK( $\infty, \emptyset$ ) to  $P(a_i)$ 
66.       send STOPEXPLORE to  $\forall a_c \in C(a_i) \wedge \text{srch\_val}_i^c \neq \text{null}$ 
67.       sleep and wait for a CPA message to wake it up
Function GetEstReports( $d_i, a_c$ ): //计算单面的子问题代价预估
68.    $EstRep \leftarrow \emptyset$ 
69.   foreach  $a_j \in Desc(a_c) \cup \{a_c\}$  do
70.       if  $x_j \in EstRep_i \wedge a_j \in AC(a_i)$  then
71.            $EstRep \leftarrow EstRep \cup \{(x_j, EstRep_i(x_j) \otimes f_{ij}(d_i)_{\perp x_j})\}$ 
72.       else if  $x_j \in EstRep_i \wedge a_j \notin AC(a_i)$  then
73.            $EstRep \leftarrow EstRep \cup \{(x_j, EstRep_i(x_j))\}$ 
74.       else if  $x_j \notin EstRep_i \wedge a_j \in AC(a_i)$  then
75.            $EstRep \leftarrow EstRep \cup \{(x_j, f_{ij}(d_i)_{\perp x_j})\}$ 
76.   return  $EstRep$ 

```

图 4.2 和算法 4.2 分别给出 PT-ISABB 搜索阶段的流程图和伪代码。该阶段开始于根节点通过 CPA 消息（1-3 行，57-60 行）发送第一个可行赋值给孩子节点。当收到父节点的 CPA 消息后， a_i 存储收到的 Cpa_i 、阈值以及预估汇报，并依据公式（4.4-4.11）初始化相关的变量（5-6 行，49-52 行）。若 Cpa_i 不可能是最优解的一部分， a_i 发送无穷大的代价以及一个空的子问题赋值，回溯至父节点（7-8 行）。否则， a_i 将第一个可行的赋值 d_i 通过 RESD_REQ 消息发送给所有的父节点，请求它们发送对应赋值下的剩余约束代价（9-12 行）。当收到（伪）孩子节点的 RESD_REQ 消息后， a_i 发送 RESD 消息回复相应的剩余约束代价（13 行）。

当收到赋值为 d_i 的 RESD 消息后， a_i 将接收到的剩余约束代价加入 $high_cost_i(d_i)$ 中（14 行）。当收到所有（伪）父节点的 RESD 消息后， a_i 更新其上、下界并检测是否满足回溯条件（15-16 行）。如果满足，则将其探索获得的解（即当 Cpa_i 可行情况下的最优代价或者为空赋值情况下的无穷大的代价）通过 BACKTRACK 消息回溯至父节点（17-18 行，61-65 行）。否则， a_i 继续探索解空间。对于叶子节点 a_i 而言，其仅需要切换至下一个可行的赋值 d'_i ，并且请求（伪）父节点在该赋值下的剩余约束代价即可（20-22 行）。如果 a_i 是一个非叶子节点且 d_i 仍然是值得探索时， a_i 扩展 Cpa_i 并将扩展过后的 Cpa_i 发送给将要探索 d_i 的孩子节点（29-30 行）。否则， d_i 被证明是次优的。此时， a_i 切换至下一个可行的赋值 d'_i ，并请求获得关于 d'_i 的剩余约束代价（24-28 行）。

当从孩子节点 a_c 收到关于 d_i 的 BACKTRACK 消息后， a_i 利用收到的最优代价 opt_c 更新对应的上、下界，并且合并收到的部分赋值 Spa_c （31 行）。然后， a_i 决定是否需要进行回溯（32 行）。如果需要且自己不是根节点时， a_i 继续回溯（36-37 行）。若需要且自己是根节点， a_i 告知其孩子停止运行，随后自己也终止运行（33-35 行）。如果不进行回溯， a_i 为 a_c 确定下一个待探索的赋值 d'_i （38-44 行）。具体地，如果 d'_i 存在并且已经从 $AP(a_i)$ 节点处收到所有关于 d'_i 的 RESD 消息，那么 a_i 直接告知 a_c 去探索 d'_i （39-42 行）；否则， a_i 向 $AP(a_i)$ 节点请求获取关于 d'_i 的剩余约束代价（43-44 行）。

4.2.3 私有偏好泄露分析

接下来，以非对称 MaxDCSPs 问题为例，分析 PT-ISABB 算法由于消息传递造成的私有偏好泄露。由于变量 x_i 的消除是由父节点处执行的，父节点很容易地从收到的效用表中发现哪些赋值对是可行的。具体地，对于在 a_i 一侧的约束，父节点可直接从 a_i 发送效用表中为零的位置推断出可行的赋值对。因此，在最坏情况下， a_i 发送的 UTIL 消息可能会直接造成在 a_i 一侧树边的约束泄露。对于 a_i 接收到的 CPA 消息，仅在如下的两种情况下可以推断出（伪）父节点的私有约束信息：1) 当收到 x_i 的悲观累积预估等于零时， a_i 可以推断出自己与（伪）父节点之间的约束是可行的。2) 当收到 x_i 的乐观累积代价预估等于 $AP(a_i)$ 大小时， a_i 与 $AP(a_i)$ 的约束是不可行的。由于汇报最优子问题代价等于零时也会发生私有偏好泄露，所以 BACKTRACK 消息也会泄露一些私有偏好。RESD_REQ 消息由于仅包含消息发送者与接收者的赋值消息，因此并不会发生私有约束的泄露。另一个严重的私有偏好泄露是由 RESD 消息造成。具体而言，当收到 x_i 的乐观累积代价预估为零时，从（伪）父节点发送给 a_i 的 RESD 消息将直接泄露具体 x_i 与 x_j 赋值对应的私有约束代价。同样收到的剩余约束代价不为零时，也存在私有约束的直接泄露。

4.3 理论分析

本节的内容安排如下：首先，4.3.1 节证明在最大内存维度限制为 $k_{mb} = w^* + 1$ 时，PT-ISABB 算法提供的下界至少和 AsymPT-FB 算法提供的下界一样紧；然后，4.3.2 节从理论上证明 PT-ISABB 算法的完备性；最后，4.3.3 节对 PT-ISABB 算法进行复杂度分析。

4.3.1 提供下界紧度分析

性质 4.1：在最大内存维度限制为 $w^* + 1$ 时，给定任意一个部分解 Cpa_i ， a_i 分支 a_c 在 d_i 处的初始下界 lb_i^c 大于等于 AsymPT-FB 算法提供的下界。

证明：在 AsymPT-FB 算法中，当收到 a_c 分支中所有的 LB_REPORT 消息后， a_c 的下界是在 Cpa_i 下 a_c 所有后代单面的本地约束代价之和，即：

$$\begin{aligned} SubtreeLB_i^c(d_i) = & \sum_{a_j \in Desc(a_c)} \min_{x_j} \sum_{a_l \in Sep(a_c) \cap PP(a_j)} f_{jl}(x_j, d_l) \\ & + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \end{aligned}$$

为简单起见，我们将向量 x_c 和其后代变量定义为 \mathbf{x}_c 。接下来，我们将证明 $lb_i^c(d_i) \geq Subtree_i^c(d_i)$ 。由于 $k_{mb} = w^* + 1$ ，推理阶段不会丢弃任何维度，我们有：

$$\begin{aligned} lb_i^c(d_i) &= util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_j)^- \\ &= util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in Desc(a_c) \cup \{a_c\}} \sum_{a_l \in Sep(a_c)} \min_{d_l \in D_l} f_{jl}(d_j, d_l) \\ &\geq util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) \\ &= \min_{\mathbf{x}_c} \sum_{a_j \in Desc(a_c)} \left(\sum_{a_l \in AP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \sum_{a_l \in AP(a_j) \cap Desc(a_c)} f_{jl}(x_j, x_l) \right. \\ &\quad \left. + \sum_{a_l \in C(a_j)} f_{jl}(x_j, x_l) + \sum_{a_l \in PC(a_j)} \min_{x_l} f_{jl}(x_j, x_l) \right) + \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\ &\quad + f_{ic}(d_i, x_c) + \sum_{a_l \in C(a_c)} f_{cl}(x_c, x_l) + \sum_{a_l \in PC(a_c)} \min_{x_l} f_{cl}(x_c, x_l) \\ &\geq \min_{\mathbf{x}_c} \sum_{a_j \in Desc(a_c)} \sum_{a_l \in AP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \end{aligned}$$

由于 $x_c \in \mathbf{x}_c$ 且 $PP(a_i) \subset AP(a_i)$ ，上述公式可以表示为：

$$\begin{aligned} lb_i^c(d_i) &\geq \min_{\mathbf{x}_c \setminus x_c} \sum_{a_j \in Desc(a_c)} \sum_{a_l \in AP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\ &\geq \min_{\mathbf{x}_c \setminus x_c} \sum_{a_j \in Desc(a_c)} \sum_{a_l \in PP(a_j) \cap Sep(a_c)} f_{jl}(x_j, d_l) + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\ &\geq \sum_{a_j \in Desc(a_c)} \min_{x_j} \sum_{a_l \in Sep(a_c) \cap PP(a_j)} f_{jl}(x_j, d_l) + \min_{x_c} \sum_{a_l \in AP(a_c)} f_{cl}(x_c, d_l) \\ &= SubtreeLB_i^c(d_i) \end{aligned}$$

从而证明该性质。

4.3.2 完备性证明

本节首先证明 PT-ISABB 算法的终止性和最优性，然后证明算法的完备性。

引理 4.1: PT-ISABB 算法在有限次数的迭代后会终止。

证明: 从算法 4.2 的伪代码可知，推理阶段只需要线性数量的消息便终止。因此，为证明算法能终止，只需要证明相同的部分解并不会在搜索阶段被探索两次即可，即一个智能体不会收到两个相同的 Cpa 。由于根节点不会收到任何 Cpa ，因此该命题在根节点处成立。当一个节点 a_i 收到来自父节点发送的 Cpa ，它将发送多个 Cpa 给其孩子节点 $a_c \in C(a_i)$ 。由于每一个 Cpa 所包含 a_i 的赋值均不同，因此孩子节点

$a_c \in C(a_i)$ 不会收到两个相同 Cpa 。因为根节点不会收到 Cpa ，且对于一个部分解，任意一个节点发送给子节点的 Cpa 都是不同的。因此，算法终止性得以保证。

引理 4.2: 给定任意一个部分解 Cpa_i 与 a_i 的赋值 (x_i, d_i) ，其子树任意赋值 Spa_i 与所产生的代价 $cost(Spa_i)$ 不小于下界 $lb_i(d_i)$ 。这里， $cost(Spa_i)$ 可形式化为：

$$cost(Spa_i) = \sum_{a_l \in \{a_i\} \cup Desc(a_i)} \sum_{a_j \in AP(a_l)} (f_{jl}(d_j, d_l) + f_{lj}(d_l, d_j))$$

其中， d_l 表示 x_l 在 Cpa_i 或 Spa_i 中的赋值。

证明：该引理将分如下两种情况进行证明：

情况 1：如果 a_i 是叶子节点， $cost(Spa_i)$ 可以被表示为：

$$\begin{aligned} cost(Spa_i) &= \sum_{a_j \in AP(a_i)} f_{ij}(d_i, d_j) + f_{ji}(d_j, d_i) \\ &\geq \sum_{a_j \in AP(a_i)} f_{ij}(d_i, d_j) + \sum_{a_j \in AP(a_i)} \min_{x_i} f_{ji}(d_j, x_i) \\ &= \sum_{a_j \in AP(a_i)} f_{ij}(d_i, d_j) + EstRep_i(x_i)^- \\ &\leq lb_i(d_i) \end{aligned}$$

情况 2：如果 a_i 是非叶子节点，在收到来自 a_c 关于 d_i 的 BACKTRACK 消息后， a_i 会用真实的代价代替初始的下界。因此，为证明该引理，仅需证明对于所有 a_c 而言，其初始下界 $lb_i^c(d_i)$ 不大于 $cost(Spa_c)$ 即可。其中， $Spa_c \subset Spa_i$ 是根为 a_c 子树对应的部分解。下面我们将采用数学归纳法进行证明。

归纳基础：当 a_i 的孩子是叶子节点，对于其每个孩子节点 a_c ，有：

$$\begin{aligned} cost(Spa_c) &= \sum_{a_j \in AP(a_c)} f_{jc}(d_j, d_c) + f_{cj}(d_c, d_j) \\ &= \sum_{a_j \in AP(a_c)} f_{jc}(d_j, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) \\ &\geq \min \left(\sum_{a_j \in AP(a_c)} f_{jc}(d_j, x_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) \right) \\ &= \min \left(f_{ic}(d_i, x_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) + \sum_{a_j \in PP(a_c)} f_{jc}(d_j, x_c) \right) \\ &\geq \min \left(f_{ic}(d_i, x_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) \right) + \sum_{a_j \in PP(a_c)} \min_{x_c} f_{jc}(d_j, x_c) \\ &\geq util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in PP(a_c)} \min_{x_c} f_{jc}(d_j, x_c) \\ &= util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + EstRep_i(x_c)^- \\ &= util_i^{c-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_j \in Desc(a_c) \cap \{a_c\}} EstRep_i(x_j)^- \\ &= lb_i^c(d_i) \end{aligned}$$

当最大内存维度限制 $k_{mb} \geq |Sep(a_i)| + 1$ 时，上式的第四步到第五步成立。由于 a_c 为叶子节点，有 $Desc(a_c) = \emptyset$ ，因此，定理的第六步到第七步成立。据此，该引理

的归纳基础成立。

现假设上述命题对所有的 $a_c \in C(a_i)$ 成立。接下来，将证明命题对所有 a_i 也成立。对于每一个孩子 a_c ，有：

$$\begin{aligned}
 cost(Spa_c) &= \sum_{a_k \in \{a_c\} \cup Desc(a_c)} \sum_{a_j \in AP(a_k)} f_{jk}(d_j, d_k) + f_{kj}(d_k, d_j) \\
 &= \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) + \sum_{a_k \in Desc(a_c)} \sum_{a_j \in AP(a_k)} f_{jk}(d_j, d_k) + f_{kj}(d_k, d_j) \\
 &= \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) \\
 &\quad + \sum_{a_c' \in C(a_c)} \sum_{a_k \in Desc(a_c') \cup \{a_c\}} \sum_{a_j \in AP(a_k)} f_{jk}(d_j, d_k) + f_{kj}(d_k, d_j) \\
 &= \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) + \sum_{a_c' \in C(a_c)} cost(Spa_{c'}) \\
 &\geq \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + f_{jc}(d_j, d_c) + \sum_{a_c' \in C(a_c)} lb_c^{c'}(d_c) \\
 &\geq f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_c' \in C(a_c)} lb_c^{c'}(d_c) + \sum_{a_j \in PP(a_c)} \min_{x_c} f_{jc}(d_j, x_c) \\
 &= f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_c' \in C(a_c)} lb_c^{c'}(d_c) + EstRep_i(x_c)^- \\
 &= f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_c' \in C(a_c)} \left(util_c^{c^-}(Cpa_{i[Sep(a_c)]}, d_c) \right. \\
 &\quad \left. + \sum_{a_l \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_l)^- \right) + EstRep_i(x_c)^-
 \end{aligned}$$

由于 $Desc(a_c) = \{a_j | \forall a_j \in Desc(a_c) \cup \{a_c\}, \forall a_{c'} \in C(a_c)\}$ ，所有关于乐观累积代价预估的项可以进行合并，即：

$$\begin{aligned}
 cost(Spa_c) &\geq f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(d_c, d_j) + \sum_{a_c' \in C(a_c)} util_c^{c^-}(Cpa_{i[Sep(a_c)]}, d_c) \\
 &\quad + \sum_{a_l \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_l)^- \\
 &\geq \min_{x_c} \left(f_{ic}(d_i, d_c) + \sum_{a_j \in AP(a_c)} f_{cj}(x_c, d_j) + \sum_{a_c' \in C(a_c)} util_c^{c^-}(Cpa_{i[Sep(a_c)]}, x_c) \right) \\
 &\quad + \sum_{a_l \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_l)^- \\
 &= util_i^{c^-}(Cpa_{i[Sep(a_c)]}, d_i) + \sum_{a_l \in Desc(a_c) \cup \{a_c\}} EstRep_i(x_l)^- \\
 &= lb_i^c(d_i)
 \end{aligned}$$

综上，上述命题成立。

引理 4.3：给定任意一个部分解 Cpa_i 与 a_i 赋值 (x_i, d_i) ，其子树任意赋值 Spa_i 与所产生的代价 $cost(Spa_i)$ 不大于上界 $ub_i(d_i)$ 。

证明: 由于该命题的证明同引理 4.2 的证明类似, 为避免冗余, 我们忽略该引理的证明过程。该命题证明与引理 4.2 证明的最主要的区别在于最小效用表和乐观累积代价预估被最大效用表和悲观累积代价预估替换。

命题 4.1: 给定任意一个部分解 Cpa_i , a_i 最优子问题赋值 $Spa_i(d_i^*)$ 所产生的代价等于下界 LB_i 和上界 UB_i , 即 $LB_i = cost(Spa_i(d_i^*)) = UB_i$ 。

证明: 基于算法 4.2 的第 61 行, a_i 汇报 $Spa_i(d_i^*)$ 的条件是 $LB_i = UB_i$ 。进一步, 根据算法 4.2 的第 62-63 行以及公式 (4.11), 有 $ub_i(d_i^*) = LB_i = UB_i$ 。由公式 (4.8-4.11) 可知 $lb_i(d_i^*) \geq \max_{d_i \in D_i} lb_i(d_i) = LB_i$ 和 $lb_i(d_i^*) \leq ub_i(d_i^*)$, 因此可推导出 $lb_i(d_i^*) = ub_i(d_i^*) = LB_i = UB_i$ 。此外, 引理 4.2 和引理 4.3 也已证明 $lb_i(d_i^*) \leq cost(Spa_i(d_i^*))$ 以及 $cost(Spa_i(d_i^*)) \leq ub_i(d_i^*)$ 。因此, 有 $LB_i = cost(Spa_i(d_i^*)) = UB_i$, 且命题 4.1 得证。

引理 4.4: 给定任意一个部分解 Cpa_i , a_i 子树中任意一个代价大于 $\min(TH_i, UB_i)$ 的完整赋值都不可能出现在代价小于整个问题已知搜索上界的完整解中。

证明: 记 a_i 子树的完整赋值为 Spa_i , 其对应代价为 $cost(Spa_i)$ 。上述命题等价于证明: 如果一个 Spa_i 满足 $cost(Spa_i) > \min(TH_i, UB_i)$, 那么对于父节点 $a_j = P(a_i)$ 一定有 $cost(Spa_j) > \min(TH_j, UB_j)$, 其中 $Spa_j \subset Spa_i$ 。

从算法 4.2 的第 5 行, 可知 TH_i 是 a_j 通过 CPA 消息发送的子问题阈值。而且通过命题 4.1 已证明, 当 a_i 回溯至 a_j 时有 $UB_i = cost(Spa_i)$ 。此外, 若 a_i 通过回溯汇报子问题的完整赋值, 那么该完整赋值的代价不大于 TH_i 。因此, a_i 不可能回溯 Spa_i 。根据算法 4.2 的第 58 行, 如果从 a_j 处收到 TH_i , 有:

$$\min(TH_j, UB_j) = TH_i + high_cost_j(d_j) + \sum_{a_c \in C(a_j) \wedge c \neq i} lb_j^c(d_j)$$

因此, 当 $cost(Spa_j) > \min(TH_j, UB_j)$, 必然有 $cost(Spa_j) > \min(TH_j, UB_j)$ 。

定理 4.1: PT-ISABB 算法是完备的

证明: 从引理 4.1-4.4 可得, 算法会终止并且所有被剪枝的赋值都是次优的。因此 PT-ISABB 算法是完备的。

4.3.3 复杂度分析

算法的复杂度主要体现在四个方面: 1) 智能体的存储复杂度; 2) 智能体的计算复杂度; 3) 消息大小; 4) 消息数量。

在智能体的存储复杂度方面: 由于每个智能体需要存储每个孩子节点的推理结果 $util_i^{c\pm}$, 而 $util_i^{c\pm}$ 的大小与节点的诱导宽度和 k_{mb} 相关。此外, 智能体还需保存搜索结果, 包括 $lb_i^c(d_i)$, $ub_i^c(d_i)$, $lb_i(d_i)$, $ub_i(d_i)$, LB_i 和 UB_i 等数据结构。因此, a_i 总整体存储复杂度为 $O(|C(a_i)| * d_{\max}^{\min(k_{mb}, |Sep(a_i)| + 1)} + (|C(a_i)| + 1) * |D_i|)$ 。这里, $d_{\max} = \max_{x_i \in X} |D_i|$ 。

在智能体的计算复杂度方面：在推理阶段，智能体处理一条 UTIL 消息的计算复杂度为 $O(d_{\max}^{\min(k_{mb}, |Sep(a_i)|+1)})$ 。在搜索阶段，当智能体收到一条 CPA 消息后会去找第一个可行的赋值，因此 a_i 处理一条 CPA 消息的计算复杂度为 $O(|D_i|^*|AP_i|)$ 。当智能体收到 RESD_REQ 消息后，它直接查表并返回对应的剩余约束代价，因此其计算复杂度为 $O(1)$ 。当智能体收到 RESD 消息或来自孩子节点的 BACKTRACK 消息后，需要更新搜索上界，因此计算复杂度为 $O(|D_i|^*|C_i|)$ 。

对于算法产生的消息大小而言：由于每个 UTIL 消息的维度并不会超过 k_{mb} ，因此来自于 a_i 的一个 UTIL 消息的大小为 $O(d_{\max}^{\min(k_{mb}, |Sep(a_i)|+1)})$ 。对于 CPA 消息，其包含对每个智能体的赋值，阈值消息以及对该智能体所有后代智能体的所有乐观和悲观累积代价预，因此其大小为 $O(|A|)$ 。其它消息包括 RESD_REQ, RESD, BACKTRACK, STOPEXPLORER 和 TERMINATE 都只携带一些标量，因此只需要 $O(1)$ 的空间。

对于算法所需的消息数而言：不同于 DPOP/ADPOP 算法，由于没有值传播阶段，因此 PT-ISABB 算法在推理阶段仅需要 $n - 1$ 的消息数。与其它基于搜索的完备算法一样，本文提出算法在搜索阶段产生 $O(d_{\max}^n)$ 数量的消息。

4.4 基于非本地消元的推理与搜索混合的 ADCOP 非完备算法

本节提出两个 PT-ISABB 的非完备版本用于权衡算法的求解质量和算法所需的计算与通信开销，且算法求解质量受用户指定的误差限制。具体地，本节将 ADOPT 的绝对误差机制以及 BnB-ADOPT 的相对误差机制适配到 PT-ISABB 算法。其中，绝对误差机制允许用户在解的代价上指定一个绝对误差界，而相对误差机制允许用户在解的代价上指定一个相对误差界。

在基于搜索的异步完备算法（例如 ADOPT 和 BnB-ADOPT）上实现绝对误差机制和相对误差机制时，只有根节点需要通过松弛终止条件做出改变。然而在同步算法中，智能体只有在探索完其子问题后才进行回溯，因此根节点无法及时地更新其上、下界。也就是说，目前应用于异步算法终止条件的松弛策略并不适用于本章提出的 PT-ISABB 算法。为将这两种误差机制应用于 PT-ISABB，本节提出为每个智能体 $a_i \in A$ 构建误差界 gap_i 来放松其回溯条件（算法 4.1 中的第 7 行，17 行，32 行），即：

$$LB_i \geq \min(UB_i - gap_i, TH_i) \quad (4.14)$$

此外，还需对 BACKTRACK 消息的接收和发送处理上进行如下调整：

① 发送 BACKTRACK 消息后： a_i 验证 Cpa_i 为可行部分解的条件从 $LB_i = UB_i \wedge LB_i < TH_i$ 变为 $LB_i \geq UB_i - gap_i \wedge LB_i < TH_i$ 。此外，回溯的内容也需要进行调整。具体地，如果 Cpa_i 可行， a_i 发送给父节点的内容从

$(ub_i(d^*), Spa_i(d^*))$ 变为 $(lb_i(d^*) ub_i(d^*), Spa_i(d^*))$ ；若 Cpa_i 不可行， a_i 发送的上文内容从 (∞, \emptyset) 变为 $(\infty, \infty, \emptyset)$ 。

② 从 a_c 处接收到 BACKTRACK 消息后， a_i 的上、下界的更新过程由 $lb_i^c(d_i) \leftarrow \max(lb_i^c(d_i), opt_c)$ 和 $ub_i^c(d_i) \leftarrow \max(ub_i^c(d_i), opt_c)$ 变为 $lb_i^c(d_i) \leftarrow \max(lb_i^c(d_i), lb_c)$ 和 $ub_i^c(d_i) \leftarrow \max(ub_i^c(d_i), ub_c)$ 。

当 gap_i 足够大时（即 $gap_i \geq UB_i - LB_i$ ），PT-ISABB 算法的两种非完备版本可能在没有找到任意完备解的情况下终止。上述问题可以通过如下策略解决。假定在算法开始时，智能体 $a_i \in A$ 为其控制变量 x_i 选择一个默认值 d_i^0 。当一个非常大的 gap_i 被指定给非根节点 a_i 时， a_i 带着赋值 Spa_i^0 进行回溯，其中 $Spa_i^0 = \bigcup_{a_j \in Desc(a_i)} \{(x_j, d_j^0)\} \cup \{(x_i, d_i^0)\}$ 且 $d_i^* = \operatorname{argmin}_{d_i \in D_i} ub_i(d_i)$ 。这样每个变量对应的默认值即构成算法的一个完整解，且该解满足对解代价的误差要求。接下来，对该策略的正确性进行证明。

引理 4.5： 给定 $gap_i \geq UB_i - LB_i$ ，非根节点 a_i 回溯的部分解 Spa_i^0 对应的代价满足 $LB_i \leq \text{cost}(Spa_i^0) \leq LB_i + gap_i$ 。

证明：根据引理 4.2 和 4.3，有 $\text{cost}(Spa_i^0) \geq lb_i(d_i^*)$ 以及 $\text{cost}(Spa_i^0) \leq ub_i(d_i^*)$ ，其中 $d_i^* = \operatorname{argmin}_{d_i \in D_i} ub_i(d_i)$ 。基于公式 (4.10) 和 (4.11)，有 $lb_i(d_i^*) \geq LB_i$ 以及 $ub_i(d_i^*) \geq UB_i$ 。因此，可推导出 $LB_i \leq \text{cost}(Spa_i^0) \leq UB_i$ 。由于 $gap_i \geq UB_i - LB_i$ ，有 $LB_i \leq \text{cost}(Spa_i^0) \leq LB_i + gap_i$ 。因此，该引理得证。

4.4.1 绝对误差机制

ADOPT 的绝对误差机制要求用户指定一个绝对误差界 b ($0 \leq b \leq \infty$)，以使得算法求得解的代价和最优解代价的差值不超过 b 。然而对于 PT-ISABB 算法中，每个非根节点的最优子问题的解代价是严格不大于最优解代价，所以 b 并不能直接应用于放松非根节点的回溯条件。因此，基于推理阶段的推理结果，引入绝对误差界分配机制来为每个节点 a_i 分配绝对误差界 b_i 。具体地，如果 a_i 是根节点， b_i 等于用户定义的绝对误差界 b 。否则， b_i 是其父节点的误差界 $b_{P(a_i)}$ 的划分，可由公式 (4.15) 进行计算并通过 CPA 消息由父节点 $P(a_i)$ 发送给 a_i 。

$$b_i = b_{P(a_i)} * \left(\frac{u_i}{\sum_{a_j \in C(P(a_i))} u_j} \right) \quad (4.15)$$

其中 $u_i \in \{util_{P(a)}^{i+}(Cpa_i), util_{P(a)}^{i-}(Cpa_i), util_{P(a)}^{i+}(Cpa_i) - util_{P(a)}^{i-}(Cpa_i)\}$ 。公式 (4.15) 背后的设计思路为：将 $b_{P(a)}$ 平均地分配给每一个孩子节点 $a_j \in C(P(a_i))$ ，这种思路可通过依据 a_j 子问题的实际代价按比例来分配 a_j 的误差界实现。但由于 a_j 子问题的实际代价在 a_j 报告汇报之前是未知的，可采用推理阶段获得的效用表来估计实际的代价。因此，绝对误差机制可以通过设置 a_i 的误差界 gap_i 来实现，其

计算公式如下：

$$gap_i = b_i \quad (4.16)$$

我们将具有绝对误差界的 PT-ISABB 算法的非完备版本命名为 PT-ISABB_{AEM}。在该算法中，一旦根节点 a_i 的上、下界之间的差值小于用户指定的绝对误差 ($UB_i - LB_i \leq b$)， a_i 通知所有智能体终止搜索过程。算法终止时，PT-ISABB_{AEM} 算法获得的解代价 $cost_i$ 满足 $LB_i \leq cost_i \leq LB_i + gap_i$ 。

4.4.2 相对误差机制

与指定绝对误差界不同，BnB-ADOPT 的相对误差机制允许用户在求解代价上指定一个相对误差界 ρ ($1 \leq \rho \leq \infty$) 以使得算法求解代价最多是最优解代价的 ρ 倍。这项机制可以通过公式 (4.17) 计算 gap_i 的值，从而将其部署到 PT-ISABB。

$$gap_i = (\rho - 1) * LB_i \quad (4.17)$$

我们将带相对误差机制的 PT-ISABB 算法非完备版本命名为 PT-ISABB_{REM}。在该算法中，一旦根节点 a_i 的上、下界的差值小于相对误差 gap_i 时， a_i 通知所有智能体终止搜索过程。算法终止时，PT-ISABB_{REM} 算法的求解代价 $cost_i$ 满足 $LB_i \leq cost_i \leq \rho * LB_i$ 。

命题 4.2： 给定 Cpa_i ，在 PT-ISABB 的两个非完备版本 (PT-ISABB_{AEM} 和 PT-ISABB_{REM}) 中，子问题完整赋值 $Spa_i(d_i^*)$ 所产生的代价应不小于 LB_i 且不大于 $LB_i + gap_i$ ，即 $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$ 。

证明： 根据算法 4.2 中第 52-53 行、公式 (4.10) 和公式 (4.11)，有 $ub_i(d_i^*) = \min_{d_i \in D_i} ub_i(d_i) = UB_i$ 以及 $lb_i(d_i^*) \geq \min_{d_i \in D_i} lb_i(d_i) = LB_i$ 。由于引理 4.2 和引理 4.3 已经证明 $lb_i(d_i^*) \leq cost(Spa_i(d_i^*))$ 以及 $cost(Spa_i(d_i^*)) \leq ub_i(d_i^*)$ ，因此有 $LB_i \leq cost(Spa_i(d_i^*)) \leq UB_i$ 。根据公式 (4.17)、 $LB_i \leq UB_i - gap_i$ 、 $LB_i \geq \min(UB_i - gap_i, TH_i)$ ，有 $LB_i \geq UB_i - gap_i$ 。因此，可以推导出 $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$ 。

定理 4.2： 当 PT-ISABB 算法的两个非完备版本算法 (PT-ISABB_{AEM} 和 PT-ISABB_{REM}) 终止时，根节点 a_i 获得的部分解 $Spa_i(d_i^*)$ 代价 $cost(Spa_i(d_i^*))$ 满足 $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$ 。

证明： 从命题 4.2 中推导出，当算法满足终止条件时，部分解 $Spa_i(d_i^*)$ 对应的代价满足 $LB_i \leq cost(Spa_i(d_i^*)) \leq LB_i + gap_i$ 。

4.5 实验评估

本节首先通过消融实验对 PT-ISABB 的四种完备版本 (具体配置见表 4.1) 进行对比分析；随后，在随机 ADCOP、无尺度网络以及非对称 MaxDCSP 上对 PT-ISABB 与 ADCOP 完备算法 (包括 SyncABB-1ph、ATWB 以及 AysmPT-FB) 进

行性能比较。为证明推理阶段能提供紧的界，也与 SyncABB-1ph 的树型版本 PT-SABB 进行对比。最后，实验对比 PT-ISABB 的两种非完备版本。

表 4.1 四种版本 PT-ISABB 的具体配置

Table 4.1 The PT-ISABB variants

算法类别	消元机制	效用表传播机制
PT-ISABB-A	本地消元	最小效用表
PT-ISABB-B	本地消元	最大、最小效用表
PT-ISABB-C	非本地消元	最小效用表
PT-ISABB-D	非本地消元	最大、最小效用表

4.5.1 参数调优

实验一：不同智能体数下，四种完备版本的 PT-ISABB 进行参数调优实验。具体地，我们选择值域大小均为 3 的随机 ADCOP 问题，设置代价选取范围为 $[0, 100]$ ，密度为 0.25，并将智能体个数按步长 2，从 8 变化到 18。

表 4.2 PT-ISABB 四种变体在 $k_{mb} = 2$ 的性能比较Table 4.2 Performance comparison of the PT-ISABB variants given $k_{mb} = 2$

智能 体数	算 法	消息数	网络负载 (KB)	NCLOs	运行时间(ms)
8	A	370.27	3.73	236.77	29.38
	B	358.88	3.80	315.85	35.50
	C	217.81	2.16	119.04	22.58
	D	56.00	0.55	85.65	3.81
10	A	1137.85	12.71	660.85	91.27
	B	1117.19	12.96	837.69	106.15
	C	556.35	6.39	341.77	54.85
	D	307.35	4.14	372.54	24.12
12	A	3409.92	39.34	1687.00	266.35
	B	3378.35	40.09	2135.46	316.42
	C	1570.69	19.22	933.35	139.58
	D	1193.23	16.67	1210.04	101.62
14	A	13,699.42	172.78	5890.38	1236.58
	B	13,576.15	174.28	6867.81	1497.62

智能 体数	算 法	消息数	网络负载 (KB)	NCLOs	运行时间(ms)
16	C	4649.85	59.97	2450.23	468.85
	D	4266.62	59.50	3279.46	388.04
	A	101,986.73	1305.41	36,308.08	8671.31
	B	101,815.38	1312.97	42,172.08	9695.62
	C	35,529.04	471.18	16,186.73	3402.88
	D	34,622.38	485.18	21,888.38	3144.19
	A	518,513.23	6,685.46	162,332.73	36,623.77
	B	518,332.46	6730.11	197,276.46	42,277.85
18	C	173,486.27	2338.11	77,594.15	14,518.19
	D	172,672.69	2468.63	112,518.50	13,977.42

表 4.3 PT-ISABB 四种变体在 $k_{mb} = \infty$ 的性能比较Table 4.3 Performance comparison of the PT-ISABB variants given $k_{mb} = \infty$

智能 体数	算 法	消息数	网络负载(KB)	NCLOs	运行时间(ms)
8	A	370.35	3.73	237.19	26.62
	B	354.50	3.80	315.81	33.38
	C	216.92	2.16	118.81	15.58
	D	56.00	0.55	85.65	3.46
	A	1157.69	13.01	656.15	91.42
	B	1135.31	13.18	812.54	102.35
	C	732.31	8.24	367.58	62.15
	D	576.58	6.67	371.27	46.31
10	A	3928.88	46.30	1690.88	312.42
	B	3893.31	46.56	1904.35	356.15
	C	2971.73	35.44	1193.92	246.50
	D	2807.65	33.60	1249.50	232.35
12	A	17,771.62	228.38	6851.31	1732.27
	B	17,768.42	228.89	7138.12	1887.35
	C	12,913.15	165.28	4603.42	1211.38
	D	12,677.65	162.57	4659.73	1083.35
16	A	205,709.62	2798.58	65,622.88	18,202.08

智能体数	算法	消息数	网络负载(KB)	NCLOs	运行时间(ms)
18	B	205,654.12	2799.26	66,009.46	20,340.62
	C	172,231.27	2342.58	53,152.12	16,387.31
	D	171,273.50	2327.88	53,150.50	14,779.04
	A	1,184,726.46	16,150.03	307,890.77	97,720.81
	B	1,184,047.15	16,143.35	308,208.92	98,058.62
	C	898,321.54	12,193.97	230,693.65	76,554.12
	D	886,177.60	12,089.82	229,888.36	69,072.92

表 4.2 和表 4.3 给出实验一，在给定 $k_{mb} = 2$ 和 $k_{mb} = w^* + 1$ 时，四个 PT-ISABB 完备算法的性能对比。从表中可以看出，在固定 k_{mb} 的值时，带非本地消元版本算法（PT-ISABB-C 和 PT-ISABB-D）的性能要明显优于带本地消元版本算法（PT-ISABB-A 和 PT-ISABB-B）。且在当没有内存限制时，这种优势更加明显。具体而言，当 $k_{mb} = 2$ 时，PT-ISABB-C 和 PT-ISABB-D 相较于 PT-ISABB-A 和 PT-ISABB-B 提高大约 20%-40%。当 $k_{mb} = w^* + 1$ 时，性能的提升达到 50%-65%。这些现象表明在 PT-ISABB 中，带非本地消元的推理阶段能够提供更紧的下界，从而能实现高效剪枝和更好地降低搜索阶段产生的计算和通信开销。此外，在消息数、网络负载以及运行时间方面，PT-ISABB-A 效果和 PT-ISABB-B 相同，而 PT-ISABB-D 表现要优于 PT-ISABB-C。这是由于 PT-ISABB-D 中产生的上界和下界之间的差值要小于 PT-ISABB-C 中上、下界之间差值。因此，在 PT-ISABB-D 中，每个智能体能高效地利用上、下界之间较小的差值来减少算法所需探索的解空间。

4.5.2 完备算法的性能比较

实验二：不同智能体数下，ADCOP 完备算法的性能比较实验。实验的具体配置与实验一中的配置相同。图 4.3 给出实验二的实验结果。从图中可以观察到随着智能体数的增加，所有算法的通信与计算开销都呈指数增长。其中，本章提出的算法在各个度量指标中都优于其它对比算法，这体现出推理和搜索混合求解 ADCOP 的优势。此外，从图中还可以看出，即使在内存预算较小的情况下（例如 $k_{mb} = 2$ ），PT-ISABB-D 需要的通信与计算开销也明显少于 AsymPT-FB。具体而言，PT-ISABB-D ($k_{mb} = 2$) 在消息数以及网络负载方面要优于 AsymPT-FB 算法 88% 左右，在 NCLOs 和运行时间方面 PT-ISABB-D 则优于 AsymPT-FB 算法 75% 左右。这主要是由于 PT-ISABB-D 不依赖于消息传递来实现前向定界过程计算下界，此外也表明本章提出的算法即使在内存预算相对较低的情况下也能提供更紧的下界。

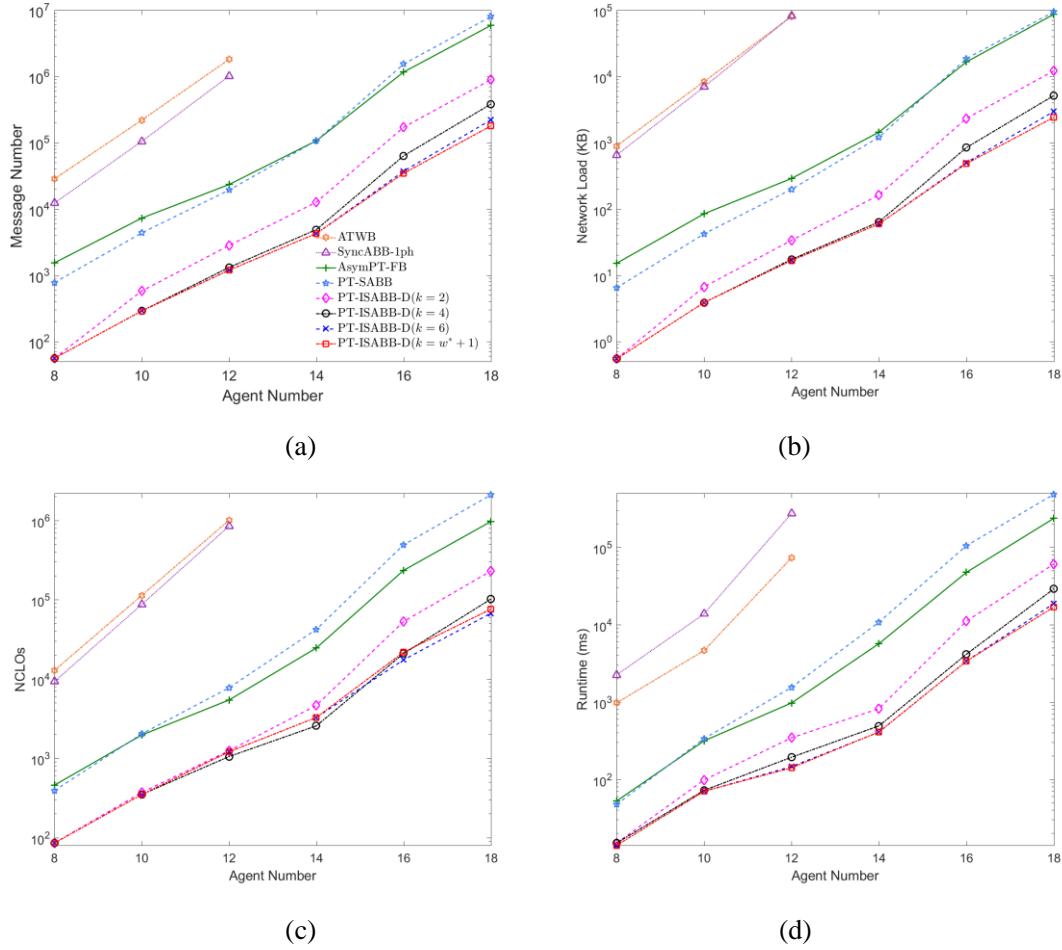


图 4.3 不同智能体数下各个算法的性能对比

Fig.4.3 Performance comparison under different agent numbers

实验三：不同密度下，各个 ADCOP 完备算法的性能比较实验。该项实验的目的是从密度的角度增加问题求解的难度，对比各个算法的性能。具体地，我们选择值域大小均为 8 的随机 ADCOP 问题，代价选取范围为 $[0, 100]$ ，固定智能体个数为 8，并将密度按步长 0.15，从 0.25 变到 1。图 4.4 给出实验三的实验结果，其中实验中所建伪树的平均诱导宽度为 1~6。在这个配置中，搜索空间的大小不会改变，问题的复杂度由拓扑结构反映。从图中可以看出，所有基于树的算法在稀疏问题上都表现良好，但随着约束图密度的增加这种优势逐渐减小。这是由于约束图的密度会影响算法所构造伪树的结构并因此影响搜索的并行性。具体而言，稠密问题通常产生更少分支的伪树，使得这些基于伪树的算法需要比 SyncABB-1ph 产生更多的消息。即便如此，当问题是全连接时（密度为 1.0），本章所提出的 PT-ISABB-D 仍然比 SyncABB-1ph 表现要好，这也说明更紧下界的对剪枝的必要性。至于 SyncABB-1ph，其在除运行时间外的其他所有指标上表现都要优于 ATWB。这是由于 SyncABB-1ph 可以通过回溯 Cpa 信息给所有已被赋值的

智能体来实现双边代价累积。而在 ATWB 中，该过程是通过同时发送 Cpa 的副本给已被赋值的智能体来实现的。另外在稀疏问题 $\rho < 0.4$ 中，PT-ISABB-D 在不同 k_{mb} 下的表现相似，但在稠密问题 $\rho \geq 0.4$ 中其表现变化较大。这是由于在求解稀疏问题时，产生的伪树具有较小的诱导宽度进而使得 PT-ISABB-D 在推理阶段需要近似的维度较少。

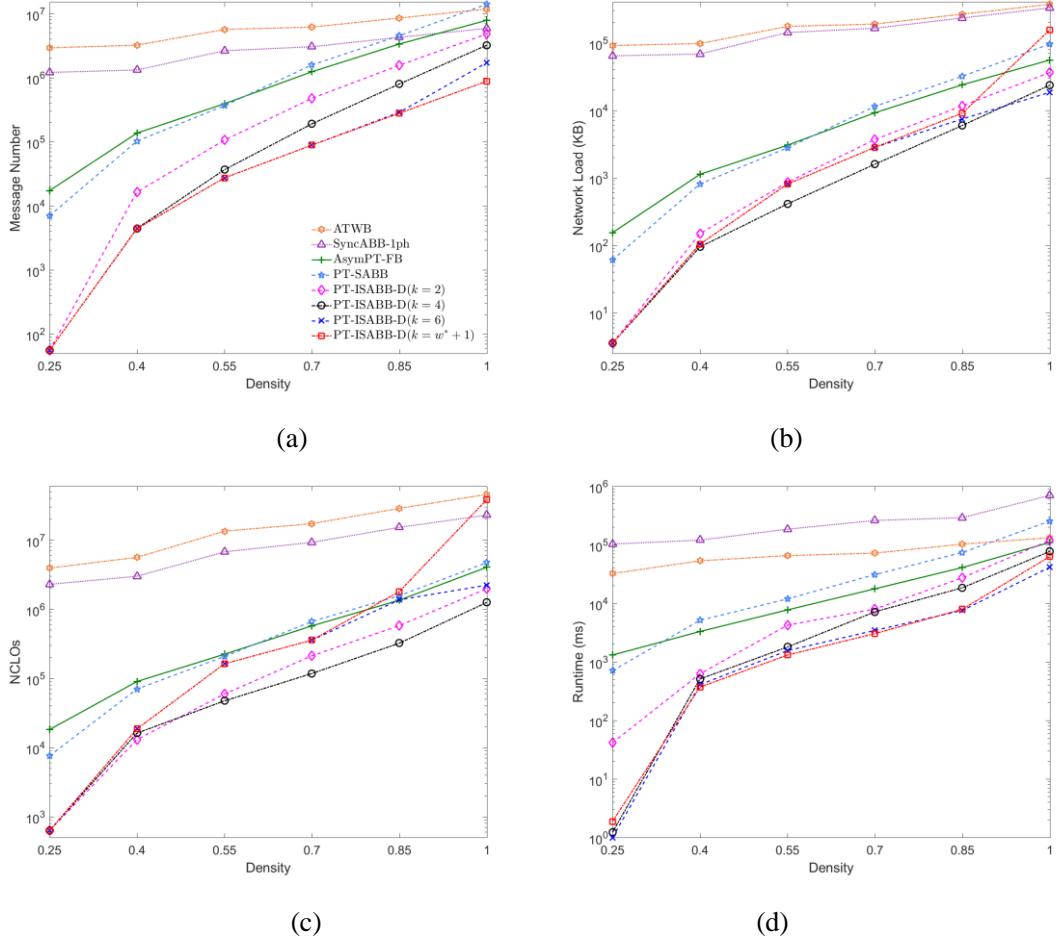


图 4.4 不同密度下各个算法的性能对比

Fig.4.4 Performance comparison under different densities

实验四：不同算法在求解无尺度网络问题时的性能对比。该实验的目的是对比各个 ADCOP 完备算法在求解结构化问题时的性能。具体地，我们选择值域大小均为 3 的无尺度网络问题，设置代价选取范围为 $[0, 100]$ ，固定智能体个数为 16， m_0 为 10，并将 m_1 按步长 2，从 2 变到 10。图 4.5 给出在不同 m_1 下各种算法在无尺度网络问题上的运行结果。由于 SyncABB-1ph 和 ATWB 在 15 分钟内无法求解 $m_1 = 2$ 的无尺度网络问题，所以未将这两个算法的实验结果放在实验对比图中。从图中可知，PT-ISABB-D 性能优于其他对比算法，并且这种优势在当内存预算的

增加更加明显。此外，虽然 AsymPT-FB 在 NCLOs 和运行时间指标上要优于 PT-SABB，但它们在消息数和网络负载等评价指标上表现相似。这种现象再次说明前向定界过程虽然能构建较紧下界，但其需要较大的网络开销。

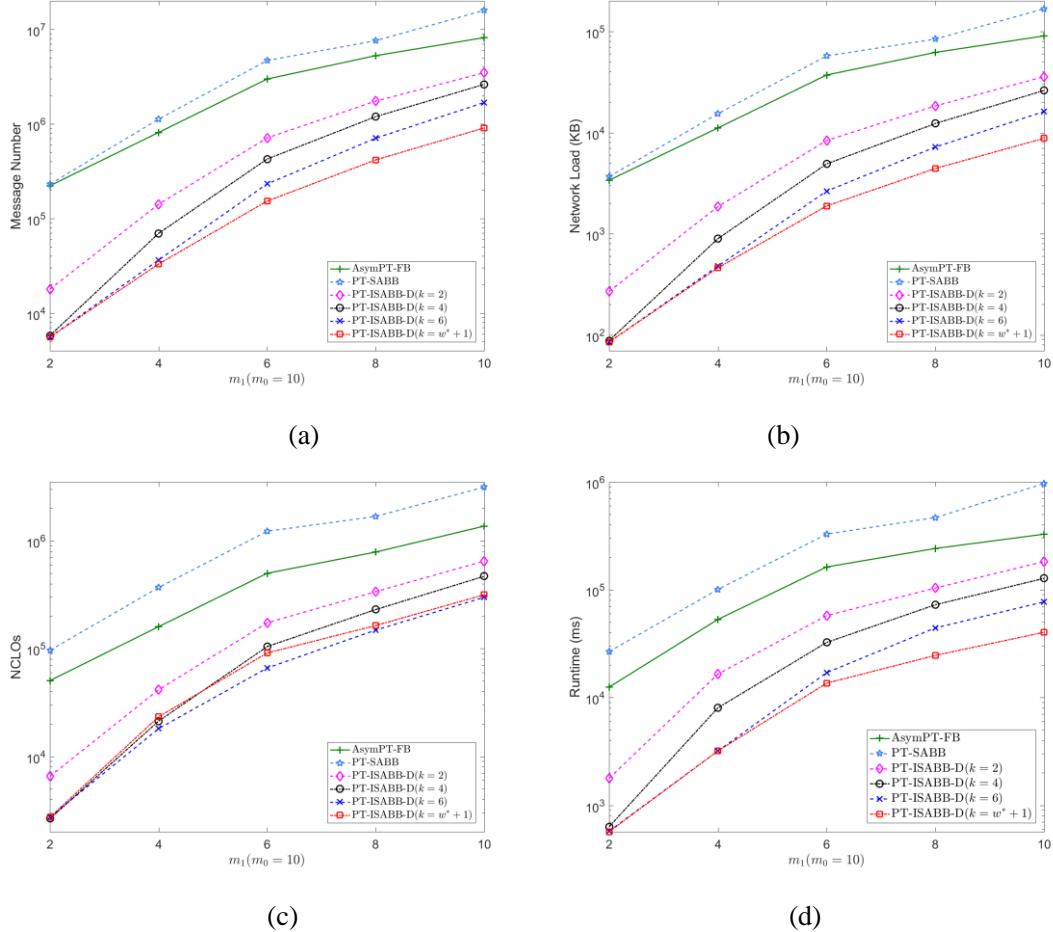
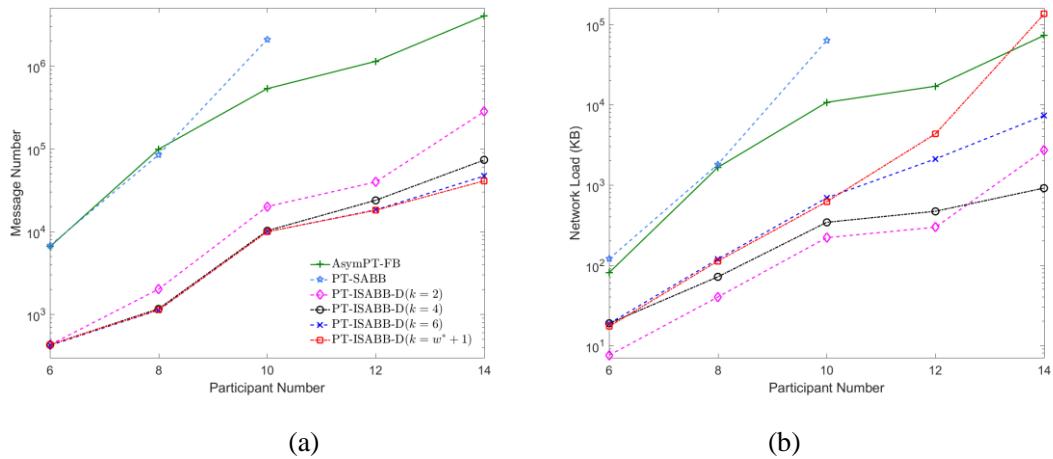


图 4.5 各个算法在求解无尺度网络问题时的性能对比

Fig.4.5 Performance comparison on scale-free networks



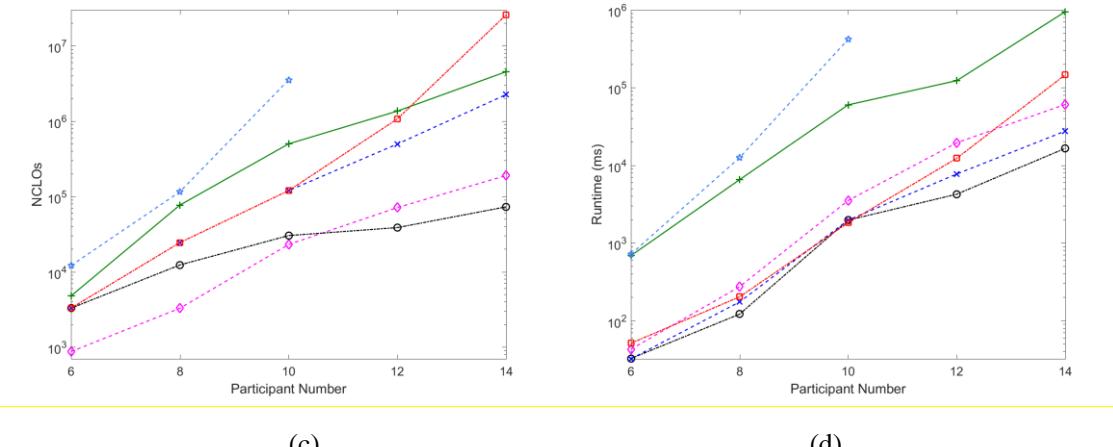


图 4.6 各个算法在求解会议调度问题时的性能对比

Fig.4.6 Performance comparison on meeting schedule

实验五：不同算法在求解会议调度问题时的性能对比。该实验的目的是对比各个 ADCOP 完备算法在求解实际应用问题时的性能。具体地，我们设置参会人数按步长 2，从 4 变化到 14；固定会议数为 10；每个人从 10 个会议中随机选择 2 个会议进行参与；每个会议的持续时间随机从 2 到 6 个时间槽进行选择；会议可安排的时间槽为 8。

图 4.6 给出在不同参会下各种算法在会议调度问题上的运行结果。由于 SynchABB-1ph 和 AWTB 在 15 分钟内无法求解参会人数为 4 的会议调度问题，所以未将这两个算法的实验结果放在实验对比图中。从图中可以看出，我们提出的 PT-ISABB-D 在消息数上明显优于其他对比算法，且这种优势随着内存预算的增加更加明显，如图 4.5 (a) 所示。这说明 PT-SABB-D 算法在计算下界的紧度优于目前最好的搜索算法 AsymPT-FB，且内存预算的增加能提升 PT-SABB-D 算法所提供的下界紧度。但在其他评价指标（例如网络负载或 NCLOs）上，算法的性能在 或时性能最优。这是由于 PT-SABB-D 算法由推理阶段和搜索阶段两部分组成，推理阶段所产生的消息大小和 NCLOs 都是指数于内存预算。而搜索阶段，每个消息大小以及所需的计算量都是线性的。因此，最优的内存预算和带求解的问题规模以及实际的应用需求相关，是需要针对实际应用场景进行调优。

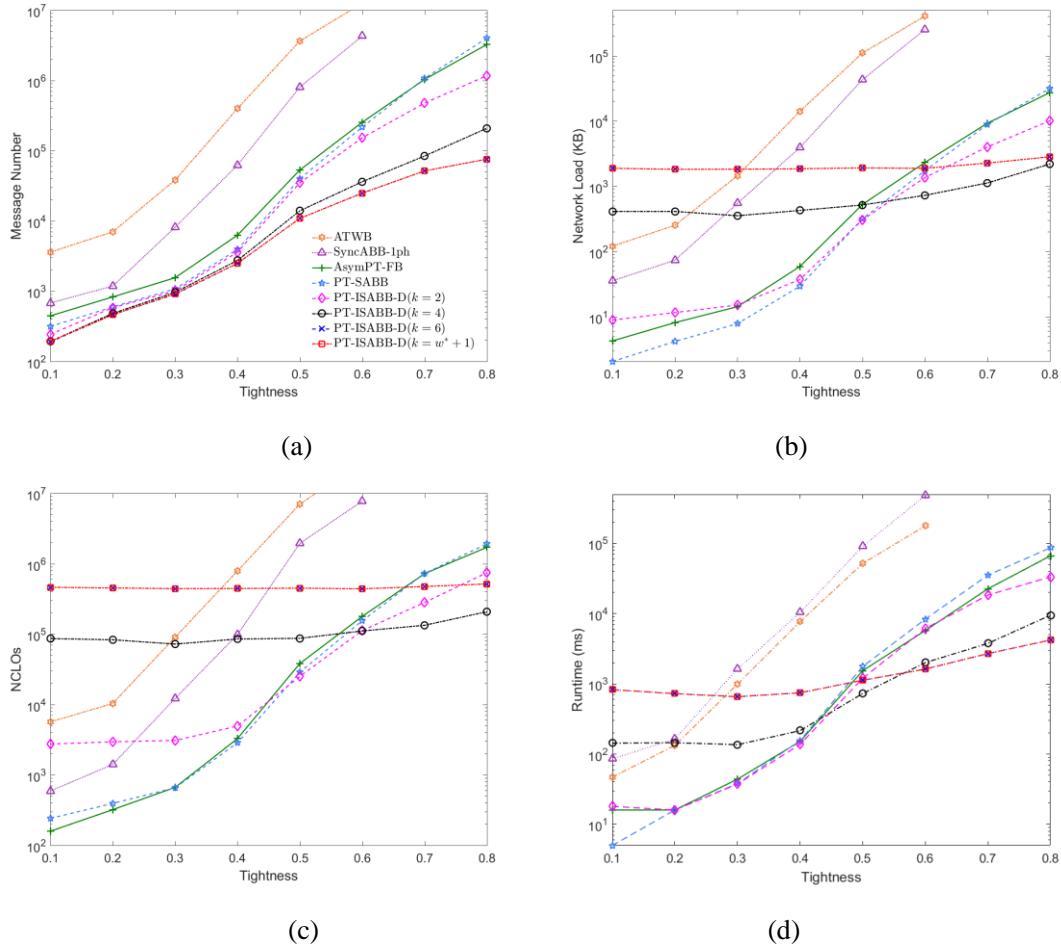


图 4.7 各个算法在求解不同紧度下非对称 MaxDCSP 问题时的性能对比

Fig.4.7 Performance comparison for asymmetric MaxDCSPs under different tightness

实验六：不同紧度下，算法求解约束满足问题的性能比较。该项实验的目的是从约束满足问题紧度的角度来增加问题求解的难度，对比各个算法的求解性能。具体地，我们选择值域大小均为 10 的非对称 MaxDCSP 问题，固定智能体个数为 10，密度为 0.4，并将紧度按步长 0.1，从 0.1 变到 0.8。

图 4.7 给出实验六的实验结果。在该实验配置中，既没有增加算法所需探索的搜索空间，也没有影响问题的拓扑结果，但却增加算法剪枝的难度。从图中可以发现除 ATWB 以外，所有算法在紧度较低的情况下都仅需要较少的消息数。这是由于在解决这些问题时，搜索算法能够快速找到较低的上界，从而能通过剪枝过程降低算法所需探索的解空间。但随着紧度的增加，非零的组合数也会增加，使得基于搜索的算法不再能很快地找到高质量的上界，从而导致算法需要搜索更多的解空间来获得最优解。此外，由于 SyncABB-1ph 和 ATWB 没有利用拓扑结构信息来加速搜索过程，它们在解决紧度达到 0.6 时表现较差。另一方面，基于树的算法在每个分支节点处能将问题分解成为多个子问题并实现子问题的并行求解。在

这些基于树的算法中，本章所提出的 PT-ISABB-D 在 $k_{mb} \geq 4$ 能产生更小的计算和网络负载开销，这说明推理阶段能够提供更紧的下界。换句话说，尽管根据性质 4.1，在 $k_{mb} = w^* + 1$ 时 PT-ISABB-D 产生的下界在理论上同 AsymPT-FB 产生的下界紧度相同，但在实际中计算这样的下界所需消耗的内存要小得多。此外，从图中还可以看出，在解决低紧度问题时 PT-SABB-D 相比于 AsymPT-FB 需要更小的通信开销，这也再次说明前向定界在消息传播过程中会产生巨大的网络负载开销。另外，在解决低紧度问题时，具有较大 k_{mb} 值的 PT-ISABB 相比于其他算法产生更多的网络负载、NCLOs 以及运行时间。这是由于在求解值域较大的问题时，推理阶段所需的计算和通信开销都较大。然而在相同情况下，基于搜索的算法能在即使下界较差的情况下也能很快地找到一组可行解。

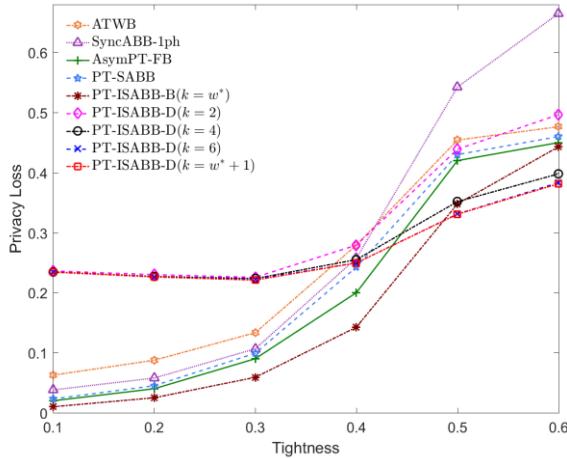


图 4.8 各个算法在求解不同紧度下非对称 MaxDCSP 问题时的私有偏好泄露

Fig.4.8 Privacy loss for asymmetric MaxDCSPs under different tightness

实验七：不同紧度下，各算法求解约束满足问题的私有偏好保障性能比较。该项实验的目的是测试各个 ADCOP 完备算法在求解不同紧度的约束满足问题时所泄露的私有偏好对比。具体地，考虑紧度按步长 0.1，从 0.1 变到 0.6，其他参数配置与实验五相同。

图 4.8 给出在不同紧度下各个算法的在问题求解过程中的私有偏好泄露的对比情况，在本实验中也包含 PT-ISABB-B（即本地消元版本的 PT-ISABB），并在约束保护方面与非本地消元版本的 PT-ISABB 以及本地消元版本的 PT-ISABB 进行比较。不同于其他的对比算法，PT-ISABB-D 的私有偏好泄露来自于推理与搜索两个阶段。如 4.4 节所述，在推理阶段，当父节点执行变量消元时，最坏情况下会导致一半的私有偏好泄露。尽管如此，这依然比直接使用 DPOP 求解 ADCOP 更加有

效，这是因为直接采用 DPOP 求解会泄露至少一半的私有信息。在搜索阶段，在本节所提出的预估汇报机制中，（伪）父节点仅需向（伪）孩子节点提供其累积的约束代价，这在一定程度上可以减少私有偏好泄露。但在某些特定条件下，智能体仍然可能推测出另一边实际的约束代价值。由于在求解低紧度问题时，该问题一般存在可行解。因此，来自孩子节点的效用表中大部分的元素都为零，因此本章提出的算法在求解该类问题时相比于其他算法会泄露更多的私有偏好。但随着问题紧度的增加，PT-ISABB-D 的优势逐渐显现。当紧度超过 0.5 时，PT-ISABB-D ($k_{mb} \geq 4$) 相比于其他算法泄露更少私有偏好。具体原因如下：一是随着紧度的增加，父节点可以推断得出为零的赋值对逐渐减少；二是推理阶段所产生的紧的下界能够在搜索阶段实现高效地剪枝。此外，在解决紧度小于 0.5 的问题时，PT-ISABB-B 在私有偏好保障方面表现得比其他算法得更好。这是因为所有的变量在给其父节点发送效用表之前就已经被消除，因此，父节点只知道其孩子节点的最优代价为零，但无法知道其对应的赋值。

4.5.3 非完备算法的性能比较

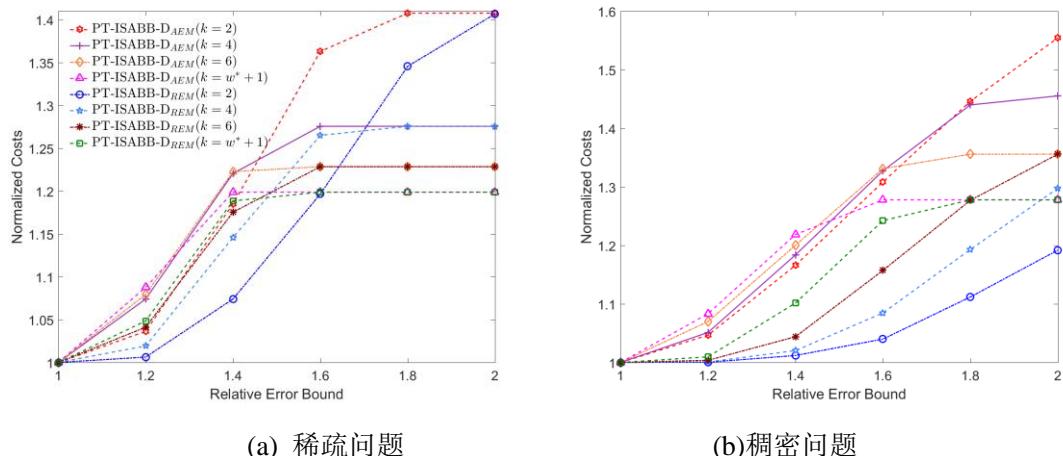


图 4.9 不同相对误差下，各个算法在求解稀疏和稠密问题时的解的归一化代价值

Fig.4.9 Normalized costs of the sparse problems and sense problems under different relative error bound

实验八：不同相对误差界下，ADCOP 非完备算法在稀疏和稠密配置的解质量对比。具体地，我们选择值域大小为 3 的随机 ADCOP 问题，设置代价选取范围为 $[0, 100]$ ，并将智能体个数为 24，密度为 0.15 的问题设为稀疏问题；将智能体个数为 14，密度为 0.6 的问题设为稠密问题。此外，以相对误差区间为 1 到 2 来评估两个非完备版本算法求解的解质量和性能。这里，PT-ISABB-D_{REM} 的相对误差界为 ρ ，PT-ISABB-D_{AEM} 的绝对误差为 b 最优解与 ρ 的乘积。由于公式 (4.15) 中三

种配置会导致相同的结果，我们选取 $v_i = \text{util}_{P(a_i)}^{i+}(Cpa_i) - \text{util}_{P(a_i)}^{i-}(Cpa_i)$ ，并预先计算平均的最优解代价给 b 赋值。在实验中，对于稀疏问题平均最优解代价为 2741，对于稠密问题的平均最优解代价为 4234。

图 4.9 给出实验八的实验结果。这里，稀疏问题中所建立伪树的平均诱导宽度为 6，稠密问题中所建立伪树的平均诱导宽度为 8。可以看出两个非完备版本算法的归一化求解代价随着相对误差界的增加而增大。然而，算法求解的代价仍然要比误差界要小得多。在相同的相对误差界下，PT-ISABB-D_{AEM} 的归一化求解代价总是比 PT-ISABB-D_{REM} 的归一化求解代价要大。这是由这两个非完备算法的终止条件所致。对于这两个非完备版本算法而言，其根节点 a_i 有 $LB_i \leq cost_i \leq LB_i + gap_i$ ，其中 $cost_i$ 是求解代价。在 PT-ISABB-D_{AEM} 中 $gap_i = (\rho - 1) * optCost_i$ ，在 PT-ISABB-D_{REM} 中 $gap_i = (\rho - 1) * LB_i$ ，这里 $optCost_i$ 是最优解代价。由于 $optCost_i \geq LB_i$ ，PT-ISABB-D_{AEM} 的绝对误差界 gap_i 不会超过 PT-ISABB-D_{REM} 的 gap_i 。此外，当相对误差比较小时，可以注意到 PT-ISABB-D_{AEM} 在不同的 k_{mb} 下，求解代价有一些不同，但随着相对误差界的增加，更大 k_{mb} 的优势也就逐渐显现。其原因是随着 k_{mb} 值的增加，在用户指定的误差界较大时，推理阶段产生的上界和下界的差值更接近所指定的误差界。当不同 k_{mb} 下，PT-ISABB-D_{REM} 解决稀疏问题时这种现象同样会发生，但在稠密问题上却有所不同。这是由于在 $k_{mb} \leq 6$ 时，PT-ISABB-D_{REM} 中的启发式上、下界对对算法的终止作用很小。这主要是由于稠密问题中伪树的平均诱导宽度要高于稀疏问题中伪树的诱导宽度。另外，由于在推理阶段较大的 k_{mb} 能够产生更紧的下界来快速终止 PT-ISABB-D_{REM}，这也使得在更大 k_{mb} 值下 PT-ISABB-D_{REM} 算法的效果不佳。

实验九：不同相对误差界下，ADCOP 非完备算法在稀疏和稠密配置的性能对比。该项实验的目的是通过调整相对误差界来增加算法的求解难度，测试算法在问题求解时的性能对比。具体实验参数配置与实验七相同。

图 4.10 给出在实验九下，两种非完备算法在解决稠密问题和稀疏问题时的性能比较，其评价指标为消息数和 NCLOs。从图中可以看出，随着相对误差界的增加，PT-ISABB-D_{AEM} 的性能并不总是下降，而是在相对误差界为 1.2 时 PT-ISABB-D_{REM} 产生最大的消息数和 NCLOs。这是由于在相对误差界等于 1.2 时，搜索阶段无法进行无效的剪枝，从而使得 PT-ISABB-D 在 $\rho = 1$ 和 $\rho = 1.2$ 时产生相同数量的消息数和 NCLOs。具体而言，在较大相对误差界下，两个非完备算法无法很快获得较低的下界，但其仍然需要付出不超过最优解代价的 1.2 倍之内的代价找到一组解。并且在相对误差界较小时，相同 k_{mb} 下的 PT-ISABB-D_{AEM} 产生比 PT-ISABB-D_{REM} 更多的信息和 NCLOs，但这种现象随着相对误差界的增加发生改变。这是由于在相对误差界较小时，PT-ISABB-D 中绝对误差界分配机制没有平均

地分配绝对误差界。但随着相对误差界的增加，分配逐渐趋于平均。此外，从图 4.10 (a) 和 4.9 (a) 可以看出，在相对误差界足够大时， $k_{mb} \geq 4$ 下的非完备算法在指定的误差界下能很快地找到稀疏问题的解。

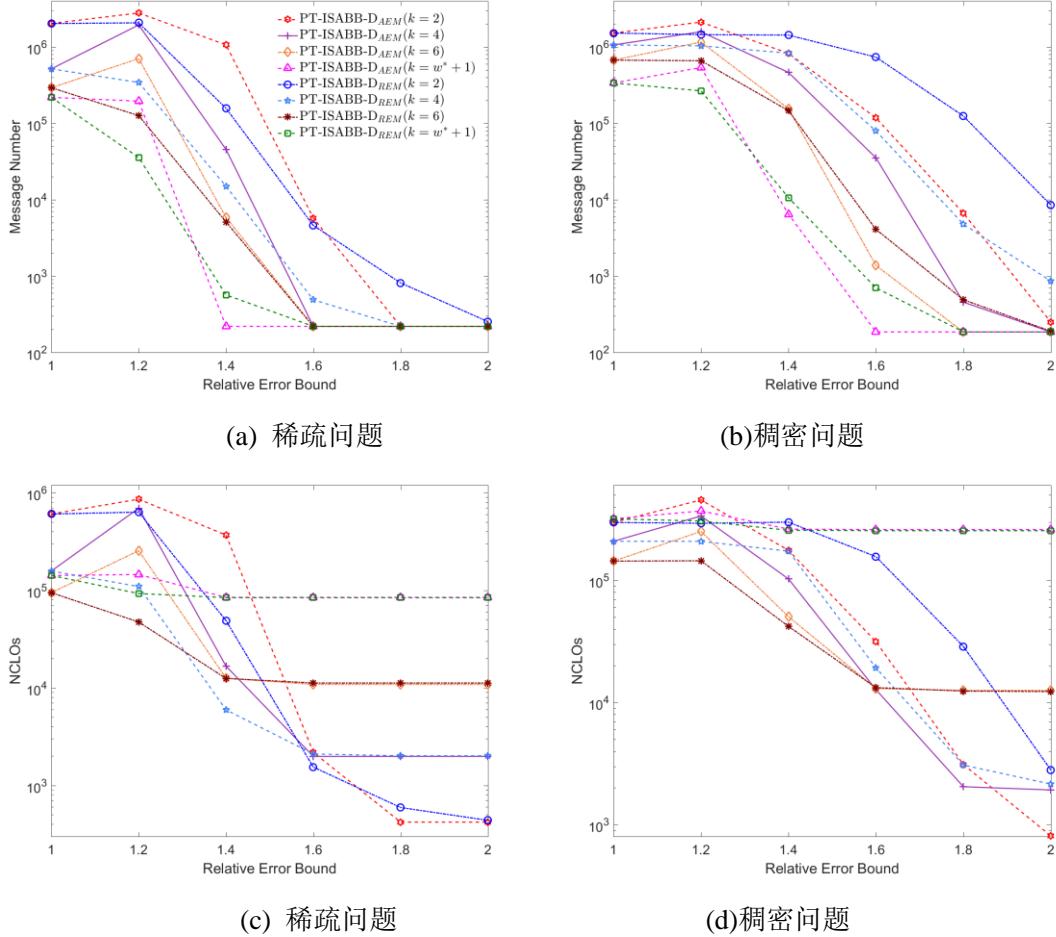


图 4.10 不同相对误差下，各个算法在求解稀疏和稠密问题时的性能对比

Fig.4.10 Performance comparison of the sparse problems and sense problems under different relative error bound

4.6 本章小结

由于私有偏好需要保障的问题，用于求解 DCOP 的 DPOP/ADPOP 无法直接用于求解 ADCOP。在本章中，通过与基于树的 SyncABB-1ph 进行结合，采用 ADPOP 来求解 ADCOP，并且提出搜索-推理混合的 ADCOP 完备算法 PT-ISABB。在推理阶段，执行非本地消元版本的 ADPOP 求解部分约束代价函数，保存推理结果为搜索阶段建立上、下界。在搜索阶段，运行基于树的 SyncABB-1ph 版本来穷举解空间。进一步，引入预估汇报机制来计算完整的上界，同时避免直接泄露私有约束代价。此外，通过引入 PT-ISABB-D_{AEM} 和 PT-ISABB-D_{REM} 两个 PT-ISABB 的非完

备版本来权衡解的质量和计算开销。这里， $\text{PT-ISABB-D}_{\text{AEM}}$ 和 $\text{PT-ISABB-D}_{\text{REM}}$ 能够保证分别在用户指定的绝对误差界和相对误差界下找到符合条件的解。在 $\text{PT-ISABB-D}_{\text{AEM}}$ 中，引入绝对误差界分配机制来为所有的非根节点智能体分配用户指定的绝对误差界，以松弛它们的回溯条件。实验结果表明， PT-ISABB 优于基于搜索的算法以及它的本地消元版本算法，并且在解决复杂问题时泄露更少的私有偏好。同时，可以从实验结果中发现，当相对误差界高于 1.2 时， PT-ISABB 的非完备版本能够在用户指定的界误差内以较少的计算开销找到符合条件的解。

5 基于广义非本地消元的 ADCOP 完备推理算法^①

5.1 引言

现有的 ADCOP 完备算法（例如 SyncABB, ATWB, AsymPT-FB 和 PT-ISABB）都采用系统地搜索策略来穷举整个解空间，导致这些算法在求解过程中需要指数级的消息数，从而无法求解较大规模问题。本章的实验结果显示，目前最好的完备搜索算法仅能求解智能体数小于 20 的随机 ADCOP 问题。

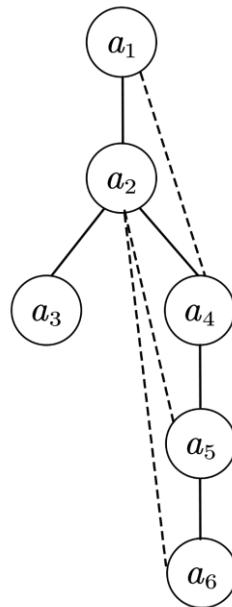


图 5.1 说明 ADCOP 完备推理算法运行的伪树结构实例

Fig. 5.1 A pseudo tree instance for Inference-based complete algorithms for ADCOP

另一方面，基于推理的 DCOP 完备算法（例如 DPOP）在问题求解过程中仅需要线性的消息数，但不依靠 PEAV 框架其无法直接用于求解 ADCOP。这主要是因为，DPOP 在执行本地消元策略过程中，需要累积与消元变量相关的全部约束。若直接应用于求解 ADCOP，（伪）父节点需要将它们所拥有的私有约束代价函数转移到消元变量处，进而保证消元时所有与消元变量相关的约束代价函数获得累积。但将私有约束代价函数直接泄露的方式将导致一半以上的私有约束函数泄露，因此这种方式求解 ADCOP 是不可取的。以图 5.1 中的智能体 a_6 为例，为执行本地消元（即 $\min_{x_6} (f_{62} \otimes f_{65} \otimes f_{56} \otimes f_{26})$ ），其需要获得父节点 a_5 的私有约束代价函数 f_{56}

^① 本章内容已被期刊《Artificial Intelligence Review》录用

和伪父节点 a_2 的私有约束代价函数 f_{26} ，这将直接导致（伪）父节点私有约束泄露。

为克服上述缺陷，通过推迟变量消元的位置（即广义推迟消元，为本文第四章所提出非本地消元机制的推广），本节提出基于推理的 ADCOP 完备算法 AsymDPOP。不同于本地消元机制（例如 DPOP），变量的消元位置是在本地节点上。不同于非本地消元机制（例如 PT-ISABB），变量的消元位置是在父节点上。在广义推迟消元中，变量的消元位置推迟到最高的（伪）父节点。此外，为避免由于推迟消元带来计算量以及内存消耗的增加，本章提出通过传播一组小的效用表集合（TSPS，Table Set Propagation Scheme）而不是一个大的联合后的效用表来降低算法的内存占用；通过分批次的消元而不是联合消元来降低算法的计算量（MBES，Mini Batch Elimination Scheme）。此外，为使得本章提出的算法适用于内存有限的环境，本章提出将内存有界推理思想适配到本章提出的算法（即 AsymDPOP 和带 TSPS 的 AsymDPOP）中，提出 RMB-AsymDPOP 和带 TSPS 的 RMB-AsymDPOP 算法。

本章的内容安排如下：首先，5.2 节描述本章所需的算法背景；随后在 5.3 节介绍基于推理的 ADCOP 完备算法 AsymDPOP，并理论分析该算法的内存占用情况，并在 5.4 节提出 TSPS 和 MBES 两个权衡机制分别用于降低 AsymDPOP 算法的内存占用和计算量，并理论分析带 TSPS 的 AsymDPOP 算法的内存占用情况；接着，5.5 节描述两种内存有界的 AsymDPOP 算法，即：RMB-AsymDPOP 和带 TSPS 的 RMB-AsymDPOP 算法，并在 5.6 节分析本文提出算法的私有偏好泄露情况；最后，5.7 节通过实验验证本章提出的 ADCOP 完备算法的优势，并在 5.8 节对本章工作进行小结。

5.2 算法背景

本节将介绍与本章提出算法相关基础算法背景，即内存有界推理的完备求解算法 MB-DPOP 和 RMB-DPOP。

如 3.2.1 节所述，DPOP 算法由于其内存消耗指数于伪树的诱导宽度，因此无法求解大规模问题。随后，有学者提出 MB-DPOP (Memory-Bounded DPOP) 算法，其通过在问题求解过程中传播更多的消息数来降低算法的内存消耗。MB-DPOP 是一种内存有界的算法，即算法运行过程中产生的最大消息的维度不超过用户指定的内存预算 k_{mb} 。因此，伪树中诱导宽度大于 k_{mb} 的节点需要执行不同于传统 DPOP 算法中的推理操作，这些节点组成的区域称为“簇”，其形式化定义如下：

定义 5.1（簇）：给定维度限制为 k 。在伪树中，一个簇（Cluster）是一个连通子图 $\langle CR, V, E \rangle$ ，其中， CR (Cluster Root) 为该簇的簇头，其满足 $|Sep(CR)| \leq k_{mb}$ 且 $a_i \in Desc(CR), \forall a_i \in V \setminus \{CR\}$ ；簇中除 CR 节点外的剩余节点满足

$|Sep(a_i)| > k_{mb}, \forall a_i \in V \setminus \{CR\}$; $e_{ij} \in E$ 为连接 $a_i \in V$ 和 $a_j \in V$ 的边。特别地，我们称 $C_i^{in} = C(a_i) \cap V$ 为簇内节点 a_i 的簇内孩子节点； $\{a_i \in V | C_i^{in} = \emptyset\}$ 为簇的叶子节点，称之为 CL (Cluster Leaf) 节点。

为使得簇内节点执行推理操作，MB-DPOP 提出两种额外的阶段，即：标记阶段和有界效用表传播阶段。其中，标记阶段用于将伪树中诱导宽度大于 k_{mb} 的区域分为多个簇，且为每个簇选择对应的 CC 节点 (Cycle-Cut node)。每个簇都有一个簇头 (Cluster Root, CR)，一个诱导宽度不大于 k_{mb} 的分割节点。此外，对于每个簇而言，一旦 CC 节点被移除，簇内任意节点的诱导宽度不大于 k_{mb} 。

有界效用传播阶段采用迭代传播内存有界效用表的方式来实现 DPOP 中维度大于 k_{mb} 效用表的传播。这里，MB-DPOP 利用 CC 节点的赋值信息来保证迭代传播的效用表维度不大于 k_{mb} (即内存有界的效用表)。具体地，CR 节点负责枚举所有 CC 节点的赋值实例，并将该赋值实例向下传播给簇内的所有节点。簇内节点收到该赋值实例后，利用该赋值实例去降低传播的效用表维度 (即获得有界效用表)，并将该有界效用表至下而上地传播到 CR 节点。一旦 CR 节点枚举完所有 CC 节点的赋值，其联合所有簇内孩子节点的有界效用表以及簇外孩子节点的效用表，执行变量消除操作，并将消元后的效用表向上传播。

由于 MB-DPOP 无法利用问题的结构信息，导致其在有界效用传播阶段传播大量冗余的效用表。因此，RMB-DPOP (Refined MB-DPOP) 提出三种改进机制，包括迭代选择机制 (ISM, Iterative Selection Mechanism)、分布式枚举机制 (DEM, Distributed Enumeration Mechanism) 和缓存机制 (CACHE, CACHE Mechanism)。其中，ISM 提出的目的是优化 CC 节点的选择，使得每个 CC 节点能覆盖簇内尽可能多的节点，从而降低每个簇对应 CC 节点的个数。DEM 利用伪树中分支的独立性来降低迭代有界效用表迭代传播的次数，即：不同于 MB-DPOP，仅 CR 节点枚举所有 CC 节点的赋值。在 DEM 中，每个簇内节点都有可能参与 CC 节点赋值的枚举。CACHE 机制提出的目的就是在有界效用传播阶段中利用历史推理结果来进一步降低重复推理。

5.3 非对称伪树优化过程

在本节，首先描述 AsymDPOP 算法的两个阶段，即：效用传播阶段和值传播阶段；随后，理论分析 AsymDPOP 算法的内存占用情况。

5.3.1 基于广义非本地消元的效用传播阶段

在 DCOP 完备推理算法 (例如 DPOP 和 MB-DPOP) 中，当收到所有子节点的 UTIL 消息时，智能体可执行“本地消元” (即智能体负责消除自身所控制的变量) 而不会影响算法的完备性。这是因为在执行变量消元时，与该变量相关的所有约

束代价函数都已被累积。然而，这一结论并不适用于 ADCOP 求解，因为涉及该变量的约束代价函数包含两面的私有约束代价函数（即节点自己一面的私有约束代价函数以及在（伪）父节点那一面的私有约束代价函数）。因此，对该变量进行本地消元时，无法将（伪）父节点那一面的私有函数考虑在内，进而在消元过程中会引入偏差且无法保证消元的完备性。以图 5.1 为例，由于私有函数 f_{56} 和 f_{26} 分别由智能体 a_5 和 a_2 持有，因此智能体 a_6 在执行本地消元时无法累积这些私有约束代价函数。实际上，和变量 x_6 相关的函数包括 f_{56} 、 f_{26} 、 f_{65} 和 f_{62} 。因此，对 x_6 执行本地消元可能会导致消元偏差，从而无法保证算法的完备性。一种简单的解决方案是让所有父节点将它们与本地变量相关的私有约束代价函数直接泄露给孩子节点，以促进本地变量的消元（例如： a_6 执行本地消元时，节点 a_5 和 a_2 将它们持有的私有函数 f_{56} 和 f_{26} 直接转发给 a_6 ），这将导致一半以上的私有约束函数泄露。

受非本地消元的启发，考虑另一种私有约束函数保障方案来累积约束代价函数，即：将变量的消元延迟到它的最高（伪）父节点。并将该消元方案称为广义非本地消元机制（GNLE，Generalized Non-Local Elimination）。在 GNLE 中，智能体 a_i 负责消除变量集合 EV_i ， EV_i 中的变量最高（伪）父节点为 a_i ，即：

$$EV_i = \{a_j \in AC(a_i) \mid a_k \notin AP(a_j), \forall a_k \in Sep(a_i)\} \quad (5.1)$$

由于与 EV_i 变量相关的所有约束代价函数都已在 a_i 处被累积，因此 EV_i 可以在 a_i 的累积效用表 $util_i$ 中被最优地消除。这里， $util_i$ 由 a_i 本地私有函数以及从孩子节点收到的效用表 $util_{c \rightarrow i}, \forall a_c \in C(a_i)$ 组成，即：

$$util_i = \left(\bigotimes_{a_j \in N(a_i)} f_{ij} \right) \otimes \left(\bigotimes_{a_c \in C(a_i)} util_{c \rightarrow i} \right) \quad (5.2)$$

从 $util_i$ 中消除 EV_i 后，可获得 a_i 发送给父节点的效用表 $util_{i \rightarrow p}$ ，即：

$$util_{i \rightarrow p} = \min_{EV_i} \left(\bigotimes_{a_j \in N(a_i)} f_{ij} \right) \otimes \left(\bigotimes_{a_c \in C(a_i)} util_{c \rightarrow i} \right) \quad (5.3)$$

根据公式 (5.3)，尽管传播给祖先节点的效用表 $util_{i \rightarrow p}$ 包含 a_i 的私有约束代价函数且没有执行本地消元（根据公式 (5.2) 有 $a_i \notin EV_i$ ），但祖先节点很难从收到的效用表 $util_{i \rightarrow p}$ 中推断出 a_i 私有约束代价函数（详细分析见 5.5 节）。

由于采用公式 (5.3) 计算传播效用表时，没有考虑各分支的独立性，存在巨大的计算开销浪费。事实上，可以通过分配消元顺序对公式 (5.3) 进行优化，即：

$$util_{i \rightarrow p} = \left(\bigotimes_{a_j \in AP(a_i)} f_{ij} \right) \otimes \left(\bigotimes_{a_c \in C(a_i)} \left(\min_{EV_i^c} \left(\left(\bigotimes_{a_e \in B_i^c} f_{ej} \right) \otimes util_{e \rightarrow i} \right) \right) \right) \quad (5.4)$$

其中， B_i^c 为 a_i 在 a_c 分支下的所有孩子和伪孩子节点集合，即：

$$B_i^c = \{a_e\} \cup (PC(a_j) \cap Desc(a_c)) \quad (5.5)$$

EV_i^c 是 EV_i 中属于分支 a_c 的变量集合，可形式化定义为：

$$EV_i^c = \{a_j \in B_i^c \mid a_k \notin AP(a_j), \forall a_k \in Sep(a_i)\} \quad (5.6)$$

由于 $EV_i^c \cap EV_i^{c'} = \emptyset, \forall a_c, a_{c'} \in C(a_i), c \neq c'$ 且 $EV_i^c = \bigcup_{a_c \in C(a_i)} EV_i^c$, 依据公式 (5.3) 和公式 (5.4) 计算获得效用表 $util_{i \rightarrow p}$ 是相同的。

算法 5.1 给出 AsymDPOP 在效用传播阶段的伪代码。算法开始由叶节点将其效用表通过 UTIL 消息发送给父节点 (1-2 行, 9-10 行)。这里, a_i 的效用表 $util_{i \rightarrow p}$ 由公式 (5.4) 计算。当收到所有孩子节点发送的 UTIL 消息后, 若 a_i 为非根节点, 则计算并传播效用表 $util_{i \rightarrow p}$ 给父节点。若 a_i 为根节点, 则开启值传播阶段。

算法 5.1 AsymDPOP 效用传播阶段的伪代码

Algorithm 5.1 Sketch of utility propagation phase in AsymDPOP

Utility propagation phase in AsymDPOP for each agent a_i

输入: 智能体 a_i 的约束函数集合

输出: 智能体 a_i 的效用表

When Initialization:

1. **if** a_i is a leaf **then**
 2. **PropUtil()**
 3. **When received** ($util_c$) from $a_c \in C(a_i)$: //处理接收到的效用表
 4. $util_{c \rightarrow i} \leftarrow util_c$
 5. **if** a_i has received all UTIL from $C(a_i)$ **then**
 6. **if** a_i is the root **then**
 7. start **Value propagation phase**
 8. **else**
 9. **PropUtil()**
 10. **Function PropUtil()** //通过执行非本地效用, 计算并发送效用表
 11. compute $util_{i \rightarrow p}$ according to Eq.(5.4)
 12. send UTIL($util_{i \rightarrow p}$) to $P(a_i)$
-

5.3.2 值传播阶段

与 DPOP 为本地消元变量选择最优赋值不同, 由于智能体 a_i 在效用传播阶段消除 $EV_i^c, \forall a_c \in C(a_i)$ 中的变量。因此, 在值传播阶段 a_i 也需要为这些变量选择最优赋值, 即:

$$X_i^{c*} = \underset{EV_i^c}{\operatorname{argmin}} \left(\left(\bigotimes_{a_j \in B_i^c} f_{ij} \right) \otimes util_{c \rightarrow i} \right) (PA^*) \quad (5.7)$$

此外，由于在算法在效用传播阶段中 $Sep(a_i)$ 已消除部分 $Desc(a_i)$ 中的变量（即下层的接口结点 ID_i ）。因此， a_i 发送给孩子节点 a_c 的赋值不仅将包含 $Sep(a_c)$ 的赋值，还包含 $ID_c \cup EV_i^c$ 的赋值。这里， ID_i 被形式化定义为：

$$ID_i = \{a_j \in Desc(a_i) \mid \exists a_k \in Sep(a_i), s.t., a_k \in AP(a_j)\} \quad (5.8)$$

算法 5.2 AsymDPOP 值传播阶段的伪代码

Algorithm 5.2 Sketch of value propagation phase in AsymDPOP

Value propagation phase in AsymDPOP for each agent a_i

输入：智能体 a_i 效用传播阶段得到的效用表

输出：智能体 a_i 最终的赋值

When Initialization:

11. **if** a_i is the root **then** //根节点计算效用表，并开启值传播阶段
12. compute $util_i$ according to Eq. (9)
13. $v_i^* \leftarrow \operatorname{argmin}_{x_i} util_i$
14. **PropValue**($\{(x_i = v_i^*)\}$)

When received $VALUE(PA^*)$ from $P(a_i)$:

15. **PropValue**(PA^*)
 - Function **PropValue**(PA^*): //最终赋值的计算
 16. **foreach** $a_c \in C(a_i)$ **do**
 17. $PA_i^c \leftarrow PA_{[Sep(x_c) \cup \{x_c\} \cup ID(x_c)]}^*$
 18. **if** $EV(x_i, x_c) \neq \emptyset$ **then**
 19. compute V_i^{c*} for $EV(a_i, a_c)$ according to Eq.(5.8)
 20. $PA_i^c \leftarrow PA_i^c \cup \{(x_j = V_{i[x_j]}^{c*}) \mid \forall x_j \in EV(a_i, a_c)\}$
 21. send $VALUE(PA_i^c)$ to a_c
-

算法 5.2 给出算法在值传播阶段的伪代码。该阶段开始由根节点变量为自己选择最优赋值（12-13 行）。依据收到父节点（15 行）或本地计算（14 行）的最优赋值， a_i 根据公式（5.7）为每个分支 $a_c \in C(a_i)$ 中选择最优赋值（19 行），并将该赋值与已确定的赋值一起通过 VALUE 消息传播给孩子节点 a_c （20-21 行）。当每个叶子节点接收到 VALUE 消息时，算法终止。

5.3.3 复杂度分析

本节将证明 a_i 产生的 UTIL 消息的大小与 $Sep(a_i) \cup \{a_i\} \cup ID_i$ 指数相关。在证明该命题之前，首先证明 $ID_i = (C(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c)) \setminus EV_i$ 。

引理 5.1: $ID_i = (C(a_i) \cup (\bigcup_{a_c \in C(a)} ID_c)) \setminus EV_i$

证明: 该命题的证明分为如下两步。首先证明 $ID_i \cap EV_i = \emptyset$ ，其次证明 $ID_i \cup EV_i = C(a_i) \cup (\bigcup_{a_c \in C(a)} ID_c)$ 。

由公式 (5.9) 可知：

$$\begin{aligned} EV_i &= \{a_j \in AC(a_i) \mid a_k \notin AP(a_j), \forall a_k \in Sep(a_i)\} \\ &= \{a_j \in Desc(a_i) \mid a_i \in AP(a_j), a_k \notin AP(a_j), \forall a_k \in Sep(a_i)\} \end{aligned}$$

又由于 $\{a_i\} \cap Sep(a_i) = \emptyset$ ，可以推出 $ID_i \cap EV_i = \emptyset$ 。

基于上述分析，可以得出以下结论：

$$\begin{aligned} ID_i \cup EV_i &= \{a_j \in Desc(a_i) \mid \exists a_k \in Sep(a_i) \cup \{a_i\}, s.t., a_k \in AP(a_j)\} \\ &= \{a_j \in Desc(a_i) \mid \exists a_k \in Anc(a_i) \cup \{a_i\}, s.t., a_k \in AP(a_j)\} \\ &= \{a_j \in C(a_i) \cup (\bigcup_{a_c \in C(a_i)} Desc(a_c)) \mid \exists a_k \in Anc(a_i) \cup \{a_i\} \\ &\quad , s.t., a_k \in AP(a_j)\} \\ &= \{a_j \in C(a_i) \mid \exists a_k \in Anc(a_i) \cup \{a_i\}, s.t., a_k \in AP(a_j)\} \\ &\quad \cup \{a_j \in \bigcup_{a_c \in C(a_i)} Desc(a_c) \mid \exists a_k \in Anc(a_i) \cup \{a_i\} \\ &\quad , s.t., a_k \in AP(a_j)\} \\ &= C(a_i) \cup \left(\bigcup_{a_c \in C(a_i)} \{a_j \in Desc(a_c) \mid \exists a_k \in Anc(a_i) \cup \{a_i\}\} \right. \\ &\quad \left. , s.t., a_k \in AP(a_j)\} \right) \\ &= C(a_i) \cup \left(\bigcup_{a_c \in C(a_i)} \{a_j \in Desc(a_c) \mid \exists a_k \in Anc(a_c)\} \right. \\ &\quad \left. , s.t., a_k \in AP(a_j)\} \right) \\ &= C(a_i) \cup \left(\bigcup_{a_c \in C(a_i)} \{a_j \in Desc(a_c) \mid \exists a_k \in Sep(a_c)\} \right. \\ &\quad \left. , s.t., a_k \in AP(a_j)\} \right) \\ &= C(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c) \end{aligned}$$

由于 $Sep(a_i)$ 是与 $Desc(a_i) \cup \{a_i\}$ 存在约束的祖先节点组成，因此上式的一步到第二步成立。基于公式 (5.8) 中 ID_i 的定义有上式第八步到最后一步成立。因此，引理 5.1 得证。

定理 5.1: 由 a_i 所产生的 UTIL 消息大小为 $d_{\max}^{|Sep(a_i) \cup \{a_i\} \cup ID_i|}$ ，即 $util_{i \rightarrow p}.dims = Sep(a_i) \cup \{a_i\} \cup ID_i$ 。

证明: 采用归纳法证明该定理。

归纳基础：当 a_i 为叶子节点时，其传播的效用表 $util_{i \rightarrow p}$ 只包含 $AP(a_i) \cup \{a_i\}$ 和（算法 5.1，1-3 行）。且由于 a_i 为叶子节点，有 $Sep(a_i) = AP(a_i)$ 且 $ID_i = \emptyset$ ，因此，归纳基础成立。

现假设该定理适用于所有 $a_c \in C(a_i)$ ，接下来证明该定理也适用于非叶节点 a_i 。根据公式 (5.2) 和公式 (5.3)，有：

$$\begin{aligned}
util_{i \rightarrow p}.dims &= (N(a_i) \cup \{a_i\} \cup (\bigcup_{a_c \in C(a_i)} util_{c \rightarrow i}.dims)) \setminus EV_i \\
&= (N(a_i) \cup \{a_i\} \cup (\bigcup_{a_c \in C(a_i)} Sep(a_c) \cup \{x_c\} \cup ID_i)) \setminus EV_i \\
&= \left(AP(a_i) \cup (\bigcup_{a_c \in C(a_i)} Sep(a_c)) \cup \{a_i\} \cup AC(a_i) \right) \setminus EV_i \\
&\quad \cup C(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c) \\
&= \left(AP(a_i) \cup (\bigcup_{a_c \in C(a_i)} Sep(a_c)) \cup \{a_i\} \cup AC(a_i) \right) \setminus EV_i \\
&\quad \cup (\bigcup_{a_c \in C(a_i)} ID_c) \\
&= \left(AP(a_i) \cup (\bigcup_{a_c \in C(a_i)} Sep(a_c) \setminus \{a_i\}) \cup \{a_i\} \right) \setminus EV_i \\
&\quad \cup AC(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c) \\
&= (Sep(a_i) \cup \{a_i\} \cup AC(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c)) \setminus EV_i
\end{aligned}$$

依据假设，上述公式第一步到第二步成立。由于 $(Sep(a_i) \cup \{a_i\}) \cap EV_i = \emptyset$ ，有：

$$\begin{aligned}
util_{i \rightarrow p}.dims &= Sep(a_i) \cup \{a_i\} \cup (AC(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c)) \setminus EV_i \\
&= Sep(a_i) \cup \{a_i\} \cup (PC(a_i) \cup C(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c)) \setminus EV_i \\
&= Sep(a_i) \cup \{a_i\} \cup (PC(a_i) \setminus EV_i) \cup (C(a_i) \cup (\bigcup_{a_c \in C(a_i)} ID_c)) \setminus EV_i \\
&= Sep(a_i) \cup \{a_i\} \cup (PC(a_i) \setminus EV_i) \cup ID_i \\
&= Sep(a_i) \cup \{a_i\} \cup ID_i
\end{aligned}$$

基于引理 5.1，上述公式第三步到第四步成立。由于 $PC(a_i)$ 中的变量在 a_i 或者 $Sep(a_i)$ 处被消除，有 $PC(a_i) \subseteq EV_i \cup ID_i$ 。进一步基于在引理 5.1 中已被证明的结论 $ID_i \cap EV_i = \emptyset$ ，有 $PC(a_i) \setminus EV_i \subseteq ID_i$ ，因此上述公式第四步到最后一步成立。因此，定理 5.1 得证。

5.4 TSPS 与 MBES 两种权衡机制

本节首先说明 AsymDPOP 在内存和计算方面都存在严重的可扩展性问题，然后提出两种权衡机制来提升 AsymDPOP 算法的性能。

5.4.1 桶集合传播机制

如 5.4.3 节所示，智能体 a_i 发送的 UTIL 消息大小是指数级的。因此，当所构建的伪树结构较差时，算法会产生巨大的内存消耗。考虑图 5.2 所示的伪树。由于每个节点都与根节点存在约束关系，所以 GNLE 只能在根节点 a_1 处对每个变量进行消元，而 a_2 节点在计算传播效用表时的联合操作会产生 $O(d_{\max}^n)$ 的内存消耗。此外，公式 (5.4) 中的联合与消元操作也会产生巨大的计算开销和一个内存占用过大的效用表。

因此，通过避免公式 (5.4) 中不必要的联合操作（即能去除的那些联合操作而不影响变量消元的最优性）可降低每个传播效用表大小，即：每个变量只需要将一组内存占用较小的效用表传递给其父节点，而非传播联合后内存占用过大的

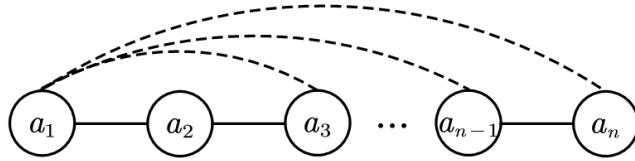


图 5.2 链式结构的伪树

Fig. 5.2 A chain-like pseudo tree

效用表。但完全丢弃联合操作获得的效用表集合会导致严重的私有约束函数泄露。以图 5.6 为例, 如果 a_n 选择不执行联合操作而是直接传递 $f_{n(n-1)}$ 和 f_{n1} 给 a_{n-1} , 那么节点 a_{n-1} 将会知道 a_n 的私有函数。因此, 本章提出通过参数 k_p 来控制每个本地效用表的最大维数从而在内存消耗和私有约束信息保障之间权衡, 并将这一权衡机制称为效用表集合传播方案 (TSPS, Table Set Propagation Scheme)。在 TSPS 中, a_i 发送给其父节点的效用表集合包含其本地效用表集合 $utilSet_{i \rightarrow p}^i$ 和处理收到孩子节点收到的效用表集合获得的结果 $utilSet_{i \rightarrow p}^c, \forall a_c \in C(a_i)$, 即:

$$utilSet_{i \rightarrow p} = \{utilSet_{i \rightarrow p}^j | \forall a_j \in \{a_i\} \cup C(a_i)\} \quad (5.9)$$

其中 $utilSet_{i \rightarrow p}^i$ 可形式化定义为:

$$utilSet_{i \rightarrow p}^i = \text{PartF}\left(\bigcup_{a_j \in AP(a_i)} \{f_{ij}\}, k_p\right) \quad (5.10)$$

$utilSet_{i \rightarrow p}^c$ 可形式化定义为:

$$utilSet_{i \rightarrow p}^c = \text{Elim}\left(\text{JoinSet}\left(\bigcup_{a_j \in B_i^c} \{f_{ij}\}, utilSet_{c \rightarrow i}\right), EV_i^c\right) \quad (5.11)$$

其中, 函数 **PartF** 用于将本地私有函数划分为多个本地效用表, 使得划分出的每个效用表的维度大小都不大于 k_p 。这里, TSPS 仅保证计算获得的每个本地效用表大小不大于 k_p , 不保证整个算法的内存消耗小于 k_p 。函数 **Elim** 和 **JoinSet** 分别被用于实现效用表集合的联合和消元操作。由于仅丢弃公式 (5.4) 中不必要的联合操作, 根据公式 (5.9) 计算效用表集合有 $util_{i \rightarrow p} = \otimes_{u \in utilSet_{i \rightarrow p}} u$ 。因此, TSPS 依然能够保证算法的完备性。

图 5.3 和算法 5.3 分别给出带 TSPS 的 AsymDPOP 的流程图和对应的伪代码。该算法包括效用表集合传播阶段和值传播阶段。不同于 AsymDPOP 中的效用传播阶段, 该效用表集合传播阶段通过执行 TSPS 来传播多个小的效用表集合, 即: 智能体 a_i 传递给其父节点的效用表集合 $utilSet_{i \rightarrow p}$ 由公式 (5.9) 计算 (12-13 行)。具体而言, a_i 首先调用函数 **JoinSet** 联合其私有函数与接收到孩子节点的效用表集合 (20-23 行); 然后调用函数 **Elim** 消除 $EV_i^c, \forall a_c \in C(a_i)$ 中的变量; 最后, 在本地效用表集合中加入消元结果获得传播的效用表集合 $utilSet_{i \rightarrow p}$ 。其中, 本地效用

表集合由函数**PartF**计算获得。在该函数中，与 $AP(a_i)$ 相关的私有函数依据 k_p 被划分为多个本地效用表（26-31行）。如果有任何剩余函数，则将它被加入到本地效用表集合中（32-33行）。

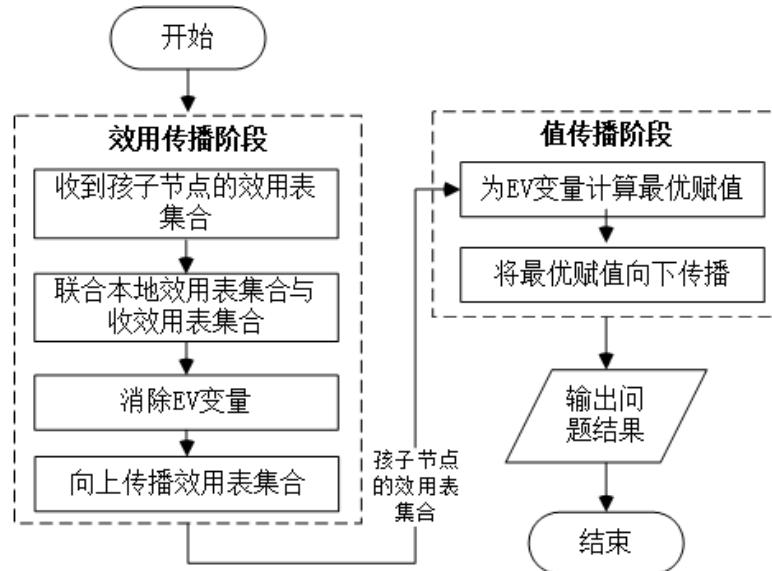


图 5.3 带 TSPS 的 AsymDPOP 流程图

Fig. 5.3 Flow chart of AsymDPOP with TSPS

算法 5.3 带 TSPS 的 AsymDPOP 伪代码

Algorithm 5.3 Sketch of AsymDPOP with TSPS

AsymDPOP with TSPS for each agent a_i

输入：智能体 a_i 的值域以及约束函数集合

输出：智能体 a_i 最终的赋值

When Initialization:

1. **if** a_i is a leaf **then**

2. **PropUtil()**

When received UTILSET($utilSet_c$) from $a_c \in C(a_i)$: //处理收到的效用表集合

3. $utilSet_{c \rightarrow i} \leftarrow utilSet_c$

4. **if** a_i has received all UTILSET from $C(a_i)$ **then**

5. **if** a_i is the root **then**

6. compute $utilSet_i$ according to Eq.(5.11)

7. $v_i^* \leftarrow \operatorname{argmin}_{x_i} (\otimes_{u \in utilSet_i} u)$

8. **PropValue**($\{(x_i = v_i^*)\}$)

AsymDPOP with TSPS for each agent a_i

9. **else**
 10. **PropUtil()**
When received $\text{VALUE}(PA^*)$ from $P(a_i)$:
 11. **PropValue**(PA^*)
Function PropUtil():
12. compute $utilSet_{i \rightarrow p}$ according to Eq.(5.11)
13. send UTILSET($utilSet_{i \rightarrow p}$) to $P(a_i)$
Function PropValue(PA^*): //计算自身的赋值并向下传播
14. **foreach** $a_c \in C(a_i)$ **do**
15. $PA_i^c \leftarrow PA_{[Sep(a_c) \cup \{a_c\} \cup ID(a_c)]}^*$
16. **if** $EV(a_i, a_c) \neq \emptyset$ **then**
17. compute V_i^{c*} for $EV(a_i, a_c)$ according to Eq.(5.12)
18. $PA_i^c \leftarrow PA_i^c \cup \{(x_j = V_{i[x_j]}^{c*}) \mid \forall x_j \in EV(a_i, a_c)\}$
19. send VALUE(PA_i^c) to a_c
Function Join(U, U_F):
20. **foreach** $u_f \in U_F$ **do**
21. **if** $\exists u \in U, s.t., u_f.dim \subset u.dim$ **then**
22. $u \leftarrow u \otimes u_f$
23. **return** U
Function Elim(U, EV): //执行非本地消元操作
24. $U_{EV} \leftarrow \{u \mid \forall u \in U, u.dim \cap EV \neq \emptyset\}$
25. **return** $U \setminus U_{EV} \cup \{\min_{EV}(\otimes_{u \in U_{EV}} u)\}$
Function PartF(U_F, k_p): //对自身的效用表进行划分
26. order U_F according to $AP(a_i)$'s levels
27. $U \leftarrow \emptyset, u \leftarrow null$
28. **foreach** $u_f \in U_F$ **do**
29. **if** $|u.dim| \geq k_p$ **then**
30. $U \leftarrow U \cup \{u\}, u \leftarrow u_f$
31. $u \leftarrow u \otimes u_f$
32. **if** $u \neq \emptyset$ **then**
33. $U \leftarrow U \cup \{u\}$
34. **return** U
-

算法的值传播阶段与 AsymDPOP 中的大致相同。由于效用表集合传播阶段使用 TSPS，收到效用表集合 $utilSet_{c \rightarrow i}$ 包含多个效用表。因此， a_i 必须在为 EV_i^c 选择最优赋值之前联合 $utilSet_{c \rightarrow i}$ 中的所有效用表（17 行），即：

$$V_i^{c*} = \operatorname{argmin}_{EV_i^c} \left(\left(\bigotimes_{a_j \in B_i^c} f_{ij} \right) \otimes \left(\bigotimes_{u \in utilSet_{c \rightarrow i}} u \right) \right) (PA^*) \quad (5.12)$$

以图 5.2 中所示伪树为例，假设 $k_p = 3$ ，智能体 a_n 将会将效用表集合 $util_{n \rightarrow n-1} = \{f_{n(n-1)} \otimes f_{n1}\}$ 传递给 a_{n-1} 。由于 a_{n-1} 不进行消元，所以不需要执行联合操作，直接将效用表集合 $util_{n-1 \rightarrow n-2} = \{f_{(n-1)(n-2)} \otimes f_{(n-1)1}, f_{(n-1)n} \otimes util_{n \rightarrow n-1}\}$ 传递给 a_{n-2} 。可以看出在本例中 TSPS 只需要 $O(n * d_{\max}^3)$ 的内存空间消耗，这比采用 GNLE 的内存空间消耗要小得多。因此，有以下定理：

定理 5.2： 在带有 TSPS 的 AsymDPOP 中，智能体 a_i 传播的每个效用表不大于

$$d^{\max(\min(|AP(a_i)|, k_p), \max_{a_c \in C(a_i)} (|Sep(a_c) \cup (ID_i \cap (ID_c \cup \{a_c\}))|))}$$

证明：根据公式 (5.9) 可知， a_i 传播的效用表集合包含如下两部分：

- ① 本地效用表集合 $U_{i \rightarrow p}^i$ ；
- ② 处理收到孩子节点的效用表集合 $U_{i \rightarrow p}^c, \forall a_c \in C(a_i)$ 。

对于本地效用表集合而言，由于在 **PartF** 函数中，本地约束代价函数会根据参数 k_p 被划分为多个效用表，所以最大的本地效用表的大小为：

$$d^{\min(|AP(a_i)|, k_p)}$$

对于处理收到孩子节点 a_c 的效用表集合而言，根据定理 5.1，从 a_c 处收到的效用表维度为 $Sep(a_c) \cup \{a_c\} \cup ID_c$ 的子集。在公式 (5.9) 中，TSPS 去除不必要的联合操作，因此对 EV_i^c 消元后的维度属于 $(Sep(a_c) \cup \{a_c\} \cup ID_c) \setminus EV_i^c$ 。又由于 $Sep(a_c) \cap EV_i^c = \emptyset$ ，因此有 $(Sep(a_c) \cup \{a_c\} \cup ID_c) \setminus EV_i^c = (\{a_c\} \cup ID_c) \setminus EV_i^c \cup Sep(a_c)$ 。进一步，基于引理 5.1，可以推导出：

$$\begin{aligned} ID_i &= \left(C(a_i) \cup \left(\bigcup_{a_c \in C(a_i)} ID_c \right) \right) \setminus EV_i \\ &= \left(\bigcup_{a_c \in C(a_i)} (\{a_c\} \cup ID_c) \right) \setminus EV_i \\ &= \left(\bigcup_{a_c \in C(a_i)} (\{a_c\} \cup ID_c) \right) \setminus \left(\bigcup_{a_c \in C(a_i)} EV_i^c \right) \\ &= \bigcup_{a_c \in C(a_i)} ((\{a_c\} \cup ID_c) \setminus EV_i^c) \end{aligned}$$

又由于 $((\{a_c\} \cup ID_c) \setminus EV_i^c) \cap ((\{a_{c'}\} \cup ID_{c'}) \setminus EV_i^{c'}) = \emptyset, \forall x_c, x_{c'} \in C(x_i), c \neq c'$ ，可推断出：

$$\begin{aligned}
 (\{a_c\} \cup ID_c) \setminus EV_i^c &= ((\{a_c\} \cup ID_c) \setminus EV_i^c) \cap ID_i \\
 &= ((\{a_c\} \cup ID_c) \cap ID_i) \setminus (EV_i^c \cap ID_i) \\
 &= ((\{a_c\} \cup ID_c) \cap ID_i) \setminus \emptyset \\
 &= (\{a_c\} \cup ID_c) \cap (ID_i \setminus EV_i^c) \\
 &= (\{a_c\} \cup ID_c) \cap ID_i
 \end{aligned}$$

据此，有 $Sep(a_c) \cup ((\{a_c\} \cup ID_c) \setminus EV_i^c) = Sep(a_c) \cup (ID_i \cap (\{a_c\} \cup ID_c))$ 。因此，处理收到孩子节点 $a_c \in C(a_i)$ 效用表集合的内存消耗为：

$$d^{\max_{a_c \in C(a_i)} (|Sep(a_c) \cup (ID_i \cap (ID_c \cup \{a_c\}))|)}$$

因此，该定理得证。

5.4.2 小批次消元机制

在 TSPS 中，一个联合后大的效用表被划分为一组小的效用表。因此，在执行变量消元时，可通过分治策略来降低消元过程所需的计算量。以图 5.2 为例，在带 TSPS 的 AsymDPOP 算法中， a_1 根据公式 (5.9) 对变量 x_2, \dots, x_n 执行消元，即：

$$\min_{\{x_2, \dots, x_n\}} (f_{12} \otimes f_{21} \otimes \dots \otimes f_{1n} \otimes f_{n1} \otimes f_{n(n-1)} \otimes f_{(n-1)n})$$

上述操作需要 $O(d_{max}^n)$ 的操作数。事实上，可通过在效用表之间重新排列最小化操作来降低计算复杂度，即：

$$\min_{x_2} (f_{12} + f_{21} + \dots + \min_{x_n} (f_{1n} + f_{n1} + f_{n(n-1)} + f_{(n-1)n}))$$

从 x_n 到 x_2 递归消元来实现上述联合消元的问题，其计算复杂度降低为 $O(n * d_{max}^3)$ 。因此，利用效用表之间的消元独立性可减少不必要的约束表联合，从而降低消元过程的计算复杂性。

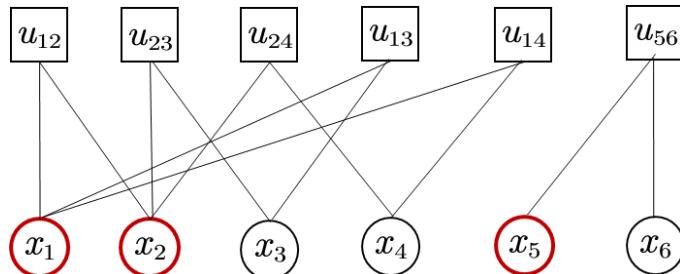


图 5.4 效用表集合

Fig. 5.4 A set of utility tables

由于消元操作可能会隐式地将多个效用表联合为一个大的、不可分割的效用表。因此，将消元操作完全分布到每个变量中会导致巨大的内存消耗。虽然该问题可以通过精心排列消元操作来缓解，但在实际中可能无法找到最优的消元序列。以图 5.4 中给出的因子图为例，其中正方形节点表示效用表，圆形节点表示变量，红色圆圈表示需要消元的变量。显然，无论如何安排消元顺序，消去 x_1 或 x_2 时，

必然会出现一个 3 元的效用表。相反，如果联合消除 x_1 和 x_2 ，最大效用表的维数为 2。因此，引入参数 k_e 来解决上述问题，该参数用于指定在一次消元过程中消除的最小变量数（即消元变量批次的大小），并将这一权衡机制称为小批量消元方案（MBES，Mini-Batch Elimination Scheme）。

算法 5.4 MBES 的伪代码

Algorithm 5.4 Sketch of MEBS

MEBS

输入：效用表集合，待消元变量集合以及算法参数 k_e

输出：多个元素个数不超过 k_e 的变量集合

Function Elim_MBES(U, EK, k_e):

1. $GEV \leftarrow \text{GroupEV}(U, EV)$
2. $EVSet \leftarrow \bigcup_{EV \in GEV} \text{PartEV}(EV, k_e)$
3. **foreach** $EV \in EVSet$ **do**
4. $U_{EV} \leftarrow \{u \mid \forall u \in U, u.dim \cap EV \neq \emptyset\}$
5. $U \leftarrow (U \setminus U_{EV}) \cup \{\min_{EV}(\otimes_{u \in U_{EV}} u)\}$
6. **return** U

Function GroupEV(U, EV): //依据独立性，对变量进行分组

7. $G_{EV} \leftarrow \{u.dim \cap EV \mid \forall u \in U\}, RM_{EV} \leftarrow \emptyset$
8. **while** $\exists EV', EV'' \in G_{EV}, EV', EV'' \notin RM_{EV}, s.t. EV' \cap EV'' \neq \emptyset$ **do**
9. $EV' \leftarrow EV' \cup EV'', RM_{EV} \leftarrow RM_{EV} \cup EV''$
10. **return** $G_{EV} \setminus RM_{EV}$

Function PartEV(EV, k_e): //依据参数 k_e ，对组内变量进一步进行划分

11. $EVSet \leftarrow \emptyset, EV' \leftarrow \emptyset$
 12. **foreach** $ev \in EV$ **do**
 13. **if** $|EV'| \geq k_e$ **then**
 14. $EVSet \leftarrow EVSet \cup \{EV'\}, EV' \leftarrow \{ev\}$
 15. **else**
 16. $EV' \leftarrow EV' \cup \{ev\}$
 17. **if** $EV' \neq \emptyset$ **then**
 18. random select an element $EV'' \in EVSet$
 19. $EV'' \leftarrow EV'' \cup EV'$
 20. **return** $EVSet$
-

为将小批量消元方案 (MBES) 部署到带 TSPS 的 AsymDPOP 中, 仅需将算法 5.3 中第 8 行的 **Elim** 函数替换为 **Elim_MBES** 函数即可。算法 5.4 给出函数 **Elim_MBES** 的伪代码, 该函数主要由如下三部分组成。第一部分用于将消元变量划分为不相交变量组。即, 每个组中的消元变量至少共享一个效用表, 该部分由函数 **GroupEV** 实现 (1 行, 7-10 行)。这部分是极其必要的, 因为联合消除互相独立的一批变量等同于单独消除这些变量。以图 5.3 为例, 设置 $k_e = 2$ 以及令 x_2, x_5 成为一批消元变量, 尽管 x_2, x_5 被联合消元, 但仍然会产生一个关于 x_1, x_3, x_4, x_5 的 4 元效用表。第二部分根据 k_e 将每个组中的消元变量划分为若干个变量集合 $EVSet$, 该部分由函数 **PartEV** 实现 (2 行, 11-20 行)。最后一部分, 通过对 $EVSet$ 进行遍历来实现对效用表的消元操作 (3-5 行)。具体而言, 对 $EVSet$ 中的每个消元变量集合 EV , 对与 EV 相关的效用表执行消元操作, 并将这些效用表替换为消元后的效用表。当消元变量集被遍历完时, 算法终止。

5.5 两种内存有界的 ADCOP 完备算法

虽然 TSPS 可以在一定程度上减少 AsymDPOP 算法的内存消耗, 但带 TSPS 的 AsymDPOP 算法中最大的 UTILSET 消息大小仍是指数级的 (如定理 5.3 所示), 即: 在内存预算有限的条件下, AsymDPOP 算法和带 TSPS 的 AsymDPOP 算法都无法求解较大规模问题。在指定的内存预算下, 内存有界推理算法 (例如 RMB-DPOP) 仍然可实现问题求解。因此, 将 RMB-DPOP 算法与 AsymDPOP 算法结合, 5.4.1 节提出 RMB-AsymDPOP 算法用于在内存有界的环境下求解 ADCOP。此外, 为提高 RMB-AsymDPOP 算法的可扩展性, 5.4.2 节提出带有 TSPS 的 RMB-AsymDPOP 算法。

5.5.1 内存有界的 AsymDPOP

与 RMB-DPOP 算法类似, RMB-AsymDPOP 算法可利用循环去除思想来实现内存有界的推理, 并结合 RMB-DPOP 算法中提出的三种改进机制来减少冗余推断。这三种改进机制包括: 分布式枚举机制 (DEM, Distributed Enumeration Mechanism) 通过因子化 CC 节点的实例来执行异步内存有界推理; 迭代选择机制 (ISM, Iterative Selection Mechanism) 通过考虑候选 CC 节点的有效性和在伪树中的相对位置来确定 CC 节点; 缓存机制 (CACHE, Caching Mechanism) 通过考虑历史推理结果与当前 CC 节点实例的兼容性, 利用历史推理结果来减少不必要的推理。其中 ISM 和 CACHE 无需调整可直接部署到 RMB-AsymDPOP 算法中。当调整 ISM 以适用于 RMB-AsymDPOP 算法时, 需要特别注意的是在 AsymDPOP 算法中由智能体 a_i 传播的效用表的维数为 $Sep(a_i) \cup \{a_i\} \cup ID(a_i)$, 而不是 DPOP 算法中的 $Sep(a_i)$ 。由于 GNLE, 在 RMB-AsymDPOP 算法中适配 DEM 和计算有界的效用表也要进行调

整。在 DEM 中，簇内节点负责枚举 CC 节点的赋值。具体而言，在 a_i 处枚举的 CC 节点 CC_i 可以通过如下公式计算：

$$CC_i = \begin{cases} CClust_i \cap \left(\bigcup_{a_c \in C(a_i)} Sep(a_c) \right) & a_i \text{ is a CR node} \\ CClust_i \cap \{a_i\} & \text{otherwise} \end{cases} \quad (5.13)$$

其中， $CClust_i$ 是包含 a_i 的簇对应的 CC 节点。与 RMB-DPOP 中的本地消元不同，GNLE 中的智能体 a_i 需要对 $EV(a_i)$ 中的所有变量进行消元，且其传播效用表的维度为 $Sep(a_i) \cup \{a_i\} \cup ID(a_i)$ 。因此，一种简单的 CC_i 计算方法如下：

$$CC_i = \begin{cases} CClust_i \cap \left(\bigcup_{a_c \in C(a_i)} (Sep(a_c) \cup \{a_c\} \cup ID_i) \right) & a_i \text{ is a CR node} \\ CClust_i \cap EV_i & \text{otherwise} \end{cases} \quad (5.14)$$

若按照公式 (5.14) 计算 CC 节点，则无法考虑智能体在消元时候的分支独立性，即 $EV_i^c \cap EV_i^{c'} = \emptyset, \forall a_c, a_{c'} \in C(a_i), c \neq c'$ 。也就是说，按照公式 (5.14) 计算获得的 a_i 负责枚举的 CC 节点仍会产生冗余的推理。以图 5.1 中的 a_2 为例，给定 $k_p = 0$ ，所有智能体在一个簇中且该簇 CC 节点为 $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ 。根据公式 (5.14)，可知 $CC_2 = \{a_3, a_5, a_6\}$ 。因此，即便 a_3 传播效用表中并不包含 $\{a_5, a_6\}$ 中的维度（即 $a_j \notin Sep(a_3) \cup \{a_3\} \cup ID_3, \forall a_j \in \{a_5, a_6\}$ ）以及 a_4 传播的效用表中并不包含 a_3 的维度（即 $a_3 \notin Sep(a_4) \cup \{a_4\} \cup ID_4$ ）， a_3 和 a_4 还是会收到 $\{a_3, a_5, a_6\}$ 的所有实例。因此，本节提出独立枚举不同分支中的 CC 节点，将此枚举方案称为分支独立的分布式枚举机制 (BI-DEM, Branch-Independent Distributed Enumeration Mechanism)。在 BI-DEM 中，智能体 a_i 仅为其孩子节点 a_c 枚举属于该分支的 CC 节点（即 CC_i^c ）。这里， CC_i^c 可由公式 (5.15) 计算获得。

$$CC_i^c = \begin{cases} CClust_i \cap (Sep(a_c) \cup \{a_i\} \cup ID(a_c)) & a_i \text{ is a CR node} \\ CClust_i \cap EV(a_i, a_c) & \text{otherwise} \end{cases} \quad (5.15)$$

根据公式 (5.14) 和公式 (5.15) 可知 $CC_i = \bigcup_{a_c \in C_i^{in}} CC_i^c$ 。这里， C_i^{in} 为 a_i 的簇内孩子节点。因此， a_3 和 a_4 分别只会收到 $CC_2^3 = \{a_3\}$ 和 $CC_2^4 = \{a_5, a_6\}$ 的实例。接下来，我们将介绍在 RMB-AsymDPOP 算法中计算有界效用表的过程。

当智能体 a_i 从其父节点 $P(a_i)$ 接收到 Ins_i 且收到所有子节点的推理结果后，可根据公式 (5.16) 计算有界效用表 $util_{i \rightarrow p}$ 。

$$util_{i \rightarrow p} = \left(\bigotimes_{a_j \in AP(a_i)} f_{ij}(Ins_i) \right) \otimes \left(\bigotimes_{a_c \in C(a_i)} util_{i \rightarrow p}^c (Ins_i \cup Ins_i^c) \right) \quad (5.16)$$

其中， Ins_i^c 是 CC_i^c 的赋值实例。 $util_{i \rightarrow p}^c$ 是处理效用表 $util_{c \rightarrow i}$ 后获得的结果，即：

$$util_{i \rightarrow p}^c = \begin{cases} \textbf{Update}\left(util_{i \rightarrow p}^c, \min_{EV_i^c \setminus CC_i^c} \left(\left(\bigotimes_{a_j \in B_i^c} f_{ij} \right) \otimes util_{c \rightarrow i} \right) \right) & a_c \in CC_i^{in} \\ \min_{EV_i^c} \left(\left(\bigotimes_{a_j \in B_i^c} f_{ij} \right) \otimes util_{c \rightarrow i} \right) & \text{otherwise} \end{cases} \quad (5.17)$$

接下来，我们将介绍 RMB-AsymDPOP 算法的具体运行过程。

图 5.5 和算法 5.5 分别给出 RMB-AsymDPOP 算法的流程图和对应的伪代码。这里，省略对值传播阶段的描述，因为该阶段与 MB-DPOP 算法中的值传播阶段类似。在标记阶段完成后，智能体 a_i 获得簇中的 CC 节点。随后， a_i 标记自己的节点类型，并根据公式 (5.15) 为簇内孩子节点计算 CC_i^c (4 行, 42-51 行)。最后，CR 节点将 CC 实例发送给簇内孩子节点 (5-6 行, 36-41 行)。

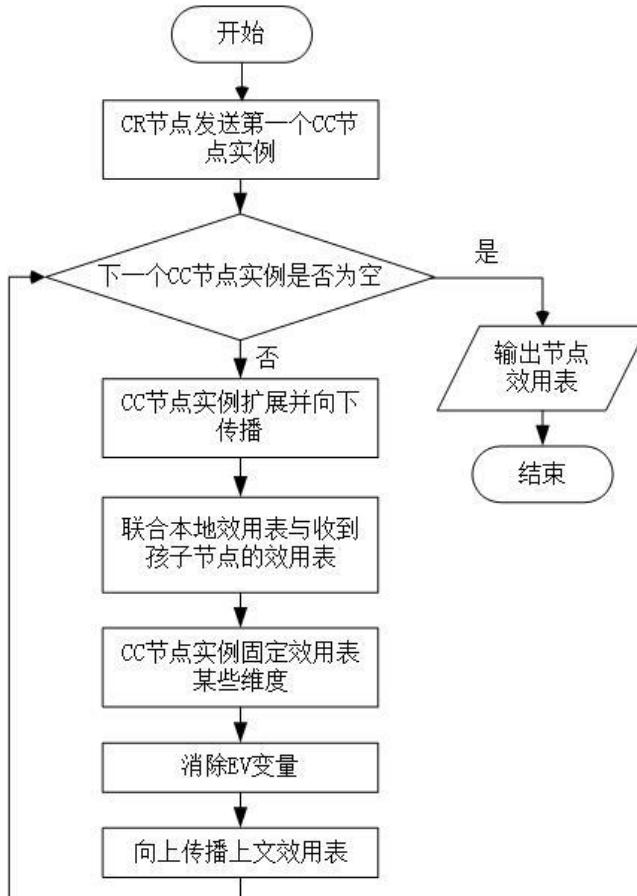


图 5.5 RMB-AsymDPOP 的流程图

Fig. 5.5 Flow chart of RMB-AsymDPOP

当 a_i 收到 CC 实例 Ins_i 时，如果它是 CL 节点，则通过公式 (5.16) 计算效用表 $util_{i \rightarrow p}$ 发送给父节点 (15-16 行)。否则，它用第一个 CC 赋值 Ins_i^c 来扩展 Ins_i ，并发送给簇内孩子节点 (18 行, 36-41 行)。一旦从簇内孩子节点 a_c 接收到

BOUNDEDUTIL 消息后, a_i 根据公式 (5.17) 更新 $util_{i \rightarrow p}^c$, 随后为 a_c 查找下一个需要探索的 CC_i^c 赋值。如果下一个赋值存在, a_i 将新的实例发送给 a_c (20-22 行); 否则, 将 a_c 标记为在 Ins_i 下完成内存有界效用传播的节点 (20 行)。当在 Ins_i 下, 所有簇内节点完成内存有界效用传播时, a_i 通过 BOUNDEDUTIL 消息将内存有界效用表向上传播。

算法 5.5 RMB-AsymDPOP 的伪代码

Algorithm 5.5 Sketch of RMB-AsymDPOP

RMB-AsymDPOP for each agent a_i

输入: 智能体 a_i 的值域和约束函数集合

输出: 智能体 a_i 最终的赋值

When Initialization:

1. start **Labeling phase**

When Labeling phase finished: //计算 CC 节点, 并开启效用传播阶段

2. initialize $done_i$ and $util_{i \rightarrow p}^c, \forall a_c \in C(a_i)$

3. $C_i^{in} \leftarrow \{a_c | \forall a_c \in C(a_i), |Sep(a_c) \cup \{a_c\} \cup ID_c| > k_{mb}\}$

4. **LabelItself()** and compute CC_i^c by Eq. (5.15)

5. if $TYPE(a_i) = CR$ then

6. **PropInstantiation()**

7. else if $TYPE(a_i) = NORMAL \wedge a_i$ is a leaf then

8. **PropUtil()**

When received NORMAL_UTIL($util_c$) from $a_c \in C(a_i)$: //处理接收到的效用表

9. $util_{c \rightarrow i} \leftarrow util_c$ and compute $util_{i \rightarrow p}^c$ by Eq. (5.17)

10. if a_i has received all NORMALUTIL from $C(a_i)$:

11. if a_i is the root then

12. start **Value propagation phase**

13. else if $TYPE(x_i) = NORMAL$ then

14. **PropUtil()**

When received INSTANTIATION(Ins_i) from $P(a_i)$: //发送 CC 节点对应的实例

15. if $TYPE(a_i) = CL$ then

16. **PropUtil()**

17. else

18. **PropInstantiation()**

RMB-AsymDPOP for each agent a_i

When received **BOUNDED_UTIL**($util_c$) from $a_c \in C_i^{in}$: //处理接收到的上文效用表

```

19.    $util_{c \rightarrow i} \leftarrow util_c$  and update  $util_{i \rightarrow p}^c$  by Eq. (5.17)
20.   if  $Ins_i^c.next() \neq null$  then
21.      $Ins_i^c \leftarrow Ins_i^c.next()$ 
22.     send INSTANTIATION( $Ins_i^c \cup Ins_{i[util_{c \rightarrow i}.dims]}$ ) to  $a_c$ 
23.   else
24.      $done_i \leftarrow done_i \cup \{a_c\}$ 
25.     if  $|done_i| = |C_i^{in}|$  then
26.       initialize  $done_i$  and  $util_{i \rightarrow p}^c, \forall a_c \in C_i^{in}$ 
27.       if  $a_i$  is the root then
28.         start Value propagation phase
29.       else
30.         PropUtil()

```

Function PropUtil():

```

31.   compute  $util_{i \rightarrow p}^c$  by Eq. (5.16)
32.   if  $TYPE(a_i) \in \{NORMAL, CR\}$  then
33.     send NORMALUTIL( $util_{i \rightarrow p}^c$ ) to  $P(a_i)$ 
34.   else
35.     send BOUNDUTIL( $util_{i \rightarrow p}^c$ ) to  $P(a_i)$ 

```

Function PropInstantiation(): //CC 节点赋值的计算

```

36.   Foreach  $a_c \in C_i^{in}$  do
37.      $Ins_i^c \leftarrow$  the first instantiation of  $CC_i^c$ 
38.     if  $TYPE(a_i) = CR$  then
39.       send INSTANTIATION( $Ins_i^c$ ) to  $a_c$ 
40.     else
41.       send INSTANTIATION( $Ins_i^c \cup Ins_{i[util_{c \rightarrow i}.dims]}$ ) to  $a_c$ 

```

Function LabelItself(): //确定节点类型

```

42.   if  $|util_{i \rightarrow p}.dims| \leq k_{mb}$  then
43.     if  $C_i^{in} = \emptyset$  then
44.        $TYPE(a_i) \leftarrow NORMAL$ 
45.     else
46.        $TYPE(a_i) \leftarrow CR$ 
47.   else

```

RMB-AsymDPOP for each agent a_i

```

48.      if  $C_i^{in} = \emptyset$  then
49.           $TYPE(a_i) \leftarrow CL$ 
50.      Else
51.           $TYPE(a_i) \leftarrow CN$ 

```

5.5.2 带桶集合作传播机制的内存有界 AsymDPOP

尽管 RMB-AsymDPOP 算法可以在内存有界的情况下实现对 ADCOP 求解，但由于在求解过程中会产生巨大的通信开销，仍然无法求解较大规模的问题。本章的实验结果显示，它只能解决智能体数量小于 20 的问题。据此，提出将 RMB-AsymDPOP 算法与 TSPS 相结合，以提高算法的可扩展性。由于 MBES 会增加算法的内存消耗，从而使得算法的内存消耗超过内存预算，因此没有将 MBES 与 RMB-AsymDPOP 算法结合。

由于带 TSPS 的 RMB-AsymDPOP 算法是 RMB-AsymDPOP 算法的扩展，因此可直接借鉴 RMB-AsymDPOP 算法的内存有界推理思想以及三种改进机制。此外，每个智能体在 AsymDPOP-TSPS 中传播的是效用表集合，因此在将 TSPS 部署到 RMB-AsymDPOP 时有如下两个问题待解决：1) 传播效用表集合的维度是在效用表集合产生的时候才确定，在标记阶段之前这些维度是未知的，但这些维度对于在标记阶段确定簇和 CC 节点是必不可少的。因此，本节提出预处理阶段用以传播带 TSPS 的 AsymDPOP 算法中效用表的维度。2) 若直接将内存有界推理应用于 RMB-AsymDPOP-TSPS 时，仍然存在冗余推理。具体而言，在内存有界的效用表集合作传播过程中，每个簇内节点通过 CC 节点的赋值来降低其传播效用表的维度，然后将有界的效用表集合迭代地向上传播。实际上，对于初始维度不超过内存限制 k_{mb} 的效用表而言，它们不需要 CC 节点的赋值来限制其维度。因此，本节提出两阶段的效用表集合作传播方案，其包括常规效用集合作传播阶段和有界效用集合作传播阶段。下面，我们将介绍预处理阶段和两阶段效用集合作传播阶段。

图 5.6 和算法 5.6 分别给出预处理阶段的流程图和对应的伪代码。该阶段开始于叶子节点通过 DIMSET 消息将其本地效用表维度发送给父节点（2-3 行）。其中传播的效用表维度是通过函数 **PartD** 获得的，其目的是实现 TSPS（1 行，14-22 行）。一旦从子节点 $a_c \in C(a_i)$ 收到 DIMSET 消息， a_i 存储收到的每个效用表维度，并将与 a_c 分支相关的本地私有函数添加到收到的效用表中（4-7 行）。随后， a_i 调用函数 **ElimD** 来模拟效用消元过程，以此获得 a_c 传播效用的维度 $utilSet_{i \rightarrow p}^c$ （8 行，23-28 行）。当收到所有子节点的 DIMSET 消息后， a_i 将所有子节点消元后的效用表维度与本地效用表维度进行合并。若 a_i 不是根节点，则 a_i 将联合后的效用表维度

发送给其父节点（11 行）。否则，预处理阶段结束。

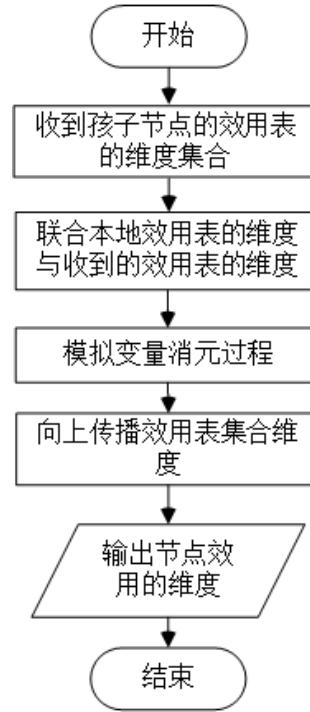


图 5.6 带 TSPS 的 RMB-AsymDPOP 预处理阶段流程图

Fig. 5.6 Flow chart of Preprocessing phase in RMB-AsymDPOP with TSPS

算法 5.6 带 TSPS 的 RMB-AsymDPOP-TSPS 预处理阶段伪代码

Algorithm 5.6 Sketch of Preprocessing phase in RMB-AsymDPOP with TSPS

Preprocessing phase in RMB-AsymDPOP with TSPS for each agent a_i

输入：智能体 a_i 的约束函数集合

输出：智能体 a_i 的效用表维度集合

When Initialization:

1. $utilSet_{i \rightarrow p}^i \leftarrow \text{PartD}(\cup_{a_j \in AP(a_i)} \{f_{ij}\}, k_p)$
2. **if** a_i is a leaf **then**
3. send DIMSET($utilSet_{i \rightarrow p}^i$) to $P(a_i)$

When received $DIMSET(utilSet_c)$ from $a_c \in C(a_i)$: //处理收到的效用表维度

4. $utilSet_{c \rightarrow i} \leftarrow utilSet_c$
 5. **foreach** $u_f \in F_i^c$ **do**
 6. **if** $\exists u \in utilSet_{c \rightarrow i}, s.t., u_f.dims \subset u.dims$ **then**
 7. $u.joint \leftarrow u.joint \cup \{u_f\}$
-

Preprocessing phase in RMB-AsymDPOP with TSPS for each agent a_i

8. $utilSet_{i \rightarrow p}^c \leftarrow \textbf{ElimD}(utilSet_{c \rightarrow i}, EV_i^c)$
9. **if** a_i has received all DIMSET from $C(a_i)$ **then**
10. **if** a_i is not the root **then**
11. send DIMSET($\cup_{a_j \in \{a_i\} \cup C(a_i)} utilSet_{i \rightarrow p}^j$) to $P(a_i)$
12. **else**
13. end **Perprocessing phase**

Function PartD(U_F, k_p): //划分效用表

14. $U \leftarrow \emptyset, u \leftarrow emptybucket$
15. order U_F according to $AP(a_i)$'s levels
16. **foreach** $u_f \in U_F$ **do**
17. **if** $|u.dims| \geq k_p$ **then**
18. $U \leftarrow U \cup \{u\}, u \leftarrow u_f.dims$
19. $u.dims \leftarrow u.dims \cup u_f.dims$
20. **if** $|u.dims| > 0$ **then**
21. $U \leftarrow U \cup \{u\}$
22. **return** U

Function ElimD(U, EV): //执行非本地消元

23. $GEV_i^c \leftarrow \textbf{GroupEV}(U, EV)$
24. **foreach** $EV \in GEV_i^c$ **do**
25. $U_{EV} \leftarrow \{u | \forall u \in U, u.dims \cap EV \neq \emptyset\}$
26. $u_{EV} \leftarrow empty bucket with dimensions that (\cup_{u \in U_{EV}} u.dims) \setminus EV$
27. $U \leftarrow (U \setminus U_{EV}) \cup \{u_{EV}\}$
28. **return** U

Function GroupEV(U, EV): //对效用表进行分组

29. $G_{EV} \leftarrow \{u.dims \cap EV | \forall u \in U\}$
30. **while** $\exists EV', EV'' \in G_{EV}, s.t. EV' \cap EV'' \neq \emptyset$ **do**
31. $EV' \leftarrow EV' \cup EV'', G_{EV} \leftarrow G_{EV} \setminus EV''$
32. **return** U

带 TSPS 的 RMB-AsymDPOP 算法的效用集合作传播阶段流程图和伪代码分别如图 5.7 和算法 5.7 所示。当标记阶段结束时，智能体 a_i 标记它自己的节点类型（4 行， 58-64 行），并为簇内孩子节点计算 CC 节点。之后，叶子节点 a_i 通过 NORMALUTILSET 消息将维度不大于 k_{mb} 的效用表发送给父节点（5-6 行， 52-57 行）。

行)。这里,剩余的效用表需要 CC 节点的赋值来限制其维度,并在有界效用传播阶段进行迭代地传播。当接收到来自 $a_c \in C(a_i)$ 的 NORMALUTILSET 消息时, a_i 合并接收到的效用表并将它们与相应的私有函数进行联合(7-9 行)。然后,调用函数 **ELimNU** 对收到的效用表执行变量消元操作,并将消元结果合并到 $utilSet_{i \rightarrow p}^c$ 中(10 行)。当收到所有孩子节点的 NORMALUTILSET 消息后,若 a_i 是 CR 节点,则开启有界效用集合传播阶段(12-13 行)。否则, a_i 继续将联合效用表传播给父节点(17-18 行)。

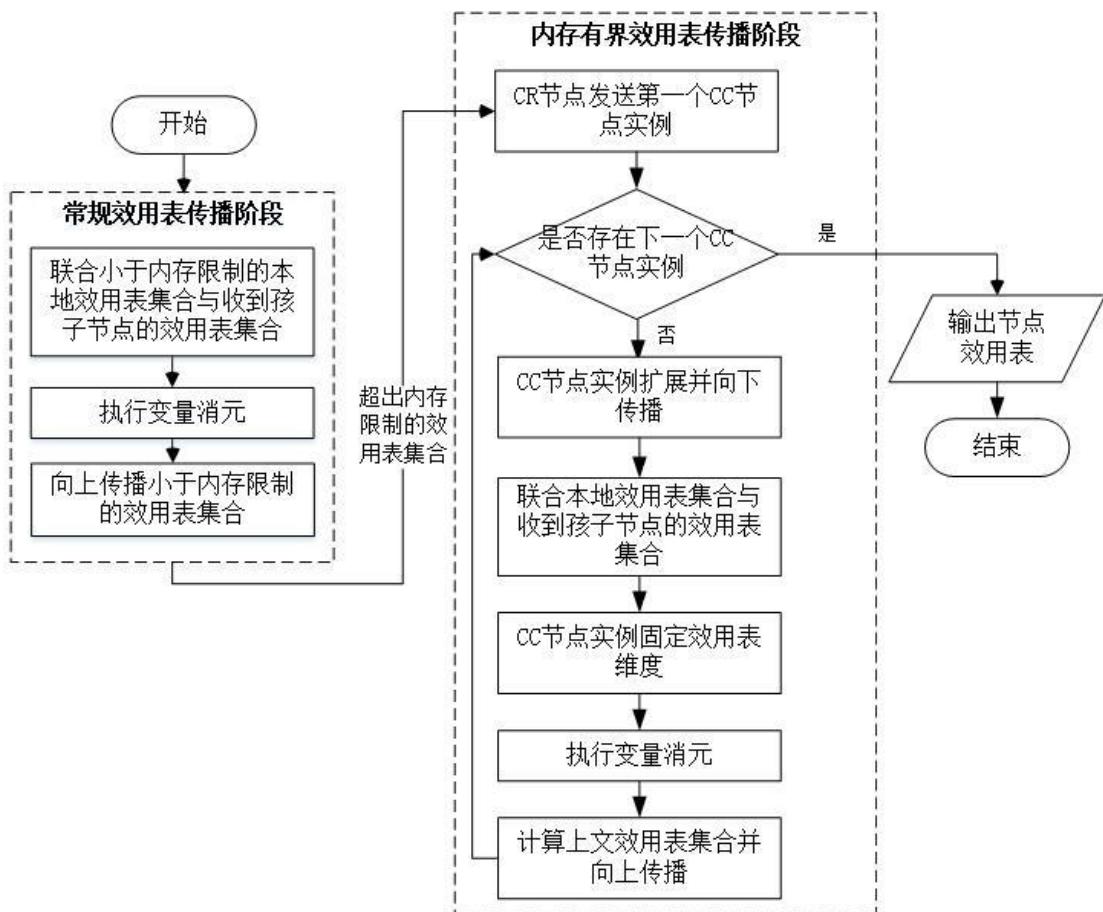


图 5.7 带 TSPS 的 RMB-AsymDPOP 效用集合传播阶段的流程图

Fig. 5.7 Flow chart of utility set propagation phase in RMB-AsymDPOP with TSPS

有界效用传播阶段开始于 CR 节点 a_i 将 CC_i^c 的实例 Ins_i^c 发送给所有簇内孩子节点(12-13 行, 51-57 行)。当节点 a_i 收到来自父节点的实例 Ins_i , 若 a_i 为 CL 节点, 则其调用函数 **ELimBU** 对接收到的效用表执行消元操作, 并将获得的有界效用表通过 BOUNDEDUTILSET 消息发送给父节点(19-22 行, 48-50 行); 否则, a_i 用 CC_i^c 的第一个赋值 Ins_i^c 来扩展 Ins_i , 并将扩展后的实例传播给簇内孩子节点(23-24 行, 51-57 行)。一旦接收到来自 a_c 的有界效用表, a_i 联合相应的私有本地约束代

价函数（25-27 行），调用函数**ELimBU**来对收到的效用表执行变量消元，并用消元结果更新效用表 $utilSet_{i \rightarrow p}^c$ （28 行）。随后， a_i 为 a_c 查找 CC_i^c 的下一个赋值。如果下一个赋值存在， a_i 用该赋值替换 Ins_i^c 并将新实例传播给 a_c （29-31 行）。否则，它将 a_c 标记为完成内存有界效用集合传播的节点（32 行）。当所有簇内孩子节点都已完成时， a_i 向上传播将有界推理结果（33 行，42 行，48-50 行）。

算法 5.7 带 TSPS 的 RMB-AsymDPOP 效用集合传播阶段伪代码

Algorithm 5.7 Sketch of utility set propagation phase in RMB-AsymDPOP with TSPS

Utility set propagation phase in RMB-AsymDPOP with TSPS for each agent a_i

输入：智能体 a_i 的值域、约束函数集合以及预处理阶段得到的效用表维度集合

输出：智能体 a_i 最终的赋值

When Initialization:

1. start **Labeling phase**

When Labeling phase finished :

2. Initialize $done_i$

3. $C_i^{in} \leftarrow \{a_c | \forall a_c \in C(a_i), \exists u \in utilSet_{c \rightarrow i}, s.t., |u.dims| > k_{mb}\}$

4. **LabelItself()** and compute CC_i^c by Eq. (5.15)

5. if a_i is a leaf then

6. **PropNormalUtil()**

When received NORMALUTILSET($utilSet_c$) from $a_c \in C(a_i)$: //处理接收到的效

用表集合

7. $utilSet_{c \rightarrow i} \leftarrow \text{Merge}(utilSet_{c \rightarrow i}, utilSet_c)$

8. **foreach** $u \in utilSet_{c \rightarrow i}, u.data \neq null, u_f \in u.joint$ **do**

9. $u.data \leftarrow u.data \otimes u_f, u.joint \leftarrow u.joint \setminus \{u_f\}$

10. $utilSet_{i \rightarrow p}^c \leftarrow \text{Merge}(utilSet_{i \rightarrow p}^c, \text{ElimNU}(x_c))$

11. if a_i has received all NORMALUTIL from $C(a_i)$ then

12. if $TYPE(a_i) = CR$ then

13. **PropInstantiation()**

14. else

15. if a_i is the root then

16. start **Value propagation phase**

17. else

18. **PropNormalUtil()**

Utility set propagation phase in RMB-AsymDPOP with TSPS for each agent a_i

When received **INSTANTIATION**(Ins_i) from $P(a_i)$: //处理 CC 节点实例

```

19.   if  $TYPE(a_i) = CL$  then
20.     foreach  $a_c \in C(a_i)$  do
21.        $utilSet_{i \rightarrow p}^c \leftarrow \text{ElimBU}(a_c)$ 
22.     PropBoundedUtil()
23.   else
24.     PropInstantiation()

```

When received **BOUNDEDUTILSET**($utilSet_c$) from $a_c \in C_i^{in}$: //处理接收到的有
界效用表集合

```

25.    $utilSet_{c \rightarrow i} \leftarrow \text{Merge}(utilSet_{i \rightarrow p}^c, utilSet_c)$ 
26.   foreach  $u \in utilSet_{c \rightarrow i}, u_f \in u.\text{joint}$  do
27.      $u.data \leftarrow u.data \otimes u_f(Ins_i \cup Ins_i^c)$ 
28.    $utilSet_{i \rightarrow p}^c \leftarrow \text{Update}(utilSet_{i \rightarrow p}^c, \text{ElimBU}(a_c))$ 
29.   if  $Ins_i^c.\text{next}() \neq null$  then
30.      $Ins_i^c \leftarrow Ins_i^c.\text{next}()$ 
31.     send INSTANTIATION( $Ins_i^c \cup Ins_{i[aug\_dim_i^c]}$ ) to  $a_c$ 
32.   else
33.      $done_i \leftarrow done_i \cup \{a_c\}$ 
34.   if  $|done_i| = |C_i^{in}|$  then
35.     initialize  $done_i$  and  $utilSet_{i \rightarrow p}^c, \forall a_c \in C_i^{in}$ 
36.   if  $TYPE(a_i) = CR$  then
37.     if  $a_i$  is the root then
38.       start Value propagation phase
39.     else
40.       PropNormalUtil()
41.   else
42.     PropBoundedUtil()

```

Function PropNormalUtil(): //效用表的传播

```

43.   foreach  $u \in utilSet_{i \rightarrow p}^i, |u.\text{dims}| \leq k_{mb}$  do
44.      $u.data \leftarrow \otimes_{x_j \in u.\text{dims}, i \neq j} f_{ij}$ 
45.      $nU \leftarrow \{u | \forall u \in utilSet_{i \rightarrow p}^j, u.data \neq null, \forall a_j \in \{a_i\} \cup C(a_i)\}$ 
46.      $utilSet_{i \rightarrow p}^j \leftarrow utilSet_{i \rightarrow p}^j \setminus (nU \cap utilSet_{i \rightarrow p}^j), \forall a_j \in \{a_i\} \cup C(a_i)$ 
47.     send NORMALUTILSET( $nU$ ) to  $P(x_i)$ 

```

Utility set propagation phase in RMB-AsymDPOP with TSPS for each agent a_i

Function PropBoundedUtil(): //有界效用表的传播

```

48.   foreach  $u \in utilSet_{i \rightarrow p}^i$  do
49.      $u.data \leftarrow \otimes_{x_j \in u.dims, i \neq j} f_{ij}(Ins_i)$ 
50.   send BOUNDEDUTIL( $\cup_{x_j \in \{x_i\} \cup C(x_i)} utilSet_{i \rightarrow p}^i$ ) to  $P(a_i)$ 

```

Function PropInstantiation(): //CC 节点实例的计算与传播

```

51.   foreach  $a_c \in C_i^{in}$  do
52.      $Ins_i^c \leftarrow$  the first instantiation of  $CC_i^c$ 
53.     if  $TYPE(a_i) = CR$  then
54.       send INSTANCE(Ins_i^c) to  $a_c$ 
55.     else
56.        $aug\_dim_i^c \leftarrow \{u.dims | \forall u \in utilSet_{c \rightarrow i}, |u.dims| > k_{mb}\}$ 
57.       send INSTANCE( $Ins_i^c \cup Ins_{i[aug\_dim_i^c]}$ ) to  $a_c$ 

```

Function LabelItself(): //节点类型的计算

```

58.   if  $|u.dims| \leq k_{mb}, \forall u \in (\cup_{a_j \in \{a_i\} \cup C(a_i)} utilSet_{i \rightarrow p}^j)$  then
59.     if  $C_i^{in} = \emptyset$  then
60.        $TYPE(a_i) \leftarrow NORMAL$ 
61.     else
62.        $TYPE(a_i) \leftarrow CR$ 
63.   else
64.     if  $C_i^{in} = \emptyset$  then
65.        $TYPE(a_i) \leftarrow CL$ 
66.     else
67.        $TYPE(a_i) \leftarrow CN$ 

```

Function ElimNU(a_c): //对收到的效用表集合执行消元操作

```

68.    $U \leftarrow \{u | \forall u \in utilSet_{c \rightarrow i}, u.data \neq null\}$ 
69.   foreach  $EV \in GEV_i^c$  do
70.      $U_{EV} \leftarrow \{u | \forall u \in utilSet_{c \rightarrow i}, u.dims \cap EV \neq \emptyset\}$ 
71.      $U \leftarrow U \setminus U_{EV}, d_{EV} \leftarrow \cup_{u \in U_{EV}} u.dims \setminus EV$ 
72.     if  $|d_{EV}| \leq k_{mb}, u.data \neq null, \forall u \in U_{EV}$  then
73.        $utilSet_{c \rightarrow i} \leftarrow utilSet_{c \rightarrow i} \setminus U_{EV}, GEV_i^c \leftarrow GEV_i^c \setminus \{EV\}$ 
74.        $U \leftarrow U \cup \{\min_{EV}(\otimes_{u \in U_{EV}} u)\}$ 
75.      $utilSet_{c \rightarrow i} \leftarrow utilSet_{c \rightarrow i} \setminus U$ 

```

Utility set propagation phase in RMB-AsymDPOP with TSPS for each agent a_i

```

76.    return  $U$ 

Function ElimBU( $a_c$ ): //对收到的有界效用表集合执行消元操作
77.     $U \leftarrow \{u | INS_i \cup Ins_i^c\} | \forall u \in utilSet_{c \rightarrow i}\}$ 
78.    foreach  $EV \in GEV_i^c$  do
79.         $U_{EV} \leftarrow \{u | \forall u \in U, u.dims \cap EV \neq \emptyset\}$ 
80.         $U \leftarrow U \setminus U_{EV} \cup \{\min_{EV \setminus CC_i^c} (\otimes_{u \in U_{EV}} u)\}$ 
81.    return  $U$ 

```

5.6 私有偏好泄露分析

接下来, 以非对称 MaxDCSPs 问题为例, 分析本章提出算法由于消息传递造成的私有约束函数泄露。由于 VALUE 消息仅包含分割点集 (对于智能体 a_i 而言, 即 $Sep(a_i)$) 和未被消除的那些子孙节点 (对于 a_i 而言, 即 ID_i), 因此该消息不会泄露关于私有约束的任何信息。也就是说, 对于 AsymDPOP 算法而言, 所有私有约束信息泄露都来自于 UTIL 消息; 对于带 TSPS 的 AsymDPOP 算法而言, 所有的私有约束信息泄露都来自于 UTILSET 消息。由于 RMB-AsymDPOP 是以迭代的方式传播与 AsymDPOP 中 UTIL 消息一样的内容, 所以没有对 RMB-AsymDPOP 中 BOUNDEDUTIL 消息泄露的私有约束代价函数进行讨论; 由于带 TSPS 的 RMB-AsymDPOP 算法仅是采用迭代的传播与带 TSPS 的 AsymDPOP 算法中 UTILSET 消息一样的内容, 所以也没有对带 TSPS 的 RMB-AsymDPOP 算法中 BOUNDEDUTILSET 消息泄露的私有约束代价函数进行讨论。

智能体 a_c 到 a_i 的 UTIL/UTILSET 消息 (即 $util_{c \rightarrow i}/utilSet_{c \rightarrow i}$) 会导致私有约束 $f_{ij}, \forall a_j \in B_i^c$ 的泄露。最差情况下, 可能导致一半的约束代价函数的泄露, 即私有约束 $f_{ij}, \forall a_j \in B_i^c$ 能被 a_i 推断出来。具体地: a_c 发送给 a_i 的 UTIL 消息 (即 $util_{c \rightarrow i}$), 基于定理 5.1, 有 $B_i^c \subseteq util_{c \rightarrow i}.dims$ 。因此, a_i 通过将收到的效用表 $util_{c \rightarrow i}$ 对除 x_i, x_j 之外的其他变量进行消除, 获得仅包含 x_i, x_j 两个维度的效用表。基于消元后的效用表中为零的赋值对, 能推断出其下层邻居的私有约束代价函数 $f_{ij}, \forall a_j \in B_i^c$ 中对应位置的私有信息。对于 a_c 到 a_i 的 UTILSET 消息而言 (即 $utilSet_{c \rightarrow i}$), 基于公式 (5.9), 可得 $util_{c \rightarrow i} = \otimes_{u \in utilSet_{c \rightarrow i}} u$ 。又由于 GNLE, B_i^c 都在 $(\{a_i\} \cup Sep(a_i)) \cap AP(a_j)$ 中被消除。因此, 对于私有约束代价函数 $f_{ij}, \forall a_j \in B_i^c$, 必定存在一个效用表 $u \in utilSet_{c \rightarrow i}$ 满足 $f_{ij}.dims \subseteq u.dims$ 。也就是说, a_i 依然能从 $utilSet_{c \rightarrow i}$ 中对应的效用表推断出约束代价函数 $f_{ij}, \forall a_j \in B_i^c$ 的信息。

5.7 实验评估

本节中,首先对比 AsymDPOP 与带 TSPS 与 MBES 的 AsymDPOP,以观察 TSPS 与 MBES 中不同参数对带 TSPS 与 MBES 的 AsymDPOP 的影响;随后,在不同的内存预算下对比 RMB-AsymDPOP 与带 TSPS 的 RMB-AsymDPOP,并研究参数 k_p 对带 TSPS 的 RMB-AsymDPOP 的影响;最后,在随机 DCOP、无尺度网络以及非对称 MaxDCSPs 上对本章提出的算法(包括 AsymDPOP、带 TSPS 与 MBES 的 AsymDPOP、RMB-AsymDPOP 以及带 TSPS 的 RMB-AsymDPOP)与目前最好的 ADCOP 完备算法(包括 SyncABB-1ph、ATWB、AysmPT-FB 以及 PT-ISABB)进行性能比较。此外,还于带 PEAV 的 DPOP 算法(即 PEAV_DPOP)进行对比。

5.7.1 参数调优

实验一: 不同智能体数下, AsymDPOP 与带 TSPS 与 MBES 的 AsymDPOP 的对比以及带 TSPS 与 MBES 的 AsymDPOP 的调参实验。该项实验的目的是从智能体数的角度来增加问题求解的难度, 测试不同配置下带 TSPS 与 MBES 的 AsymDPOP 性能比较以及与 AsymDPOP 的对比, 并为带 TSPS 与 MBES 的 AsymDPOP 选择综合性能最好的一组参数。具体地, 我们选择值域大小均为 3 的随机 ADCOP 问题, 设置代价选取范围为 $[0, 100]$, 密度为 0.25, 并将智能体个数按步长 4, 从 8 变化到 24。

图 5.8 中给出实验一的实验结果。从图中可以看出, 在不限制内存的情况下, AsymDPOP 仅能求解智能体数为 20 以下的问题, 且在所有评估指标上 AsymDPOP 的求解性能也远差于带 TSPS 和 MBES ($k_p = \infty$) 的 AsymDPOP。这是因为 AsymDPOP 的内存消耗不仅与智能体分割点集数成指数相关, 而且还与 ID 节点集合指数级相关(如定理 5.1 所示)。该现象也说明 TSPS 可以降低不必要的联合操作, 进而减少 AsymDPOP 产生的内存消耗。具体地, 当求解 16 个智能体的问题时, AsymDPOP 产生效用表的最大维度为 13, 而带 TSPS 和 MBES ($k_p = \infty$) 的 AsymDPOP 产生效用表的最大维度仅为 7。此外, 随着 k_p 的减小, 计算传播效用表过程中更多的联合操作被避免, 因此效用表维度变得更小。值得注意的是, 较小 k_p 可以极大地提高带 TSPS 和 MBES 的 AsymDPOP 的性能, 但也存在大量的私有代价函数泄露(如图 5.14(d)所示)。对于 MBES, 当 k_e 较小时, 带 TSPS 和 MBES 的 AsymDPOP 所需计算量显著地降低, 但同时算法也产生较大中间效用表结果, 这表明 MBES 会使得算法在计算量与内存消耗之间进行权衡。此外, 当与较大 k_p 的 TSPS 结合时, MBES 的性能会显著退化。这是因为较大的 k_p 会使得算法产生包含更多维度的效用表, 从而在消除变量时会更频繁地遍历效用表。因此, 设置带 TSPS 和 MBES 的 AsymDPOP 的参数为 $k_p = 2$ 和 $k_e = 1$ 。

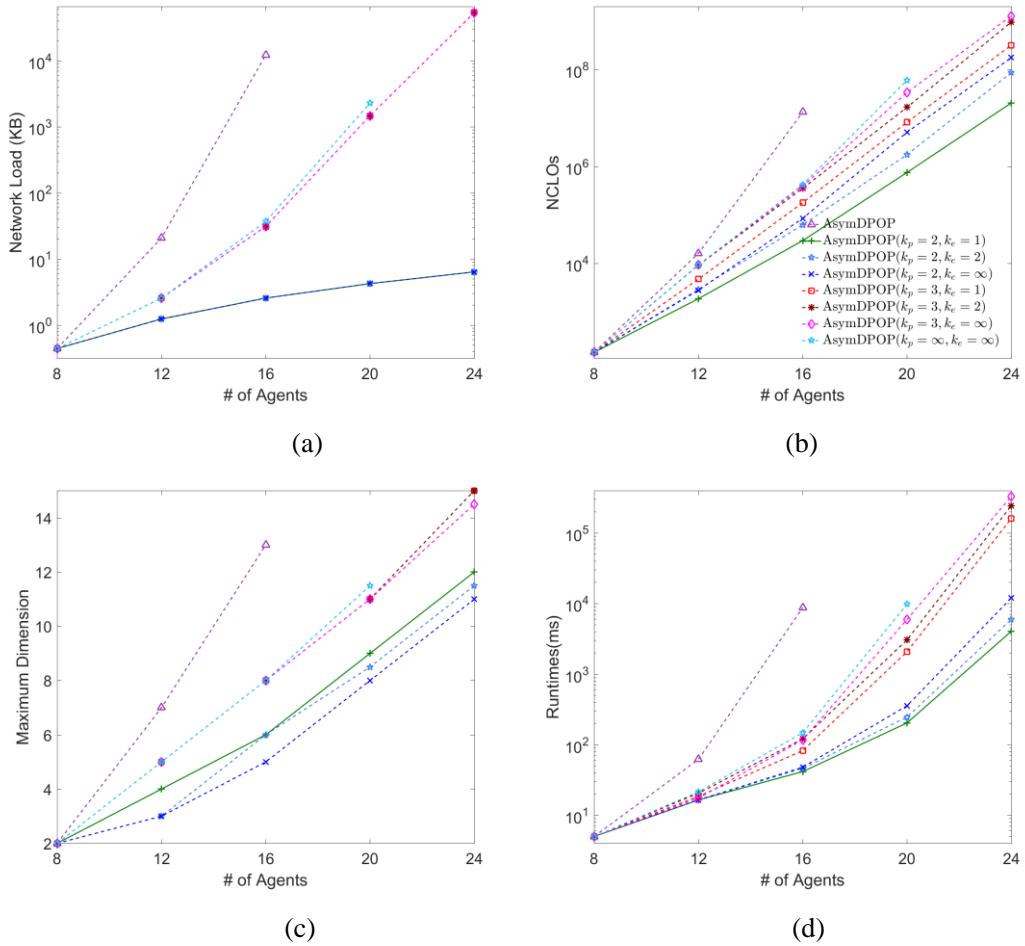


图 5.8 不同智能体数下 AsymDPOP 以及在不同 k_p 和 k_e 下带 TSPS 和 MBES 的 AsymDPOP 性能对比

Fig.5.8 Performance comparison for AsymDPOP and AsymDPOP with TSPS and MBES(k_p, k_e) under different agent numbers

实验二：不同智能体数下，RMB-AsymDPOP 与带 TSPS 的 RMB-AsymDPOP 的对比以及带 TSPS 的 RMB-AsymDPOP 的调参实验，即：RMB-AsymDPOP 的 k_{mb} 和带 TSPS 的 RMB-AsymDPOP 的 k_p 和 k_{mb} 。这里，变化 k_{mb} 为 3~9，变化 k_p 为 2~3。该实验的目的是从智能体数的角度来增加问题求解的难度，测试不同配置带 TSPS 的 RMB-AsymDPOP 性能比较以及其与 RMB-AsymDPOP 的对比，最终为带 TSPS 的 AsymDPOP 选择综合性能最好的一组参数。该实验中，设置智能体个数按步长 4，从 8 变化到 28，其他配置与实验一中的配置相同。

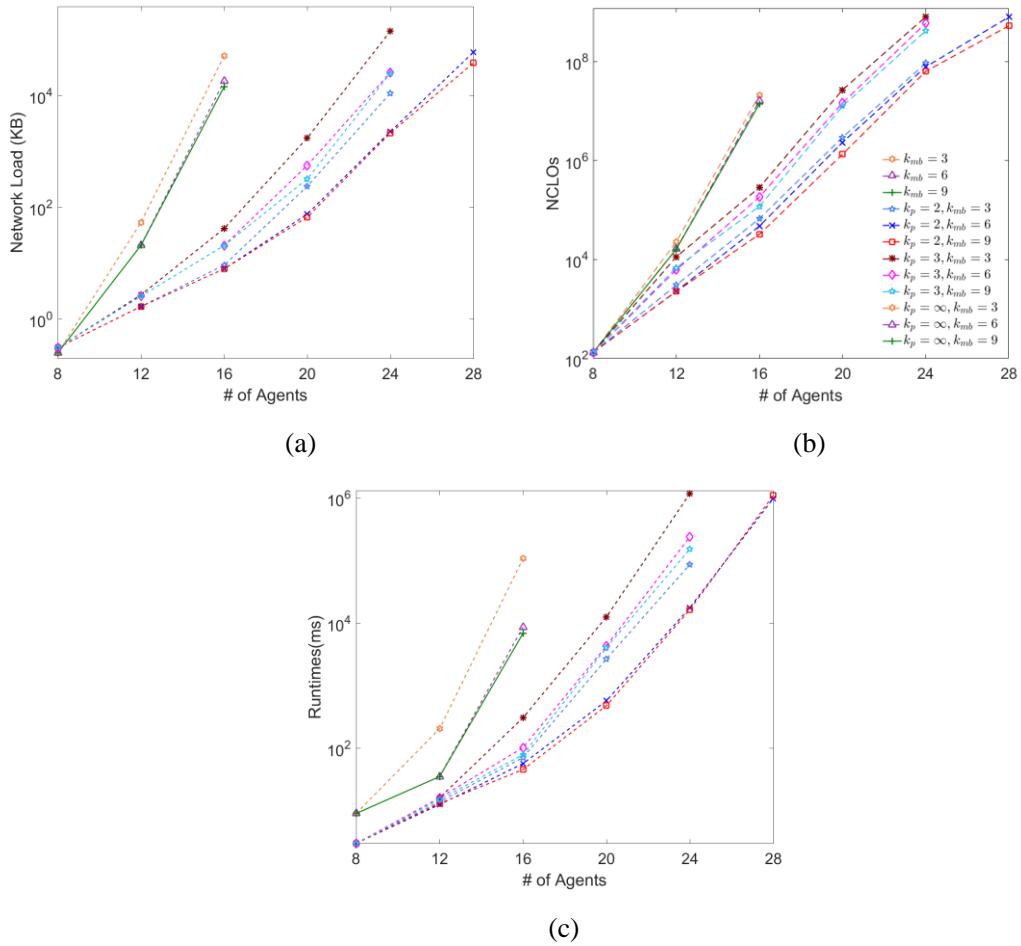


图 5.9 不同智能体数下在不同参数 k_{mb} 下 RMB-AsymDPOP 以及在不同参数 k_{mb} 和 k_p 下带 TSPS 的 RMB-AsymDPOP 性能对比

Fig.5.9 Performance comparison for RMB-AsymDPOP(k_{mb}) and RMB-AsymDPOP with TSPS(k_{mb}, k_p) under different agent numbers

图 5.9 中给出实验二的实验结果。从图中可以看出，即使结合内存有界推理，AsymDPOP 仍只能求解智能体数小于 20 的问题。但将 RMB-AsymDPOP 与 TSPS 进行结合时，算法的可扩展性和性能都获得极大的提升。具体地，RMB-AsymDPOP 在求解 16 个智能体的问题时至少是带 TSPS 的 RMB-AsymDPOP 产生网络负载的 1500 倍。给定 $k_p = 2$ 且 $k_{mb} \geq 6$ ，带 TSPS 的 RMB-AsymDPOP 能求解的问题可以扩展到 28 个智能体。此外，从图中还可以看出，较大的 k_{mb} 可以在所有评价指标上提升本章所提出算法的性能。这是因为更大的内存预算将导致更少的 CC 节点，并且这些算法所需的通信和计算开销也将减少。因此，设置带 TSPS 的 RMB-AsymDPOP 的参数为 $k_p = 2$ 和 $k_{mb} = 9$ ；RMB-AsymDPOP 的参数为 $k_{mb} = 9$ 。

5.7.2 实验结果与分析

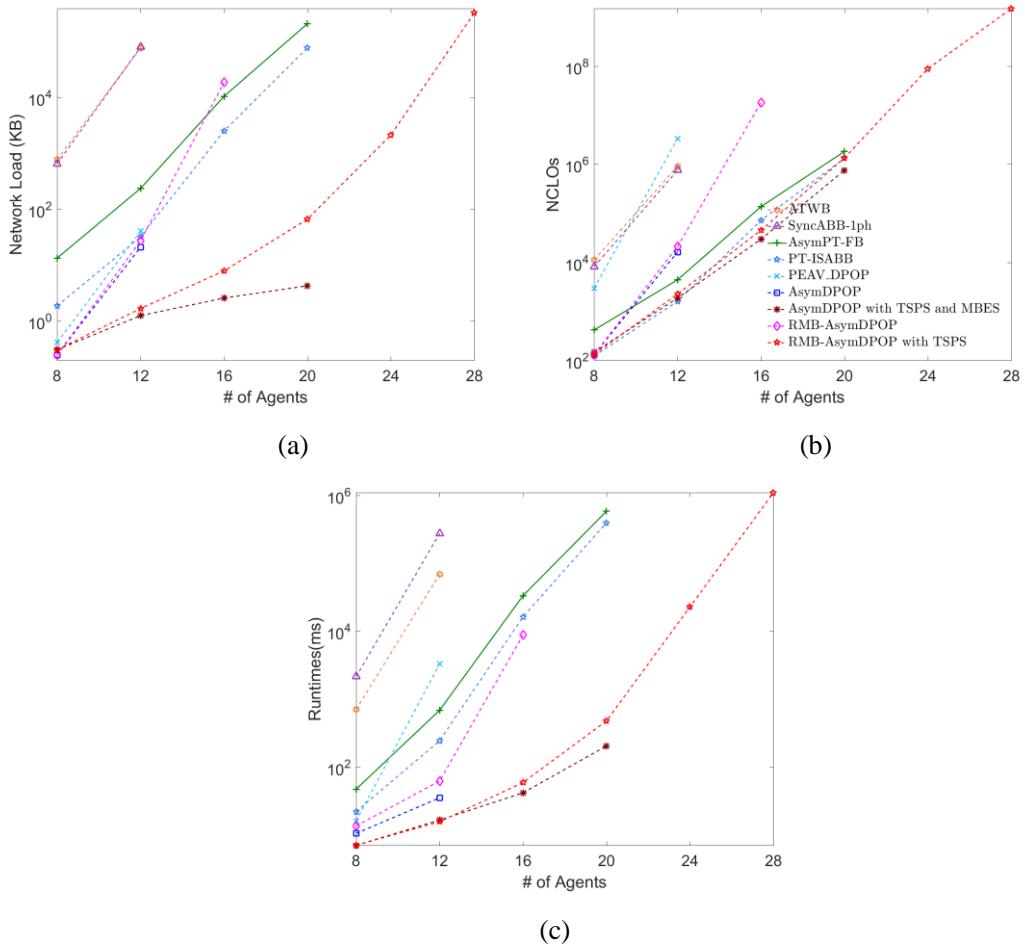


图 5.10 不同智能体数下各个算法的性能对比

Fig.5.10 Performance comparison under different agent numbers

实验三：不同智能体数下，ADCOP 完备算法的性能比较实验。该项实验的目的是测试不同智能体数下，各个算法的性能。该实验的具体配置与实验二中的配置相同。为对比的公平性，本节设置内存维度限制为 $k_{mb} = 9$ 。

图 5.10 给出实验三中 ADCOP 完备算法的对比实验结果。可以看出 AsymDPOP 能求解的问题规模和 ATWB, AsyncABB-1ph, PEAV_DPOP 类似，仅能求解 12 个智能体的问题，且带 TSPS 和 MBES 的 AsymDPOP 也仅能求解 20 个智能体的问题。这种现象表明，即使将 AsymDPOP 与 TSPS 结合，算法的内存消耗也是指数级的（如定理 5.2 所示）。一旦结合内存有界推理，算法的可扩展性得到提升。这表明在内存受限的情况下，TSPS 对于 RMB-AsymDPOP 和 RMB-AsymDPOP 的必要性。此外，与基于搜索的 ADCOP 完备算法相比，带 TSPS 的 AsymDPOP 在网络负载和运行时间方面展示出巨大的优势。这是由于基于搜索算法使用消息传递的方式实现问题求解，使得通信负载消耗的代价非常大。尤其是在求解智能体数多的问题时，代价非常巨大。相比之下，本章提出的算法（如带 TSPS 和 MBES

的 AsymDPOP 和带 TSPS 的 RMB-AsymDPOP) 由于采用推理策略, 它们仅需要线性的消息数, 进而减少算法所需的通信开销与运行时间。另一方面, 尽管 PEAV_DPOP 也使用推理策略, 但是它仍然存在严重的可扩展性问题, 且仅能求解智能体数小于 12 的问题。这是因为带有 PEAV 的 DPOP 引入镜像变量来确保原始变量与对应镜像变量赋值的一致性, 这增加算法求解的复杂度。

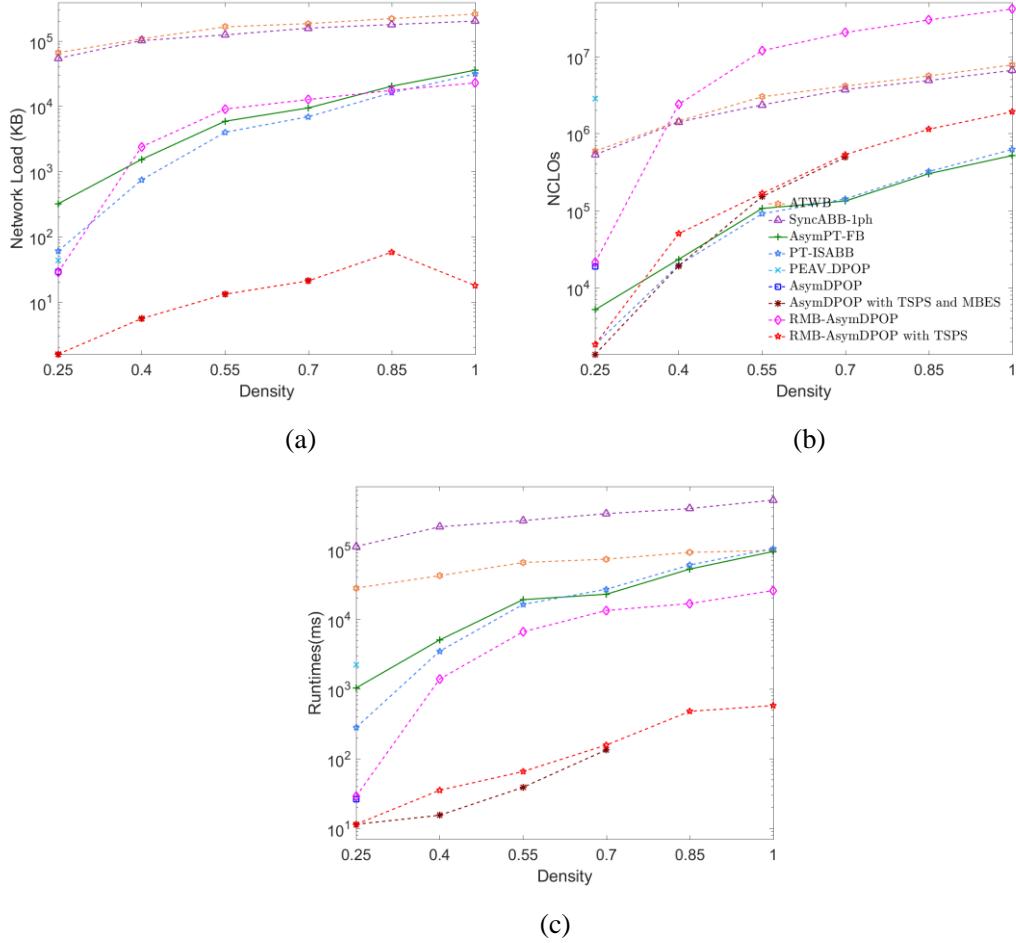


图 5.11 不同密度下各个算法的性能对比

Fig.5.11 Performance comparison under different densities

实验四：不同密度下，各个 ADCOP 完备算法的性能比较实验。具体地，我们选择值域大小均为 3 的随机 ADCOP 问题，设置代价选取范围为 [0, 100]，固定智能体个数为 12，并将密度按步长 0.15，从 0.25 变到 1。

图 5.11 给出实验四的实验结果。从图中可以看出，求解稠密问题时，相比于 AsymDPOP 和 RMB-AsymDPOP，带 TSPS 的 AsymDPOP 和 RMB-AsymDPOP 能较大地降低算法所产生的通信开销。这说明 TSPS 能有效地避免在传播效用表计算中不必要的联合操作。此外，可以看到当求解密度为 1.0 的问题时，它们的网络负

载小于求解密度为 0.85 问题时的网络负载。这是由于当密度等于 1.0 时，所有变量都将在根节点上消除。因此，带 TSPS ($k_p = 2$) 的算法可以避免所有的联合操作。由于带 TSPS 的算法避免不必要的联合操作，变量消除是维度增加的主要来源。因此，在带 TSPS 的 AsymDPOP 和 RMB-AsymDPOP 中，智能体在大多数时候都仅需传播较低维度大小的效用表。对于 NCLOs，在求解稠密问题时，基于搜索算法由于具有较强的解空间剪枝效率，所以优于 AsymDPOP 及其变体。

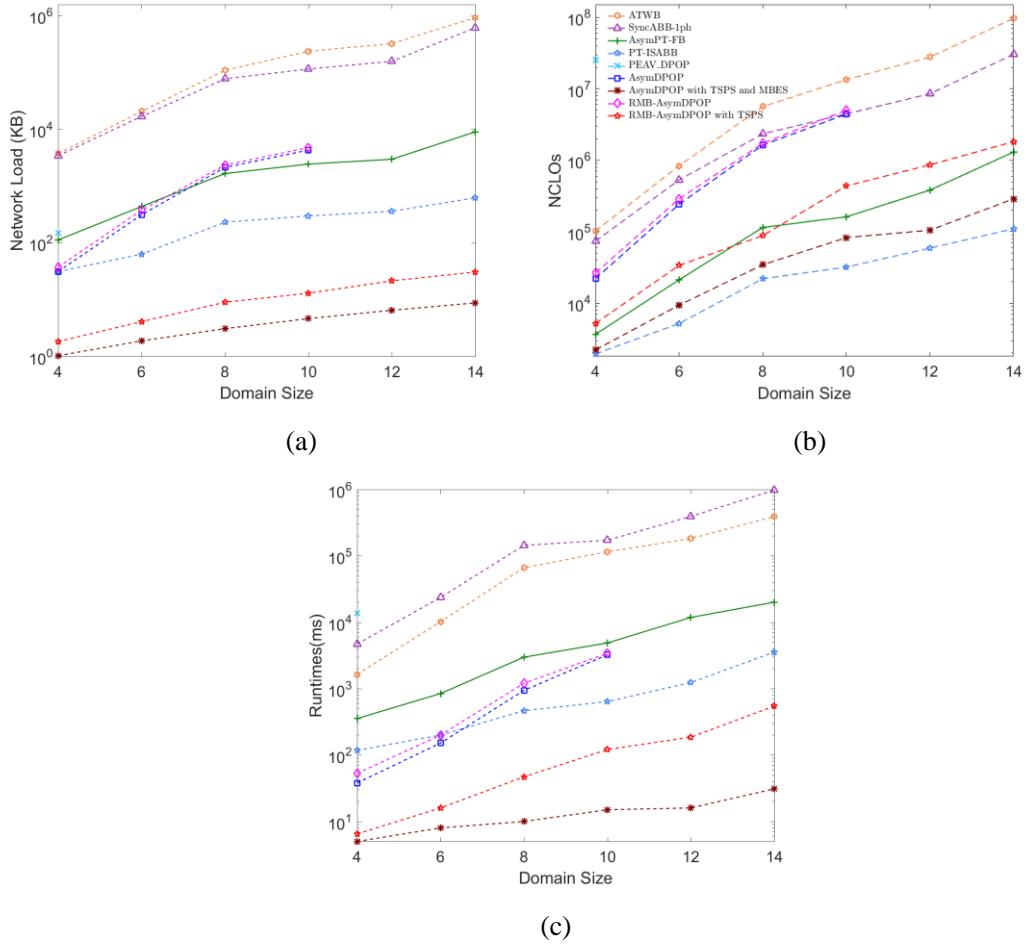


图 5.12 不同值域下各个算法的性能对比

Fig.5.12 Performance comparison under different domain size

实验五：不同值域下，各个 ADCOP 完备算法的性能比较实验。该项实验的目的是从值域的角度增加问题求解的难度，对比各个算法的性能。具体地，我们选择密度为 0.4 和智能体数为 8 的随机 ADCOP 问题，设置代价选取范围为 $[0, 100]$ ，并将值域大小按步长 2，从 4 变化到 14。

图 5.12 给出实验五的实验结果。从图中可以看出，随着值域大小的增加，所有算法的通信和计算开销都呈指数增长。其中，AsymDPOP 和 RMB-AsymDPOP

无法求解值域大小大于 10 的问题。这是因为基于推理的完备算法中的最大消息大小是指数级的，其基数是值域大小。因此，它们在求解值域较大的问题时会耗尽运行内存。一旦结合 TSPS，AsymDPOP 和 RMB-AsymDPOP 的可扩展性和性能获得极大的提升。尽管基于推理的算法往往产生比基于搜索的算法更大的消息，但与基于搜索的 ADCOP 算法相比，带 TSPS 的算法在网络负载和运行时间方面也展示出极大的优势。具体地，带 TSPS 的 RMB-AsymDPOP 所需的网络负载和运行时间仅是 PT-ISABB 的百分之 5。

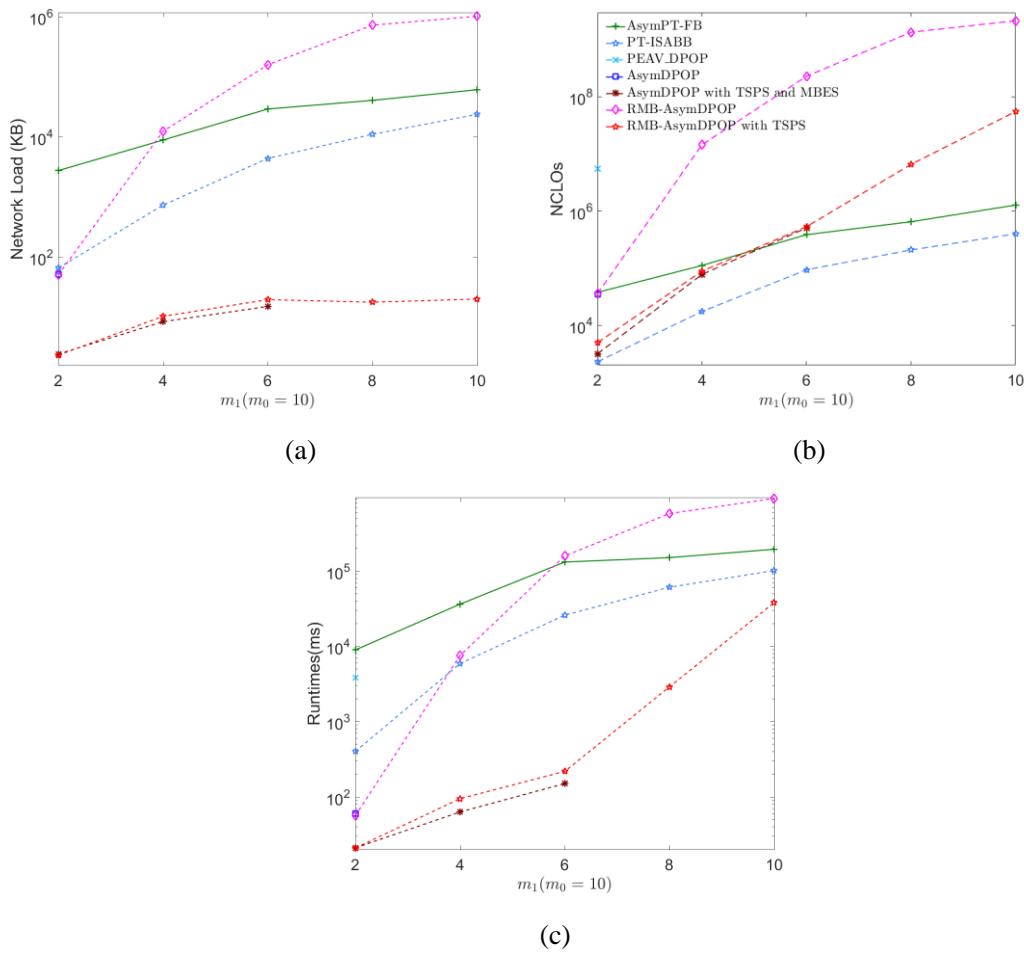


图 5.13 各个算法在求解无尺度网络问题时的性能对比

Fig. 5.13 Performance comparison on scale-free networks

实验六：不同算法在求解无尺度网络问题时的性能对比。该实验的目的是对比各个 ADCOP 完备算法在求解结构化问题时的性能。具体地，我们选择值域大小均为 3 的无尺度网络问题，设置代价选取范围为 $[0, 100]$ ，固定智能体个数为 16， m_0 为 10，并将 m_1 按步长 2，从 2 变到 10。

图 5.13 给出实验六的实验结果。由于 SynchABB-1ph 和 AWTB 无法在 30 分钟

内不能求解 $m_1 = 2$ 的问题，因此，实验图中没有包括这两种算法。从图中可以看出，由于内存预算有限，带 TSPS 和 MBES 的 AsymDPOP 和 AsymDPOP 存在非常严重的可扩展性问题。具体地，AsymDPOP 仅能求解 $m_1 = 2$ 的问题，且 AsymDPOP 与 TSPS 和 MBES 结合时最多也仅能求解 $m_1 = 6$ 的问题。一旦合并内存有界推理，AsymDPOP 以及带 TSPS 的 AsymDPOP 可以求解该配置中的所有问题，这也证明内存有界推理对于 AsymDPOP 和带 TSPS 的 AsymDPOP 是必不可少的。从图中也可以看出，在求解 m_1 值较大的问题时，RMB-AsymDPOP 在所有的评估指标上都要差于大部分基于搜索的算法。这是因为 AsymDPOP 需要巨大的通信和计算开销。

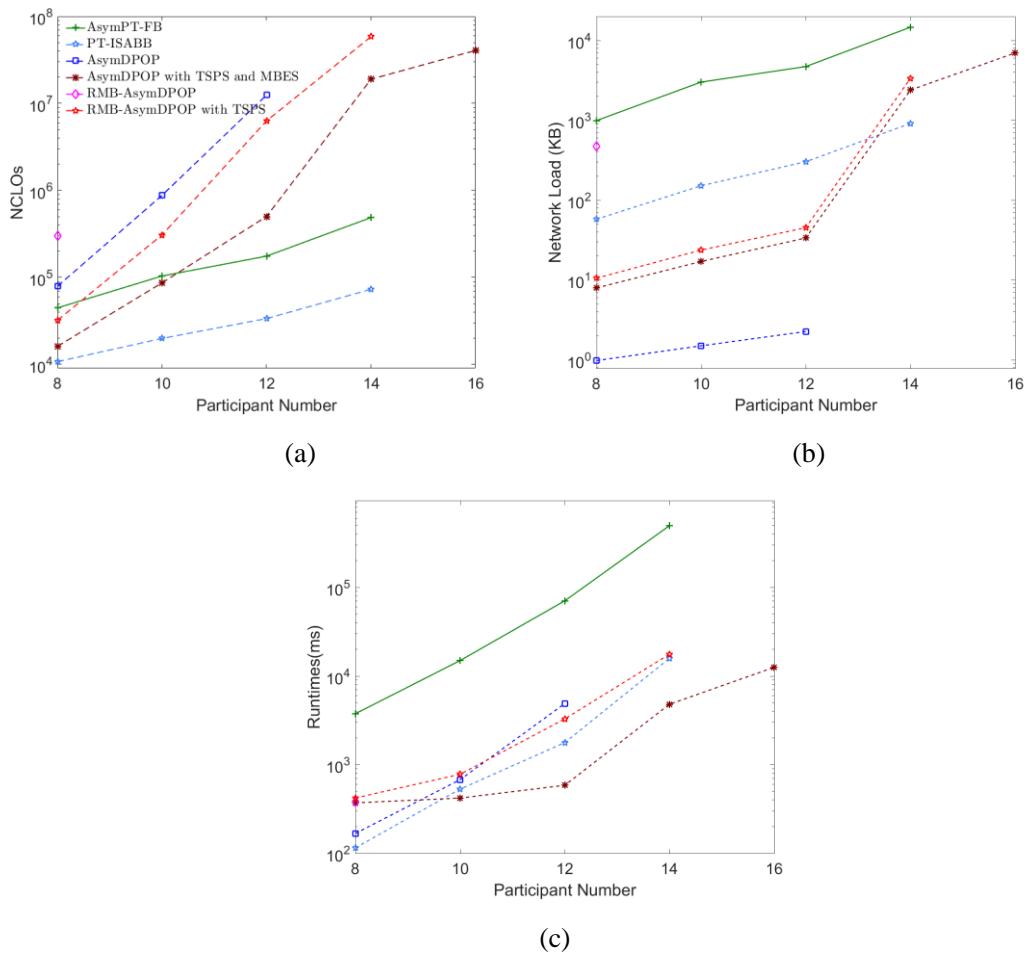


图 5.14 各个算法在求解会议调度问题时的性能对比

Fig. 5.14 Performance comparison on meeting schedule

实验七：不同算法在求解会议调度问题时的性能对比。该实验的目的是对比各个 ADCOP 完备算法在求解实际应用问题时的性能。具体地，我们设置参会人数按步长 2，从 8 变化到 16；固定会议数为 10；每个人从 10 个会议中随机选择 2 个会议进行参与；每个会议的持续时间随机从 2 到 6 个时间槽进行选择；会议可安排

的时间槽为 8。

图 5.14 给出在不同参会下各种算法在会议调度问题上的运行结果。由于 SynchABB-1ph 和 AWTB 在 15 分钟内无法求解参会人数为 4 的会议调度问题，所以未将这两个算法的实验结果放在实验对比图中。从图中可以看出，所有算法所需的计算和通信复杂度随着参会人的增加呈指数级增长。这是由于会议调度问题建模所对应的智能体个数正比于参会人数。由于所提出的推理算法（例如 AsymDPOP、RMB-AsymDPOP 和带 TSPS 和 MBES 的 AsymDPOP）在求解过程中，产生每条推理消息的大小以及所需的计算量都是指数级的，因此基于推理的求解算法在 NLCOs 和网络负载两个评价指标上差于基于搜索的求解算法（例如 AsymPT-FB 和 PT-ISABB）。但由于推理的求解算法仅需传播线性级的消息数，与消息的处理时间相比，消息传播的时间所需的时间可忽略不计。而基于搜索的求解算法其消息的处理时间仅为线性级的，但消息数却是指数级，因此消息的传播时间与处理时间一起构成了这类算法运行所需的时间。正如图 5.14 (c) 所示，基于推理的算法在运行时间上明显优于现有基于搜索的求解算法。此外，在结合本章提出的两种加速机制（TSPS 和 MBES），基于推理的算法在问题求解规模上也优于基于搜索的求解算法。

实验八：不同紧度下，算法求解约束满足问题的性能和私有偏好保障性能的比较。该项实验的目的是从约束满足问题紧度的角度来增加问题求解的难度，对比各个 ADCOP 完备算法的性能和求解问题时所泄露的私有信息对比。具体地，我们选择值域大小均为 10 的随机非对称 MaxDCSP 问题，固定智能体个数为 10，密度为 0.4，并将紧度按步长 0.1，从 0.1 变到 0.8。对于本章提出的算法，仅选择带 TSPS 的 AsymDPOP，没有选择 AsymDPOP 和 RMB-AsymDPOP 是因为它们在所有的问题上都耗尽内存；也没有选择带 TSPS 的 RMB-AsymDPOP 是因为在给定 k_p 值的情况下，它们与带 TSPS 的 AsymDPOP 导致相同的私有偏好泄露。

图 5.15 给出实验八中，各个算法的性能对比。从图中可以看出随着紧度的增长，所有基于搜索的算法所需的计算和通信开销都呈指数增长，但带 TSPS 的 AsymDPOP 似乎不受影响。这是因为基于搜索的完备算法采用系统地搜索来不重不漏地枚举解空间，并且随着紧度的提高，对解空间剪枝的难度逐渐增加。而带 TSPS 的 AsymDPOP 由于采用动态规划策略实现问题的求解，因此它的求解性能不受紧度变化的影响。至于私有偏好保障，从图 5.15 (c) 可以看出，随着紧度的增加，基于搜索的算法会泄漏更多的私有偏好，而基于推理的算法泄漏的私有偏好较少。这是由于基于搜索的算法依靠直接泄露机制来实现双边私有约束的累积，且在求解高紧度的问题时需要遍历更多的搜索空间。相反，基于推理的算法通过在伪树自下而上地传播和累积效用，其仅在如下两种情况下存在私有偏好泄露：1)

当接收包含 x_i 和 x_c 维度的效用表时, 智能体 a_i 可以从该效用表中为零的位置处推断出其孩子或伪孩子 $a_c \in AC(a_i)$ 的私有约束; 2) 当该效用表是未经过消元且是仅包含 x_i 和 x_c 两个维度时, a_i 可以直接推断出 a_c 完整的私有约束。因此, 带 TSPS ($k_p = 2$) 的 AsymDPOP 几乎泄漏一半的私有偏好。另一方面, 在求解高紧度问题时, 带 TSPS 的 AsymDPOP ($k_p \geq 3$) 的私有偏好泄露也随之降低。这是因为随着紧度的增加, 问题中非零代价的赋值组合数量也会增加, 即: 情况 1) 出现的几率逐渐降低。

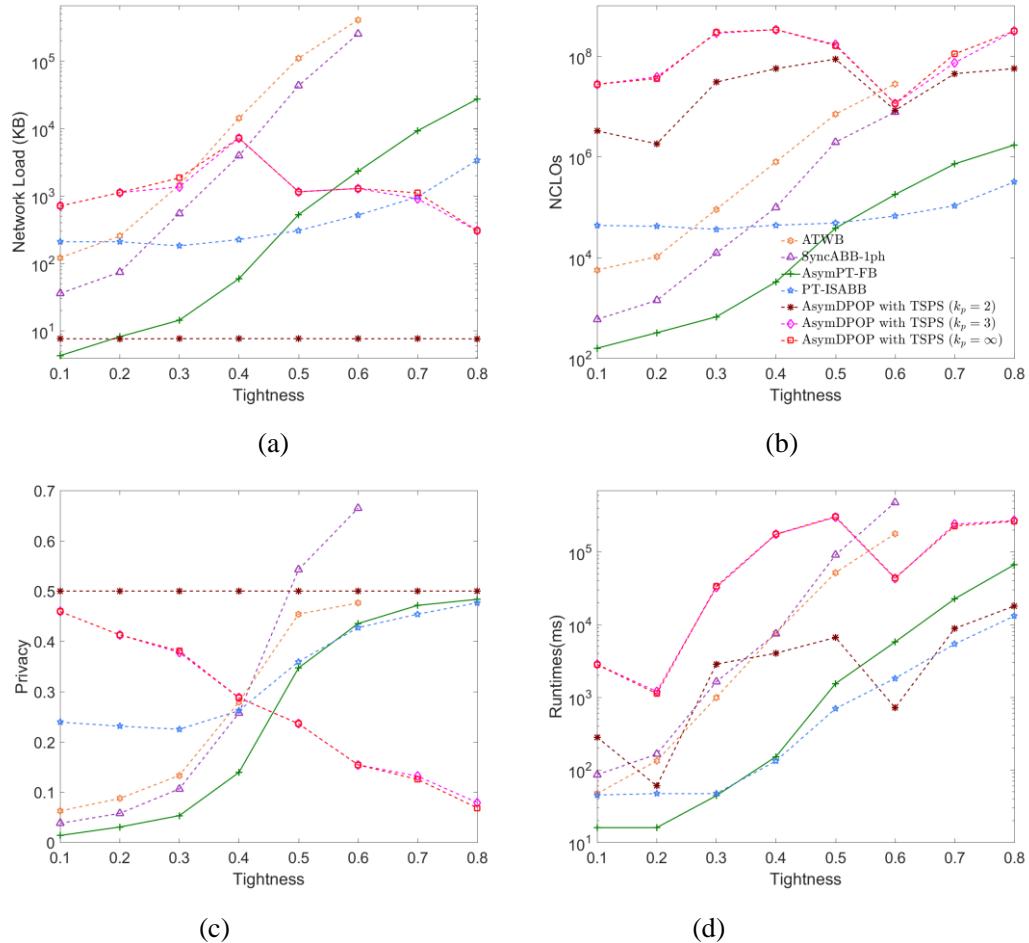


图 5.15 各个算法在求解不同紧度下非对称 MaxDCSP 问题时的性能对比

Fig. 5.15 Performance comparison for asymmetric MaxDCSPs under different tightness

5.8 本章小结

基于本章提出广义非本地消元机制, 将推理策略用于 ADCOP 完备求解中, 提出 AsymDPOP 算法。为提高 AsymDPOP 的性能和可扩展性, 引入效用表集合传播方案 (TSPS) 来减少内存消耗, 引入小批量消除方案 (MBES) 来减少效用传播阶段中计算效用表的计算量。为保证提出的算法在有限的内存预算下仍能求解较

大规模问题,我们将内存有界推理适配到 AsymDPOP 和带 TSPS 的 AsymDPOP 中,其中引入分支独立的分布式枚举机制来减少内存受限推理中的冗余,并引入两阶段效用集合传播机制来减少有界效用集传播阶段的冗余。在理论上,分析提出算法(例如 AsymDPOP 和带 TSPS 的 AsymDPOP)的空间复杂度。实验结果表明,本章提出算法优于目前最好的 ADCOP 完备算法,也优于 PEAV_DPOP。此外,一旦与内存有界推理和 TSPS 结合,AsymDPOP 在内存有限环境下能求解的问题规模和性能都获得提升。

6 总结与展望

6.1 本文工作总结

布式约束优化问题（DCOP）是多智能体系统协调优化的重要框架，具有重要的研究意义和实用价值。非对称分布式约束优化问题（ADCOP）是在 DCOP 基础上增加非对称特性，具有更强的建模能力。完备算法由于保障能找到问题最优解且具有较强的理论性质，是目前 DCOP/ADCOP 求解算法研究的热点。本文从近似推理与搜索混合的 DCOP 算法为基础，研究内存预算对这类算法求解性能的影响，据此提出上文推理与搜索混合的 DCOP 求解算法。另外，考虑 ADCOP 的非对称特性，研究推理策略用于 ADCOP 求解的可能性，创新性地提出一种基于推理与搜索混合的 ADCOP 完备算法，并提出针对非对称特性的消元策略以提升基于推理与搜索混合的 ADCOP 完备算法的求解效率。以本文提出非对称特性的消元策略为基础，研究采用推理策略实现 ADCOP 的完备求解，率先提出仅采用推理策略的 ADCOP 完备算法。具体地，本文研究工作内容如下：

① 提出上文推理与搜索混合的 DCOP 完备算法

利用在内存受限情况下，上文推理依然可以提供紧的下界的优势，研究如何有效地将上文推理与搜索进行混合，提出上文推理与搜索混合的 DCOP 完备算法 HS-CAI。在该算法中，上文推理与搜索互相辅助、并行执行。其中，搜索通过上文的扩展与传播实现对解空间的探索，同时为上文推理提供迭代推理所需的上文进而避免指数级的上文推理所带来的巨大网络负载消耗。上文推理利用搜索提供的上文保证传播效用表的更加完整，从而为搜索提供紧的下界来提升搜索的剪枝效率同时减少上文推理的次数。进一步，本文提出上文筛选机制在不影响算法性能的同时，降低上文推理带来的网络负载。本文理论证明在相同内存预算的条件下，上文推理过程相比于预处理推理过程能够提供更紧的下界，且实验结果证明该算法在多种测试问题上均优于目前性能最好的 DCOP 完备算法。

② 提出基于非本地消元的推理与搜索混合的 ADCOP 算法

基于工作一对推理与搜索混合 DCOP 算法理解，考虑 ADCOP 的非对称特性，研究如何将推理策略用于 ADCOP 求解，提出基于推理与搜索混合的 ADCOP 完备算法 PT-ISABB。该算法在推理阶段求解问题的部分约束，然后在搜索阶段采用基于伪树的分布式分支定界算法求解整个问题。具体地，推理阶段求出问题单面约束用作搜索阶段的下界以实现剪枝。给定推理阶段获得的下界，搜索阶段在伪树不同分枝独立并行地执行分布式分支定界搜索策略。为进一步提升推理阶段提供下界的紧度，本文提出非本地消元机制，即：将智能体的消元位置推迟到父节点

以获得更完整的效用表。此外，通过结合绝对误差机制和相对误差机制，提出两种 PT-ISABB 的非完备扩展版本。这两种非完备算法适用于在用户分别指定绝对误差界和相对误差界的前提下，以较小的计算量和消息数来快速地找到满足条件的一组近似解。本文理论证明本文所提出算法的正确性；在内存维度小于问题诱导宽度时，PT-ISABB 提供下界的紧度优于现有性能最好的 ADCOP 完备搜索求解算法；且实验结果表明该算法在多种测试问题上均优于最好的 ADCOP 完备算法。

③ 提出基于推理的 ADCOP 完备算法

基于工作二提出的非本地消元机制，本文提出广义非本地消元机制，即：智能体的消元位置推迟到其在伪树上最高的（伪）父节点，以确保在私有约束函数不直接泄露的同时也能保证变量消元的完备性。基于该机制，本文率先提出基于推理的 ADCOP 完备算法 AsymDPOP。进一步，为解决由于 AsymDPOP 算法由于推迟消元带来的内存占用以及计算量增加，本文提出桶集合的传播策略 TSPS 和小批次的消元策略 MBES 来分别减少算法的内存占用和计算量。同时为使得提出的算法适用于内存有限的环境，本文将内存有界推理思想适配到提出的算法中，进而提升算法的可扩展性。此外，本文还提出分支独立性枚举机制和两阶段的效用集合传播机制来减少内存有界推理中的冗余推理。本文理论分析所提出算法的内存占用，且实验结果表明本文所提出的算法在多种测试问题上均优于目前性能最好的 ADCOP 完备算法，且在求解的问题规模上远超现有的 ADCOP 完备算法。

6.2 未来研究展望

在前人的基础上，本人针对 DCOP/ADCOP 算法展开一些有益的探索和研究，创新性地提出一些 DCOP/ADCOP 完备算法。但文中难免存在着一定的不足之处，需要进一步的研究的问题还很多。在未来的工作中，我们将如下几个方面展开进一步的研究。

针对基于搜索的完备算法，进一步研究的方向有如下几点：① 在算法求解过程中对解空间进行排序，更快地获得更紧的上界；② 内存有界的条件下，设计针对于特定分布式搜索算法的一套统一的存储机制与换页策略以减少对子问题的重复搜索；③ 搜索阶段如何有效地执行多个搜索过程，提升算法的并行性。

针对基于推理的完备算法，进一步研究的方向有如下几点：① 采用剪枝技术，减少传播效用表的大小，从而降低算法所产生的网络负载；② 研究高效的效用表计算方案。算法中消元和联合操作都涉及大量的高维矩阵索引操作，设计高效的矩阵遍历算法能有效地提升算法的运行速度。③ 内存有界的条件下，采用函数近似表示超出内存限制的效用表，在保证算法的求解质量的同时，提升算法的性能。

参考文献

- [1] Russell, Stuart J. Artificial Intelligence: A Modern Approach[M]. 人民邮电出版社, 2002.
- [2] Wooldridge M. An introduction to multiagent systems[M]. John Wiley & Sons, 2009.
- [3] Weiss, Gerhard, ed. Multiagent systems[M]. MIT press, 2013.
- [4] Sycara K P. Multiagent systems[J]. AI magazine, 1998, 19(2): 79-79.
- [5] Shoham Y, Leyton-Brown K. Multiagent systems: Algorithmic, game-theoretic, and logical foundations[M]. Cambridge University Press, 2008.
- [6] Cerquides J, Farinelli A, Meseguer P, et al. A tutorial on optimization for multi-agent systems[J]. The Computer Journal, 2014, 57(6): 799-824.
- [7] Leite, A.R., Enembreck, F., Barthes, J.P.A. Distributed constraint optimization problems: review and perspectives[J]. Expert Systems with Applications, 2014, 41(11): 5139-5157.
- [8] Fioretto, F., Pontelli, E., & Yeoh, W. Distributed constraint optimization problems and applications: a survey[J]. Journal of Artificial Intelligence Research, 2018, 61: 623-698.
- [9] Petcu, A., Faltings, B. Distributed constraint optimization applications in power networks[J]. International Journal of Innovations in Energy Systems and Power, 2008, 3(1): 1-12.
- [10] Fioretto, F., Yeoh, W., Pontelli, E., Ma, Y., Ranade, S.J. A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response[C]. Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, 2017: 999–1007.
- [11] Cheng, S., Raja, A., Xie, J. Dynamic multi-agent load balancing using distributed constraint optimization techniques[J]. Web Intelligence and Agent Systems: An International Journal, 2014, 12(2), 111-138.
- [12] Monteiro, T.L., Pujolle, G., Pellenz, M.E., Penna, M.C., Enembreck, F., Souza, R.D., 2012. A multi-agent approach to optimal channel assignment in WLANs[C]. IEEE Wireless Communications and Networking Conference, 2012: 2637–2642.
- [13] Enembreck, F., Barths, J.P.A. Distributed constraint optimization with MULBS: a case study on collaborative meeting scheduling[J]. Journal of Network and Computer Applications, 2012, 35(1): 164-175.
- [14] Hirayama, K., Miyake, K., Shiota, T., Okimoto, T. DSSA+: Distributed collision avoidance algorithm in an environment where both course and speed changes are allowed[J]. TransNav, International Journal on Marine Navigation and Safety of Sea Transportation 2019, 13: 117–123.

- [15] Zhang, W., Wang, G., Xing, Z., Wittenburg, L. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks[J]. Artificial Intelligence, 2005, 161(1): 55-87.
- [16] Pereira, Igor Avila, Lisane B. de Brisolara, and Paulo Roberto Ferreira. Application-Level Load Balancing for Reactive Wireless Sensor Networks: An Approach Based on Constraint Optimization Problems[J]. Brazilian Conference on Intelligent Systems, 2020: 63-76.
- [17] Ruben Stranders, Long Tran-Thanh, Francesco Maria Delle Fave, Alex Rogers, and Nick Jennings. DCOPs and bandits: Exploration and exploitation in decentralised coordination. Proceedings the 11th International Conference on Autonomous Agents and Multi-Agent Systems. 2012: 289-297.
- [18] Lisý V, Zivan R, Sycara K, et al. Deception in networks of mobile sensing agents[C]. Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. 2010: 1031-1038.
- [19] Grinshpoun T, Grubshtain A, Zivan R, et al. Asymmetric Distributed Constraint Optimization Problems[J]. Journal of Artificial Intelligence Research, 2013, 47: 613-647.
- [20] Mailler, Roger, and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation[C]. Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, 2004: 438-445.
- [21] Grinshpoun T, Meisels A. Completeness and performance of the APO algorithm[J]. Journal of Artificial Intelligence Research, 2008, 33: 223-258.
- [22] Petcu A, Faltings B, Mailler R. PC-DPOP: A New Partial Centralization Algorithm for Distributed Optimization[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2007, 7: 167-172.
- [23] Hirayama K, Yokoo M. Distributed partial constraint satisfaction problem[C]. Proceedings of International Conference on Principles and Practice of Constraint Programming, 1997: 222-236.
- [24] Grinshpoun T, Tassa T. A privacy-preserving algorithm for distributed constraint optimization[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2014: 909-916.
- [25] Grinshpoun T, Tassa T. P-SyncBB: A privacy preserving branch and bound DCOP algorithm[J]. Journal of Artificial Intelligence Research, 2016, 57: 621-660.
- [26] Gershman A, Meisels A, Zivan R. Asynchronous forward bounding for distributed COPs[J]. Journal of Artificial Intelligence Research, 2009, 34(1): 61-88.

- [27] Ezzahir, R., Bessiere, C., Benelallam, I. and Belaissaoui, M. Asynchronous breadth first search DCOP algorithm[J]. *Applied Mathematical Sciences*, 2008, 2(37-40): 1837-1854.
- [28] Netzer A, Grubshtain A, Meisels A. Concurrent forward bounding for distributed constraint optimization problems[J]. *Artificial Intelligence*, 2012, 193: 186-216.
- [29] Modi, Pragnesh Jay, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. An asynchronous complete method for distributed constraint optimization[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2003: 161-168.
- [30] Modi P J, Shen W M, Tambe M, et al. ADOPT: asynchronous distributed constraint optimization with quality guarantees[J]. *Artificial Intelligence*, 2005, 161(1): 149-180.
- [31] Ali, Syed, Sven Koenig, and Milind Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT[C]. Proceedings of the 4th International Joint Conference on Artificial Intelligence, 2005: 1041-1048.
- [32] Ali S M, Koenig S, Tambe M. Preprocessing techniques for distributed constraint optimization[C]. International Conference on Principles and Practice of Constraint Programming, 2004: 706-710.
- [33] Matsui T, Matsuo H. A Distributed Cooperative Search Algorithm Using Multiple Contexts and Pruning[C]. Proceedings of the 26th International Conference on Computers and Their Applications, 2011: 1-8.
- [34] Gutierrez P, Meseguer P. Saving messages in ADOPT-based algorithms[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010: 53-64.
- [35] Yeoh W, Felner A, Koenig S. IBD-ADOPT: A depth-first search DCOP algorithm[C]. International Workshop on Constraint Solving and Constraint Logic Programming, 2008: 132-146.
- [36] Silaghi M C, Yokoo M. Nogood based asynchronous distributed optimization (ADOPT-ng)[C]. Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems, 2006: 1389-1396.
- [37] Silaghi M C, Yokoo M. Dynamic DFS tree in ADOPT-ing[C]. Proceedings of the AAAI Conference on Artificial Intelligence, 1999, 2007, 22(1): 763.
- [38] Silaghi M C, Yokoo M. ADOPT-ing: unifying asynchronous distributed optimization with asynchronous backtracking[J]. *Autonomous Agents and Multi-Agent Systems*, 2009, 19(2): 89-123.
- [39] Yeoh W, Felner A, Koenig S. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm[J]. *Journal of Artificial Intelligence Research*, 2010, 38: 85-133.

- [40] Gutierrez P, Meseguer P. Saving redundant messages in BnB-ADOPT[C]. Proceedings of the 24th AAAI Conference on Artificial Intelligence, 2010: 1259-1260.
- [41] Gutierrez P, Meseguer P. Removing redundant messages in n-ary BnB-ADOPT[J]. Journal of Artificial Intelligence Research, 2012, 45: 287-304.
- [42] Yeoh W, Felner A, Koenig S. BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm[J]. Journal of Artificial Intelligence Research, 2010, 38: 85-133.
- [43] Lee J H M, Meseguer P, Su W. Adding laziness in BnB-ADOPT+[J]. Constraints, 2015, 20(2): 274-282.
- [44] Gutierrez P, Meseguer P, Yeoh W. Generalizing ADOPT and BnB-ADOPT[C]. Proceedings of International Joint Conference on Artificial Intelligence, 2011: 554-559.
- [45] Chen Z, He C, He Z, et al. BD-ADOPT: a hybrid DCOP algorithm with best-first and depth-first search strategies[J]. Artificial Intelligence Review, 2018, 50(2): 161-199.
- [46] Chechetka A, Sycara K. No-commitment branch and bound search for distributed constraint optimization[C]. Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems, 2006: 1427-1429.
- [47] Litov O, Meisels A. Forward bounding on pseudo-trees for DCOPs and ADCOPs[J]. Artificial Intelligence, 2017, 252: 83-99.
- [48] Matsui T, Silaghi M C, Hirayama K, et al. Directed soft arc consistency in pseudo trees[C]. Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, 2009(2): 1065-1072.
- [49] Matsui T, Silaghi M C, Hirayama K, et al. Applying soft arc consistency to distributed constraint optimization problems[J]. Transactions of the Japanese Society for Artificial Intelligence, 2010, 25(3): 410-422.
- [50] Gutierrez P, Meseguer P. Improving BnB-ADOPT+-AC[C]. Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, 2012(1): 273-280.
- [51] Gutierrez P, Meseguer P. Distributed constraint optimization problems related with soft arc consistency[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2011: 2812-2813.
- [52] Gutierrez P, Lee J H M, Lei K M, et al. Maintaining soft arc consistencies in BnB-ADOPT+ during search[C]. International Conference on Principles and Practice of Constraint Programming. Springer, Berlin, Heidelberg, 2013: 365-380.
- [53] Adrdor R, Ezzahir R, Koutti L. Connecting AFB_BJ+ with soft arc consistency[J]. International Journal of Computing and Optimization, 2018, 5(1): 9-20.

- [54] Matsui T, Matsuo H, Iwata A. Efficient Methods for Asynchronous Distributed Constraint Optimization Algorithm[C]. Artificial Intelligence and Applications. 2005: 727-732.
- [55] Chechetka A, Sycara K P. An Any-space Algorithm for Distributed Constraint Optimization[C]. AAAI Spring Symposium: Distributed Plan and Schedule Management, 2006: 33-40.
- [56] Yeoh W, Varakantham P, Koenig S. Caching schemes for DCOP search algorithms[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2009(1): 609-616.
- [57] Petcu A, Faltings B. A scalable method for multiagent constraint optimization[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2005, 5: 266-271.
- [58] Dechter R, Rish I. Mini-buckets: A general scheme for bounded inference[J]. Journal of the ACM (JACM), 2003, 50(2): 107-153.
- [59] Dechter R. Bucket elimination: A unifying framework for reasoning[J]. Constraints, 2013, 2(1):51-55.
- [60] Léauté T, Faltings B. Protecting privacy through distributed computation in multi-agent decision making[J]. Journal of Artificial Intelligence Research, 2013, 47: 649-695.
- [61] Petcu A, Faltings B. MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2007: 1452-1457.
- [62] Chen Z, Zhang W, Deng Y, et al. RMB-DPOP: Refining MB-DPOP by Redundant Inferences[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2020: 249-257.
- [63] Petcu A, Faltings B. ODPOP: an algorithm for open/distributed constraint optimization[C]. Proceedings of the AAAI Conference on Artificial Intelligence, 2006: 21(1): 703-708.
- [64] Ottens B, Faltings B. Asynchronous open DPOP[C]. Proceedings of the 10th International Workshop on Distributed Constraint Reasoning, 2008: 36-42.
- [65] Atlas J, Decker K. A complete distributed constraint optimization method for non-traditional pseudotree arrangements[C]. Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems. 2007: 1-8.
- [66] Brito I, Meseguer P. Cluster tree elimination for distributed constraint optimization with quality guarantees[J]. Fundamenta Informaticae, 2010, 102(3-4): 263-286.
- [67] Vinyals M, Rodriguez-Aguilar J A, Cerquides J. Generalizing DPOP: action-GDL, a new complete algorithm for DCOPs[C]. Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems, 2009: 1239-1246.

- [68] Vinyals M, Rodriguez-Aguilar J A, Cerquides J. Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law[J]. Autonomous Agents and Multi-Agent Systems, 2011, 22(3): 439-464.
- [69] Paskin M, Guestrin C, McFadden J. A robust architecture for distributed inference in sensor networks[C]. Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, 2005: 55-62.
- [70] Chen Z, He Z, He C. An improved DPOP algorithm based on breadth first search pseudo-tree for distributed constraint optimization[J]. Applied Intelligence, 2017, 47(3): 607-623.
- [71] Kumar A, Petcu A, Faltings B. H-DPOP: Using Hard Constraints for Search Space Pruning in DCOP[C]. Proceedings of the AAAI Conference on Artificial Intelligence, 2008: 325-330.
- [72] Fioretto F, Le T, Yeoh W, et al. Improving DPOP with branch consistency for solving distributed constraint optimization problems[C]. Proceedings of International Conference on Principles and Practice of Constraint Programming, 2014: 307-323.
- [73] Rashik M, Rahman M M, Khan M M, et al. Speeding up distributed pseudo-tree optimization procedures with cross edge consistency to solve DCOPs[J]. Applied Intelligence, 2021, 51(3): 1733-1746.
- [74] Sánchez M, Larrosa J, Meseguer P. Tree decomposition with function filtering[C]. Proceedings of the International Conference on Principles and Practice of Constraint Programming, 2005: 593-606.
- [75] Sánchez-Fibla M, Larrosa J, Meseguer P. Improving Tree Decomposition Methods With Function Filtering[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2005: 1537-1538.
- [76] Brito I, Meseguer P. Improving DPOP with function filtering[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems. 2010, 1435: 141-148.
- [77] Pujol-Gonzalez M, Cerquides J, Meseguer P, et al. Communication-constrained DCOPs: message approximation in GDL with function filtering[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2011: 379-386.
- [78] Pujol-Gonzalez M, Cerquides J, Meseguer P, et al. Improving function filtering for computationally demanding DCOPs[C]. Proceedings of the International Joint Conference on Artificial Intelligence Workshop, 2011: 99-101.
- [79] Atlas J, Warner M, Decker K. A memory bounded hybrid approach to distributed constraint optimization[C]. Proceedings of 10th International Workshop on Distributed Constraint Reasoning, 2008: 37-51.

- [80] Kim Y, Lesser V. DJAO: A Communication-Constrained DCOP algorithm that combines features of ADOPT and Action-GDL[C]. Proceedings of the 28th AAAI Conference on Artificial Intelligence, 2014: 2680-2687.
- [81] Yeoh W, Sun X, Koenig S. Trading off solution quality for faster computation in DCOP search algorithms[C]. Proceedings of the 21st AAAI Conference on Artificial Intelligence. 2009.
- [82] Petcu A, Faltings B. Approximations in distributed optimization[C]. Proceedings of the International Conference on Principles and Practice of Constraint Programming. Springer, Berlin, Heidelberg, 2005: 802-806.
- [83] Petcu A, Faltings B. LS-DPOP. A Hybrid of Inference and Local Search for Distributed Combinatorial Optimization[C]. Proceedings of the International Conference on Intelligent Agent Technology, 2007: 342-348.
- [84] Zhang W, Wang G, Xing Z, Wittenburg L. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks[J]. Artificial Intelligence, 2005, 161(1): 55-87.
- [85] Maheswaran R T, Pearce J P, Tambe M. Distributed algorithms for DCOP: a graphical-game-based approach[C]. Proceedings of ISCA PDGS, 2004: 432-439.
- [86] Maheswaran R T, Pearce J P, Tambe M. A family of graphical-game-based algorithms for distributed constraint optimization problems[C]. Coordination of large-scale multiagent systems, 2006: 127-146.
- [87] Steven, O., Roie, Z., Aviv, N. Distributed breakout: Beyond satisfaction[C]. Proceedings of the 25th International Joint Conference on Artificial Intelligence, 2016:447453.
- [88] Yokoo M, Hirayama K. Distributed breakout algorithm for solving distributed constraint satisfaction problems[C]. Proceedings of the Second International Conference on Multi-Agent Systems. 1996: 401-408.
- [89] Yokoo M, Durfee E H, Ishida T, et al. The distributed constraint satisfaction problem: Formalization and algorithms[J]. IEEE Transactions on knowledge and data engineering, 1998, 10(5): 673-685.
- [90] Hirayama K, Yokoo M. The distributed breakout algorithms[J]. Artificial Intelligence, 2005, 161(1): 89-115.
- [91] Fioretto, F., Campeotto, F., et al. Large neighborhood search with quality guarantees for distributed constraint optimization problems[C]. Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems, 2015: 1835-1836.

- [92] Hoang K D, Fioretto F, Yeoh W, et al. A large neighboring search schema for multi-agent optimization[C]. Proceedings of the International Conference on Principles and Practice of Constraint Programming, 2018: 688-706.
- [93] Chapman A C, Alex R, Jennings N R, et al. A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems[J]. Knowledge Engineering Review, 2011, 26(4):411-444.
- [94] Pearce J P, Tambe M. Quality Guarantees on k-Optimal Solutions for Distributed Constraint Optimization Problems[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2007: 1446-1451.
- [95] Pearce J P, Maheswaran F T, Tambe M. How local is that optimum? k-optimality for DCOP[C]. Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems, 2005: 1303-1304.
- [96] Bowring E, Pearce J P, Portway C, et al. On k-optimal distributed constraint optimization algorithms: new bounds and algorithms[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2008: 607-614.
- [97] Vinyals M, Shieh E, Cerquides J, et al. Quality guarantees for region optimal DCOP algorithms[C]. Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, 2011, 1: 133-140.
- [98] Okimoto T, Joe Y, Iwasaki A, et al. Pseudo-tree-based incomplete algorithm for distributed constraint optimization with quality bounds[C]. Proceedings of the International Conference on Principles and Practice of Constraint Programming, 2011: 660-674.
- [99] Pearce J P, Tambe M, Maheswaran R. Solving multiagent networks using distributed constraint optimization[J]. AI Magazine, 2008, 29(3): 47-47.
- [100] Katagishi, H, Pearce, J. P. KOPT: distributed DCOP algorithm for arbitrary k-optimal with monotonically increasing utility[C]. Proceedings of DCR-07, 2007: 34-42.
- [101] Kiekintveld C, Yin Z, Kumar A, et al. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, 10: 133-140.
- [102] Tassa T, Zivan R, Grinshpoun T. Preserving Privacy in Region Optimal DCOP Algorithms[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2016: 496-502.
- [103] Grinshpoun T, Tassa T, Levit V, et al. Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs[J]. Artificial Intelligence, 2019, 266: 27-50.

- [104] Chen Z, He J, et al. A Partial Decision Scheme for Local Search Algorithms for Distributed Constraint Optimization Problems[C]. Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, 2017: 187-194.
- [105] Zilberstein S. Using anytime algorithms in intelligent systems[J]. AI magazine, 1996, 17(3): 73-83.
- [106] Zivan R, Okamoto S, Peled H. Explorative anytime local search for distributed constraint optimization[J]. Artificial Intelligence, 2014, 212: 1-26.
- [107] MS , A. Farinelli, A. Rogers, A. Petcu, N.R. Jennings, Decentralized coordination of low-power embedded devices using the max-sum algorithm[C]. Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, 2008: 639 – 646.
- [108] Aji S M, McEliece R J. The generalized distributive law[J]. IEEE Transactions on Information Theory, 2000, 46(2): 325-343.
- [109] Kschischang F R, Frey B J, Loeliger H A. Factor graphs and the sum-product algorithm[J]. IEEE Transactions on information theory, 2001, 47(2): 498-519.
- [110] Rogers, A. Farinelli, R. Strandersa and N.R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm[J]. Artificial Intelligence, 2011, 175(2): 730-759.
- [111] Rollon, E., Larrosa, J. Improved bounded Max-sum for distributed constraint optimization[C]. Proceedings of the International Conference on Principles and Practice of Constraint Programming, 2012: 624-632.
- [112] Rollon, E., Larrosa, J. Decomposing utility functions in bounded max-sum for distributed constraint optimization[C]. Proceedings of the International Conference on Principles and Practice of Constraint Programming, 2014:646-654.
- [113] Zivan R, Peled H. Max/min-sum distributed constraint optimization through value propagation on an alternating DAG[C]. Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, 2012, 1: 265-272.
- [114] Zivan R, Parash T, Cohen L, et al. Balancing exploration and exploitation in incomplete Min/Max-sum inference for distributed constraint optimization[J]. Autonomous Agents and Multi-Agent Systems, 2017, 31(5): 1165-1207.
- [115] Chen Z, Deng Y, Wu T. An iterative refined Max-sum_AD algorithm via single-side value propagation and local search[C]. Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, 2017: 195-202.
- [116] Z. Chen, Y. Deng, T. Wu, et al. A class of iterative refined Max-sum algorithms via non-consecutive value propagation strategies[J]. Autonomous Agents and Multi-Agent Systems, 2018, 32(6): 822-860.

- [117] Cohen L, Zivan R. Max-sum Revisited: The Real Power of Damping[C]. Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. 2017: 1505-1507.
- [118] Cohen L, Galiki R, Zivan R. Governing convergence of Max-sum on DCOPs through damping and splitting[J]. Artificial Intelligence, 2020, 279: 1-22.
- [119] Zivan R, Lev O, Galiki R. Beyond Trees: Analysis and Convergence of Belief Propagation in Graphs with Multiple Cycles[C]. Proceedings of the 34th AAAI Conference on Artificial Intelligence, 2020: 7333-7340.
- [120] Tassa T, Zivan R, Grinshpoun T. Max-Sum Goes Private[C]. Proceedings of the International Joint Conference on Artificial Intelligence, 2015, 1360: 425-431.
- [121] Tassa T, Grinshpoun T, Zivan R. Privacy preserving implementation of the Max-Sum algorithm and its variants[J]. Journal of Artificial Intelligence Research, 2017, 59: 311-349.
- [122] Stranders R, Farinelli A, Rogers A, et al. Decentralised coordination of mobile sensors using the Max-sum algorithm[C]. Proceedings of the 21st International Joint Conference on Artificial Intelligence, 2009: 299-304.
- [123] Chen Z, Jiang X, Deng Y, et al. A generic approach to accelerating belief propagation based incomplete algorithms for DCOPs via a branch-and-bound technique[C]. Proceedings of the 33th AAAI Conference on Artificial Intelligence, 2019: 6038-6045.
- [124] Macarthur K S, Stranders R, Ramchurn S, et al. A distributed anytime algorithm for dynamic task allocation in multi-agent systems[C]. Proceedings of the 25th AAAI Conference on Artificial Intelligence, 2011: 701-706.
- [125] Kim Y, Lesser V. Improved Max-sum algorithm for DCOP with n-ary constraints[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent systems, 2013: 191-198.
- [126] Khan M M, Tran-Thanh L, Jennings N R. A generic domain pruning technique for GDL-based DCOP algorithms in cooperative multiagent systems[C]. Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems, 2018: 1595-1603.
- [127] Khan M. Speeding up GDL-based distributed constraint optimization algorithms in cooperative multiagent systems[D]. University of Southampton, 2018.
- [128] Deng, Y., & An, B. Speeding Up Incomplete GDL-based Algorithms for Multi-agent Optimization with Dense Local Utilities[C]. Proceedings of the 29th International Joint Conference on Artificial Intelligence, 2020: 31-38
- [129] Ottens B, Dimitrakakis C, Faltings B. Duct: An upper confidence bound approach to distributed constraint optimization problems[C]. Proceedings of the 26th AAAI Conference on Artificial Intelligence, 2012: 528–533.

- [130] Ottens B, Dimitrakakis C, Faltings B. DUCT: An upper confidence bound approach to distributed constraint optimization problems[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2017, 8(5): 1-27.
- [131] Auer P, Cesa-Bianchi N, Fischer P. Finite-time analysis of the multiarmed bandit problem[J]. Machine learning, 2002, 47(2-3): 235-256.
- [132] Kocsis L, Szepesvári C. Bandit based monte-carlo planning[C]. Proceedings of the 17th European Conference on Machine Learning, 2006: 282-293.
- [133] Nguyen D T, Yeoh W, Lau H C. Distributed Gibbs: a memory-bounded sampling-based DCOP algorithm[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2013: 167-174.
- [134] Geman S, Geman D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images[J]. IEEE Transactions on pattern analysis and machine intelligence, 1984 (6): 721-741.
- [135] Nguyen D T, Yeoh W, Lau H C, et al. Distributed Gibbs: A linear-space sampling-based DCOP algorithm[J]. Journal of Artificial Intelligence Research, 2019, 64: 705-748.
- [136] Kumar A, Yeoh W, Zilberstein S. On message-passing, MAP estimation in graphical models and DCOPs[C]. Proceedings of the Distributed Constraint Reasoning Workshop, 2011: 57-70.
- [137] Ghosh S, Kumar A, Varakantham P. Probabilistic inference based message-passing for resource constrained DCOPs[C]. Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015: 411-417.
- [138] 贺利坚, 张伟. 基于约束图分片求解 DCOP 的 Agent 组织结构[J]. 计算机研究与发展, 2007, 44(3): 434-438.
- [139] 黄晶, 刘大有, 杨博等. 自组织分治求解分布式约束优化问题[J]. 计算机研究与发展, 2008, 45(11): 1831-1839.
- [140] Vinyals M, Pujol M, Rodriguez-Aguilar JA, Cerquides J. Divide-and-coordinate: DCOPs by agreement[C]. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, 10: 149-156.
- [141] Hatano D, Katsutoshi H. DeQED: an efficient divide-and-coordinate algorithm for DCOP[C]. In Proceedings of the 23rd International Joint Conference on Artificial Intelligence, 2013: 566-572.
- [142] Chen Z, Wu T, Deng Y, et al. An ant-based algorithm to solve distributed constraint optimization problems[C]. Proceedings of the AAAI Conference on Artificial Intelligence, 2018: 4654-4661.

- [143] Chen Z, Liu L, He J, Yu Z. A genetic algorithm based framework for local search algorithms for distributed constraint optimization problems[J]. Autonomous Agents and Multi-Agent Systems, 2020, 34(2): 41-75.
- [144] Mahmud, S., Choudhury, M., Khan, M., Tran-Thanh, L., & Jennings, N. R. (2019). AED: An Anytime Evolutionary DCOP Algorithm[C]. Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, 2020: 825-833.
- [145] Leite A R, Enembreck F. Using Collective Behavior of Coupled Oscillators for Solving DCOP[J]. Journal of Artificial Intelligence Research, 2019, 64: 987-1023.
- [146] Deng Y, Yu R, Wang X, et al. Neural Regret-Matching for Distributed Constraint Optimization Problems[C]. Proceedings of the 33st International Joint Conference on Artificial Intelligence, 2021:146-153.
- [147] Deng Y, Kong S, An B. Pretrained Cost Model for Distributed Constraint Optimization Problems[C]. Proceedings of the 36th AAAI Conference on Artificial Intelligence, 2022.
- [148] Greenstadt, Rachel, Jonathan P. Pearce, and Milind Tambe. Analysis of privacy loss in distributed constraint optimization[C]. Proceedings of the AAAI Conference on Artificial Intelligence, 2006, 6: 647-653.
- [149] Yokoo, M., Suzuki, K., & Hirayama, K. (2002, September). Secure distributed constraint satisfaction: Reaching agreement without revealing private information[C]. Proceedings of the International Conference on Principles and Practice of Constraint Programming, 2002: 387-401.
- [150] Maheswaran R T, Tambe M, Bowring E, et al. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling[C]. Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, 2004, 1: 310-317.
- [151] Zivan R, Parash T, Naveh Y. Applying Max-Sum to Asymmetric Distributed Constraint Optimization[C]. Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015: 432-439.
- [152] R. Zivan, T. Parash, L. Cohen-Lavi, Y. Naveh, Applying Max-sum to asymmetric distributed constraint optimization problems[J]: Autonomous Agents and Multi-Agent Systems, 2020, 34 (1): 1-29.
- [153] Wahbi M, Ezzahir R, Bessiere C, et al. Nogood-based asynchronous forward checking algorithms[J]. Constraints, 2013, 18(3): 404-433.

- [154] Matsui T, Matsuo H. A constraint based formalisation for distributed cooperative sensor resource allocation[J]. International Journal of Intelligent Information and Database Systems, 2010, 4(4): 307-321.
- [155] Monteiro T L, Pellenz M E, Penna M C, et al. Channel allocation algorithms for WLANs using distributed optimization[J]. AEU-International Journal of Electronics and Communications, 2012, 66(6): 480-490.
- [156] Chen J K, De Veciana G, Rappaport T S. Site-specific knowledge and interference measurement for improving frequency allocations in wireless networks[J]. IEEE Transactions on Vehicular Technology, 2008, 58(5): 2366-2377.
- [157] Barab ási A L, Albert R. Emergence of scaling in random networks[J]. Science, 1999, 286(5439): 509-512.

附录

A. 作者在攻读学位期间发表的论文目录：

- [1] Dingding Chen, Yanchen Deng, Ziyu Chen, Zhongshi He, Wenxing Zhang. HS-CAI: A Hybrid DCOP Algorithm via Combining Search with Context-based Inference[C]. In AAAI, 2020: 7087-7094. (CCFA 会议)
- [2] Dingding Chen, Yanchen Deng, Ziyu Chen, Zhongshi He, Wenxing Zhang. A Hybrid Tree-based Algorithm to Solve Asymmetric Distributed Constraint Optimization Problems[J]. Autonomous Agents and Multi-Agent Systems, 2020, 34(2): 1-42. (CCF B 期刊)
- [3] Dingding Chen, Ziyu Chen, Yanchen Deng, Zhongshi He, Lulu Wang. Inference-based complete algorithms for asymmetric distributed constraint problems[J]. Artificial Intelligence Review, 2022. (SCI 2 区)
- [4] Dingding Chen, Ziyu Chen, Zhongshi He, Junsong Gao, Zhizhuo Su. Learning Heuristic for WCSP through Deep Reinforce Learning[J]. Applied Intelligence, 2022: 1-20. (SCI 3 区)
- [5] Jie Wang, Dingding Chen, Ziyu Chen, Junsong Gao, Xiangshuang Liu. Completeness Matters: Towards Efficient Caching in Tree-based Synchronous Backtracking Search for DCOPs [C]. In CP, 2022. (CCF B 会议)
- [6] Xiangshuang Liu, Ziyu Chen, Dingding Chen, Junsong Gao. A Bound-Independent Pruning Technique to Speeding up Tree-Based Complete Search Algorithms for Distributed Constraint Optimization Problems[C]. In CP, 2021: 1-17. (CCF B 会议)
- [7] Haiyan Wang, Zhongshi He, Yiman He, Dingding Chen, Huang Yuanwen. Average-face-based virtual inpainting for severely damaged statues of Dazu Rock Carvings[J]. Journal of Cultural Heritage, 2019, 36: 40-50. (SCI 2 区, SSCI, A&HCI)
- [8] Haiyan Wang, Zhongshi He, Dingding Chen, Huang Yuanwen, He Yiman. Virtual Inpainting for Dazu Rock Carvings Based on a Sample Dataset[J]. Journal on Computing and Cultural Heritage, 2019, 12(3): 1-17. (SCI 4 区, A&HCI)

B. 作者在攻读学位期间取得的科研成果目录：

- [1] 主持重庆市研究生科研创新项目 (CYS17023)：大足石刻的虚拟修复研究。
- [2] 参与重庆市研究生科研创新项目 (CYS18047)：非对称分布式约束优化问题算法研究。
- [3] 参与重庆市前沿与应用基础研究(一般)项目 (cstc2017jcyjAX0030)：非对称分布式约束优化问题算法及其应用研究。

C. 学位论文数据集:

关键词		密级	中图分类号		
分布式约束优化问题; 非对称 分布式约束优化问题; 完备算 法; 完备搜索; 完备推理		公开	TP		
学位授予单位名称	学位授予单位代码	学位类别	学位级别		
重庆大学	10611	学术学位	博士		
论文题名	并列题名	论文语种			
基于推理的分布式约束优化 问题完备求解算法研究	无	中文			
作者姓名	陈定定	学号	20171401006		
培养单位名称	培养单位代码				
重庆大学	10611				
学科专业	研究方向	学制	学位授予年		
计算机科学与技术	分布式人工智能	3	2022		
论文提交日期	2022 年 9 月	论文总页数	132		
导师姓名	何中市	职称	教授		
答辩委员会主席		李传东 教授			
电子版论文提交格式					
文本 <input checked="" type="checkbox"/> 图像 <input type="checkbox"/> 视频 <input type="checkbox"/> 音频 <input type="checkbox"/> 多媒体 <input type="checkbox"/> 其他 <input type="checkbox"/>					

致 谢

七年的硕士和博士生涯即将结束。回首这些年的经历，感慨万千：研究入门时的迷茫，论文读不懂时的焦虑，思路迸发时的激情澎湃，代码复现和论文书写受阻时的沉闷和自我怀疑，论文投稿后等待结果的紧张，和论文接收后的喜悦。这一切的酸甜苦辣咸，已成为我生命中最重要的一部分，激励着我不断前行。谨以此文感谢这一期间，一直关心我、帮助我的人。

首先，我要感谢我的恩师，何中市教授。没有何老师接收我入门，就没有我这一时期的成长和蜕变。“踏踏实实做人，实实在在做事”，何老师对学生的这一培养准则早已通过耳濡目染，慢慢地渐入我内心深处，成为我的人生信条。从研究生入学时，“壁画分类识别”项目中对我工程实践以及与人合作方面的指导；初期，让我加入“人脸识别”项目，对我动手和前沿科学研究探索能力的锻炼；中期，让我加入机器学习团队“大足石刻数字化修复”项目，您的指导与团队协作，激发我对科研的热情和兴趣，让我体会到团队合作在科研方面取得“大的突破”所获得的成就感。感谢您，在研二末，继续接收我，让我有机会在“科研探索之路”上继续前行。在博士初期，让我自由探索研究方向、选择自己要做的方向；之后的科研实践方面，对于文献的整理、思路的梳理、论文的书写、算法公式的规范表达，都给予我极大的帮助。在生活最困难期间，何老师也给予我极大的帮助；在我人生最迷茫的时候，总会站在我角度，给予一些看法和建议，帮助我走出低谷。何老师严肃的科学态度、深厚的理论功底、渊博的学识、无私的奉献精神、精益求精的工作作风和儒雅的为人处世风格，让我受益匪浅。

其次，我要感谢陈自郁老师。感谢您在我博士初期科研选题感到迷茫时，给予我无私的开导，接收我，让我能将“分布式约束优化完备求解算法研究”作为我的研究方向。之后，在学术与科研上给予我很大的帮助。在论文的复现、思路的探索与整理、论文的写作上，您的严谨、认真、执着、勇于探索、追求真理、严格要求自我的精神，一直影响着我对待科研、以及做人做事的态度上。也感谢您在这一期间，在学术和生活上的帮助和鼓励，让我总能在失意中重新找到自己，全心全意投入到科研中，让我有勇气去攀登学术的高峰。

我也要感谢机器学习研究组其他教师：邹东升老师、何静媛老师和伍星老师。你们在研究和生活方面，都给予了我大量的帮助和启发，让我能够开阔视野，活跃思维。

我要感谢王海燕师姐。最初的接触始于“大足石刻数字化修复”项目，和您合作的期间是我研究生期间最具成就感的阶段之一。您的“文科”思维方式，为我

打开宁一扇认知世界的窗户。您和善、平易近人的待人处世方式，令我印象深刻。您对“大足石刻修复”的热情以及专业的知识储备，让我佩服和敬仰。也是您的支持，让我有勇气向何老师申请“攻读博士学位”，才有了我之后在学术和科研上探索。在您身上，我学到了很多计算机专业之外的知识，也改变了我很多对世界的认知和人生的态度。

我要感谢一直陪伴在身边的战友，邓衍晨。你的学识、专注做事以及对人包容的态度，让我影响深刻，是我想要追寻的目标。有缘能在博士期间一起在“分布式约束优化”这一方向上耕耘，才有一直以来“斗志昂扬”的思路探讨、分享与合作。这种科研方面的合作，也让我们在日常生活的细节、人生态度和理想与追求上有了更多的探讨。对于科研上遇到的难题，你总能看到其中最关键的点，然后给予耐心的指导和极具建设性的意见。和你的每一次学术探讨，都能给我极大的触动和帮助，让我在科研上有不断努力和前行的动力。

感谢机器学习研究组的祝华正师兄、贾媛媛师姐、郭飞师兄、铉静师姐、石美凤师姐、李英豪师兄、张航师兄、周泽寻师兄、杜井龙师兄、王璐璐师妹等。你们的热心和包容让我感到无比温暖。无论是在研究工作中还是在日常生活中，你们都给予了我大量的帮助和照顾，正是有你们的陪伴，我才能够充满信心，坚定信念，度过一个个难关，一步一步地前进。感谢研究生同门刘智鹏、姜伶伶，研究组的师弟师妹王翰林、贾晓光、傅俞、张文昕、刘力贞、王力珩、刘向爽、张子旭、乔小雨、李维佳、张辰交、何伟东、江雪聪、林雨婷、张婉琪、陈伟、高俊松、黄锦辉、王杰、唐源强、王惊涛、吴光磊，感谢你们对我的帮助、包容和理解。感谢王臣、赵毅、苏智卓等室友的陪伴与鼓励。

感谢实习期间，给予我帮助和鼓励的导师刘少腾、郑若斌，同事刘磊、陈祥玉、田甄。

感谢谭依依，在我最失落的时候给我安慰和支持，帮助我走出低谷。

感谢我的家人对我的关心和照顾。特别是我的父母，您们含辛茹苦地抚养我、培养我、默默地支持我。感谢你们的无私关爱，祝你们健康快乐。

感谢参加我博士论文预答辩，并提出宝贵意见的周尚波教授、冯亮教授、钟将教授、李佳教授！

在此，向所有关心和帮助过我的领导、老师、同学和朋友表示由衷的谢意！

衷心地感谢在百忙之中评阅论文和参加答辩的各位专家、教授！

陈定定

二〇二二年八月 于重庆