求解分布式约束优化问题的推理算法研究

重庆大学硕士学位论文

（学术学位）

学生姓名：邓衍晨

指导教师：陈自郁 讲师

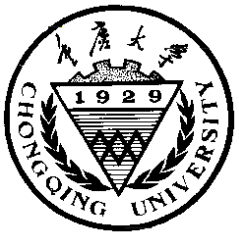
专　　业：计算机科学与技术

学科门类：工　学

重庆大学计算机学院

二〇一八年四月

The Study on Inference-based Algorithms for Distributed Constraint Optimization Problems



A Thesis Submitted to Chongqing University

in Partial Fulfillment of the Requirement for the

Master's Degree of Engineering

By Deng Yanchen

Supervised by Lec. Chen Ziyu

Specialty: Computer Science and Technology

College of Computer Science of

Chongqing University, Chongqing, China.

April 2018

摘　要

分布式约束优化问题（DCOP）是多智能体系统（MAS）的基本框架，是对分布式问题解决、多智能体协作的重要建模方式，现已成功应用于任务调度、电力系统等领域。非对称分布式约束优化问题（ADCOP）在 DCOP 的基础上增加了 Agent 的私有偏好，具有更强的建模的能力和更大的应用前景。以最大和算法（Max-sum）为代表的推理算法作为求解 DCOP/ADCOP 的重要手段，广泛地应用于各种实际场景中。然而，现有的非完备推理算法普遍存在着难以收敛、解的质量较差的问题。此外，由于 ADCOP 对隐私性的要求，传统用于求解 DCOP 的完备推理算法无法直接用于 ADCOP，而现有的求解 ADCOP 的完备搜索算法普遍存在着求解问题规模较小、隐私性较差等问题。

针对以上问题，本文拟从求解 DCOP 的非完备推理算法和求解 ADCOP 的完备算法开展研究。具体研究内容如下：

① 深入分析了值传播机制 Max-sum 类算法的影响。本文从理论上证明了虽然值传播可以极大地提高算法的性能，但是其同时阻碍了 Max-sum 类算法的信念传播。特别地，本文证明了当在换向有向无环图上连续执行值传播机制时，智能体将完全无法利用全局累加的信念，因此算法将等价于一个顺序的贪心局部搜索算法。由此，提出了在值传播机制下如何有效地平衡探索与利用这一重要的科学问题。

② 为了解决上述问题，本文提出了一系列基于非连续值传播的 Max-sum 类算法，包括基于单向值传播的 Max-sum_AD 算法（Max-sum_ADSSVP），基于混合信念/值传播的 Max-sum 算法（Max-sum_HBVP）和基于概率值传播的 Max-sum_AD 算法。这些算法通过打破在有向无环图上反复执行值传播这一桎梏，使得智能体在作出决策时可以同时兼顾个体利益和全体利益。本文还从理论上说明了上述算法不会等价于贪心的局部搜索算法，并分析了其时空复杂度。实验结果表明，上述算法显著优于传统的非完备推理算法，且对值传播机制开始启用的时机不敏感。

③ 充分考虑了 ADCOP 的特点，针对 ADCOP 对隐私性的要求，创新性地提出了一种基于搜索-推理混合的完备算法 PT-ISBB。该算法利用完备推理算法速度较快这一优势，预先求解一面的约束，并将推理结果存储；在搜索阶段，利用伪树中不同分支间相互独立这一事实，不断将问题划分为更小的子问题。在每一个节点上，都使用子树的推理结果作为取值的下界，以实现高效率的剪枝。本文同时在理论上证明了其完备性，并分析了其复杂度。实验结果表明，PT-ISBB 在多

个测试问题上均优于传统的搜索算法。

关键词：分布式约束优化问题，非对称约束优化问题，推理算法，最大和，值传播，伪树，完备搜索算法

ABSTRACT

Distributed Constrained Optimization Problems (DCOP) is the fundamental framework of Multi-Agent System (MAS), which plays an important role in modeling distributed problem solving and multi-agent coordination. DCOP has been successfully applied to task scheduling, power system and other fields. Asymmetric Distributed Constraint Optimization Problems (ADCOP) extends DCOP by adding private preferences for each agent. Thus it has stronger modeling capabilities and greater application prospects. Inference-based algorithms, represented by the Max-sum algorithm, are important techniques for solving DCOP/ADCOP and are widely applied into many real applications. However, the existing incomplete inference-based algorithms usually fail to converge and cannot produce high-quality solutions. In addition, due to the privacy concerns in ADCOP, the traditional inference-based complete algorithms for DCOP cannot directly solve ADCOP. On the other hand, the existing complete search-based algorithms for ADCOP cannot solve large scale problems and usually leak a lot of private information.

Against the above background, this paper aims to study the incomplete inference-base algorithms for solving DCOP and complete algorithms for solving ADCOP. The main contributions of the paper are listed as follows:

① The paper provides substantial analysis of the impact of the value propagation mechanism to Max-sum. We theoretically show that although value propagation can greatly improve the performance of Max-sum, it also blocks the belief propagation of the Max-sum. In particular, we prove that when the value propagation is continuously performed on the alternated directed acyclic graphs, the agent cannot utilize the global accumulated beliefs and the algorithm will be equivalent to a sequential greedy local search algorithm. Thus, how to efficiently balance exploration and exploitation after enabling value propagation becomes an ugent need.

② To solve the above problem, the paper proposes a class of Max-sum algorithms based on non-consecutive value propagation strategies, including Max-sum_AD with Single-side Value Propagation(Max-sum_ADSSVP), Max-sum with hybrid belief/value propagation (Max-sum_HBVP) and Max-sum_AD with probabilistic value propagation (Max-sum_ADPVP). By avoiding to continuously perform value propagation on directed acyclic graphs, these algorithms allow agents to

consider both individual benefits and the global benefit when making decisions. We then theoretically show that the above algorithms cannot be equivalent to a greedy local search algorithm, and analyze their space-time complexity. The experimental results demonstrate that our proposed algorithms are significantly superior to the traditional incomplete inference-based algorithms, and are less sensitive to the timing of starting value propagation.

③ Considering the privacy requirement of ADCOP, we propose a novel algorithm called PT-ISBB that is based on both search strategy and inference strategy. The algorithm first employs a complete inference-based algorithm to solve constraints in a direction. In the search phase, a problem is divided into smaller sub-problems in every branch node, and each sub-problem can be solved independently. On each node, the subtrees' inference results are used as a lower bound to accelerate pruning. We then prove its completeness theoretically and analyze its complexity. Experimental results show that PT-ISBB outperforms traditional search algorithms on various benchmarks.

Keywords: Distributed Constraint Optimization Problems, Asymmetric Distributed Constraint Optimization Problems, Inference algorithm, Max-sum, Value propagation, Pseudo-tree, Complete search algorithm

目　录

1 绪论

1.1 研究背景、目的及意义

多智能体系统（Multi-agent System，MAS）[1]是由多个可以互相交互的自治智能体（Agent）所组成的计算系统。在一个多智能体系统中，智能体可以相互协作去完成一个共同目标（如流水线组装任务），也可以相互竞争来使得个体利益最大化（如围棋）。由于它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，多智能体系统已经成为分布式人工智能的重要分支。多智能体系统体现了人类的社会智能，其本身也已经成为决策理论、博弈论及约束规划等领域的研究热点。

分布式约束优化问题（Distributed Constraint Optimization Problems, DCOPs）[2]作为多智能体系统协作问题的一个重要框架，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际价值。在 DCOP 中，每个智能体控制一组变量；全局目标函数分布在变量间的约束关系上，且每个智能体仅能知道关于自己所控制变量上的约束关系。智能体需要通过互相协作为其所控制的变量赋值，使得全局函数最优（如使得所有约束代价之和最小）。与传统的集中式优化不同，DCOP 更加强调通过局部交互获得全局最优。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网控制[3]、资源分配[4]、任务调度[5]和传感器网络[6,7]等领域的建模。

然而，DCOP 的对称性限制了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。所谓的对称性是指智能体与与其有约束关系的特征、值域空间及智能体的约束代价函数有完全且准确的信息。即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个体之间没有隐私性。因此，对于非对称或具有偏好特征的问题，用 DCOP 建模和求解将无法满足实际需求。非对称分布式约束优化问题（Asymmetric Distributed Constraint Optimization Problems，ADCOPs）[8]是在 DCOP 的基础上增加了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，给求解算法带来了挑战。例如更复杂的搜索空间、求解时的私密性保障等等。

随着 DCOP 研究的深入，近年来越来越多优秀的求解算法被提出。其中，推理算法作为求解 DCOP 的重要技术，成功地被应用在多种实际问题中。其中，最

大和算法（Max-sum）[9]是非完备推理算法的典型代表。但是，Max-sum 算法仅在树形结构问题上保证收敛性，并且在求解有环问题时无法得到较好的解。尽管学者们针对其收敛性和解的质量提出了一系列基于值传播的改进算法[10]，但它们在提高解的质量的同时，也带来了探索与利用间权衡的问题。因此，研究如何有效地 Max-sum 算法中的平衡探索与利用，不仅可以进一步提高 Max-sum 算法的求解质量，同时也为解决强化学习[11]中探索-利用困境提供新的思路。此外，由于 ADCOP 对隐私性的要求，传统的完备推理算法无法直接用于求解 ADCOP，而现有基于搜索的完备算法普遍存在着消息数过多、隐私性差等问题。因此，研究如何利用推理算法帮助搜索算法剪枝，进而设计具有更好隐私性的搜索-推理混合求解算法对于 ADCOP 的求解理论和通用搜索算法的设计都有着重要的学术意义和实用价值。

1.2 国内外研究现状

1.2.1 DCOP 求解算法研究现状

DCOP 求解算法近年来得到了长足的发展，图 1.1 给出了 DCOP 求解算法的大致分类。

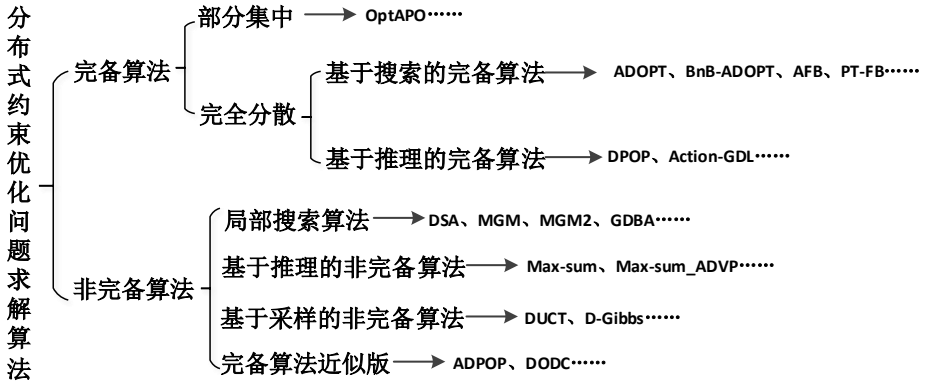


图 1.1 DCOP 求解算法分类

Fig. 1.1 Taxonomy of DCOP algorithms

根据能否能够保证求得最优解，DCOP 求解算法大体上可分为完备算法和非完备算法。其中，完备算法又分为部分集中式和完全分散式算法。OptAPO[12]是一种常见的采用部分集中思想求解 DCOP 的算法，通过将问题划分，在每个子问题中用分支定界法得到最优解，并通过选择某一个 Agent 作为调停者来解决重叠区域的子问题。由于部分集中方法需要 Agent 收集其他 Agent 的值域和约束函数信息，这样造成了信息隐私性方面的损失。从另一方面来说，部分集中式算法中

仅有一些节点在工作，使得算法不能充分利用多智能体系统的计算资源。因此，完全分散式算法成为研究的热点。其中，基于搜索的完备算法旨在通过解空间进行不重、不漏的遍历来获得最优解。这就要求算法在求解前首先对解空间进行有序地排列，这种排列反应到智能体间的具体交互上就是通信结构。同步分枝定界算法（Synchronous Branch and Bound，SBB）[13]就是一种早期的基于链式通信结构的完备搜索算法，是分枝定界算法的分布式实现。SBB 算法通过在链式结构上对解空间进行展开，并完成完整解代价的累积，且不断地通过回溯收紧全局解代价的上界。通过该上界实现对解空间的剪枝，最终找到最优解。异步前向定界算法（Asynchronous Forward Bounding，AFB）[14]是对 SBB 算法的异步改进版。它通过引入局部最优代价作为下界有效地提高了算法剪枝的效率。此外，它还在搜索过程中引入了时间戳机制，可以使得算法更早的放弃次优解。然而在链式结构中，智能体只能顺序地串行计算，即在每个时刻仅有一个智能体在计算。这极大地限制了算法的并行性，也对多智能体系统的计算资源造成了极大的浪费。相比之下，树形通信结构具有更低的搜索深度和更高的并发性，相对于链式结构极大的优势。因此，学者们提出了一系列基于树形结构的完备搜索算法。

ADOPT 算法[15]是基于深度优先伪树结构最重要的完备搜索算法。在算法中，智能体自上而下地进行异步搜索。对于树形结构的每一个子问题，算法使用最佳优先搜索策略（Best-first Strategy）和分布式回溯的方式不断重构当前解，并更新上下界，直到搜索到最优解。为了避免频繁重构解带来的额外消息，学者们相继提出了 ADOPT+[16]、ADOPT-k[17]、BnB-ADOPT[18]等改进算法。近期，Omer 等人提出了一种基于伪树结构的前向界定算法 PT-FB[19]。该算法利用伪树中不同分支相互独立这一事实，不断地将问题划分为更小的子问题，而后并行地用 AFB 算法并行地求解，极大地提高了问题的求解效率。然而，该算法仍然存在着消息量大、打破了 DCOP 只能局部通信的限制等诸多问题。

不同于完备搜索算法中显式地遍历解空间，完备推理算法则是通过对约束关系进行消元来求得最优解。分布式伪树优化过程（Distributed Pseudo-tree Optimization Procedure，DPOP）[20]是基于伪树的一个典型的完备推理算法，它是桶消除算法（Bucket Elimination）[21]的分布式实现。在该算法中，智能体利用动态规划技术自下而上地计算并传播赋值组合对应的效用（Utility）并自上而下地传播赋值信息。相比于完备搜索算法，DPOP 算法的消息数仅随着智能体数线性增长，但其消息大小和内存消耗则随着伪树的诱导宽度呈指数级增长。为了减少内存消耗，相继提出了 MB-DPOP[22]、ODPOP[23]等算法。此外，Action_GDL[24]是另外一种基于广义分配率（Generalized Distribution Law，GDL）[25]的完备推理算法，它是 DPOP 算法在分布式联合树上的泛化。

由于 DCOP 是 NP-Hard 问题，随着问题规模的增加，这些完备算法的求解代价（如消息数、消息大小等）将呈指数级增长。这就限制了此类算法在真实场景下的应用。与完备算法不同，非完备算法可以以较小的计算代价得到一个较优的解。这使得非完备算法可以广泛应用于求解大规模的实际问题，并成为当前 DCOP 求解算法研究的热点。

当前非完备算法主要包括基于决策的局部搜索算法、基于采样的近似算法和基于消息传递的推理算法三大类。其中，基于决策的局部搜索算法如分布式随机算法（Distributed Stochastic Algorithm，DSA）[6]、最大增益消息算法（Maximal Gain Message，MGM）[26]、MGM2[26]等是当前研究较为活跃的非完备算法。在这些算法中，智能体不断地与其邻居交换自身的状态信息（如取值、增益等），然后根据邻居的状态决定其下一轮的赋值。它们之间的主要区别在于赋值改变的方式。例如在 DSA 中，智能体依概率改变自己的赋值，而在 MGM 中，只有在所有邻居中增益最大的智能体改变自己的赋值。最近，Okamoto 等人提出了泛化分布式打破算法（Generalized Distributed Breakout Algorithm，GDBA）[27]。该算法通过规定计算有效代价的方式、约束违反的定义和修饰矩阵的变化范围等，将原本用于求解分布式约束满足问题（Distributed Constraint Satisfaction Problems，DCSPs）[28]的 DBA[29]算法扩展到求解 DCOP，取得了较好的效果。由于局部搜索算法具有类似的结构，为了更好地分析和比较各种算法，Chapman 等人提出了一种面向局部搜索算法的统一框架[30]。在这个框架中，一个局部搜索算法被抽象为 3 个阶段，即状态评估、决策准则和调度调整。为了提高局部搜索算法收敛时解的质量，Pearce 等人提出了 KOPT 算法[31]。该算法通过协调 k 个智能体组成的联合体的赋值来保证解的质量。具体地说，联合体中的每一个智能体将自己所涉及到的约束传导联合体的调停者，而后该调停者通过集中式的完备搜索为各个智能体计算最优的取值。最终算法会收敛到 k 优（k-optimal）状态[32]，即任意改变 k 个或者更少变量的取值都无法提高解的质量。显然，k 越大则解的质量越好，但相应的求解代价也越大。特别地，完备算法是 N 优的，其中 N 是变量的个数。此外，因为当全局最优时无法保证每一个智能体的本地利益是最优的，所以很多局部搜索算法无法返回其在求解过程中所探索的最优解（即算法不具备 Anytime 性质[33]）。因此，Zivan 等人提出了用于保证局部搜索算法的 Anytime 性质的 ALS（Anytime Local Search）框架[34]。该框架通过一个宽度优先树（Breadth-first Tree）收集各个智能体的局部代价来固定当前探索到的最优解。由于局部搜索算法仅依赖邻居的状态确定下一回合的赋值，它们很容易陷入局部最优。Yu 等人提出了部分决策框架（Partial Decision Scheme）[35]。该框架通过忽略邻居的取值来扩大智能体的决策空间，进而打破算法的局部最优。

基于采样的近似算法是根据统计推断对搜索空间的采样来逼近最优解。分布式 UCT（Distributed Upper Confidence Tree，DUCT）[36]算法是一个典型的基于采样的近似算法。在该算法中，每个智能体为每一个取值构造一个置信界，然后选择置信界最小的取值。其中，每一个置信界都是其对应取值在当前上文下对其收益的乐观的估计。然而，由于智能体需要为每一个上文构造若干置信界，DUCT 算法对内存的占用是指数级的，这就使得该算法不能用于大规模的实际问题。为此，Nguyen 等人提出了一种线性内存占用的采样算法，即分布式 Gibbs（Distributed Gibbs，D-Gibbs）算法[37]。该算法通过将 DCOP 映射成为极大似然估计（Maximum Likelihood Estimation，MLE）问题，并用分布式的 Gibbs 算法求解该问题。

最大和（Max-sum）[9]算法是一个重要的基于推理的非完备算法。该算法通过在因子图（Factor graph）[25]上传播和累积信念（Belief）来实现全局效用函数的边际化。具体地，每个智能体对于其每一个可能的取值都维护着一个信念，该信念反映了取该值时的效用或者收益，并且这些信念会根据邻居发来的消息进行实时更新。当智能体做决策时，它直接选取效用最大的取值。然而，该算法仅在求解树形结构问题时保证收敛，且通常无法在有环问题上取得较好的解。

有界最大和（Bounded Max-sum，BMS）算法[38]通过去除有环因子图中多余的边来解决上述问题。具体地，算法首先把对解的质量影响较小的二元关系通过对变量取极小转换为一元函数，最终达到一个近似问题。这个过程被称为松弛（Relaxation）阶段。因此，算法就将一个有环因子图转换成为一棵树，再用 Max-sum 算法求解该近似问题，并给出最优解近似比的上界（称为限界阶段）。然而，BMS 给出的近似比的上界通常过大，这反映出其置信度不高。为此提出了改进的 BMS（Improved BMS，IBMS）算法[39]。与 BMS 算法不同，该算法在松弛阶段不仅产生一个最小化的近似问题，同时也产生一个极大化的近似问题。在限界阶段，该算法使用两个问题中求解质量最好的去计算近似比。由于在松弛阶段产生近似问题时会引入一定的误差，上述的两种 BMS 算法均无法取得较好的效果。Rollon 等人提出将多元函数精确或者近似地分解成多个一元函数。他们据此提出的精确分解 IBMS（Exact Decomposition IBMS，ED-IBMS）算法和近似分解（Approximate Decomposition IBMS，AD-IBMS）算法相对于原始的 IBMS 算法取得了良好的效果[40]。

不同于 BMS 类算法通过移除多余的边来使得问题无环，换向有向无环图最大和（Max-sum on Alternating Directed acyclic graph，Max-sum_AD）[10]算法通过将有环因子图转换成为一有向无环图使得算法收敛。具体地，与 Max-sum 算法中智能体向所有邻居发送消息的做法不同，Max-sum_AD 中的智能体只向排在其后面的邻居发送消息。可以证明，该算法可以在线性迭代轮次后达到单向收敛状态，

即所谓的单阶段收敛（Single Phase Convergence，SPC）。为了可以充分利用两侧的约束关系，当算法单侧收敛后，算法将消息传播的顺序反向后继续进行消息传播，如此循环直到算法结束。然而，效用相同（Utility ties）和无效取值假设使得 Max-sum_AD 无法得到高质量的解。为此，Zivan 等人提出利用两阶段的值传播来打破效用重复并消除无效取值假设的问题，即带值传播的 Max-sum_AD 算法（Max-sum_AD with value propagation，Max-sum_ADVP）[10]。具体地说，变量节点（Variable node）不仅传播效用，也传播其当前取值。函数节点（Function node）在计算消息时，考虑其所收到的取值信息。相比于 Max-sum_AD，Max-sum_ADVP 更加倾向于利用且可以保证跨阶段收敛（Cross Phase Convergence，CPC）。但是，尽管它能极大地提高 Max-sum_AD 的解的质量，值传播同时也限制了算法的探索能力。因此，一些增强探索性的方法被引入来缓解该问题。这些方法大体上可以分为基于选值的探索方法和基于消息的探索方法。前者通过在根据信念选值时引入随机性来避免陷入局部最优，典型的方法如 K 邻居（K neighbors）、K 深度（K depth）、递减偏置（Decreasing bias）。模拟退火等[10]。后者是通过利用不同类型的 Max-sum 算法的消息来起到平衡探索和利用的目的。然而，这些方法的效果不甚理想，仅基于消息的探索方法略微优于 Max-sum_ADVP。

近期，Cohen 等人[41]研究了当求解 DCOP 时，消息衰减对 Max-sum 性能的影响。消息衰减技术通过降低循环消息传播的效应来增加信念传播在有环因子图中收敛的可能性。实验结果表明，Max-sum 在高衰减因子及 Anythime 机制下可以得到明显优于局部搜索算法和其他版本 Max-sum 算法的解。

1.2.2 ADCOP 算法研究现状

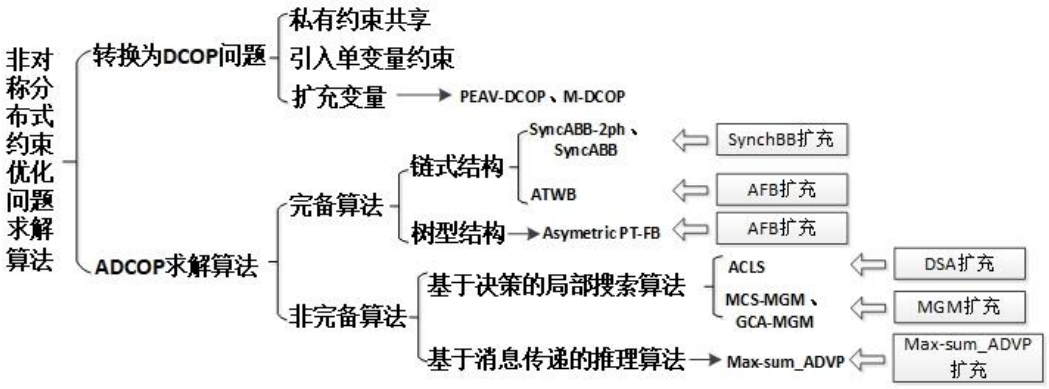


图 1.2 ADCOP 求解算法分类

Fig. 1.2 Taxonomy of ADCOP algorithms

ADCOP 求解算法的研究尚处于起步阶段，相应的成果不多。目前求解 ADCOP 的主要思路是模型转换和直接求解 ADCOP，其大致分类如图 1.2 所示。模型转换

是指将 ADCOP 直接转换为 DCOP 后利用 DCOP 算法求解。私有约束共享是最直接的模型转换方式，它通过共享约束各方的私有约束函数来消除非对称的特性。但这种方式会极大地泄露隐私[42-44]，不符合 ADCOP 对求解算法的要求。另一种可能的模型转换方式是引入一元约束，即将一个非对称的约束关系用一个对称约束关系和若干一元约束关系表示。然而，可以证明存在一种 ADCOP 无法通过上述方法转换为 DCOP，亦即该方法的表示能力不足以表示非对称关系。目前比较成熟的模型转换方式是私有事件作为变量（Private Events As Variables，PEAV）[45]。该方式通过在 Agent 中增加虚拟变量来映射邻居 Agent 的变量，采用修改约束函数或引入可信任的中心结点处理私有信息。这种方式显然增加了建模的难度和复杂度，不适合大规模问题。

因此，直接求解 ADCOP 成为 ADCOP 算法研究的主流。现有的 ADCOP 完备算法均是基于搜索的算法。其中，两阶段非对称 SBB 算法（SyncABB-2ph）[46]和非对称 SBB 算法（SyncAB）[46]是在 SBB 的基础上引入阶段检测和反向检测的回溯搜索算法。双向界定算法（Asymmetric Two Ways Bounding，ATWB）[46]则加入异步机制，是 AFB 算法的扩充。近期，Omer 等人提出利用暴露父节点代价的方式实现用 PT-FB 算法求解 ADCOP，即 AsymPT-FB[19]。该方法可以保证隐私泄露量不超过 50%。然而，这些基于搜索的完备算法能求解的问题的规模较小，而可以求解更大规模问题的完备推理算法由于 ADCOP 隐私性的限制无法直接求解 ADCOP。**因此，如何用完备推理算法辅助求解 ADCOP 仍然是一个开放问题。**

ADCOP 非完备算法多使用基于决策的局部搜索算法。非对称协调局部搜索（Asymmetric Coordinated Local Search，ACLS）[46]算法在 DSA 的基础上，通过交换最佳响应的单面约束代价值实现协调决策；最小约束共享 MGM（Minimum Constraint Sharing，MCS-MGM）[46] 算法则基于 MGM 算法，通过不断更新邻居约束来指导决策。保证收敛的非对称 MGM（Guaranteed Convergence Asymmetric MGM）[46] 算法通过放宽 MCS-MGM 中交换约束代价的条件来保证算法的收敛性。这些基于局部搜索的 ADCOP 算法在求解过程中暴露了大量的隐私，且 ACLS 和 MCS-MGM 无法保证收敛性。Zivan 等人最近提出使用 Max-sum_ADVP 求解 ADCOP[47]，并证明了在特定因子图排列方式下，用 Max-sum_ADVP 求解 ADCOP 等价于求解消除非对称性质的对应 DCOP。

1.3 论文的主要研究内容和创新之处

1.3.1 论文的主要研究内容

本文从基于推理的 DCOP 求解算法入手，重点研究了以下科学问题：1) 如何通过对值传播进行细粒度的控制来有效的平衡 Max-sum 类算法的探索和利用；2)

如何利用推理算法加速求解 ADCOP 的完备搜索算法。本文的具体研究内容如下：

① 研究值传播对 Max-sum 类算法的影响。本文从理论上证明了值传播会限制 Max-sum 类算法的探索能力。换句话说，值传播会限制全局效用的传播和累积，最终使得智能体仅能够用本地信息作出决策。基于上述理论分析，本文提出了三种可以平衡探索与利用的非连续值传播 Max-sum 算法，即基于单向值传播的 Max-sum_AD（Max-sum_AD with Single-side Value Propagation，Max-sum_ADSSVP）算法、基于混合信念/值传播的 Max-sum（Max-sum with Hybrid Belief/Value Propagation，Max-sum_HBVP）算法和基于概率值传播的 Max-sum_AD 算法（Max-sum_AD with Probabilistic Value Propagation,Max-sum_ADPVP）算法。其中，Max-sum_ADSSVP 始终只在一个方向上做值传播，即当消息传播方向为正方向时，其执行信念传播来探索更有潜力的解，反之则执行值传播来保证解的质量。Max-sum_ADPVP 则是通过依概率进行值传播的方式平衡探索与利用。Max-sum_HBVP 通过同时执行值传播和信念传播来加速优化过程。本文给出了以上算法中变量节点的决策函数，并分析了其时空复杂度。

② 充分考虑 ADCOP 的非对称特性，研究了完备推理算法用于求解 ADCOP 的可能性。提出了一种基于混合推理-搜索的求解 ADCOP 的完备算法 PT-ISBB。该算法首先利用完备推理算法求解问题的一面的约束，然后用基于伪树的 SBB 算法求解整个问题。在算法的搜索阶段，利用深度优先伪树中不同分枝的搜索互相独立的性质，将 SBB 算法的搜索并行化，极大地提高了求解的效率。此外，智能体可以利用已经由推理算法求出的子树单面效用作为搜索的下界来实现提前剪枝，进一步提高了算法的整体性能。

1.3.2 论文的创新之处

① 尽管值传播已经广泛用于 Max-sum 类算法，但现有的文献仅停留在从实验上评估值传播的效果。**本文首次通过理论上的分析，指出值传播会限制 Max-sum 类算法的探索能力。**特别地，当值传播被应用在 Max-sum_AD 算法中时，算法将等价于一个顺序地贪心局部搜索算法。**由此，提出了当值传播应用在 Max-sum 类算法时的探索与利用问题。**

② 基于上述分析，提出了三种可以平衡探索与利用的基于值传播的 Max-sum 算法。本文同时分析了算法中变量节点决策函数，指出这些算法中的变量节点始终考虑全局利益和局部利益，因而不会等价于顺序的局部搜索算法。同时，实验结果表明本文提出的算法显著优于双向值传播（即纯利用）和没有值传播（即纯探索）的算法。

③ 由于 ADCOP 对隐私性的要求，传统的完备推理算法（如 DPOP）无法直接求解 ADCOP。本文创新性地提出了一种基于混合推理-搜索的两阶段的完备求

解算法。与传统的直接利用搜索算法求解不同，本文提出的算法利用推理算法求解速度快的优势，预先求解一面的约束。在正式进入搜索阶段后，这些由推理算法求解出的子树效用可以作为搜索算法的更紧的下界。**本文提出的算法是目前为止，完备推理算法首次应用于 ADCOP 求解。**

1.4 论文的组织结构

本文共分为 5 章，具体内容安排如下：

第一章，绪论。首先介绍了本文研究背景与研究意义。然后讲述了国内外研究现状，分析当前各个研究成果的优点和不足之处。最后介绍了本文的主要研究内容和创新之处，以及本文的组织结构。

第二章，分布式约束优化研究基础。给出了 DCOP 和 ADCOP 的形式化定义并介绍了 DCOP 算法测试的平台、标准化测试问题及算法性能指标。然后重点介绍了本文研究的算法基础，包括 DPOP 算法、Max-sum 类算法和 SBB 算法等。

第三章，基于非连续值传播的最大和算法。首先从理论上证明了值传播会限制 Max-sum 类算法的探索能力，并可以使得 Max-sum_AD 算法等价于一个顺序的贪心局部搜索算法。然后用一个例子说明了 Max-sum_ADVP 算法对初始取值（即开始值传播的时机）较为敏感且最终会陷入局部最优的问题。为了解决连续值传播所导致的探索-利用失衡的问题，本章提出了三个基于非连续值传播的 Max-sum 类算法，并在理论上分析了各算法中变量节点的决策函数，指出本章提出的方法不会等价于局部搜索算法。实验结果表明，本章所提出的算法显著优于 Max-sum_ADVP 且对开始值传播的时机具有较强的鲁棒性。

第四章，求解非对称约束优化问题的搜索-推理混合算法。首先分析了求解 ADCOP 的完备搜索算法泄露私有偏好的本质，指出提高算法隐私性的重要途径是提高算法的剪枝性能。基于上述分析，提出了一种搜索与推理相结合的求解 ADCOP 的完备算法 PT-ISBB。该算法通过利用完备推理算法预先求解一面约束，并作为搜索算法的下界来大幅提高算法的剪枝性能。然后给出了一个例子说明算法的工作流程。在理论分析中，证明了算法的完备性，并分析了其时空复杂度。实验结果表明，PT-ISBB 不仅在性能上优于其他完备搜索算法，在隐私性上也显著优于其他算法。

第五章，总结与展望。首先对本文的研究工作进行了概括和总结，并给出了未来可能的工作方向。

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

分布式约束优化问题（Distributed Constraint Optimization Problems，DCOPs）可以用一个四元组$\langle A,X,D,F\rangle$组成，其中：

① $A=\{a_1,...,a_n\}$是 Agent 的集合，一个 Agent 负责为一个或多个变量取值；

② $X=\{x_1,...,x_m\}$是变量的集合；

③ $D=\{D_1,...,D_m\}$是值域的集合，其中变量x_i从值域D_i中取值；

④ $F=\{f_1,...,f_q\}$是约束代价函数的集合，约束$f_i:D_{i_1}\times...\times D_{i_k}\rightarrow\mathbb{R}^+\cup\{0\}$是从任意$k$个变量$(x_{i_1},...,x_{i_k})$的赋值组合到一个非负代价的映射。

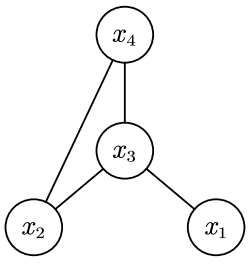
不失一般性地，DCOP 的一个解是一个对所有变量的赋值，使得所有约束代价之和最小。即：

$$X^*=\underset{X\in D}{\operatorname{argmin}}\sum_{f_i\in F}f_i \tag{2.1}$$

为了便于理解和讨论，我们假设一个 Agent 只控制一个变量且所有的约束关系都是二元关系。因此，在本文中 Agent，节点和变量可以被认为是同一个概念，且可以相互替换。二元约束关系是指仅涉及到两个变量的约束函数，即$f_{ij}:D_i\times D_j\rightarrow\mathbb{R}^+\cup\{0\}$。因此 DCOP 的解可以由式(2.2)定义。

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

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 Agent，一条边则代表一个约束关系。图 2.1 给出了一个 DCOP 的实例，其中左边是其约束图，右边是对应的约束矩阵。在这个实例中，每个变量的值域都是$\{0,1\}$，且当各个变量分别按$\{x_1=1,x_2=1.x_3=1,x_4=0\}$取值时可以达到最优解，其对应的全局代价是 13。



(a) 约束图

x_1 \ x_3	0	1
0	1	7
1	9	3

x_2 \ x_3	0	1
0	7	3
1	8	3

x_3 \ x_4	0	1
0	4	7
1	6	6

x_2 \ x_4	0	1
0	3	3
1	1	5

(b) 约束矩阵

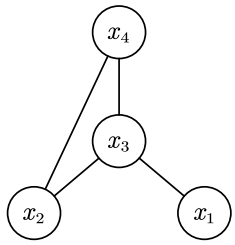
图 2.1 DCOP 实例

Fig. 2.1 A DCOP instance

非对称约束优化问题（ADCOP）在 DCOP 的基础上增加了个体偏好和隐私性的要求。即对于每一个约束关系，每个 Agent 都有自己的约束代价函数（偏好），且不希望该代价函数被其他 Agent 所知。因此，ADCOP 中的约束代价函数可以表示为：

$$f_i: D_{i_1} \times, \ldots \times, D_{i_k} \rightarrow \{\mathbb{R}^+ \cup \{0\}\}^k \tag{2.3}$$

即同一个约束关系为约束各方给出的代价值不同。ADCOP 的求解目标同样是使得所有约束代价之和最小。图 2.2 给出了一个 ADCOP 的实例。与 DCOP 不同，求解该 ADCOP 需要考虑每个 Agent 的私有约束代价函数，即找到使 8 个约束函数之和最优的变量赋值。



(a) 约束图

x_3 \ x_1	0	1
0	1	7
1	9	3

x_1 \ x_3	0	1
0	3	1
1	4	2

x_3 \ x_2	0	1
0	7	3
1	8	3

x_2 \ x_3	0	1
0	0	1
1	5	2

x_4 \ x_3	0	1
0	4	7
1	6	6

x_3 \ x_4	0	1
0	5	9
1	8	1

x_4 \ x_2	0	1
0	3	3
1	1	5

x_2 \ x_4	0	1
0	3	3
1	7	2

(b) 约束矩阵

图 2.2 ADCOP 实例

Fig. 2.2 An ADCOP instance

2.2 算法背景

2.2.1 DCOP 算法的通信结构

在 DCOP 完备算法中，为了能对解空间进行不重、不漏的遍历，必须对解空间进行排序。这种排序体现在 Agent 内部时就是对其值域搜索顺序的排序，而体现在 Agent 之间时就是通信结构。换句话说，通信结构就是一种特殊的、有序的约束图，各个 Agent 按照通信结构所规定的顺序收发消息和作出决策，完成对解空间的遍历。

常见的通信结构有：无结构、链式结构和伪树结构。无结构通常适用于那些不需要完整遍历解空间的非完备算法，如 DSA、MGM 等。此类算法一般没有全局信息，仅通过局部调整来优化全局目标。链式结构是指各个 Agent 按照某个优先级排成一条链，广泛地应用于早期的完备搜索算法中，如 SBB、AFB 等。由于在链式结构中，一个时刻仅有一个 Agent 在计算，因此其并发性较差，极大浪费了分布式系统的计算资源与优势。此外，根据问题的拓扑结构和优先级排序，一个 Agent 可能会与另外一个不相邻的 Agent 通信，这就打破了 DCOP 中利用局部

交互获得全局最优的理念，增加了隐私泄露的风险。

为了克服链式结构的上述问题，提出了基于伪树的通信结构。其通过对原约束图进行深度优先遍历将所有的约束边分为树边和伪边（即非树边）。需要指出的是，在一棵伪树中，一个 Agent 的各个树边分支是相互独立的，因此算法可以利用这一特性不断地将问题划分为更小的子问题来并行地求解。所以，很多完备算法都采用伪树作为其通信结构。图 2.3 给出了图 2.1 中约束图的一个可能的伪树结构，其中实线代表树边，虚线代表伪边。

为了更好地介绍背景算法和本文工作，下面以图 2.3 为例介绍伪树中的基本概念。

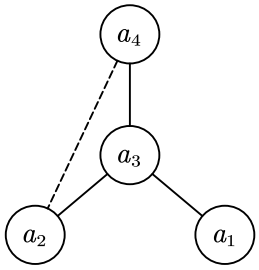


图 2.3 伪树结构实例

Fig. 2.3 A pseudo tree instance

① P_i：P_i是 Agent a_i的父节点（Parent），即与a_i通过树边直接相连的上层节点，如$P_3 = a_4$；

② PP_i：PP_i是a_i的所有伪父节点（Pseudo Parent）的集合。其中伪父节点是指与a_i通过伪边直接相连的上层节点，如$PP_2 = \{a_4\}$；

③ C_i：C_i是a_i的所有孩子节点（Child）的集合。其中孩子节点是指与a_i通过树边直接相连的下层节点，如$C_4 = \{a_3\}$；

④ PC_i：PC_i是a_i的所有伪孩子节点（Pseudo Child）的集合。其中伪孩子节点是指与a_i通过伪边直接相连的下层节点，如$PC_4 = \{a_2\}$。

⑤ N_i：N_i是a_i的所有邻居节点（Neighbor）的集合。其中邻居节点是指与a_i有约束关系的节点，如$N_4 = \{a_2, a_3\}$。

2.2.2 分布式伪树优化过程

分布式伪树优化过程（Distributed Pseudo tree Optimization Procedure，DPOP）是一种典型的基于动态规划的完备推理算法。该算法通过在伪树上进行一个自下而上的效用传播过程（UTIL Phase）和一个自上而下的值传播过程（VALUE Phase）实现问题的求解。在效用传播阶段，Agent 将其收到的子节点效用和自己本地约

束的效用联合后，消除自己的维度并将该消元后的效用继续向父节点传播，直至根节点。在值传播阶段，Agent 根据上层节点给出的取值和自己本地的效用确定自己的取值，并把该取值与上层节点的取值一起传播给子节点。与基于搜索的算法相比，DPOP 仅需要与 Agent 个数成线性关系的消息数即可完成问题的求解，因而可以求解更大规模的问题。为了更好地说明 DPOP 算法的原理，考虑以下定义：

定义 2.1（联合）：设函数f和g的定义域分别是DV_f和DV_g，则二者的联合定义为：

$$(f \otimes g)(d) = f(d_f) + g(d_g),\ d \in DV_{f \otimes g} \tag{2.4}$$

其中，$DV_{f \otimes g} = DV_f \times DV_g$是联合后函数的定义域空间；$d_f$和$d_g$分别是自变量$d$在$DV_f$和$DV_g$上的投影。

定义 2.2（消元）：设函数f是定义在变量集合$Dim_f = (x_1,\ldots,x_k)$的函数，则该函数针对第i维消元的定义为：

$$\underset{x_i}{\oplus} f = \min_{x_i} f(x_i, x_{-i}) \tag{2.5}$$

其中$x_{-i} = Dim_f \setminus \{x_i\}$。

在 DPOP 中，一个 Agent 的联合操作完成了其各个孩子节点效用及其本地约束的效用的合并，而消元操作则计算了其效用函数中所有上层节点的所有可能的赋值组合下其最佳的代价值。具体地，算法描述如下：

① 效用传播阶段。当一个 Agent a_i收到来自所有子节点的效用时，a_i将其与父亲节点和所有伪父亲节点的约束关系与来自子节点的效用联合，得到其本地的效用函数，最后对自身变量消元并向上传播消元后的效用。显然，在效用传播的过程中，联合操作和消元操作交替进行，且每经过一个 Agent 就会消除对应变量的维度。因此，当根节点收到所有分支的效用时（即效用传播阶段结束时），其联合后的本地效用是仅关于自身的一元函数。

② 值传播阶段。当效用传播阶段结束后，根节点根据其本地的效用确定自己的最优取值并向下传播。当收到来自父节点的值消息时，一个 Agent 会根据收到的值消息固定其本地效用函数中上层节点的取值，然后再找到一个可以使本地效用函数最优的取值作为取值，并将收到的值消息和自己的取值一起发送给子节点。当所有子节点收到值消息并取值后，算法结束。

下面以图 2.3 为例，给出 DPOP 执行的详细过程。

效用传播阶段：

$$x_2 \to x_3 : u_{2\to 3}(x_3, x_4) = \underset{x_2}{\oplus}(f_{23} \otimes f_{24})$$

$$x_1 \to x_3 : u_{1\to 3}(x_3) = \underset{x_1}{\oplus} f_{13}$$

$$x_3 \to x_4 : u_{3\to 4}(x_4) = \underset{x_3}{\oplus}(f_{34} \otimes u_{2\to 3}(x_3, x_4) \otimes u_{1\to 3}(x_3))$$

值传播阶段：

$x_4 \to x_3 : \{x_4 = x_4^*\}, x_4^* = \underset{d_4 \in D_4}{\operatorname{argmin}} u_{3\to 4}(d_4)$

$x_3 \to \{x_1, x_2\} : \{x_4 = x_4^*, x_3 = x_3^*\}, x_3^* = \underset{d_3 \in D_3}{\operatorname{argmin}} u_{2\to 3}(d_3, x_4^*) + u_{1\to 3}(d_3) + f_{34}(d_3, x_4^*)$

$x_2 : x_2^* = \underset{d_2 \in D_2}{\operatorname{argmin}} f_{24}(d_2, x_4^*) + f_{23}(d_2, x_3^*)$

$x_1 : x_1^* = \underset{d_1 \in D_1}{\operatorname{argmin}} f_{13}(d_1, x_3^*)$

2.2.3 最大和算法

最大和（Max-sum）算法是一种基于因子图的非完备推理算法。因子图是 DCOP 的一种二部图表示。在因子图中，变量节点对应 DCOP 的变量；函数节点对应 DCOP 中的约束关系，且一个变量节点仅和涉及到它的函数节点相连。图 2.4 给出了图 2.1 所示的 DCOP 实例所对应的因子图。

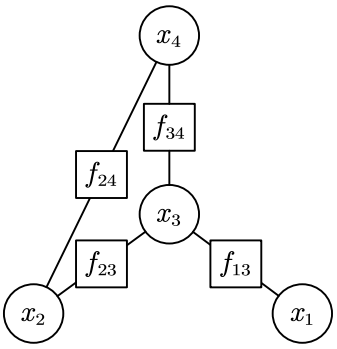


图 2.4 因子图实例

Fig. 2.4 A factor graph instance

不同于局部搜索算法中 Agent 通过本地约束和邻居状态进行决策，在 Max-sum 算法中变量节点根据其当前的信念完成决策。信念在因子图中的传播和更新是通过查询消息（Query message）和响应消息（Response Message）完成的。查询消息是指从变量节点到函数节点的消息。当一个变量节点计算查询消息时，它会将所有除了目标节点外的邻居节点发送的消息累加起来。具体地，式(2.6)给出了变量节点 x_i 发送给函数节点 f_j 的查询消息的计算方式。

$$q_{i\to j}(x_i) = \alpha_{ij} + \sum_{n \in N_i \setminus j} r_{n\to i}(x_i) \tag{2.6}$$

其中，$N_i \setminus j$ 是 x_i 的除了目标函数节点 f_j 的邻居下标的集合；$r_{n\to i}(x_i)$ 是从函数节点 f_n 发送给变量节点 x_i 的消息；α_{ij} 是一个正则化因子，其作用是避免消息在有环因子图中无限制地增大，其定义由式(2.7)给出。

$$\alpha_{ij} = -\frac{1}{|D_i|}\sum_{d_i \in D_i} q_{i\to j}(d_i) \tag{2.7}$$

响应消息是指从函数节点到变量节点的消息，其包含了在当前信念和本地函数下，对于目标变量节点的每一个取值所对应的最佳代价。在一般的 Max-sum 算法中，响应消息的计算过程分为加和过程（即 sum）和取极大（即 Max）过程。这里，为了与 DCOP 的最小化求解目标相统一，我们描述取极小的版本的算法（即 Min-sum）。在加和过程中，一个函数节点累加除目标变量节点外的所有节点发来的消息，得到当前的信念。在取极小过程中，变量节点对当前信念和本地函数的加和按除目标节点之外的变量取极小。式(2.8)给出了函数节点f_j给变量节点x_i的响应消息的计算方式

$$r_{j\to i}(x_i) = \min_{\mathbf{x}_j \setminus x_i}\left[f_j(\mathbf{x}_j) + \sum_{n \in N_j \setminus x_i} q_{n\to j}(x_n)\right] \tag{2.8}$$

其中，$\mathbf{x}_j \setminus i = \{x_k : k \in N_j \setminus i\}$是除了$x_i$外的所有$f_j$涉及到的变量的集合。

当一个变量节点做决策时，它首先通过累加来自所有邻居的消息去计算其每一个取值所对应的信念：

$$z_i(x_i) = \sum_{n \in N_i} r_{n\to i}(x_i) \tag{2.9}$$

然后根据该信念选择一个代价最小的赋值，即：

$$x_i^* = \underset{d_i \in D_i}{\operatorname{argmin}} z_i(d_i) \tag{2.10}$$

2.2.4 换向有向无环图最大和算法

事实上，Max-sum 算法仅在无环问题上保证收敛，在求解有环问题时，不能收敛且通常得不到高质量的解。为此，提出了换向有向无环图最大和算法（Max-sum on Alternating Directed acyclic graph，Max-sum_AD），其伪代码如算法 2.1 所示。该算法通过严格控制有环信息传播的这一事实来克服上述问题。具体地说，算法首先根据一个预先定义的优先级将有环因子图转换为一有向无环图，然后在该有向无环图上进行 Max-sum 的消息传递。换句话说，在 Max-sum_AD 算法中，所有节点仅向排在其后面的节点进行消息传递。可以证明，该算法将在线性于 Agent 个数的轮次后单向收敛，此时我们称为算法的一个阶段（Phase）。为了保证算法单向收敛，一般选择有向无环图中的最长路径的长度作为单向最大的迭代轮次。此外，为了可以利用另外一边的约束关系，当算法单向收敛后，该有向无环图反向并继续执行消息传递。上述过程反复执行直到算法结束。图 2.5 给出了图 2.4 的一个可能的有向无环图例子。

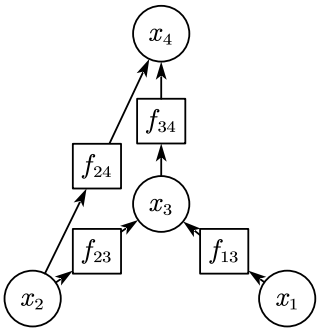


图 2.5 有向无环因子图实例

Fig. 2.5 A directed acyclic factor graph instance

需要指出的是，在计算每一条消息时，Max-sum_AD 算法像标准 Max-sum 算法一样，考虑全部的邻居，而不仅仅是上游的邻居。换句话说，Max-sum_AD 中的节点在产生消息时，始终考虑在本阶段产生的上游节点的消息和在上一阶段产生的下游节点消息。这就使得信念可以跨阶段传播，且 Agent 可以考虑全部的约束而不仅仅是一个方向上的约束。

算法 2.1 Max-sum_AD 的伪代码

Algorithm 2.1 Sketch of Max-sum_AD

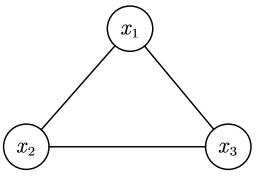
Max-sum_AD (node *n*)

1. $current\_order \leftarrow$ select an order on all nodes in the factor graph;
2. **While** *no termination condition is met* **do**
3. $N_{prev\_n} \leftarrow \{\hat{n} \in N_n : \hat{n}$ is ordered before n in $current\_order\}$;
4. $N_{follow\_n} \leftarrow N_n \setminus N_{prev\_n}$;
5. **for** *k iterations* **do**:
6. **foreach** $n' \in N_{follow\_n}$ **do**:
7. **if** *n is a variable node* **then**:
8. produce message $m_{n'}$ using Equation (2.6);
9. **else if** *n is a function node* **then**:
10. produce message $m_{n'}$ using Equation (2.8);
11. send $m_{n'}$ to n';
12. $current\_order \leftarrow reverse(current\_order)$;

2.2.5 带值传播的换向有向无环图最大和算法

由于无效假设的存在，尽管 Max-sum_AD 可以保证单侧收敛，它仍然无法得

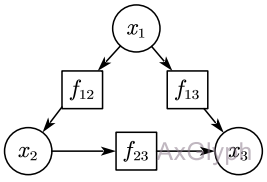
到高质量的解。考虑一个简单的图着色问题的例子。在该问题中，有三个两两相邻的顶点需要涂色，共有红绿蓝三种颜色供选择。若相邻顶点被涂成了相同颜色，则会导致代价 1。图 2.6 给出了该问题的约束图、约束矩阵和一个可能的有向无环因子图。



(a) 约束图

x_i \ x_j	R	G	B
R	1	0	0
G	0	1	0
B	0	0	1

(b) 约束矩阵



(c) 有向无环因子图

图 2.6 图着色实例

Fig. 2.6 A graph coloring instance

当按照 Max-sum_AD 算法求解时，因子图中的各个节点会根据式(2.6)和式(2.8)计算消息。例如x_1会按以下方式计算发送给f_{12}的消息：

$$q_{x_1 \to f_{12}}(x_1) = r_{f_{13} \to x_1}(x_1) = [0,0]$$

在上式中，由于初始状态下，x_1尚未收到$r_{f_{13} \to x_1}(x_1)$，因此将其视为全零向量。而f_{12}会按以下方式计算发送给x_2的消息：

$$r_{f_{12} \to x_2}(x_2) = \min_{x_1} \left(f_{12}(x_1, x_2) + q_{x_1 \to f_{12}}(x_1) \right) = [0,0]$$

根据图 2.6 (c)所示的有向无环图，在一个阶段内会产生如下消息[1]。

第 1 轮：$x_1 \to f_{12}$:$[0,0,0]$　　$x_1 \to f_{13}$:$[0,0,0]$

第 2 轮：$f_{12} \to x_2$:$[0,0,0]$　　$f_{13} \to x_3$:$[0,0,0]$

第 3 轮：$x_2 \to f_{23}$:$[0,0,0]$

第 4 轮：$f_{23} \to x_3$:$[0,0,0]$

根据式(2.9)，不难发现，四轮消息传递后，各个变量节点的信念如下：

$$z_1 = [0,0,0] \quad z_2 = [0,0,0] \quad z_3 = [0,0,0]$$

因为各个取值的效用相同，各个节点根据信念都将选择涂为红色（即第一个取值），而这种策略反而导致了最大的代价。换句话说，Max-sum_AD 无法对该问题做任何的优化。

这种问题的根源在于各个变量节点错误地估计了上游节点的取值。例如根据变量节点x_2的信念，当其选择红色时，对应代价为 0。然而这种代价只有当x_1和x_3选择绿色和蓝色才能达到，而显然，在这个例子该假设是不成立的。事实上，

[1] 事实上，Max-sum_AD 中的每一个节点始终并行操作，且每一轮都会发送消息。为了便于讨论，这里只给出了稳定、不变且没有被正则化的消息，但这不影响算法的性质。

Max-sum 算法同样存在该问题，且更加严重。因为在 Max-sum 算法中，信念会在环中反复传播。因此，无效的取值假设将会严重影响算法的性能。

为了解决该问题，一种可能的方法是在变量节点中引入取值偏好（Preference）。这样，当效用相同时，变量节点可以根据其取值偏好来选值。例如在上面这个例子中，假设三个变量节点的取值偏好分别是：

$$b_1 = [1,0,0] \quad b_2 = [0,1,0] \quad b_3 = [0,0,1]$$

根据该偏好，三个节点可以分别选择红色、绿色、蓝色，最终使得整体代价为 0。

但是，偏好的选择需要先验知识，而在完全分布式的环境下很难做到选择恰当的个体偏好。如果偏好选择地不好，则无法提高算法的性能，甚至反而可能会降低解的质量。事实上，该方法只是解决了效用相同这一表象，但是没有解决无效取值假设这一根本问题。因此，静态的取值偏好无法较好地解决上述问题。

值传播（Value Propagation）是另外一种解决效用相同和无效取值假设的方式。它通过传播变量节点的取值来直接消除下游节点中变量节点的取值假设问题。具体地，一个变量节点的查询消息不仅包含信念，还包含其当前的取值。当一个函数节点在产生响应消息时，相比于 Max-sum_AD 算法中对信念关于除目标节点外邻居取极小的操作，其直接将其他邻居取值固定为其所收到的取值。带值传播的 Max-sum_AD（Max-sum_ADVP）算法基于上述原理，在执行若干阶段的 Max-sum_AD 操作后，开启值传播直至算法结束。实验结果和相应的理论分析表明，该方法可以大幅提高解的质量，且能保证跨阶段的收敛。

2.2.6 同步分支定界算法

同步分支定界算法（Synchronous Branch and Bound，SBB）是早期的一种基于链式通信结构的完备搜索算法。该算法是分支定界算法的分布式实现，其通过在链式结构上将解空间展开，并借助不断收缩的上界对解空间进行剪枝，直至找到最优解。具体地，算法流程如下：

① 链式结构中的第一个 Agent 向其子节点发送一个包含其值域中第一个取值的部分解（Current Partial Assignment，CPA）消息给其下一个 Agent 来启动算法。此时，该 CPA 消息中的上界是无穷大，部分解对应的代价是 0。

② 当 Agent 收到来自前一个 Agent 的 CPA 消息时，它选择第一个能满足上界的取值加入部分解，更新部分解的代价并继续向下一个 Agent 传播 CPA 消息。如果其值域中不存在任何可以满足当前上界的取值，则发送回溯消息给其前一个 Agent。特别地，如果当前 Agent 是最后一个 Agent，则它选择在收到的部分解下的最优取值，更新上界并发送回溯消息给上一个 Agent。

③ 当 Agent 收到来自后一个 Agent 的回溯消息后，它切换其取值到下一个可以满足当前上界的取值，更新当前部分解的代价并发送 CPA 消息给下一个 Agent。

如果不存在可以满足当前上界的取值，则该 Agent 继续向其前一个 Agent 发送回溯消息。

2.3 分布式约束优化问题实验

2.3.1 实验测试问题

本文实验中所使用的标准测试问题包括以下四类：

① 随机 DCOP/ADCOP（Random DCOP/ADCOP）。该类测试问题代表了一般化的无结构 DCOP 和 ADCOP。在这类问题中，两个 Agent 间随机建立起约束关系，直到达到给定的约束密度。对于每一个约束关系，每一个赋值组合对应的代价都是均匀地从给定的区间范围内随机选取。具体地，该问题可以由如下参数定义：

1）Agent 个数 n，定义问题中 Agent 的个数，反映问题的规模；

2）约束图密度 p_1，定义了问题中约束的密度，反映了问题的复杂程度；

3）值域大小 $|D_i|$，定义了问题中每个变量的值域大小，反映了问题的规模；

4）代价范围[minCost, maxCost]，定义了约束函数中代价选取的范围。

② 随机非对称约束满足问题（Random ADCSP）。该类测试问题代表了一般化的无结构 ADCSP。在这类问题中，两个 Agent 间随机建立起约束关系，直到达到给定的约束密度。对于每一个约束关系，每一个赋值组合对应的代价根据紧度从 0 或者 1 中选取，其中，1 代表该约束被违反。问题的目标是找到一组解使得违反的约束数最少。具体地，该问题可以由如下参数定义：

1）Agent 个数 n，定义问题中 Agent 的个数，反映问题的规模；

2）约束图密度 p_1，定义了问题中约束的密度，反映了问题的复杂程度；

3）值域大小 $|D_i|$，定义了问题中每个变量的值域大小，反映了问题的规模；

4）紧度 p_2，定义了约束函数中每个赋值组合被禁止的概率。

③ 无尺度网络（Scale-free Networks）[48]。该类测试问题是指拓扑结构中节点的度分布为幂率分布（Power-law）的问题，反映了自然世界中普遍存在的“马太效应”。它通常用来测试算法求解拓扑高度结构化的问题时的性能。本文使用 BA 模型[48]来生成无尺度网络问题的拓扑结构。具体地，首先生成一个由 m_1 个 Agent 组成的连通图，在接下来的每一次迭代中，加入一个新的 Agent，并与当前系统中的 m_2 个 Agent 相连，直至所有 Agent 都加入系统。其中，一个 Agent 被连接的概率与该 Agent 的当前度数成正比。显然，m_1 和 m_2 反映了问题的结构化程度。当 m_1 越小、m_2 越大时，节点的度数分布越不均衡，问题的结构化程度越高。该类问题的 Agent 个数、值域大小和代价范围的定义与随机 DCOP/ADCOP 相同。

④ 加权图着色问题（Weighted Graph-coloring Problem）。该类问题是图着色

问题的扩充，用来评估算法在求解约束代价函数高度结构化的问题时的性能。在该类问题中，需要为每一个顶点（即 Agent）涂色，且每两个相邻顶点不能被涂成相同的颜色（即取值）。如果违反了该要求，则会带来相应的代价。每一个违反代价都是从给定的范围内均匀的随机选取。该类问题的 Agent 个数、值域大小、图密度和代价范围的定义与随机 DCOP/ADCOP 相同。需要指出的是，根据四色定理，4 种颜色即可完成任意图的着色。因此，在实验中，通常选取 3 中颜色作为测试问题。

2.3.2 算法评价指标

不同种类算法的性能指标不尽相同。在评价完备算法时，由于保证获得最优解，因此我们更关注求得最优解所付出的代价，如消息数、非并发约束检查、运行时间等。在评价非完备算法时，求解质量和速度是考察的首要因素，因此我们更关注最终代价和每轮代价。此外，为了评价算法的探索能力，我们还应考察算法在 Anytime 机制下的最终代价和每轮代价。最后，在考察求解 ADCOP 的算法时，除了上述指标，还应额外考虑算法求解过程中造成的隐私泄露的程度。具体地，算法评价指标如下：

① 消息数（Message number）。消息数是指算法求解过程中所交换的全部消息的数量，该指标反映了算法对网络通信带来的负载。因此，在设计算法时应尽可能减少消息发送的数量。

② 非并发约束检查（Non-concurrent Constraint Checks，*NCCCs*）[49]。该指标是处理时间和通信时间之和，反映了算法在不同的通信环境下的性能。为了计算该指标，每个 Agent 内部维护一个 *NCCCs* 变量。当一个 Agent 执行因此约束检查时，该变量就增加 1。此外，Agent 间传递的消息包含了发送者的 *NCCCs* 值。当一个 Agent 收到另外一个 Agent 的消息时，其更新自己 $NCCCs = \max\{NCCCs, NCCCs' + t\}$。其中，$NCCCs'$ 为发送者的并发约束检查值，t 为通信时间。在实验中，一般用 $t = 0$ 模拟快速通信，用 $t = 100$ 模拟慢速通信。算法最终的 *NCCCs* 值为所有 Agent 中最大的 *NCCCs*。

③ 运行时间（Run time）。运行时间是指算法从开始到终止（如找到最优解或满足某一终止条件）时在实验平台上模拟的运行时间。该时间反映了算法在求解真实问题时所耗费的时间。

④ 最终代价。最终代价反映了算法终止时所得出的解的质量。该指标用于评价和比较非完备算法的质量和进行显著性分析。

⑤ 每轮代价。每轮代价是指在同步算法的每一轮中，各个 Agent 取值所造成的全局代价。因此，该指标通常是以折线图的方式存在。每轮代价反映了算法寻优的速度，即在给定相同最大迭代轮次下，折线下面积越小则说明算法的寻优速

度越快。

⑥ 隐私损失（Privacy loss）[50]。该指标反映了算法在求解过程中所造成的隐私泄露程度。Agent 的隐私可以用信息熵（Entropy）来量化。具体地，一个 Agent a_i 的熵定义如下：

$$H_i = -\sum_{a_j \in \mathrm{N}_i} \sum_{k \in S_{ij}} p_k \log_2 p_k \tag{2.11}$$

其中，N_i 是 a_i 所有邻居的集合；S_{ij} 是 a_i 与 a_j 之间所有可能的状态集合；p_k 是状态 k 的概率。熵代表了邻居对该约束未知的信息量。为了测量算法运行时的消息损失，需要计算算法开始前整个系统对应的信息量和算法运行结束后的的信息量，二者的差值即为算法的隐私损失。

2.3.3 实验平台

为了评价算法的优劣，必须对 DCOP 算法进行仿真以得到上述各种评价指标。常见的仿真方式有多机仿真和单机仿真两种方式。前者将每一台计算机作为一个 Agent，然后通过网络互连实现 Agent 的相互通信。该方法的优点在于较好地还原了分布式计算环境，且系统计算能力较高，能够对大规模复杂问题进行仿真。但该方式成本较高，且对软件基础设施要求较高，灵活性差。因此，单机软件仿真是目前使用较多的仿真方式。目前主要的软件仿真平台有 DisChoco[51]、Frodo[52]、DCOPolis [53]等。本文所用的实验平台是研究组自行研发的 DCOPSolver 平台。该平台通过多线程模拟分布式环境，即一个 Agent 就是一个线程，Agent 间的相互通信是通过线程间的通信完成的。相比于现有的仿真平台，DCOPSolver 具有更好的可扩展性和更便于使用的接口。图 2.7 给出了该平台的 GUI 界面。

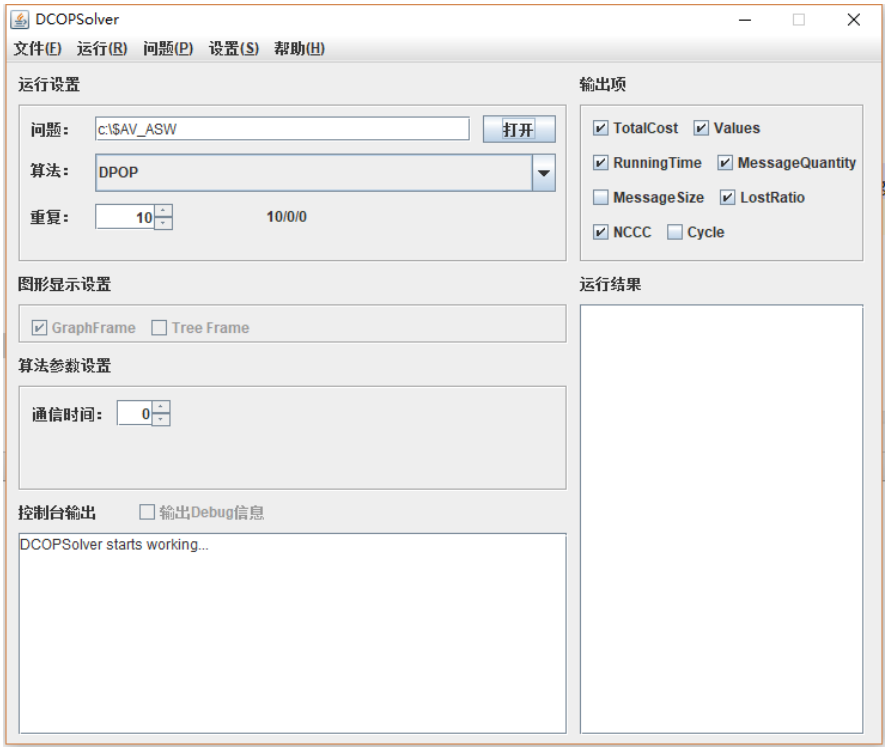


图 2.7 DCOPSolver 平台的图形化界面

Fig. 2.7 GUI of DCOPSolver

DCOPSolver 由三个模块：基础设施模块、算法模块和指标统计模块。基础设施模块为平台提供了基本的类型定义、标准测试问题生成与解析、线程间通信和图形界面等功能。其中，基本类型定义规定了平台中 Agent 的收发消息的接口规范；线程间通信则通过中心邮箱（Mailer）实现了 Agent 间的同步和异步通信。在同步通信中，Mailer 与 Agent 交替工作，即在一轮中 Agent 首先向 Mailer 中投递消息，当所有 Agent 执行完本轮的操作后，Mailer 向各个 Agent 转发消息，完成一个同步轮次。在异步通信中，Mailer 和 Agent 同时工作，当 Mailer 收到 Agent 投递的消息后，立刻转发给消息对应的目标 Agent。算法模块中提供了现有的流行算法的实现，用来比较新开发的算法的性能。算法模块包括 SBB、DPOP、ADOPT、PT-FB 等完备算法和 DSA 类、MGM 类、Max-sum 类以及 ALS 和 PDS 框架等。指标统计模块定义了各个性能指标的数据结构，并定义了这些数据结构的加、减、乘、除等算术操作，以便大规模批量实验中取平均操作。

2.4 本章小结

本章首先介绍了 DCOP 和 ADCOP 的形式化定义，给出了对应的可视化表示方式——约束图。其次，本章描述了本文研究的必要的基础知识与算法，包括通信结构的概念、DPOP 算法、Max-sum 类算法、SBB 算法。其中，重点讨论了

Max-sum 算法和 Max-sum_AD 算法的缺点，指出导致其性能较差的原因是无效取值假设的问题，并由此介绍了值传播在 Max-sum 类算法中所起到的重要作用。最后，本章介绍了 DCOP/ADCOP 算法的实验评估，包括标准化测试问题种类、性能指标及实验平台。

3　基于非连续值传播的最大和算法

3.1 引言

作为一种重要的 DCOP 非完备推理算法，Max-sum 广泛地应用于各种实际场景中。但其在有环问题中不能收敛，且经常不能得到较优的解。为此，学者们陆续提出了多种方法，将原有环因子图转换为树或者有向无环图来避免信念的循环传播。其中，Max-sum_AD 可以保证算法的单向收敛性。但正如 2.2.4 节所展示的，Max-sum_AD 不能打破效用相同，且存在无效取值假设的问题。为此，提出用值传播消除取值假设并打破效用相同，即 Max-sum_ADVP。该算法可以保证跨阶段的收敛性，并可以极大提高 Max-sum_AD 的性能。

在本章中，我们首先分析了值传播对 Max-sum 类算法的影响。证明了值传播会阻碍 Max-sum 中的信念传播，并指出 Max-sum_ADVP 最终等价于一个顺序的贪心局部搜索算法。由此，提出了如何平衡 Max-sum_ADVP 中的探索和利用的问题。为了解决上述问题，本章提出了三个基于非连续值传播的 Max-sum 类算法，并在理论上分析了各算法中变量节点的决策函数，指出本章提出的方法不会等价于局部搜索算法。实验结果表明，本章所提出的算法显著优于 Max-sum_ADVP 且对开始值传播的时机具有较强的鲁棒性。

3.2 值传播对 Max-sum 类算法的影响

值传播作为打破效用相同及无效取值假设的重要手段，在 Max-sum 类算法中得到了广泛使用。然而，尽管值传播可以保证跨阶段收敛并极大地提高解的质量，它实际上也会限制 Max-sum 类算法的探索能力。因此，Agent 最终将丧失传播和累积全局信念的能力，并只能用局部信息作出决策。下面，本文首先将从理论上证明值传播会限制 Max-sum 类算法的全局信念传播。特别地，当值传播被应用于 Max-sum_AD 算法时，将等价于一个贪心局部搜索算法。最后，本节将给出一个具体的例子说明 Max-sum_ADVP 最终会陷入局部最优，且其质量与初值密切相关。考虑以下命题：

引理 3.1：当启用值传播后，函数节点将阻碍全局信念的传播，且只能传播本地函数。

证明：不失一般性地，考虑图 3.1 所示的有向无环因子图的片段。在这个片段中，有两个变量节点x_i、x_j和一个函数节点f_{ij}。当开启值传播后，x_i不仅传播信念，同时也传播其取值。即：

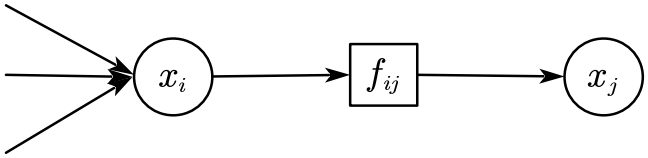


图 3.1 一个简单的有向无环因子图

Fig. 3.1 A simple directed factor graph

$$x_i \to f_{ij}: q_{x_i \to f_{ij}}(x_i) = \sum_{n \in N_i \setminus j} r_{f_{ni} \to x_j}(x_i), x_i = d_i \tag{3.1}$$

当f_{ij}收到来自x_i的消息后，它根据x_i的查询消息和取值产生向x_j的响应消息。即：

$$f_{ij} \to x_j: r_{f_{ij} \to x_j}(x_j) = f_{ij}(d_i, x_j) + q_{x_i \to f_{ij}}(d_i) \tag{3.2}$$

由于在一个信念函数上加上或者减去同一个常数并不会对变量节点的决策产生影响，我们可以去除式(3.2)中的常数$q_{x_i \to f_{ij}}(d_i)$。因此，上式可以简化为：

$$f_{ij} \to x_j: r_{f_{ij} \to x_j}(x_j) = f_{ij}(d_i, x_j) \tag{3.3}$$

事实上，式(3.3)给出的信念是本地函数f_{ij}给定变量x_i取值的结果。原命题得证。

根据引理 3.1，我们可以得到下述推论：当值传播达到第一次单向收敛后，因子图中传播的消息全部是本地函数。因此，变量节点也就无法累加并转发全局信念。换句话说，值传播限制了 Max-sum 算法的探索能力。

命题 3.1：当值传播达到第一次单向收敛后，Max-sum_ADVP 的选值策略与贪心局部搜索算法相同。

证明：考虑图 3.1 中的因子图片段。假设值传播已经启用，且算法已经从右向左达到了一次单向收敛。根据引理 3.1，x_i的第m阶段的信念是：

$$z_i(x_i) = \sum_{n \in N_i \setminus j} f_{ni}(d_n^m, x_i) + f_{ji}(d_j^{m-1}, x_i) \tag{3.4}$$

其中，d_n^m是变量节点x_n在第m阶段所选取的赋值。因此，x_i需要选择一个可以使得式(3.4)最小的取值。换句话说，x_i总是选择在给定邻居取值下的最佳响应（Best Response）。另一方面，贪心局部搜索算法（如 DSA、MGM 等）中的 Agent a_i同样是考虑所有邻居的取值来作出决策，即：

$$\sum_{n \in N_i} f_{ni}(d_n, x_i) \tag{3.5}$$

a_i选择一个可以使得式(3.5)最小的的取值作为自己的取值。对比式(3.4)和式(3.5)不难发现，Max-sum_ADVP 和贪心局部搜索算法具有相同的决策依据。因此原命题得证。

需要指出的是，命题 3.1 只是建立了 Max-sum_ADVP 和贪心局部搜索算法在决策依据上的等价关系，事实上它们在其他方面是完全不同的。例如，DSA 中的

Agent 并行地将它们的取值替换为最佳响应，而 Max-sum_ADVP 中的变量节点则顺序地作出决策。命题 3.1 同时也说明了 Max-sum_ADVP 中的变量节点无法利用全局信念，并最终产生与贪心局部搜索算法类似的行为。因此，Max-sum_ADVP 与贪心局部搜索一样会陷入局部最优。考虑图 2.1 所示的 DCOP 实例和图 2.5 所示有向无环因子图，假设在第三阶段开始执行值传播，则 Max-sum_ADVP 的执行的详细过程如下：

阶段 1（信念传播）

第 1 轮：$x_1 \to f_{13}:[0,0]$　　$x_2 \to f_{23}:[0,0]$　　$x_2 \to f_{24}:[0,0]$

第 2 轮：$f_{13} \to x_3:[1,3]$　　$f_{23} \to x_3:[7,3]$　　$f_{24} \to x_4:[1,3]$

第 3 轮：$x_3 \to f_{34}:[8,6]$

第 4 轮：$f_{34} \to x_4:[12,12]$

信念：　$z_1=[0,0]$　$z_2=[0,0]$　$z_3=[8,6]$　$z_4=[13,15]$

赋值：　$x_1^*=0$　$x_2^*=0$　$x_3^*=1$　$x_4^*=0$

阶段 2（信念传播）

第 1 轮：$x_4 \to f_{24}:[12,12]$　　$x_4 \to f_{34}:[1,3]$

第 2 轮：$f_{34} \to x_3:[5,7]$　　$f_{24} \to x_2:[15,13]$

第 3 轮：$x_3 \to f_{13}:[12,10]$　　$x_3 \to f_{23}:[6,10]$

第 4 轮：$f_{13} \to x_1:[13,13]$　　$f_{23} \to x_2:[13,13]$

信念：　$z_1=[13,13]$　$z_2=[28,26]$　$z_3=[13,13]$　$z_4=[13,15]$

赋值：　$x_1^*=0$　$x_2^*=1$　$x_3^*=0$　$x_4^*=0$

阶段 3（值传播）

第 1 轮：$x_1 \to f_{13}:[0,0],x_1=0$　　$x_2 \to f_{23}:[8,3],x_2=1$

$x_2 \to f_{24}:[1,5],x_2=1$

第 2 轮：$f_{13} \to x_3:[1,7]$　　$f_{23} \to x_3:[8,3]$　　$f_{24} \to x_4:[1,5]$

第 3 轮：$x_3 \to f_{34}:[9,10],x_3=0$

第 4 轮：$f_{34} \to x_4:[4,7]$

信念：　$z_1=[13,13]$　$z_2=[28,26]$　$z_3=[14,17]$　$z_4=[5,12]$

赋值：　$x_1^*=0$　$x_2^*=1$　$x_3^*=0$　$x_4^*=0$

阶段 4（值传播）

第 1 轮：$x_4 \to f_{24}:[4,7],x_4=0$　　$x_4 \to f_{34}:[1,5],x_4=0$

第 2 轮：$f_{34} \to x_3:[4,6]$　　$f_{24} \to x_2:[3,1]$

第 3 轮：$x_3 \to f_{13}:[12,9],x_3=0$　　$x_3 \to f_{23}:[5,13],x_3=0$

第 4 轮：$f_{13} \to x_1:[1,9]$　　$f_{23} \to x_2:[7,8]$

信念：　$z_1=[1,9]$　$z_2=[10,9]$　$z_3=[13,16]$　$z_4=[5,12]$

赋值：$x_1^*=0$ $x_2^*=1$ $x_3^*=0$ $x_4^*=0$

阶段 5（值传播）

第 1 轮：$x_1 \to f_{13}:[0,0],x_1=0$ $x_2 \to f_{23}:[3,1],x_2=1$

$x_2 \to f_{24}:[7,8],x_2=1$

第 2 轮：$f_{13} \to x_3:[1,7]$ $f_{23} \to x_3:[8,3]$ $f_{24} \to x_4:[1,5]$

第 3 轮：$x_3 \to f_{34}:[9,10],x_3=0$

第 4 轮：$f_{34} \to x_4:[4,7]$

信念：$z_1=[13,13]$ $z_2=[28,26]$ $z_3=[14,17]$ $z_4=[5,12]$

赋值：$x_1^*=0$ $x_2^*=1$ $x_3^*=0$ $x_4^*=0$

从上述执行过程中不难看出，经过阶段 3 和阶段 4 的值传播之后，算法已经收敛。亦即阶段 3 和阶段 5 中的响应消息和变量节点得到效用相同，因此算法最终收敛到一个次优解$\{x_1=0,x_2=1,x_3=0,x_4=0\}$，其代价是 14。而该问题的最优解是$\{x_1=1,x_2=1,x_3=1,x_4=0\}$，其代价是 13。此外，上述执行过程也同时说明了 Max-sum_ADVP 严重依赖于初始取值。假设在阶段 4 的结尾各个变量节点的取值分别是$\{x_1=1,x_2=1,x_3=0,x_4=0\}$，那么在一轮值传播后算法将会收敛到最优解。

事实上，因为信念传播可以帮助变量节点找到更有潜力的解，而值传播可以单调地优化这些解，我们可以分别将信念传播和值传播看作探索（Exploration）与利用（Exploitation）。然而，引理 3.1 说明了值传播和信念传播完全是相互对抗的，因此不管是 Max-sum_AD 还是 Max-sum_ADVP 都无法平衡探索与利用。所以，在接下来的内容中，本文将提出一系列可以平衡探索与利用的 Max-sum 类算法。

3.3 基于单向值传播的 Max-sum_AD 算法

3.3.1 算法描述

考虑到值传播和信念传播间的关系，一种可能的平衡探索和利用的途径就是交替执行值传播和信念传播。这样，值传播始终沿着一个方向执行，因此该算法称为基于单向值传播的 Max-sum_AD 算法（Max-sum_AD with Single Side Value Propagation，Max-sum_ADSSVP）。具体地，下述过程将在一个回合中交替执行：

① **值传播阶段（Value Propagation Phase）**：当当前消息传播方向是正向时，使用值传播以保证解的质量。

② **信念传播阶段（Belief Propagation Phase）**：当当前消息传播方向是反向时，使用信念传播来为下一轮的值传播找到更高质量的初始值。

这里，正向和反向分别指初始的消息传播方向和对应的反方向。

事实上，上述信念传播和值传播交替执行的方法可以让变量节点做决策时，不仅考虑本地函数，也考虑全局信念。具体地，我们有下面的性质：

性质 3.1：在启用值传播后 Max-sum_ADSSVP 中的变量节点始终利用全局信念和本地函数作出决策。即当阶段m是值传播阶段时，变量节点x_i在该阶段结束时的信念是

$$z_i(x_i) = \sum_{n \in P_i} f_{ni}(k_n^m, x_i) + \sum_{n \in S_i} r_{f_{ni} \to x_i}^{m-1}(x_i) \tag{3.6}$$

其中，P_i和S_i分别表示在当前方向下，排在x_i前面或和面的邻居的集合；$r_{f_{ni} \to x_i}^{m-1}(x_i)$表示在阶段$m-1$结束时函数节点$f_{ni}$发送给变量节点$x_i$的响应信息。相似地，当阶段$m$是信念传播阶段时，变量节点$x_i$在该阶段结束时的信念是：

$$z_i(x_i) = \sum_{n \in P_i} r_{f_{ni} \to x_i}^{m}(x_i) + \sum_{n \in S_i} f_{ni}(k_n^{m-1}, x_i) \tag{3.7}$$

作为一个例子，考虑图 2.5 所示的有向无环因子图。如果当前阶段m是值传播阶段，变量节点x_3使用来自其上游的邻居f_{23}和f_{13}的本地函数和来自下游邻居f_{34}的全局信念作出决策。形式化地，x_3在阶段m结束时的信念由式(3.8)给出。

$$z_3(x_3) = f_{13}(k_1^m, x_3) + f_{23}(k_2^m, x_3) + r_{f_{34} \to x_3}^{m-1}(x_3) \tag{3.8}$$

上述性质说明了 Max-sum_ADSSVP 可以通过综合考虑个体利益（即本地函数）和全局利益（全局信念）来跳出局部最优。然而，上述性质同时说明了变量节点不再完全基于本地函数作出决策，这就导致了算法不再保证单调性和收敛性。实际上，在每一次从值传播阶段到信念传播阶段的过渡时，都会引发解的质量的剧烈震荡。算法也因此需要更多轮次来抑止这种波动。

注意到在 Max-sum_AD 类算法中，为了可以让约束信息跨阶段传播，节点始终根据其所有邻居的消息来计算消息。因此，值传播阶段结束时的解会对下一轮信念传播阶段产生重要的影响。考虑到值传播的单调性，一种直接的缓解上述问题的方式是连续执行多阶段值传播。为此，提出 Max-sum_ADSSVP(*t*)算法。不同于 Max-sum_ADSSVP 中一回合由一个值传播阶段和一个信念传播阶段组成，Max-sum_ADSSVP(*t*)中的一个回合由 *t* 个连续的值传播阶段和一个信念传播阶段组成。这样，在过渡到信念传播阶段前，可以保证解已经被充分地优化，从而有效地抑制信念传播阶段的解的质量的波动。

需要指出的是，Max-sum_ADSSVP(*t*)与文献[10]提出的带消息探索方法的 Max-sum_ADVP 算法具有本质的不同。在 Max-sum_ADSSVP(*t*)中，连续执行多个阶段的值传播的目的是充分地优化解；而文献[10]则是通过交替执行多个版本的 Max-sum 算法来平衡探索和利用。此外，在 Max-sum_ADSSVP(*t*)是每一个回合中，信念传播只因被执行一次。如果连续执行两个阶段的信念传播，则后一次执行的信念传播中，消息的计算完全依赖于第一次的信念传播阶段产生的消息，也就与

前一次的值传播阶段产生的消息毫无关系。

解决上述问题的另外一种方法是在值传播阶段后面引入局部搜索，在短时间内快速提升解的质量。据此，提出带局部搜索的 Max-sum_ADSSVP 算法（Max-sum_ADSSVP with Local Search，Max-sum_ADSSVP_LS）。具体地，下面两个阶段将会在值传播阶段结束后被执行：

③ 优化阶段（Refining Phase）：该阶段通过局部搜索来对值传播产生的解进行进一步优化以产生更高质量的解。

④ 修改阶段（Modification Phase）：该阶段通过执行无决策的值传播将优化阶段产生的解应用到因子图中。

算法 3.1 Max-sum_ADSSVP_LS 的伪代码

Algorithm 3.1 Sketch of Max-sum_ADSSVP_LS

Max-sum_ADSSVP_LS (node *n*, local search, *forward_direction, k, l*)

```
1.  current_order ← forward_direction;
2.  backward_direction ← reverse(forward_direction);
3.  while no termination condition is met do
4.    for k iterations do
5.      if current_order is backward_direction then
6.        perform belief propagation;
7.      else if current_order is forward_direction then
8.        x_i^* ← current optimal decision;
9.        perform value propagation using assignment x_i^*;
10.   if current_order is forward_direction then
11.     perform l iteration local search with initial assignment x_i^*;
12.     x_i^* ← current optimal decision;
13.     for k iterations do
14.       perform value propagation using assignment x_i^*;
15.   current_order ← reverse(current_order);
```

算法 3.1 给出了 Max-sum_ADSSVP_LS 的伪代码。该算法有三个参数：优化阶段使用的局部搜索算法、优化阶段的长度l和信念传播和值传播阶段的长度k。若当前消息传播方向是反向时，Agent 执行信念传播（5-6 行），否则 Agent 执行值传播（7-8 行）。如果当前消息传播方向是正向（即 Agent 此时处于值传播阶段），优化阶段和修改阶段在值传播阶段结束后相继被触发（10-14 行）。每个 Agent 将

在传播阶段中产生的取值作为局部搜索算法的初值，然后继续l轮的局部搜索（11行）。优化阶段结束后每个 Agent 执行k轮的修改阶段将上一阶段产生的解应用到因子图中（12-14 行）。

需要指出的是只有 Max-sum_ADSSVP 可以（或需要）用局部搜索来增强。事实上，在值传播阶段后面引入局部搜索的目的是进一步提高解的质量，从而对接下来的信念传播阶段产生积极地影响。对于 Max-sum_ADVP 来说，由于它本身就可以保证单调性，且已经被前文证明了其和贪心局部搜索算法的等价性，因此它是没有必要用局部搜索来增强的。另一方面，由于 Max-sum 和 Max-sum_AD 没有值传播阶段，导致局部搜索所产生的更优解无法应用到因子图中，因此它们无法用局部搜索算法增强

3.3.2 实例分析

考虑图 2.1 所示的 DCOP 实例和图 2.5 所示的有向无环图。假设值传播在阶段 3 启用，则 Max-sum_ADSSVP 的详细执行过程如下（这里，忽略了与上文 Max-sum_ADVP 执行过程中完全一致的前两个阶段）：

阶段 3（值传播）

第 1 轮：$x_1 \to f_{13}:[0,0],x_1=0$　　$x_2 \to f_{23}:[8,3],x_2=1$

　　$x_2 \to f_{24}:[1,5],x_2=1$

第 2 轮：$f_{13} \to x_3:[1,7]$　　$f_{23} \to x_3:[8,3]$　　$f_{24} \to x_4:[1,5]$

第 3 轮：$x_3 \to f_{34}:[9,10],x_3=0$

第 4 轮：$f_{34} \to x_4:[4,7]$

信念：　$z_1=[13,13]$　$z_2=[28,26]$　$z_3=[14,17]$　$z_4=[5,12]$

赋值：$x_1^*=0$　$x_2^*=1$　$x_3^*=0$　$x_4^*=0$

阶段 4（信念传播）

第 1 轮：$x_4 \to f_{24}:[4,7]$　　$x_4 \to f_{34}:[1,5]$

第 2 轮：$f_{34} \to x_3:[5,7]$　　$f_{24} \to x_2:[7,5]$

第 3 轮：$x_3 \to f_{13}:[13,10]$　　$x_3 \to f_{23}:[6,14]$

第 4 轮：$f_{13} \to x_1:[14,13]$　　$f_{23} \to x_2:[13,14]$

信念：$z_1=[14,13]$　$z_2=[20,19]$　$z_3=[14,17]$　$z_4=[5,12]$

赋值：$x_1^*=1$　$x_2^*=1$　$x_3^*=0$　$x_4^*=0$

阶段 5（值传播）

第 1 轮：$x_1 \to f_{13}:[0,0],x_1=1$　$x_2 \to f_{23}:[7,5],x_2=1$

　　$x_2 \to f_{24}:[13,14],x_2=1$

第 2 轮：$f_{13} \to x_3:[9,3]$　　$f_{23} \to x_3:[8,3]$　　$f_{24} \to x_4:[1,5]$

第 3 轮：$x_3 \to f_{34}:[17,6],x_3=1$

第 4 轮：$f_{34} \to x_4{:}[6,6]$

信念：　$z_1 = [14,13]$　$z_2 = [20,19]$　$z_3 = [22,13]$　$z_4 = [7,11]$

赋值：$x_1^* = 1$　$x_2^* = 1$　$x_3^* = 1$　$x_4^* = 0$

对比 Max-sum_ADVP 和 Max-sum_ADSSVP 的执行过程不难发现，不同于 Max-sum_ADVP 中连续地进行值传播，本文提出的 Max-sum_ADSSVP 在阶段 4 执行了信念传播。所以，该算法就有机会得到新的、更加有希望的初始值，并最终打破了局部最优，找到了全局最优解。与此相反，Max-sum_ADVP 在启用值传播之后，一直对解进行连续单调地优化，没有机会接触到更有潜力的新解。尽管 Max-sum_ADVP 保证了单调性和收敛性，它最终陷入了局部最优。此外，由于每个信念传播阶段都可以为下一个值传播阶段提供初始值，Max-sum_ADSSVP 对值传播开始时的时机并不敏感。因此，Max-sum_ADSSVP 也解决了 Max-sum_ADVP 中值传播时机的选择问题。

3.3.3 理论分析

Max-sum_ADSSVP 所带来的额外代价主要集中于计算。具体地，在值传播阶段，一个函数节点仅需要$O(d)$的操作即可通过固定上游节点的取值为其收到的值来计算出向下游变量节点的响应消息。而在信念传播阶段，一个函数节点需要$O(d^2)$的操作来遍历上游节点的值域来计算向下游变量节点的响应消息。其中$d = \max_{D_i \in D}|D_i|$为最大的值域长度。因此，相比于 Max-sum_ADVP，本文提出的 Max-sum_ADSSVP 仅在每个信念传播阶段引入了$O(d^2)$的额外计算代价。

Max-sum_ADSSVP(*t*)和 Max-sum_ADSSVP_LS 所带来的计算的代价更小。对于 Max-sum_ADSSVP(*t*)，每一个回合包含若干个值传播阶段和一个信念传播阶段。因此如果给定相同的最大轮次，信念传播阶段的个数事实上是远远小于 Max-sum_ADVP 的。类似的，Max-sum_ADSSVP_LS 中由于信念传播阶段带来的计算代价同样小于 Max-sum_ADVP。此外，由于优化阶段通常轮次较少，且局部搜索算法操作一般较为简单，所引发的计算代价通常也较小。而修改阶段与值传播阶段操作类似，也仅需$O(d)$的操作即可计算一条消息，因此 Max-sum_ADSSVP_LS 的整体复杂度低于 Max-sum_ADVP。

3.4 基于混合信念/值传播的 Max-sum 算法

3.4.1 算法描述

由于在 Max-sum_ADSSVP 中，Agent 在一个阶段只能执行探索或者利用，因此算法需要大量的轮次去抑制从值传播阶段到信念传播阶段过渡时产生的解的质量的波动。此外，从性质 3.1 不难看出，在信念传播阶段，变量节点始终用过时的赋值信息作出决策，这同样加剧了解的质量的波动。具体地，变量节点始终考

虑上一轮收到的下游节点的赋值信息，而这些赋值可能在本轮中被更改和替换。换句话说，在信念传播阶段，变量节点考虑了无效的下游赋值信息。

本节试图通过让节点尽可能快地获取到最新的消息缓解上述问题，并提出一种全新的算法，即基于混合信念/值传播的 Max-sum 算法（Max-sum with Hybrid Belief/Value Propagation, Max-sum_HBVP）。具体地，相比于 Max-sum_ADVP 中每两轮交替执行信念传播和值传播，在 Max-sum_HBVP 中信念传播和值传播将会在有向无环图的两端同时被执行。算法 3.2 给出了 Max-sum_HBVP 的伪代码。

算法 3.2 Max-sum_HBVP 的伪代码

Algorithm 3.2 Sketch of Max-sum_HBVP

Max-sum_HBVP(node *n*)

1. $current\_order \leftarrow forward\_direction$;
2. $N_{prev\_n} \leftarrow \{\hat{n} \in N_n : \hat{n} \text{ is ordered before } n \text{ in } current\_order\}$;
3. $N_{follow\_n} \leftarrow N_n \setminus N_{prev\_n}$;
4. **while** *no termination condition is met* **do**
5. **for** *k iterations* **do**
6. collect messages from N_n;
7. **if** *n is a variable node* **then**
8. **if** *n receives all messages from* N_{prev_n} *and hasn't sent message to* N_{follow_n} **then**:
9. $x_n^* \leftarrow$ current optimal decision;
10. produce the message $\{x_n^*\}$ to n' using messages received from $N_n \setminus \{n'\}$, $\forall n' \in N_{follow\_n}$;
11. **if** *n receives all messages from* N_{follow_n} *and hasn't sent message to* N_{prev_n} **then**:
12. produce the message to n' using messages received from $N_n \setminus \{n'\}$, $\forall n' \in N_{prev\_n}$;
13. **else if** *n is a function node* **then**:
14. **if** *n receives all messages from* N_{prev_n} *and hasn't sent message to* N_{follow_n} **then**:
15. produce the message to n' using the local constraint, assignments received from $N_n \setminus \{n'\}$, $\forall n' \in N_{follow\_n}$;
16. **if** *n receives all messages from* N_{follow_n} *and hasn't sent message to* N_{prev_n} **then**:

Max-sum_HBVP(node *n*)	
17.	produce the message to n' using the local constraint, messages received from $N_n \setminus \{n'\}$, $\forall n' \in N_{prev\_n}$;
18.	clear all sending and receiving flags;

算法中每个节点都检测其有没有收到来自上游（或下游）节点的全部消息。如果满足该条件且节点还没给其下游（或上游）节点发送消息，节点就发送消息给这些节点（8-12 行，14-17 行）。当迭代次数到达 *k* 时，所有的发送和接收标志位都被重置，而后一个新的回合开始（18 行）。值传播和信念传播的混合执行是由函数节点完成的。具体地，当一个函数节点收到来自所有上游节点的消息时，它产生向下游节点的值传播消息（14-15 行）；当其收到来自所有下游节点的消息时，它产生向上游节点的信念传播消息（16-17 行）。此外，变量节点仅在收到上游节点的消息后作出决策（9 行），这就保证了变量节点始终依据最新的、有效的赋值信息作出决策，从而克服了上述问题。

3.4.2 实例分析

考虑图 2.1 所示的 DCOP 实例和图 2.5 所示的有向无环图。Max-sum_HBVP 的详细执行内容如下：

回合 1

第 1 轮：$z_1(x_1)=[0,0] \Rightarrow x_1^*=0$　　$z_2(x_2)=[0,0] \Rightarrow x_2^*=0$

$x_1 \to f_{13}:[0,0], x_1=0$　　$x_2 \to f_{23}:[0,0], x_2=0$

$x_2 \to f_{24}:[0,0], x_2=0$　　$x_4 \to f_{24}:[0,0]$

$x_4 \to f_{34}:[0,0]$

第 2 轮：$f_{13} \to x_3=[1,7]$　　$f_{23} \to x_3=[7,3]$　　$f_{24} \to x_4=[3,3]$

$f_{24} \to x_2=[3,1]$　　$f_{34} \to x_3=[4,6]$

第 3 轮：$z_3(x_3)=[12,16] \Rightarrow x_3^*=0$

$x_3 \to f_{34}:[8,10], x_3=0$　　$x_3 \to f_{13}:[11,9]$　　$x_3 \to f_{23}:[5,13]$

第 4 轮：$f_{34} \to x_4=[4,7]$　　$f_{13} \to x_1=[12,12]$　　$f_{23} \to x_2=[12,13]$

第 5 轮：$z_4(x_4)=[7,10] \Rightarrow x_4^*=0$

回合 2

第 1 轮：$z_1(x_1)=[12,12] \Rightarrow x_1^*=0$　　$z_2(x_2)=[15,14] \Rightarrow x_2^*=1$

$x_1 \to f_{13}:[0,0], x_1=0$　　$x_2 \to f_{23}:[3,1], x_2=1$

$x_2 \to f_{24}:[12,13], x_2=1$　　$x_4 \to f_{24}:[4,7]$

$x_4 \to f_{34}:[3,3]$

第 2 轮：$f_{13} \to x_3=[1,7]$　　$f_{23} \to x_3=[8,3]$　　$f_{24} \to x_4=[1,5]$

$f_{24} \to x_2 = [7,5]$　　$f_{34} \to x_3 = [7,9]$

第 3 轮：$z_3(x_3) = [16,19] \Rightarrow x_3^* = 0$

$x_3 \to f_{34}{:}[9,10], x_3 = 0$　　$x_3 \to f_{13}{:}[15,12]$　　$x_3 \to f_{23}{:}[8,16]$

第 4 轮：$f_{34} \to x_4 = [4,7]$　　$f_{13} \to x_1 = [16,15]$　　$f_{23} \to x_2 = [15,16]$

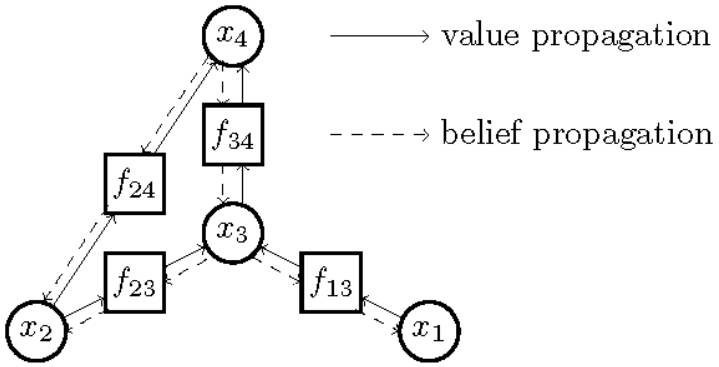


图 3.2 Max-sum_HBVP 执行示例

Fig. 3.2 An execution example of Max-sum_HBVP

第 5 轮：$z_4(x_4) = [5,12] \Rightarrow x_4^* = 0$

回合 3

第 1 轮：$z_1(x_1) = [16,15] \Rightarrow x_1^* = 1$　　$z_2(x_2) = [22,21] \Rightarrow x_2^* = 1$

$x_1 \to f_{13}{:}[0,0], x_1 = 1$　　$x_2 \to f_{23}{:}[7,5], x_2 = 1$

$x_2 \to f_{24}{:}[15,16], x_2 = 1$　　$x_4 \to f_{24}{:}[4,7]$

$x_4 \to f_{34}{:}[1,5]$

第 2 轮：$f_{13} \to x_3 = [9,3]$　　$f_{23} \to x_3 = [8,3]$　　$f_{24} \to x_4 = [1,5]$

$f_{24} \to x_2 = [7,5]$　　$f_{34} \to x_3 = [5,7]$

第 3 轮：$z_3(x_3) = [22,13] \Rightarrow x_3^* = 1$

$x_3 \to f_{34}{:}[17,6], x_3 = 1$　　$x_3 \to f_{13}{:}[13,10]$　　$x_3 \to f_{23}{:}[14,10]$

第 4 轮：$f_{34} \to x_4 = [6,6]$　　$f_{13} \to x_1 = [14,13]$　　$f_{23} \to x_2 = [13,13]$

第 5 轮：$z_4(x_4) = [7,11] \Rightarrow x_4^* = 0$

图 3.2 给出了 Max-sum_HBVP 的消息传递示意图。从上面的示意图和所传播的消息中不难发现，算法沿着有向无环图的方向执行值传播（如第 1 轮中的 $x_1 \to f_{13}$、$x_2 \to f_{23}$、$x_2 \to f_{24}$；第 2 轮中的 $f_{13} \to x_3$、$f_{23} \to x_3$、$f_{24} \to x_4$；第 3 轮中的 $x_3 \to f_{34}$ 和第 4 轮中的 $f_{34} \to x_4$）；而沿着有向无环图的反方向，算法执行信念传播（如第 1 轮中的 $x_4 \to f_{34}$、$x_4 \to f_{24}$；第二轮中的 $f_{24} \to x_2$、$f_{34} \to x_3$；第 3 轮中的 $x_3 \to f_{23}$、$x_3 \to f_{13}$ 和第 4 轮中的 $f_{23} \to x_2$、$f_{13} \to x_1$）。此外，只有收到来自所有上游节点的消息的变量节点才可以做出决策。例如，在每一个回合

中，变量节点x_3只能在第 3 轮做决策。这是因为直到第 2 轮结束它才收到来自其上游节点f_{12}和f_{13}的消息。这样，x_3就可以始终基于x_1和x_2的最新取值做出决策。因此，算法的性能被大幅提升，且仅在 3 个回合内就找到了最优解。

3.4.3 理论分析

在 Max-sum_HBVP 中，变量节点做出决策的时机是被严格控制的。因此，每个变量节点做出决策时的信念与其在有向无环图中的位置是相关的。下面，本文首先讨论变量节点做出决策的时机与与其在有向无环图中的位置的关系，再给出变量节点做出决策时的信念的形式化表述。

为了便于讨论，首先引入有向无环图中若干记号。记从一个变量节点x_i到x_j的最长路径为$path_{i,j}$；起点和终点（即没有上游邻居或没有下游邻居的节点）的集合为V；从x_i到函数节点f_{ij}的弧为(i,ij)。特别地，如果从x_i无法到达x_j，则$path_{i,j}=\emptyset$且$|path_{i,j}|=0$。

命题 3.2：在每一个回合中，变量节点x_i始终在第h_i+1轮作出决策，其中$h_i=\max_{v\in V:path_{v,i}\neq\emptyset}|path_{v,i}|$是从起始节点到$x_i$最长路径的长度。

证明：令$v_i^*=\text{argmax}_{v\in V:path_{v,i}\neq\emptyset}|path_{v,i}|$。不失一般性地，假设弧$(ij,i)\in path_{v_i^*,i}$。显然，根据算法伪代码第 8-9 行，$x_i$不可能在$h_i+1$轮前作出决策，因为它至少需要$h_i$轮才能收到上游函数节点$f_{ij}$发过来的消息。

假设x_i在$h_i'+1$轮作出决策，其中$h_i'>h_i^*$。那么一定存在（至少）一个函数节点f_{ik}在第$h_i'-1$轮时给x_i发送消息，且一定存在一个变量节点在$h_i'-2$轮给f_{ik}发送消息。上述分析过程可以递归地应用直到（至少）一个起始节点v_i'发送消息给它的下游邻居。因此，必定存在一条从v_i'到x_i的路径，且该路径的长度为h_i'。这显然与h_i的定义矛盾。因此x_i不可能在h_i+1轮后作出决策。综上，x_i始终在第h_i+1轮作出决策，原命题得证。

因为变量节点x_i需要等待所有上游节点发来的消息来作出决策，所以x_i始终考虑本回合产生的最新的、有效的赋值信息。此外，从命题 3.2 中我们可以总结出x_i也考虑在本回合的h_i+1轮前收到的来自下游节点的消息。具体地，我们有如下性质：

性质 3.2：在 Max-sum_HBVP 中，变量节点x_i在第m回合决策时的信念为：

$$z_i(x_i)=\sum_{n\in P_i}f_{ni}(k_n^m,x_i)+\sum_{\substack{n\in S_i:(i,ni)\in path_{i,v}\\|path_{i,v}|>h_i}}r_{f_{ni}\to x_i}^{m-1}(x_i)+\sum_{\substack{n\in S_i:(i,ni)\in path_{i,v}\\|path_{i,v}|\leq h_i}}r_{f_{ni}\to x_i}^{m}(x_i) \quad (3.9)$$

考虑图 3.3 所示的简单例子。从中不难发现沿着值传播的方向，从起始节点到变量节点x_2的最长路径长度是 2。因此，根据命题 3.2，x_2将会在第m回合的第三轮作出决策。在第二轮结束时，x_2收到了来自f_{12}和f_{23}的消息，但还没有收到本回合中来自f_{24}的消息。因此，x_2作出决策时的信念是

$$z_2(x_2) = f_{12}(k_1^m, x_2) + r^m_{f_{23}\to x_2}(x_2) + r^{m-1}_{f_{24}\to x_2}(x_2) \tag{3.10}$$

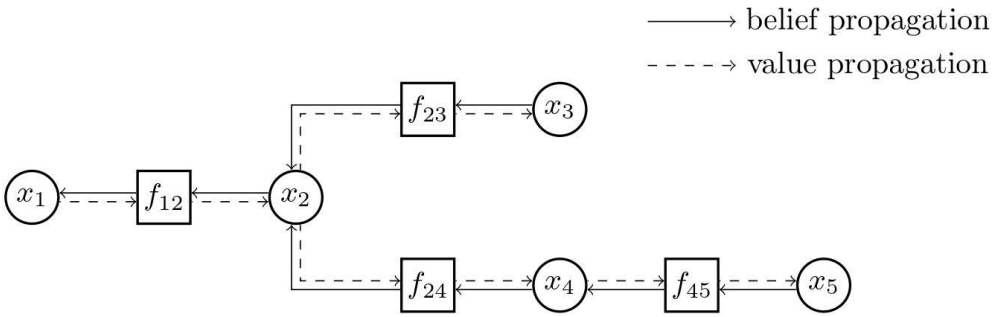


图 3.3 Max-sum_HBVP 执行的简单示例

Fig. 3.3 A simple execution example of Max-sum_HBVP

从性质 3.2 中可以看出，类似于 Max-sum_ADSSVP，Max-sum_HBVP 可以通过考虑局部函数和全局累积信念来跳出局部最优。此外，Max-sum_HBVP 中变量节点在作出决策时，总是考虑有效的取值信息。因此，解的质量的波动就可以被有效地抑制。变量节点同时也有机会考虑到本回合下游邻居产生的消息，而 Max-sum、Max-sum_AD 和 Max-sum_ADVP 均不满足该性质。

Max-sum_HBVP 的求解代价比 Max-sum_ADVP 更小。具体地，给定每个回合（或阶段）的最大轮次k，Max-sum_ADVP 在一个阶段内交换的全部消息的数量是$2k|F|$，其中F是约束函数的集合。与此不同，Max-sum_HBVP 中的节点在一个回合中只发送两次消息（一个方向一次）。因此，Max-sum_HBVP 在一个回合内交换的总消息数量仅为$4|F|$。此外，尽管计算信念传播消息会在函数节点中引入额外的计算代价，本文提出的 Max-sum_HBVP 仍然具有较大的优势，因为在一个回合中每个函数节点仅计算一次信念传播消息。

3.5 基于概率值传播的 Max-sum_AD 算法

3.5.1 算法描述

本节将从随机化的值传播入手去平衡 Max-sum_ADVP 中的探索与利用问题，并提出一种全新的基于概率值传播的 Max-sum_AD 算法（Max-sum_AD with Probabilistic Value Propagation，Max-sum_ADPVP）。具体地，在启用值传播之后，每个函数节点根据一个概率来随机决定本次采用值传播还是信念传播。显然，相比于 Max-sum_ADVP，Max-sum_ADPVP 中的变量节点有机会去考虑累加的信念。因此，本节提出的算法对值传播的初值和时机不敏感，这也就解决了 Max-sum_ADVP 中值传播时机选择的问题。算法 3.3 给出了 Max-sum_ADPVP 的伪代码。

类似于 Max-sum_ADVP，Max-sum_ADPVP 同样在两个换向有向无环图上工作。它们的区别在于函数节点上的计算过程。具体地，相比于启用值传播后 Max-sum_ADVP 中的函数节点始终考虑来自上游节点的取值信息，本文提出的算法仅在满足值传播的概率 p 时才考虑上游节点的取值，并执行值传播（11-12 行），否则执行信念传播（13-14 行）。

算法 3.3 Max-sum_ADPVP 的伪代码

Algorithm 3.3 Sketch of Max-sum_ADPVP

Max-sum_ADPVP(node *n*, probability *p*)

1. $current\_order \leftarrow$ select an order on all nodes in the factor graph;
2. **while** *no termination condition is met* **do**
3. $N_{prev\_n} \leftarrow \{\hat{n} \in N_n : \hat{n} \text{ is ordered before } n \text{ in } current\_order\}$;
4. $N_{follow\_n} \leftarrow N_n \setminus N_{prev\_n}$;
5. **for** *k iterations* **do**
6. collect messages from N_n;
7. **if** *n is a variable node* **then**
8. $x_n^* \leftarrow$ current optimal decision;
9. produce the message $\{q_{n \to n'} x_n^*\}$ to n' using messages received from $N_n \setminus \{n'\}$, $\forall n' \in N_{follow\_n}$;
10. **else if** *n is a function node* **then**:
11. **if** *random()* < *p* **then**:
12. produce the message to n' using the local constraint, assignments received from $N_n \setminus \{n'\}$, $\forall n' \in N_{follow\_n}$;
13. **else:**
14. produce the message to n' using the local constraint, messages received from $N_n \setminus \{n'\}, \forall n' \in N_{follow\_n}$;
15. $current\_order \leftarrow reverse(current\_order)$;

事实上，值传播的概率 p 决定了算法的探索能力。当 p 被设为 1 时，算法具有最强的利用能力且等价于 Max-sum_ADVP。反之，若 p 被设为 0，算法具有最强的探索能力且等价于 Max-sum_AD。因此，考虑到 p 对算法的影响，在算法的过程中对值传播的概率进行自适应更新对算法具有较大的意义。大体上来说，我们希望在优化初期，鼓励算法进行探索来发现更有潜力的解；在算法后期，我们希

望算法更加倾向于利用来保证解的质量。根据上述分析，提出以下四种概率自适应的方式。

① **线性自适应（Linear Adaptation，LA）**。LA 均匀地增加值传播的概率，使得算法从完全探索均匀转换到完全利用。具体地，算法在第m轮的值传播概率由式(3.11)给出。

$$p = \frac{m}{Max\_Iter} \tag{3.11}$$

其中，Max_Iter为最大迭代轮次。

② **正二次适应（Positive Quadratic Adaptation，PQA）**。PQA 以二次于迭代轮次的速率增加值传播概率，且其导数单调增加。相比于 LA，PQA 在前期的较长的一段时间内，值传播概率均维持在较低的水平，并在后期的较短的时间内迅速地增加值传播的概率。因此，该方法在后期较为激进。具体地，算法在第m轮的值传播概率由式(3.12)给出。

$$p = \left(\frac{m}{Max\_Iter}\right)^2 \tag{3.12}$$

③ **负二次适应（Negative Quadratic Adaptation，PQA）**。NQA 以二次于迭代轮次的速率增加值传播概率，且其导数单调减少。相比于 LA，NQA 在前期的较短的一段时间内，迅速地增加值传播的概率，并在后期的较长的时间内维持一个平稳的水平。因此，该方法在前期较为激进。具体地，算法在第m轮的值传播概率由式(3.13)给出。

$$p = -\left(\frac{m}{Max\_Iter}\right)^2 + \frac{2m}{Max\_Iter} \tag{3.13}$$

④ **指数适应（Exponential Adaptation，EA）**。EA 以指数于轮次的速率增加值传播的概率。因此，值传播概率在绝大多数轮次内都维持在一个较低的水平，并在最后极少的轮次内急剧增加。相比于前面三种适应方法，EA 下的值传播概率分布最不均衡。具体地，算法在第m轮的值传播概率由式(3.14)给出。

$$p = e^{\frac{m}{Max\_Iter}-1} \tag{3.14}$$

显然，上面所给出的概率自适应方法均满足如下性质：1）函数连续、可微；2）函数单调递增；3）函数的最小值和最大值分别是 0 和 1。这些性质使得上述适应方法可以有效的平衡探索和利用。

3.5.2 理论分析

Max-sum_ADPVP 的代价大体上介于 Max-sum_AD 和 Max-sum_ADVP 之间。具体地，因为这三个算法都基于换向有向无环图，且具有相同的消息传递规则，因此在给定最大迭代轮次的前提下，Max-sum_ADPVP 与 Max-sum_AD 和 Max-sum_ADVP 具有相同的消息数。因此，该算法的额外代价在于信念传播消息

的计算上。当算法的值传播概率较小时，函数节点更倾向于执行信念传播，从而在计算每一条响应消息时需要$O(d^2)$的操作，这与 Max-sum_AD 的复杂度类似；而当算法的值传播概率越大时，函数节点越倾向于执行值传播，因此每计算一条响应消息的复杂度是$O(d^2)$，这与 Max-sum_ADVP 的复杂度类似。综上所述，当值传播概率越大时，该算法的整体复杂度越低；当值传播概率越小时，该算法的整体复杂度越高，但最高不超过 Max-sum_AD 的复杂度。

3.6 实验结果及分析

3.6.1 实验配置

在本节中，所用到的标准测试问题的种类包括随机 DCOP、无尺度网络和加权图着色。实验问题的具体参数配置如下：

① 随机 DCOP：在该类问题中，Agent 个数 *n*=120，代价选取范围是[1, 100]；各个 Agent 的值域大小为 10。对于稀疏配置，约束图密度$p_1=0.05$；对于稠密配置，约束图密度$p_1=0.6$。

② 无尺度网络：在该类问题中，Agent 个数 *n*=120，代价选取范围是[1, 100]；各个 Agent 的值域大小为 10。在 BA 过程中，初始连通的 Agent 个数m_1为 15，每轮新加 Agent 与已有 Agent 相连的个数为$m_2=3$（稀疏配置）或$m_2=10$（稠密配置）。

③ 加权图着色：在该类问题中，节点个数 *n*=120，违反约束的代价选取范围是[1, 100]；各个节点有 3 种可用的颜色，约束图密度为$p_1=0.05$。

实验所比较的算法包括 Max-sum，Max-sum_AD，Max-sum_ADVP，带消息探索的 Max-sum_ADVP 和衰减 Max-sum（Damped Max-sum）。对于带消息探索的 Max-sum_ADVP，本文根据[10]的实验结果，选择效果最好的组合 *ADVP_AD_ADVP*。对于 Damped Max-sum，本文根据[41]的实验结果，令衰减因子为 0.9。为了保证算法单边收敛，我们设置 Max-sum 类算法的一个阶段的长度为 240 轮。对于所有的 Max-sum 类算法，本文根据[9]引入个体偏好来打破效用相同的问题。在实验中，对于每一个变量节点的每一个可能的取值，随机地从[-0.5, 0.5]中选取偏好。此外，对于运行在有向无环图中且有值传播环节的算法，除非特别指出，本文根据[10]将值传播的时机设置为第三阶段开始时，即第 481 轮后进行值传播。以下所有实验结果都是求解独立的 50 个问题，且每个问题重复求解 30 遍取平均后的结果。

3.6.2 参数调优

第一组实验的目的是探索 Max-sum_ADSSVP(*t*)的求解性能与连续值传播阶段数 *t* 的关系。在本组实验中，使用稀疏配置的随机 DCOP 作为测试问题，并依次测试算法在 *t* 从 1 到 5 时的性能。实验结果如图 3.4 所示。

从实验结果中不难看出，当 *t* 比较小时，算法在从值传播阶段向信念传播阶段转移时，解的质量会大幅度地震荡，如当 *t*=1 时的第 720 轮、1200 轮和 *t*=2 时的第 960 轮等。从另一个方面来说，当 *t* 较大时，解的质量的波动通常较小。例如，当 *t*=1 时，第一次从值传播到信念传播的过渡发生在第 720 轮，且相应的振幅为 14.6%，而当 *t*=5 时，算法从值传播到信念传播的第一次过渡发生在第 1680 轮，此时对应的振幅仅有 10.2%。然而，图 3.4 的结果同时说明了当 *t* 比较小时算法最终可以收到较好的效果。这是由于当 *t* 较大时，算法执行探索（即信念传播阶段）的机会就比较少。例如当 *t*=5 时，算法仅执行了 3 次信念传播；而当 *t*=1 时，算法共执行利率 9 次信念传播。因此，综合实验结果和上述分析，在下面的实验中，本文使用 *t*=2 作为 Max-sum_ADSSVP(*t*)的参数。

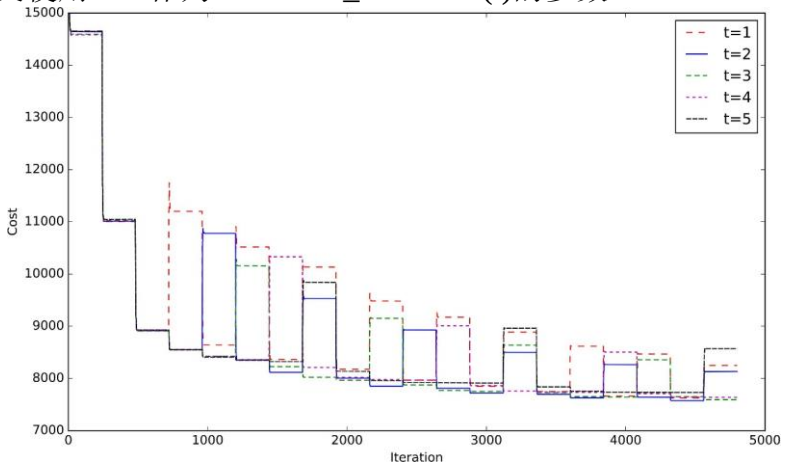


图 3.4 Max-sum_ADSSVP(*t*)算法在不同参数 *t* 下的解的质量

Fig. 3.4 Solution qualities of Max-sum_ADSSVP(*t*) under different *t*

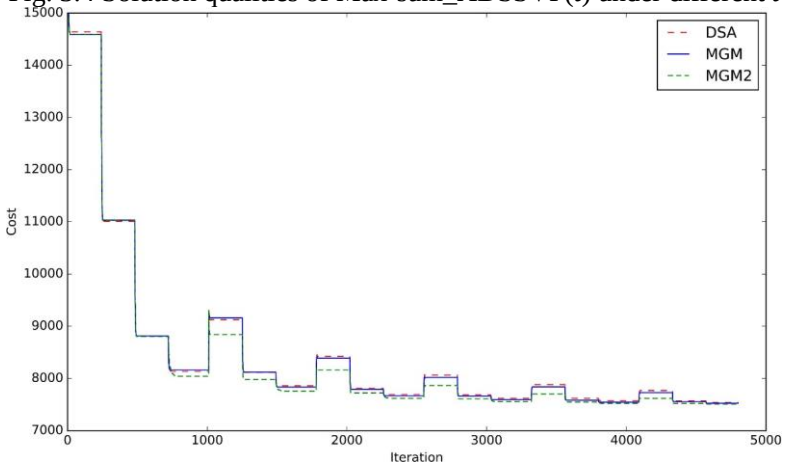


图 3.5 Max-sum_ADSSVP_LS 算法与不同局部搜索算法配合时的解的质量

Fig. 3.5 Solution qualities of Max-sum_ADSSVP_LS under different local search

第二组实验的目的是研究不同局部搜索算法对 Max-sum_ADSSVP_LS 的影响。图 3.5 展示了 Max-sum_ADSSVP_LS 算法分别与 DSA、MGM 和 MGM2 配合时求解稀疏配置的随机 DCOP 的结果。其中，本文设置优化阶段的长度为 50 轮。从图中不难看出，当 Max-sum_ADSSVP_LS 分别与 MGM 和 DSA 配合时解的质量十分类似，而与 MGM2 配合时，解的质量在每一个优化阶段都有了大幅度的提高，且从信念传播到值传播的过渡更加缓和。这同时也从侧面证明了值传播结束时解的质量会对下一阶段的信念传播产生巨大的影响。因此，在下面的实验中，本文使用 MGM2 作为 Max-sum_ADSSVP_LS 算法优化阶段的局部搜索算法。

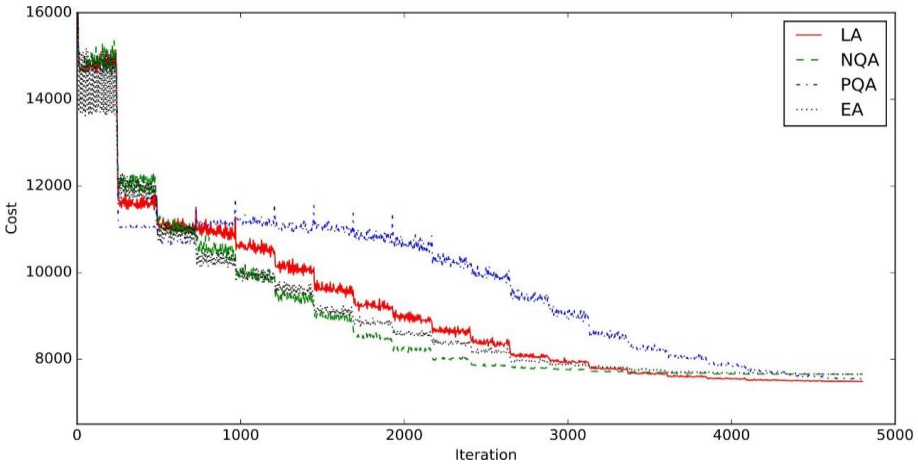


图 3.6 Max-sum_ADPVP 算法在不同自适应方法下的解的质量

Fig. 3.6 Solution qualities of Max-sum_ADPVP under different adaptations

第三组实验的目的是探索 Max-sum_ADPVP 中不同的自适应方式对算法性能的影响。图 3.6 给出了算法在不同自适应方式下求解稀疏配置时的性能。从结果中不难看出，在算法的初始阶段，除了指数适应（EA）的其他自适应方式的震荡都较为轻微。这是因为这些自适应方式在前期对应的值传播概率都比较小。因为在负二次适应（NQA）中，除去开始和结束两个时间点，值传播概率在任何时刻都严格大于线性适应（LA）中的值传播概率，因此 NQA 更倾向于利用，并因此比 LA 更快收敛。相似地，在正二次适应（PQA）中，除去开始和结束两个时间点，值传播概率在任何时刻都严格小于 LA 中的值传播概率，因此 PQA 更倾向于探索，并因此比 LA 更慢收敛。另一方面，EA 在大部分轮次下的值传播概率都维持在一个较低的水平，仅在最后若干轮急剧增加。这种不平衡解的质量大幅度震荡，并最终产生了一个较差的解。综上，在下面的实验中，本文采用 LA 作为值传播概率的自适应方法。

3.6.3 值传播时机对解的质量的影响

本节将通过实验来证明本章所提出的算法对值传播时机不敏感，而 Max-sum_ADVP 严重依赖值传播时机。在实验中，我们分别评估了算法在第 1 阶段到第 5 阶段开启值传播时，求解稀疏配置的随机 DCOP 的解的质量。其中，Max-sum_HBVP 由于从第 1 阶段就必须开始值传播，因此其值传播时机不可调，不在本节的讨论范围之内。图 3.7 给出了各个算法在不同值传播时机下的 Anytime 求解结果。

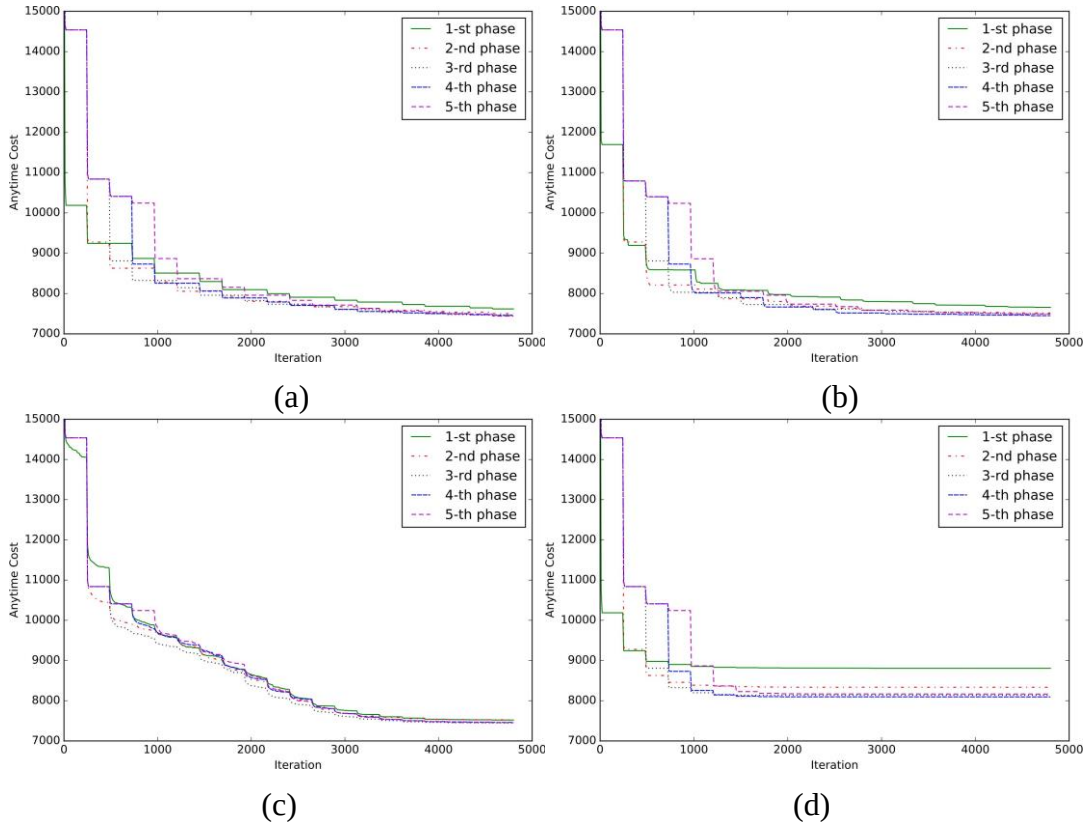


图 3.7 各算法在不同值传播时机下的解的质量：(a) Max-sum_ADSSVP(*t*=2)的 Anytime 解的质量；(b) Max-sum_ADSSVP_MGM2 的 Anytime 解的质量；(c) Max-sum_ADPVP_LA 的 Anytime 解的质量；(d) Max-sum_ADVP 的 Anytime 解的质量；

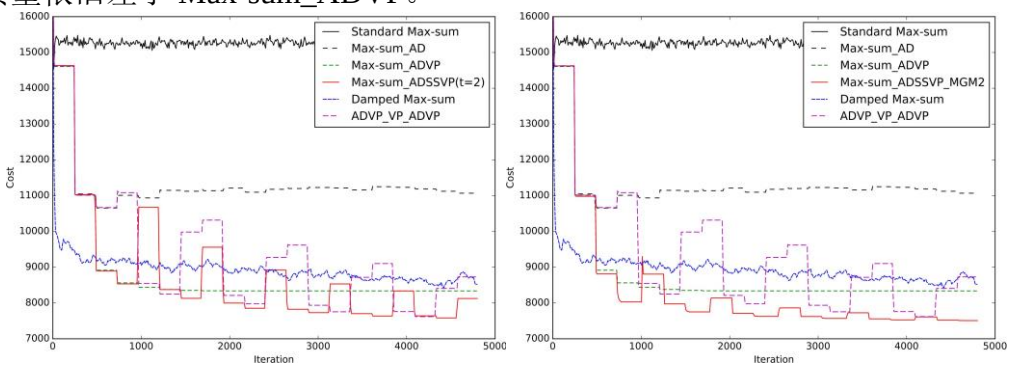
Fig. 3.7 Anytime solution qualities of the algorithms under different value propagation timings: (a) anytime solution qualities of Max-sum ADSSVP(t = 2), (b) anytime solution qualities of Max-sum ADSSVP with MGM2, (c) anytime solution qualities of Max-sum ADPVP with linear adaptation, (d) anytime solution qualities of Max-sum ADVP

从实验结果中可以看出，本章所提出的算法在不同的值传播时机下均取得了相似的性能，而 Max-sum_ADVP 在不同值传播时机下解的质量相差巨大。这就

说明了本章提出的算法对值传播的时机不敏感，从而克服了 Max-sum_ADVP 中值传播时机选择的问题。出现上述现象的主要原因在于，在本章提出的算法中，即使启用值传播之后，Agent 决策时仍然有机会考虑累加的全局信念。换句话说，Agent 不完全根据局部利益作出决策，因此可以缓解邻居的差的取值对决策的影响，并最终跳出局部最优。另一方面，在开启值传播并达到第一次单向收敛后，Max-sum_ADVP 单调地对解进行优化，即 Agent 完全依赖于邻居的取值和本地的约束关系作出决策，并最终陷入局部最优。因此，Max-sum_ADVP 对初值和值传播时机都非常敏感。此外，从实验结果可以发现，所有的算法都在第 4 阶段开始值传播时能取得最好的效果。但必须指出的是，最佳的值传播时机是和问题高度相关的，且通常是一个经验值。

3.6.4 性能比较

本节将比较本章提出的算法和当前最新的算法在求解标准测试问题时性能的优劣。图 3.8 给出了各个算法在求解稀疏配置的随机 DCOP 时的性能。从实验结果中可以得出 Max-sum 不能收敛，且始终无法得到较好的解。Max-sum_AD 尽管可以保证单阶段收敛，但也无法产生较高质量的解。其原因在于 Max-sum 和 Max-sum_AD 都没能消除无效取值假设并打破效用相同。Max-sum_ADVP 在第 2 阶段（即 481 轮）开启值传播，并在第 721 轮后单调地对解进行优化。因为值传播可以消除无效取值假设，并打破效用相同，Max-sum_ADVP 最终得到了一个相对好的解。然而，Max-sum_ADVP 无法在第 6 阶段结束后（即第 1480 轮）对解进行任何优化。这说明了其最终陷入了局部最优。*ADVP_AD_ADVP* 试图通过利用 Max-sum 类算法的不同种消息来平衡探索与利用，并缓解上述问题。然而，该算法没能有效地抑制解的质量的波动，致使即使在算法结束时，解的质量依旧非常不稳定。这是由于该算法在一个回合内始终执行两次连续的信念传播，导致后面一次的信念传播完全基于前一次的信念传播的消息进行计算，最终使得 Agent 无法利用前一次值传播的结果。Damped Max-sum 试图通过降低环路传播的信息来增加 Max-sum 算法收敛的可能性。但就上述结果来说，Damped Max-sum 的解的质量依旧差于 Max-sum_ADVP。



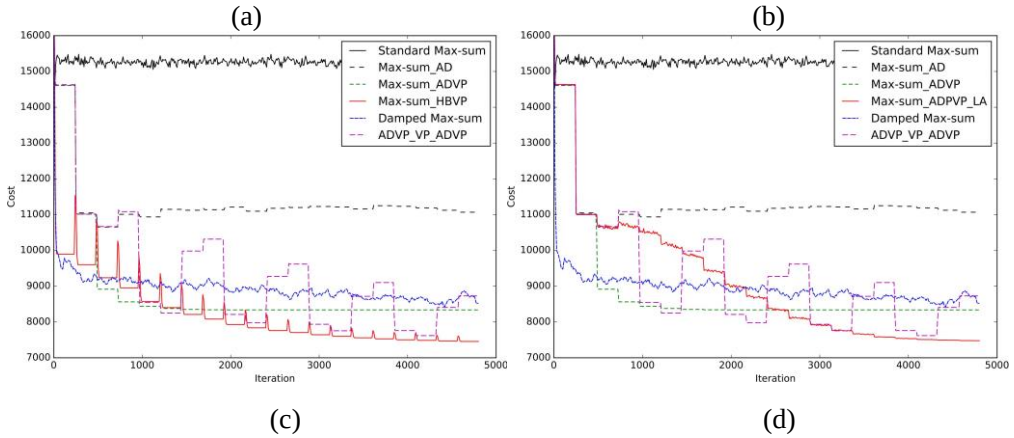


图 3.8 各算法在求解稀疏配置的随机 DCOP 的解的质量

Fig. 3.8 Solution qualities on sparse random DCOPs

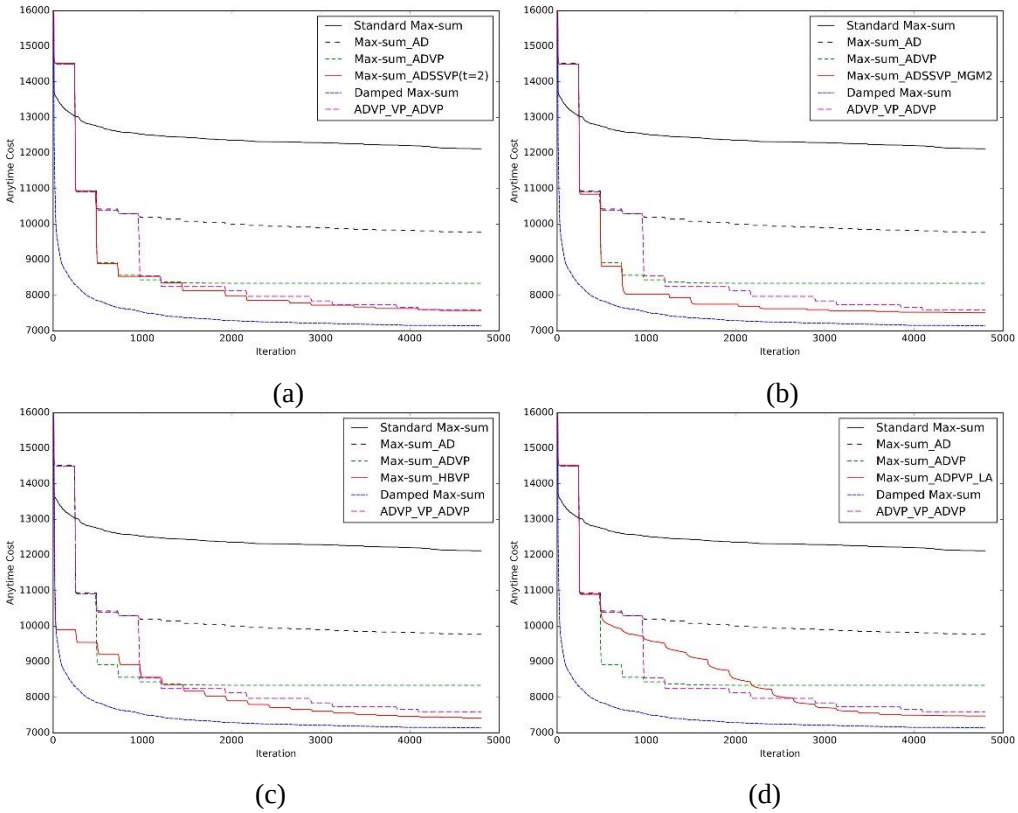


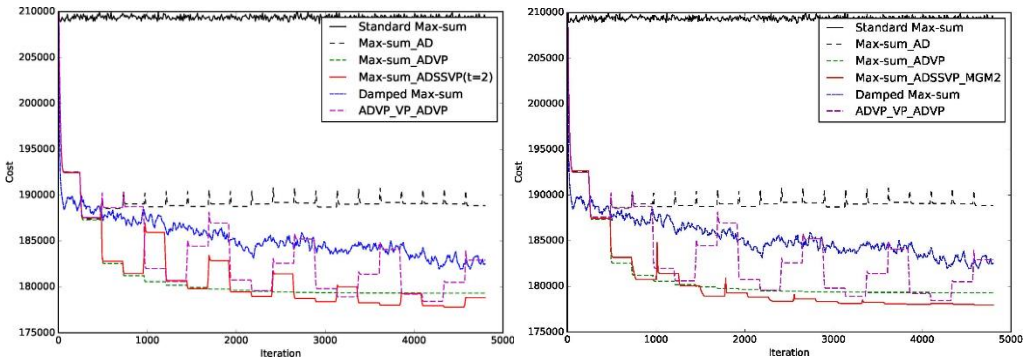
图 3.9 各算法在求解稀疏配置的随机 DCOP 的 Anytime 解的质量

Fig. 3.9 Anytime solution qualities on sparse random DCOPs

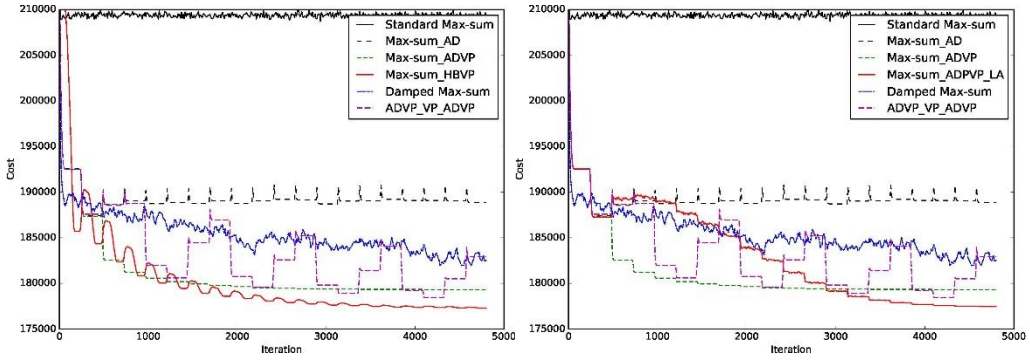
相反，本章提出的算法均起到了平衡探索与利用的作用，并极大地提高了 Max-sum_ADVP 的性能。从图 3.8 (a)和(b)中可以看出，Max-sum_ADSSVP(t=2)和 Max-sum_ADSSVP_MGM2 在开始值传播后，迭代式地对解进行优化，并最终于第 1250 轮超过所有的竞争者。值得一提的是，尽管本章所提出的算法

Max-sum_ADSSVP 和 *ADVP_AD_ADVP* 均不能保证跨阶段的收敛，前者的解质量波动要明显小于后者。此外，在本章提出的两个 Max-sum_ADSSVP 算法中，随着轮次的增加解质量的波动逐渐减少。这也说明了在这两个算法中，值传播和信念传播可以互相合作来有效压制解质量的波动。从图 3.8 (c)中可以看出，Max-sum_HBVP 最终在第 2400 轮后超过所有竞争者。该图还说明了 Max-sum_HBVP 可能保证单阶段收敛，这是因为该算法严格规定了 Agent 在一个阶段中仅能作出一次决策。根据图 3.8 (d)，Max-sum_ADPVP_LA 在前 960 轮的性能与 Max-sum_AD 类似，随后其性能急剧提升，并最终于第 2600 轮超过所有的竞争者。这是因为算法在前期时，值传播概率较小，因此算法的性能接近于 Max-sum_AD。当值传播概率增加时，算法越来越倾向于利用，并最终等价于 Max-sum_ADVP。

图 3.9 给出了各个算法在求解低密度随机 DCOP 时的 Anytime 性能。从该图中可以看出 Damped Max-sum 优于所有其他的算法，这是因为衰减触发了 Max-sum 的有效探索，且其所探索到的好的结果被 Anytime 机制及时地保存了下来。但是，对比图 3.8，我们可以看出 Damped Max-sum 实际上没能做到平衡探索与利用，因为在没有 Anytime 机制的情况下，其解的质量并不高。此外，图 3.9 (a)和(b)展示了本章提出的算法在每一个值传播和信念传播的回合中都有所提高。这说明了本章所提出的两个 Max-sum_ADSSVP 均能起到平衡探索和利用的效果。根据图 3.9 (c)，Max-sum_HBVP 在第 1680 轮后超过所有除 Damped Max-sum 外的竞争者，这个现象展示了混合信念/值传播可以极大地加速优化进程。值得一提的是，本章所提出的算法的性能分别超过 Max-sum_ADVP 约 9.1%、9.9%、10.9%和 10.2%。



(a) (b)

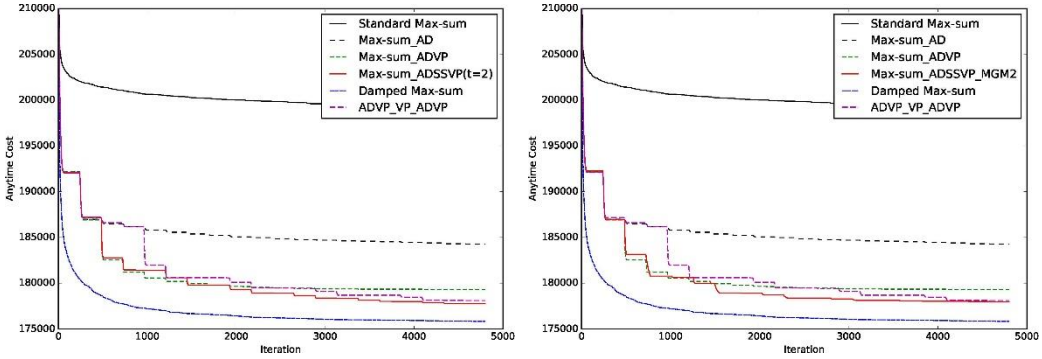


(c)　　(d)

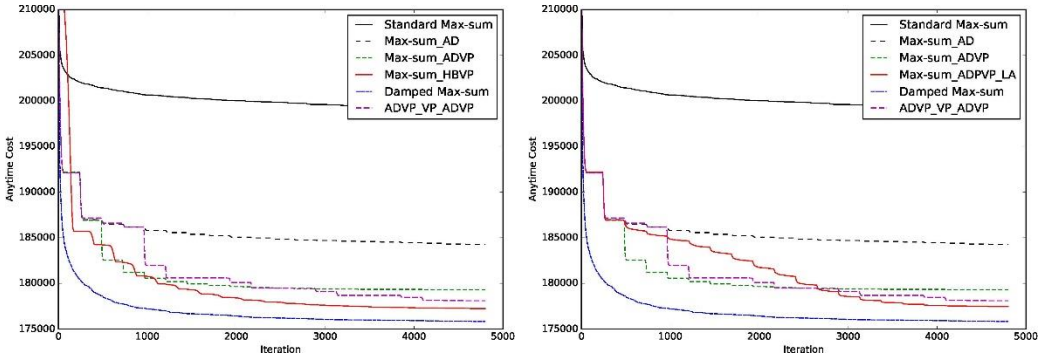
图 3.10　各算法在求解稠密配置的随机 DCOP 的解的质量

Fig. 3.10 Solution qualities on dense random DCOPs

图 3.10 和图 3.11 分别展示了各个算法在求解稠密配置的随机 DCOP 的性能和 Anytime 性能。从中不难看出，其曲线趋势与图 3.8 和图 3.9 类似，只是各个算法的差距缩小了。这部分是因为在稠密配置下，代价基数较高。尽管如此，本文提出的算法仍然较 Max-sum_ADVP 提高 0.8%、0.7%、1.2%和 1.0%。实验结果同时表明，Max-sum_ADSSVP_MGM2 比 Max-sum_ADSSVP(t=2)更快收敛。这是因为相比于值传播，MGM2 可以提供更好的短时优化效果。此外，从图 3.10 (c)中不难看出，在稠密配置下，Max-sum_HBVP 需要更多轮次来达到单阶段收敛。这是因为在稠密图中，有向无环图的直径远大于稀疏配置下直径。因此，Agent 需要等待更多轮才能收到来自所有上游节点的消息，算法因而也需要更多轮次才能收敛。



(a)　　(b)

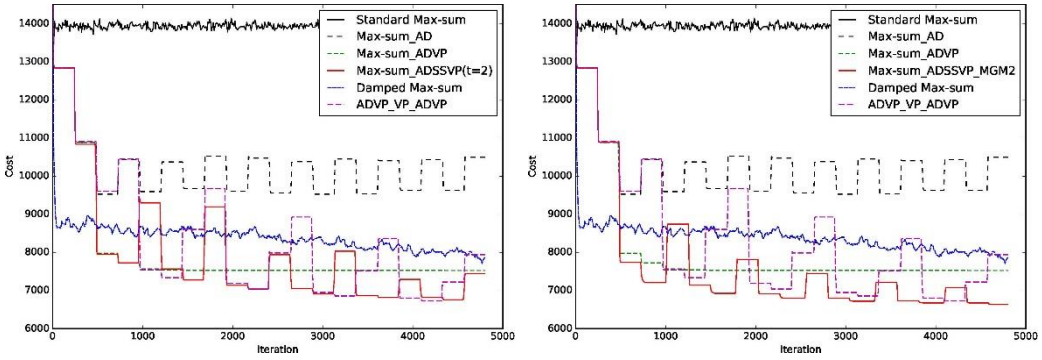


(c) (d)

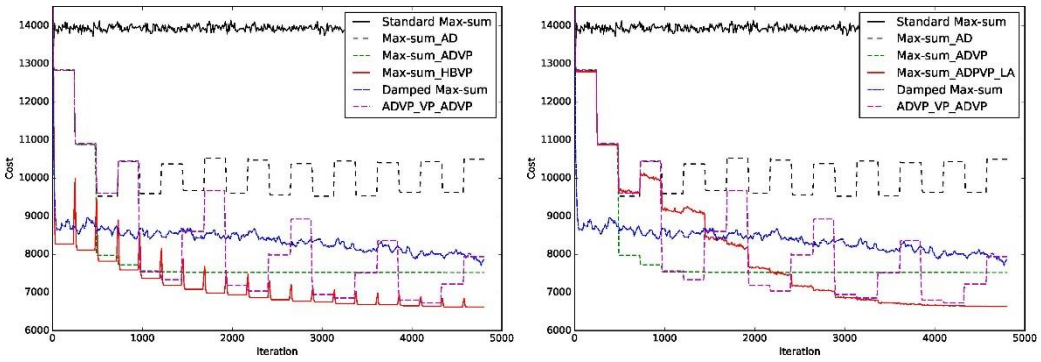
图 3.11 各算法在求解稠密配置的随机 DCOP 的 Anytime 解的质量

Fig. 3.11 Anytime solution qualities on dense random DCOPs

图 3.12 和图 3.13 分别给出了各个算法在求解稀疏配置的无尺度网络的性能和 Anytime 性能。其中，Max-sum 由于存在无效取值假设和效用相同问题，始终无法产生较好的解。Max-sum_AD 尽管在前两个阶段找到了比 Max-sum 更好的解，但在第 3 阶段和后续的阶段中，其解的质量始终在宽幅震荡。Max-sum_ADVP 收敛到一个明显优于 Max-sum_AD 的解上，且在第 3 阶段后单调地对解进行优化。在 Anytime 机制的帮助下，Damped Max-sum 在各轮都严格优于除 Max-sum_HBVP 之外的所有竞争者，但其当前性能（即没有 Anytime 机制时的性能）依旧差于 Max-sum_ADVP。*ADVP_AD_ADVP* 在第 6 阶段及其后续的值传播阶段中可以找到优于 Max-sum_ADVP 的解，但它无法抑制从值传播阶段到信念传播阶段的解的质量的波动。在 Anytime 机制下 Max-sum_ADSSVP(t=2) 和 Max-sum_ADSSVP_MGM2 分别于第 1440 轮和 770 轮时找到了比 Max-sum_ADVP 更优的解，并最终分别超过 Max-sum_ADVP 10.2%和 11.8%。Max-sum_HBVP 在经过 4 次混合执行信念传播和值传播后，超过 Max-sum_ADVP，并最终提升 12.5%。Max-sum_ADPVP 中随机的本质使得变量节点不仅仅只是通过考虑局部约束来作出决策，因此，在 Anytime 机制的帮助下，Max-sum_ADPVP_LA 在第 1920 轮后超过 Max-sum_ADVP，并最终超出 11.7%。



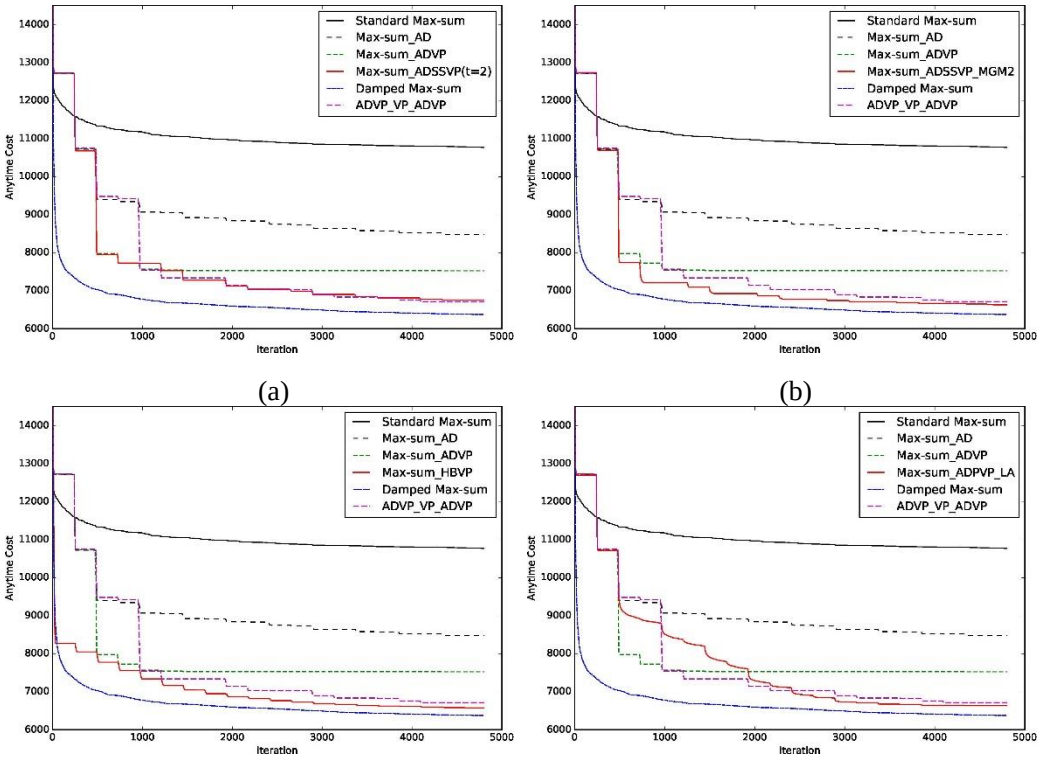
(a) (b)



(c)　(d)

图 3.12　各算法在求解稀疏配置的无尺度网络问题的解的质量

Fig. 3.12 Solution qualities on sparse scale-free networks



(a)　(b)

(c)　(d)

图 3.13　各算法在求解稀疏配置的无尺度网络问题的 Anytime 解的质量

Fig. 3.13 Anytime solution qualities on sparse scale-free networks

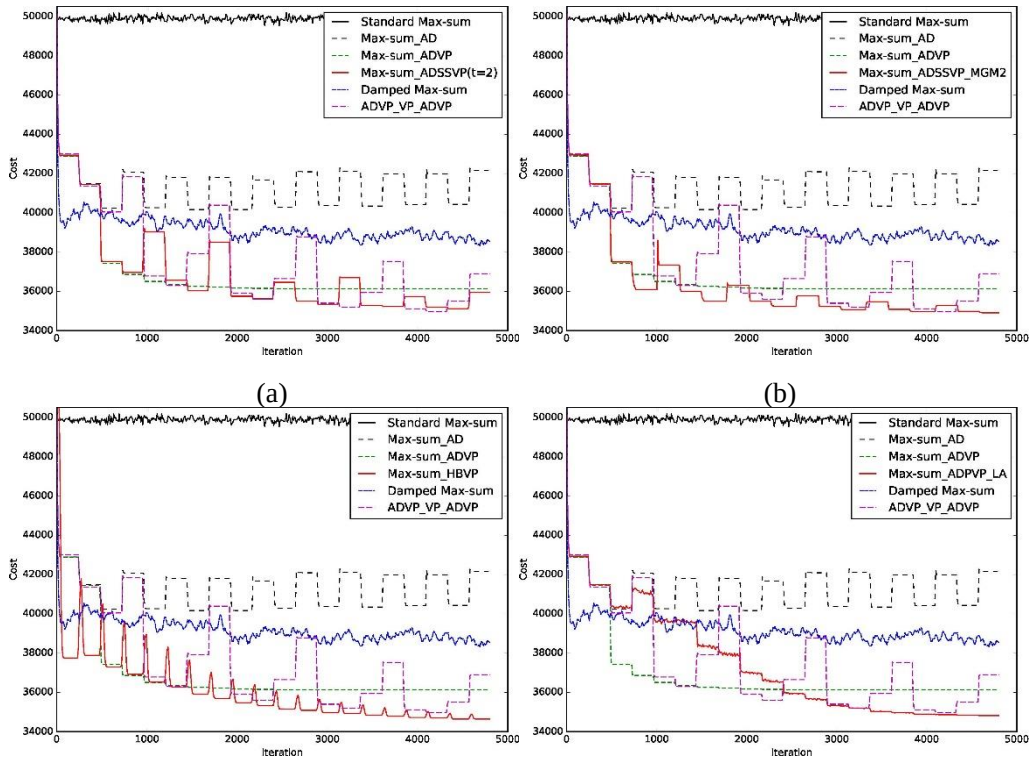


图 3.14 各算法在求解稠密配置的无尺度网络问题的解的质量

Fig. 3.14 Solution qualities on dense scale-free networks

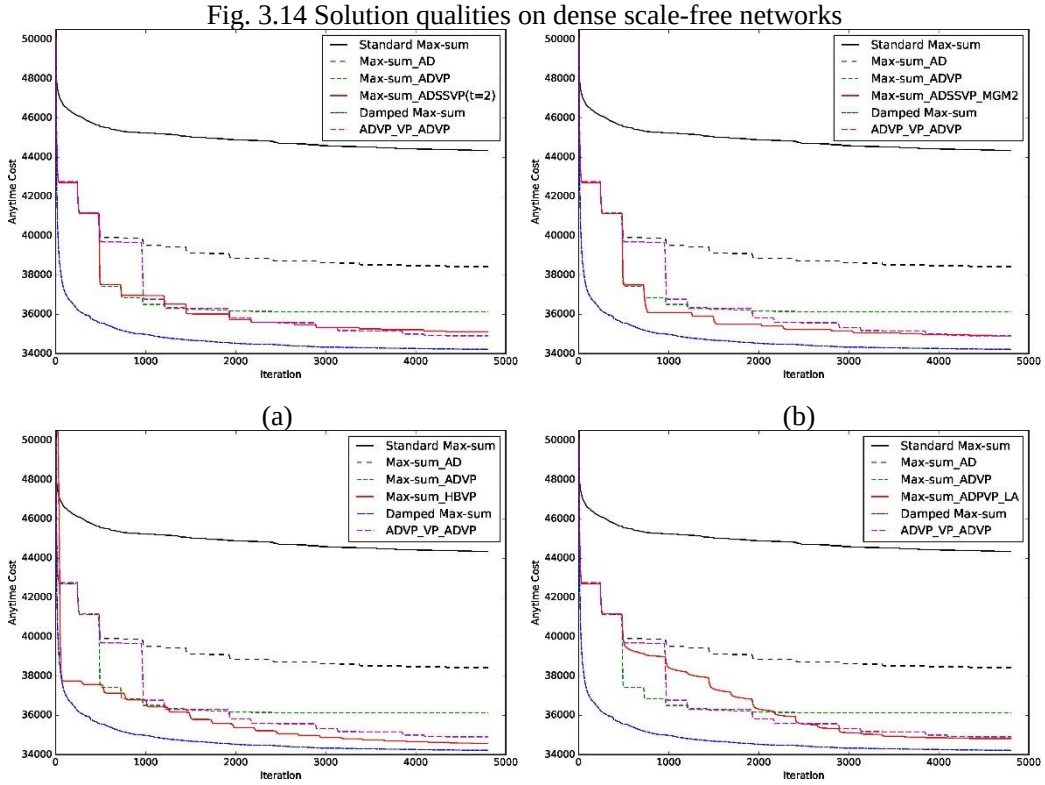


图 3.15 各算法在求解稠密配置的无尺度网络问题的 Anytime 解的质量

Fig. 3.15 Anytime solution qualities on dense scale-free networks

图 3.14 和图 3.15 给出了各算法在求解稠密配置的无尺度网络问题的实验结果。与前面的实验结果类似，Max-sum 依旧无法收敛，并始终无法产生较高质量的解。Max-sum_AD 尽管在第 3 阶段后得到了比 Max-sum 好的解，但它在后续的轮次中，无法对该解进行进一步的优化。根据图 3.15，Max-sum_ADVP 在第 3 阶段后可以找到比 Max-sum、Max-sum_AD 和 Damped Max-sum 更优的解，并在后续的阶段中，缓慢地单调优化该解。但是，它最终在第 7 阶段后陷入局部最优，并无法对解进行任何优化。Max-sum_ADVP 在启用值传播后，严格优于 Damped Max-sum。但在 Anytime 机制的帮助下，Damped Max-sum 很快找到了最好的解。*ADVP_AD_ADVP* 在第 8 阶段优于 Max-sum_ADVP。与求解稀疏配置的无尺度网络类似，Max-sum_ADSSVP(t=2)和 Max-sum_ADSSVP_MGM2 分别于第 1440 轮和第 770 轮得到比 Max-sum_ADVP 更优的 Anytime 结果，并最终分别超过 Max-sum_ADVP 2.8%和 3.4%。在 Anytime 机制下，Max-sum_HBVP 在第 1300 轮后超过 Max-sum_ADVP，并最终超出 Max-sum_ADVP 4.3%。值得注意的是，相比于图 3.10 (c)，Max-sum_HBVP 在求解稠密偏置的无尺度网络时可以更快达到单阶段的收敛。这是因为无尺度网络的直径通常远远小于普通无结构拓扑的直径。Max-sum_ADPVP_LA 可以通过均匀增加值传播概率来更精细化地平衡探索与利用，并于 2160 轮后超过 Max-sum_ADVP。Max-sum_ADPVP_LA 最终超过 Max-sum_ADVP 3.6%。

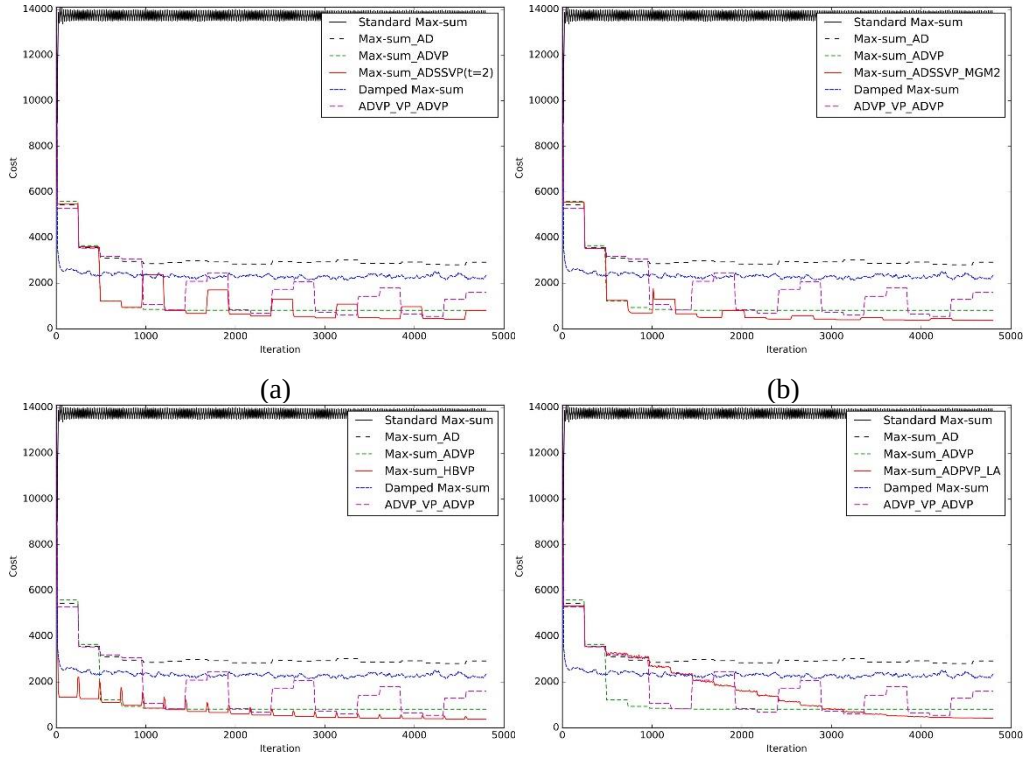


图 3.16 各算法在求解加权图着色问题的解的质量

Fig. 3.16 Solution qualities on weighted graph-coloring problems

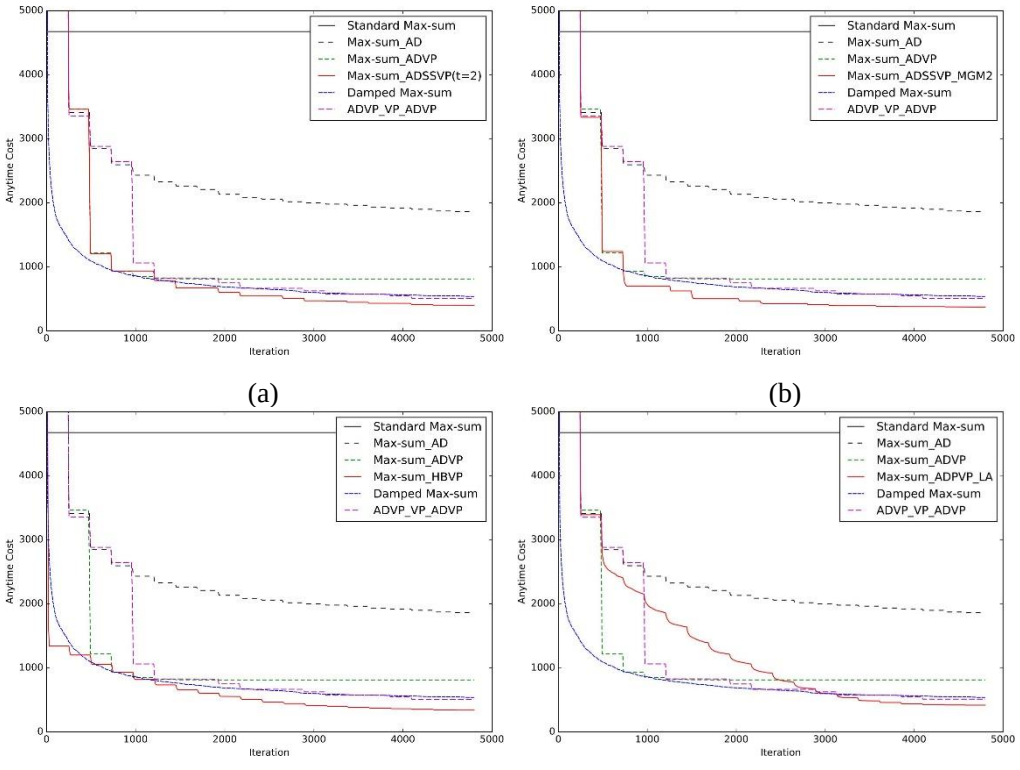


图 3.17 各算法在求解加权图着色问题的 Anytime 解的质量

Fig. 3.17 Anytime solution qualities on weighted graph-coloring problems

图 3.16 和图 3.17 分别给出了各个算法在求解加权图着色问题时的性能和 Anytime 机制下的性能。在随机个体偏好的帮助下，Max-sum 和 Damped Max-sum 可以打破相同信念，并在算法初期找到比其他运行在换向有向无环图上的竞争者。但是，正如 2.2.4 节所讨论的那样，Max-sum 中的 Agent 无法传播除个体偏好之外的任何有用信息。因此，这些 Agent 只能随机地作出决策，并因此无法得到较优的解。与此相反，尽管 Max-sum_AD 也是通过个体偏好打破效用重复，但由于其严格控制了环路信息的传播，它取得了明显优于 Max-sum 的解。Max-sum_ADVP 在开始值传播后超过了 Max-sum_AD，但同时很快陷入了局部最优。*ADVP_AD_ADVP* 在第 10 阶段优于 Max-sum_ADVP，但其在后续的阶段中无法对解进行任何的优化。在 Anytime 机制下，Max-sum_ADSSVP(t=2) 和 Max-sum_ADSSVP_MGM 分别在第 1440 轮和第 770 轮超过所有竞争对手，并最终提高 Max-sum_ADVP 49.2%和 52.5%。此外，值得一提的是，在 Anytime 机制下 Max-sum_HBVP 在算法运行的初期很快就找到优于所有竞争者的解并最终超过 Max-sum_ADVP 57.1%，这再一次证明了信念传播和值传播的混合执行可以极大地加速优化的进程。Max-sum_ADPVP_LA 通过逐渐增加值传播概率来对解进

行优化，并最终于第 3120 轮超过所有竞争对手。该算法最终超过 Max-sum_ADVP 59.1%。

3.7 本章小结

在本章中，我们首先从理论上分析了值传播对 Max-sum 类算法的影响。我们证明了尽管值传播可以极大地改善解的质量，但它阻碍了全局信念的传播。因此，Max-sum_ADVP 中的 Agent 只能完全根据本地函数作出决策。换句话说，这些 Agent 的行为等价于一个贪心的局部搜索算法。在开始值传播后，该算法也因此成为一个完全利用的算法。为了克服上述缺陷，我们在本章中提出了几种基于非连续值传播的 Max-sum 算法。这些算法可以有效地平衡探索和利用。

我们的第一次尝试是交替执行信念传播和值传播并提出一个全新的算法 Max-sum_ADSSVP。在这种模式下，值传播始终沿着一个方向执行。因此，Agent 在决策时可以考虑一部分本地函数（即个体利益）和一部分累积信念（即全局利益）。不幸的是，该算法不再保证单调性和跨阶段的收敛性。为了减轻从值传播阶段到信念传播阶段过渡时解质量的波动，我们在切换到信念传播阶段之前充分对解进行优化，并由此提出 Max-sum_ADSSVP(t)和 Max-sum_ADSSVP_LS 两个算法。其中，前者利用值传播的单调性，通过连续执行 t 阶段对解进行优化；而后者则通过局部搜索算法在短时间内对解进行快速优化。

此外，我们同时注意到过时的取值信息同样会导致解质量的波动。因此，我们同时在一个有向无环图的两端同时执行值传播和信念传播，并提出 Max-sum_HBVP 算法。在该算法中，变量节点仅在收到来自所有上游邻居的取值信息后才进行决策，这就保证了每次决策时所参考的取值信息都是最新、有效的。此外，值传播和信念传播的混合执行也极大地加速了算法的优化进程。本章提出的最后一个基于非连续值传播的 Max-sum 算法是 Max-sum_ADPVP。在该算法中，函数节点根据一个值传播概率 p 随机地决定是否执行值传播。此外，我们还提出了四种自适应方式来动态调整值传播概率的值。

本章的实验结果表明，本章所提 4 个算法在各种测试问题上均优于当前最新的 Max-sum 类算法。特别地，本章所提出的算法明显优于 Max-sum、Max-sum_AD 和 Max-sum_ADVP。这是由于在本章所提出的算法中，Agent 始终可以考虑本地函数和累积的信念。相反，其他算法中的 Agent 只能考虑本地函数（如 Max-sum_ADVP），或只能考虑累积的信念（如 Max-sum 和 Max-sum_AD 等）。此外，本章的实验结果同时表明，本章所提出的算法对值传播的时机不敏感，这也就克服了 Max-sum_ADVP 中需要选择值传播时机的重要缺陷。

4 求解非对称约束优化问题的搜索-推理混合算法

4.1 引言

由于 ADCOP 在 DCOP 的基础上引入了个体偏好和非对称性，在求解 ADCOP 时，不仅要考虑算法的性能，还应把隐私性作为算法的重要指标。事实上，如果将一个 ADCOP 中所有 Agent 的私有约束共享，那么问题将转化为一个 DCOP，并因此所有现有的 DCOP 算法均可解决该问题。但是在这个过程中，Agent 的个体偏好完全泄露，ADCOP 的建模优势将不复存在。因此，研究如何在保证 Agent 的隐私性的同时，保证算法的性能是一个极富挑战性的课题。

为了得到解的完整代价，现有的完备 ADCOP 搜索算法通常采用直接暴露的方式获取 Agent 的私有代价。因此，Agent 的隐私性完全依赖于问题的代价结构和算法的剪枝性能。然而，现有的完备 ADCOP 搜索算法均不能起到较好的剪枝作用。当问题较为复杂时，整个系统所泄露的隐私接近 50%。这就意味着近一侧约束代价完全被泄露。**因此，研究如何更高效地剪枝不仅是提高算法性能的必由之路，也是保证隐私性的重要手段。**

从另外一个方面来说，隐私性也限制了完备推理算法在求解 ADCOP 中的应用。例如在 DPOP 中，Agent 需要知道与所有上层节点（即父节点和伪父节点）之间的约束代价函数才能进行本地消元，并将消元后的效用继续向上传播。而在 ADCOP 中，一个 Agent 无法、也不应该获得上层节点的私有约束代价函数，并因此无法针对全部约束函数进行优化求解。**换句话说，完备推理算法无法直接用于 ADCOP 的求解。**

本章基于上述分析，创新性地提出了一种全新的基于搜索-推理混合的完备求解算法。在推理阶段，利用 DPOP 算法求解速度较快这一优势，预先求解一面约束，并存储每个分支的推理结果。在搜索阶段，将传统基于链式结构的 SBB 算法扩展到伪树上，并将求解结果作为搜索过程的下界。因此，相比于传统的完备搜索算法，本章所提出的算法具有更强的并发性和更紧的下界，因而具有更加的性能和隐私性。**该算法同时也是目前为止，完备推理算法首次应用于求解 ADCOP。**

4.2 基于伪树的求解非对称约束优化问题的搜索-推理混合算法

针对上述问题，提出了一种基于伪树的求解非对称约束优化问题的搜索-推理混合算法（Pseudo Tree-Inference/SBB，PT-ISBB）。在该算法中，推理阶段和搜索阶段共用一棵伪树。在推理阶段，Agent 需要保存每一分支的推理结果；在搜索

阶段，Agent 需要记录每一分支的搜索结果及位置。具体地，算法中所涉及到的重要变量如下：

① f_{ij}：Agent a_i的关于a_j的私有约束函数；

② $local\_util_i$：a_i与其上层邻居（即$PP_i \cup \{P_i\}$）间的私有约束函数的联合，即$local_util_i = \bigotimes_{a_p \in PP_i \cup \{P_i\}} f_{ip}$；

③$Subtree_utils_i^c$：a_i的孩子a_c的推理结果；

④ Cpa：当前部分解（Current Partial Assignment），包含了所有祖先节点的取值；

⑤ $Opt_i^c(v_i)$：在某个Cpa下，a_i的孩子a_c针对a_i取值为v_i时所搜索到的最优代价；

⑥ $Srch\_val_i^c$：在某个Cpa下，a_i的孩子a_c所探索到a_i值域的位置。

算法 4.1 给出了 PT-ISBB 的伪代码。

算法 4.1 PT-ISBB 的伪代码

Algorithm 4.1 Sketch of PT-ISBB

PT-ISBB For Agent a_i

```
1.  when Initialize:
2.    if a_i is a leaf then
3.      send UTIL (min_{x_i} local_util_i) to P_i;
4.  when receive UTIL (subtree_util) from a_c:
5.    Subtree_utils_i^c ← subtree_util;
6.    local_util_i ← local_util_i ⊗ subtree_util;
7.    if a_i has received all UTIL message from C_i then
8.      if a_i is the root then
9.        InitVariables();
10.       v_i ← the first element in D_i;
11.       Cpa ← (x_i = v_i);
12.       foreach a_c' ∈ C_i do
13.         Srch_val_i^c' ← v_i;
14.         send CPA ({Cpa, ∞}) to a_c';
15.     else
16.       send UTIL (min_{x_i} local_util_i) to P_i;
17. when receive CPA ({Cpa, rcv_bound}) from P_i:
```

PT-ISBB For Agent a_i

18. Store $\{Cpa, rcv\_bound\}$;
19. InitVariables();
20. $v_i \leftarrow$ the first element in D_i;
21. **while** $v_i \neq null \wedge rcv\_bound - \text{SingleSideCost}(v_i) - \text{SubtreeLB}(v_i) \leqslant 0$ **do**
22. $v_i \leftarrow$ the next element in D_i;
23. **if** $v_i \neq null$ **then**
24. send REQUEST_COST (Cpa_p, v_i) to a_p, $\forall a_p \in PP_i \cup \{P_i\}$;
25. $wait\_list_i(v_i) \leftarrow C_i$;
26. **else**
27. send BACKTRACK (∞) to P_i;
28. **when** receive REQUEST_COST (v_i, v_c) from a_c:
29. send COST ($\{f_{ic}(v_i, v_c), v_c\}$) to a_c;
30. **when** receive COST ($\{cost, v_i\}$) from a_p:
31. $High\_cost_i(v_i) \leftarrow High\_cost_i(v_i) + cost$;
32. **if** a_i has received all COST messages from $PP_i \cup \{P_i\}$ for v_i **then**
33. $lw\_bound_i \leftarrow \min(\text{LocalBound}(), rcv\_bound)$;
34. $ub_i(v_i) \leftarrow lw\_bound_i - \text{SingleSideCost}(v_i) - \text{SubtreeDelta}(v_i) - High\_cost_i(v_i)$;
35. **if** $ub_i(v_i) - \text{SubtreeLB}(v_i) \leqslant 0$ **then**
36. $v_i' \leftarrow \text{NextFeasibleAssignment}(v_i)$;
37. **if** $v_i' \neq null$ **then**
38. send REQUEST_COST (Cpa_p, v_i) to a_p, $\forall a_p \in PP_i \cup \{P_i\}$;
39. $wait\_list_i(v_i') \leftarrow wait\_list_i(v_i)$;
40. **else**
41. $Srch\_val_i^c \leftarrow null$, $\forall a_c \in wait\_list_i(v_i)$;
42. **if** $Srch\_val_i^c = null, \forall a_c \in C_i$ **then**
43. send BACKTRACK ($\text{LocalBound}()$) to P_i;
44. **else**
45. **if** a_i is a leaf **then**
46. $v_i \leftarrow \text{NextFeasibleAssignment}(v_i)$;
47. **if** $v_i \neq null$ **then**
48. send REQUEST_COST (Cpa_p, v_i) to a_p, $\forall a_p \in PP_i \cup \{P_i\}$;
49. **else**
50. send BACKTRACK ($\text{LocalBound}()$) to P_i;

PT-ISBB For Agent a_i

51. **else**
52. $TmpCpa \leftarrow Cpa \cup (x_i = v_i)$;
53. **foreach** $a_c \in wait\_list_i(v_i)$ **do**
54. $Srch\_val_i^c \leftarrow v_i$;
55. send CPA($\{TmpCpa, ub_i(v_i)\}$) to a_c;
56. **when** receive BACKTRACK ($cost^*$) from a_c:
57. $Opt_i^c(Srch\_val_i^c) \leftarrow cost^*$;
58. $v_i \leftarrow$ the element next to $Srch\_val_i^c$;
59. $lw\_bound_i \leftarrow \min(\text{LocalBound}(), rcv\_bound)$;
60. **while** $v_i \neq null$ **do**
61. **if** v_i is infeasible **then**
62. $Opt_i^{c'}(v_i) \leftarrow \infty, \forall a_{c'} \in C_i$;
63. $v_i \leftarrow$ the next element in D_i;
64. **continue**;
65. **if** a_i hasn't requested costs for v_i **then**
66. send REQUEST_COST (Cpa_p, v_i) to a_p, $\forall a_p \in PP_i \cup \{P_i\}$;
67. $wait\_list_i(v_i) \leftarrow \{a_c\}$;
68. **break**;
69. **if** a_i hasn't received all COST messages from $PP_i \cup \{P_i\}$ for v_i **then**
70. $wait\_list_i(v_i) \leftarrow wait\_list_i(v_i) \cup \{a_c\}$
71. **break**;
72. $ub_i(v_i) \leftarrow lw\_bound_i - \text{SingleSideCost}(v_i) - \text{SubtreeDelta}(v_i) - High\_cost_i(v_i)$;
73. **if** $ub_i(v_i) - \text{SubtreeLB}(v_i) \leqslant 0$ **then**
74. $Opt_i^c(v_i) \leftarrow \infty$;
75. $v_i \leftarrow$ the next element in D_i;
76. **else**
77. $TmpCpa \leftarrow Cpa \cup (x_i = v_i)$;
78. $Srch\_val_i^c \leftarrow v_i$;
79. send CPA($\{TmpCpa, ub_i(v_i)\}$) to a_c;
80. **break**;
81. **if** $v_i = null$ **then**
82. $Srch\_val_i^c \leftarrow null$;
83. **if** $Srch\_val_i^c = null, \forall a_c \in C_i$ **then**

PT-ISBB For Agent a_i

send BACKTRACK ($\mathrm{LocalBound}()$) to P_i;

function InitVariables ():

$Opt_i^c(v_i) \leftarrow \infty, \forall a_c \in C_i, v_i \in D_i$;

$High\_cost_i(v_i) \leftarrow 0, \ \forall v_i \in D_i$;

clear all cost request flags;

clear all infeasible flags;

function NextFeasibleAssignment (v_i):

$v_i \leftarrow$ the next element in D_i;

while $v_i \neq null \wedge lw\_bound_i - \mathrm{SingleSideCost}(v_i) - \mathrm{SubtreeLB}(v_i) \leqslant 0$ **do**

mark v_i as infeasible;

$v_i \leftarrow$ the next element in D_i;

return v_i;

function SingleSideCost (v_i):

return $\sum_{a_p \in \mathrm{PP}_i \cup \{P_i\}} f_{ip}(v_i, Cpa_p)$;

function SubtreeLB (v_i):

return $\sum_{a_c \in C_i : Opt_i^c(v_i) = null} Subtree\_utils_i^c(v_i, Cpa)$;

function SubtreeDelta (v_i):

return $\sum_{a_c \in C_i : Opt_i^c(v_i) \neq null} Opt_i^c(v_i)$;

function LocalBound ():

$lowest\_bound \leftarrow \infty$;

foreach $v_i \in D_i$ **do**

if $Opt_i^c(v_i) \neq null, \forall a_c \in C_i$ **then**

$tmp\_bound \leftarrow \sum_{a_c \in C_i} Opt_i^c(v_i) + High\_cost_i(v_i) + \mathrm{SingleSideCost}(v_i)$;

$lowest\_bound \leftarrow \min(tmp\_bound, lowest\_bound)$;

return $lowest\_bound$;

算法由叶子节点向父节点发送消元后的单面效用开始（2-3 行）。当父节点收到来自子节点的效用消息（UTIL 消息）后，其先将推理结果存储在数据结构 $Subtree\_utils_i$ 中（5 行），以备搜索阶段查询使用，并将子节点的推理结果与本地效用联合（6 行）。如果该节点收到了来自所有子节点的效用消息并且它不是根节

点的话，它将继续向它的父节点传送消元后的单面效用（7 行，15-16 行）。当根节点收到来自所有孩子节点的推理结果后，算法的推理阶段结束。

算法的搜索阶段由根节点触发。具体地，在推理阶段结束后，根节点首先初始化相关变量（9 行），然后将值域中的第一个取值通过 CPA 消息发送给各个子节点，并更新$Srch_val_i$中各个子节点搜索位置的记录（10-14 行）。其中，由于是第一次进行搜索，当前解的上界是无穷大。当一个节点收到来自父节点的 CPA 消息时，它首先存储收到的部分解Cpa和当前搜索上界rcv_bound（18 行）并初始化相关变量（19 行）。然后利用单边代价（SingleSideCost）和子树下界（SubtreeLB）结合当前搜索上界来确定值域中首个可行的取值（20-22 行）。其中，SingleSideCost (v_i)给出了当$x_i = v_i$时，a_i与所有上层邻居间的单面私有代价（96-97 行）；而 SubtreeLB (v_i)则给出了a_i的所有子节点中尚未返回关于$x_i = v_i$的搜索结果的子节点（即$\{a_c \in C_i | Opt_i^c(v_i) = null\}$）的代价下界（98-99 行），该下界由推理结果$Subtree_utils_i$给出。如果当前存在一个可行的取值，则向其上层邻居请求对方的单面代价（即 REQUEST_COST 消息），并将向下继续传 CPA 消息的动作暂时挂起，即将其所有孩子加入$wait_list_i$（23-25 行）。否则说明值域中不存在满足当前上界的取值，于是直接向父节点回溯（26-27 行）。当节点收到来自其子节点的的代价请求消息后，它通过直接暴露的形式将其一侧的对应代价通过代价消息（即 COST 消息）发送给子节点（28-29 行）。

当节点收到来自上层节点的代价消息后，它首先将上层节点的私有代价累加（31 行）。如果该节点已经收到来自所有上层节点关于当前取值v_i的代价消息，则该节点则可以计算关于取值v_i的本地上界（LocalBound），并与父节点给出的搜索上界相比较，取其小者为当前的上界（32-33 行）。其中，LocalBound 给出了由a_i子节点搜索结果得出的最小搜索上界（102-108 行）。因此，关于取值v_i的搜索上界$ub_i(v_i)$是当前上界减去取值v_i所造成的与上层邻居间的代价（即 HighCost (v_i) + SingleSideCost (v_i)）和关于v_i子树已经搜索完毕的结果 SubtreeDelta (v_i)（34 行）。其中，SubtreeDelta (v_i)给出了a_i的所有子节点中已经返回关于$x_i = v_i$的搜索结果的子节点（即$\{a_c \in C_i | Opt_i^c(v_i) \neq null\}$）汇报的代价之和（100-101 行）。如果$ub_i(v_i)$减去当前的子树下界后小于 0，则说明当前取值不可行，算法转而去找下一个可行解v_i'（35-36 行），即找到下一个满足当前上界、单面代价和子树预估的取值（90-95 行）。如果存在这样的v_i'，则继续向上层邻居请求关于v_i'，并将关于v_i的等待列表转移到v_i'上（37-39 行）。否则说明在当前上下文下，值域中不存在可行解。因此需要将关于v_i的等待列表中所有子节点的搜索位置记录重置，并判断执行回溯（40-43 行）。如果$ub_i(v_i)$减去子树下界后大于 0，则说明取值v_i可以继续向下搜索。如果当前节点不是叶子节点，它首先扩展部分解，更新该值对应

的等待列表中的子节点的搜索位置记录，并继续向这些子节点发送部分解消息（51-55 行）。否则它直接向其上层邻居请求下一个可行解的私有代价，如果不存在这样可行解，则直接向父节点回溯（45-50 行）。

当节点收到来自子节点的回溯消息时，它首先存储子节点汇报的搜索结果 $cost^*$（57 行），并为其选择下一个要探索的取值（58-80 行）。如果不存在这样的取值，节点需要判断是否所有子节点都搜索完毕，若是，则继续向上回溯（81-84 行）。具体地，算法首先切换取值 v_i 到值域中下一个元素并更新当前搜索上界（58-59 行）。然后算法从取值 v_i 开始找到第一个可行解。其中，当某个取值已经被标记为不可行时，填充 Opt_i 中所有关于该取值的结果为无穷大并跳过该取值（61-64 行）；当节点尚未向其父节点请求某个取值的私有代价时，向父节点发起关于该取值的代价请求，并将子节点加入该值对应的等待列表后挂起后续操作（65-68 行）。类似地，当节点已经向父节点请求私有代价，但尚未收到来自所有上层邻居的代价时，该节点把子节点加入该取值对应的等待列表中，并挂起后续操作（69-71 行）。如果某个取值同时满足在当前上下文下可行，且节点已经收到来自所有上层邻居关于该取值的私有代价，则进一步计算关于该取值的搜索上界（72 行）。如果该上界减去子树下界后仍然大于 0，则说明该值可以被进一步探索。于是该节点扩展部分解，更新子节点的搜索位置记录并发送 CPA 消息给子节点（76-80 行）。否则说明该值无需探索，继续检查下一个可能的取值（73-75 行）直至遍历完该节点的值域空间。如果不存在可以继续探索的取值（即 $v_i = null$），则更新子节点的搜索位置记录为空，并判断其他分支是否完成搜索。如果所有分枝都完成搜索，则该节点向其父节点回溯（81-84 行）。

4.3 实例分析

下面以图 2.2 所示的 ADCOP 为例，描述算法执行的过程。根据算法的 1-3 行、15-16 行，叶子节点 a_1 和 a_2 向其父节点发送消元后的单面效用并开启算法的推理阶段。具体地，在推理阶段所发送的全部消息如下：

第 1 轮：$a_2 \to a_3 : u_{2\to 3}(x_3, x_4) = \min_{x_2}(f_{23} \otimes f_{24})$　$a_1 \to a_3 : u_{1\to 3}(x_3) = \min_{x_1} f_{13}$

第 2 轮：$a_3 \to a_4 : u_{3\to 4}(x_4) = \min_{x_3}(f_{34} \otimes u_{2\to 3}(x_3, x_4) \otimes u_{1\to 3}(x_3))$

此时，根节点 a_4 收到了来自所有子节点的效用消息，推理过程结束。各个节点的子树推理结果（即 $Subtree\_utils_i^c$）如下：

x_3	0	1
	1	3

(a) $Subtree\_util_3^1$

x_4 / x_3	0	1
0	9	10
1	4	6

(b) $Subtree\_util_3^2$

x_4	0	1
	10	12

(c) $Subtree\_util_4^3$

图 4.1 各个 Agent 所存储的子树推理结果

Fig. 4.1 Subtree utilities for agents

值得一提的是，图 4.1 仅给出了a_3和a_4的子树推理结果。这是因为a_1和a_2是叶子节点，因此没有孩子节点向它们发送效用消息。根据算法的 8-14 行，a_4通过向a_3发送 CPA 消息来开始算法的搜索阶段。该消息包含了a_4的第一个取值和当前的上界，即，

第 3 轮：$a_4 \rightarrow a_3$:$\{(x_4=0),\infty\}$

并更新当前a_3的搜索位置记录为$Srch_val_4^3=0$。

当a_3收到来自a_4的 CPA 消息时，根据算法 18-22 行，a_3首先存储当前的部分解和收到的搜索上界，并初始化搜索相关变量。然后找到值域中第一个满足给定搜索上界的取值。因为当前给出的上界是∞，所以a_3的任意取值均可满足该上界。因此，a_3即将探索的取值是$v_3=0$。根据算法 23-25 行，a_3向其父节点a_4请求当$(x_4=0,x_3=0)$时a_4的私有代价，即，

第 4 轮：$a_3 \rightarrow a_4$:$(x_4=0,x_3=0)$

并将其的孩子节点加入关于$v_3=0$的等待列表，即$wait_list_3(0)=\{a_1,a_2\}$。当$a_4$收到来自$a_3$请求代价的消息后，它根据自己的私有代价函数查询在给定的取值对下自己一侧所产生的代价，并通过 COST 消息发送给a_3（29 行），即，

第 5 轮：$a_4 \rightarrow a_3$:$\{5,(x_3=0)\}$

当a_3收到来自a_4的 COST 消息后，它将收到的代价累加到记录上层邻居代价的数据结构中，即$High_cost_3(0)=5$（31 行）。同时，满足 32 行所定义的条件，即此时a_3已经能够计算取值$v_3=0$所导致的a_3与其上层邻居间的完整代价。根据算法 33-34 行，a_3需要计算取值$v_3=0$的搜索上界。由于a_3尚未收到子节点的回溯消息，其当前搜索上界为$lw_bound=\infty$，进而$ub_3(0)=\infty$。显然，$ub_3(0)$减去子树下界后仍然大于 0，说明取值$v_3=0$可以继续探索。因此，算法转而执行 51-55 行。为此，a_3首先扩展部分解，更新$wait_list_3(0)$中的子节点的搜索位置记录为$Srch_val_3^1=0$，$Srch\_val_3^2=0$，并向这些子节点继续发送 CPA 消息，即：

第 6 轮：$a_3 \rightarrow a_1$:$\{(x_4=0,x_3=0),\infty\}$ $a_3 \rightarrow a_2$:$\{(x_4=0,x_3=0),\infty\}$

与a_3收到来自a_4的 CPA 消息类似，当a_1和a_2收到来自a_3的 CPA 消息后，它们根据收到的搜索上界确定要探索的取值为值域中的第一个值，并向它们的上层邻居请求私有代价，即，

第 7 轮：$a_2 \to a_3:(x_3=0,x_2=0)$ $a_2 \to a_4:(x_4=0,x_2=0)$ $a_1 \to a_3:(x_3=0,x_1=0)$

当a_3和a_4收到来自下层邻居的请求代价消息后，查询自己的私有代价函数，并返回对应的代价，即，

第 8 轮：$a_4 \to a_2:\{3,(x_2=0)\}$ $a_3 \to a_2:\{0,(x_2=0)\}$ $a_3 \to a_1:\{3,(x_1=0)\}$

a_1和a_2在收到上层邻居的私有代价后，首先计算当前的搜索上界（33 行）。根据 102-108 行，其本地搜索上界为自己所有已探索完毕的取值中，完整代价最小者。具体地，a_1和a_2的当前搜索上界计算过程如下：

$$\begin{aligned} lw\_bound_1 &= \min(f_{13}(0,0)+High\_cost_1(0),\infty) \\ &= \min(1+3,\infty)=4 \\ lw\_bound_2 &= \min(f_{23}(0,0)+f_{24}(0,0)+High\_cost_2(0),\infty) \\ &= \min(7+3+3,\infty)=13 \end{aligned}$$

根据算法第 34 行，a_1和a_2分别为取值 0 计算上界，计算过程如下：

$$\begin{aligned} ub_1(0) &= lw\_bound_1 - f_{13}(0,0) - 0 - High\_cost_1(0) \\ &= 4-1-3=0 \\ ub_2(0) &= lw\_bound_2 - f_{23}(0,0) - f_{24}(0,0) - 0 - High\_cost_2(0) \\ &= 13-7-3-3=0 \end{aligned}$$

因为上述结果满足 35 行之条件，a_1和a_2转而寻找下一个可行解。根据 90-95 行，a_1和a_2分别评估当取值为 1 时，在现有信息下的（宽松）上界。若该上界小于 0，则说明取值$v_1=1$不可行。具体地，对于a_1，有：

$$lw\_bound_1 - f_{13}(1,0) - 0 = 4-9=-5<0$$

因为取值$v_1=1$不可行，根据 40-43 行，a_1没有任何可以继续探索的取值。因此它向父节点a_3回溯，即，

第 9 轮：$a_1 \to a_3:4$

值得一提的是，因为a_1将取值$v_1=1$剪枝了，这使得a_1不会继续向a_3请求私有代价，因此其隐私性得以保证。至此，a_1完成了在部分解$(x_4=0,x_3=0)$下的搜索。类似地，对于a_2，有：

$$lw\_bound_2 - f_{23}(1,0) - f_{24}(1,0) - 0 = 13-8-1=4>0$$

因为$v_2=1$初步可行，算法转而执行 45-50 行，即继续请求值域中下一个取值关于上层邻居的私有代价，即

第 9 轮：$a_2 \to a_3:(x_3=0,x_2=1)$ $a_2 \to a_4:(x_4=0,x_2=1)$

当a_3和a_4收到来自下层邻居的请求代价消息后，查询自己的私有代价函数，并返回对应的代价，即，

第 10 轮：$a_4 \to a_2$:$\{3,(x_2=1)\}$ $a_3 \to a_2$:$\{1,(x_2=1)\}$

此外，在本轮中，a_3同时收到了来自a_1的回溯消息。根据 57-84 行，a_3首先存储a_1的搜索结果，即：

$$Opt_3^1=[4,null]$$

然后为a_1寻找下一个要探索的取值。因为a_3尚未为取值$v_3=1$向上层邻居a_4请求私有代价，因此满足第 65 行之条件。于是a_3将a_1加入等待列表，即$wait_list_3(1)=\{a_1\}$，并向a_4请求关于$v_3=1$的私有代价：

第 10 轮：$a_3 \to a_4$:$(x_4=0,x_3=1)$

当a_2收到来自上层邻居的私有代价后，首先更新当前搜索上界。根据 102-108 行可知，当$v_2=0$时，其本地上界最小。因此，其当前搜索上界lw_bound_2仍然为 13。进而为取值$v_2=1$计算搜索上界（34 行），过程如下：

$$\begin{aligned} ub_2(1) &= lw\_bound_2 - f_{23}(1,0) - f_{24}(1,0) - 0 - High\_cost_2(1) \\ &= 13-8-1-4=0 \end{aligned}$$

由此，$v_2=1$不用继续探索，算法转而执行 36-43 行。又因为$v_2=1$是其值域中的最后一个取值，算法由此执行分支 40-43 行，即向父节点a_3发送回溯消息，即：

第 11 轮：$a_2 \to a_3$:13

至此，a_2完成了在部分解$(x_4=0,x_3=0)$下的搜索。

当a_4收到来自a_3的代价请求消息后，其查询自己的私有代价函数，并返回对应的代价，即：

第 11 轮：$a_4 \to a_3$:$\{9,(x_3=1)\}$

下面假设a_3先处理来自a_2回溯消息再处理来自a_4的代价消息。根据 57-84 行，a_3首先存储a_2的搜索结果，即：

$$Opt_3^2=[13,null]$$

然后为a_2寻找下一个要探索的取值。因为a_3尚未收到取值$v_3=1$对应的上层邻居代价，因此满足 69 行之条件，并将a_2加入关于$v_3=1$的等待列表中（70 行）。此时，等待列表为$wait_list_3(1)=\{a_1,a_2\}$。至此$a_2$回溯消息已经处理完毕。在处理$a_4$的代价消息时，$a_3$首先更新当前搜索上界。因为目前只有取值$v_3=0$探索完毕，因此其当前搜索上界由下式给出：

$$\begin{aligned} lw\_bound_3 &= \min(f_{34}(0,0)+High\_cost_3(0)+Opt_3^1(0)+Opt_3^2(0),\infty) \\ &= \min(4+5+4+13,\infty)=26 \end{aligned}$$

接着，a_3开始计算$v_3=1$时的搜索上界$ub_3(1)$（34 行）。计算过程如下：

$$ub_3(1) = lw\_bound_3 - f_{34}(1,0) - 0 - High\_cost_3(1)$$
$$= 26 - 6 - 9 = 11$$

然后利用子树下界初步评估$v_3 = 1$是否需要继续探索（35 行）。根据 98-99 行，子树下界由a_1和a_2的推理结果给出，即

$$\text{Subtree\_LB}(1) = Subtree\_utils_3^1(v_3 = 1) + Subtree\_utils_3^2(v_3 = 1, v_4 = 0)$$
$$= 3 + 4 = 7$$

这表明取值$v_3 = 1$在当前部分解下可以继续探索，于是算法执行 51-55 行，即扩展部分解，更新$wait_list_3(1)$中子节点的搜索位置，并发送 CPA 消息给这些子节点，即：

第 12 轮：$a_3 \to a_1$:$\{(x_4 = 0, x_3 = 1), 11\}$　$a_3 \to a_2$:$\{(x_4 = 0, x_3 = 1), 11\}$

a_1和a_2收到来自父节点a_3的消息后，首先在值域中找出在当前部分解下的第一个满足当前接收到的上界的取值（21-22 行）。对于a_1的第一个取值$v_1 = 0$有：

$$rcv\_bound - f_{13}(0,1) - 0 = 11 - 7 = 4 > 0$$

对于a_2的第一个取值$v_2 = 0$有：

$$rcv\_bound - f_{23}(0,1) - f_{24}(0,1) - 0 = 11 - 3 - 3 = 5 > 0$$

因此，a_1和a_2将继续探索取值 0。因为算法执行过程与上文相同，这里不再给出后续算法执行的详细过程。图 4.2 给出了算法结束时，各个节点中Opt的内容。

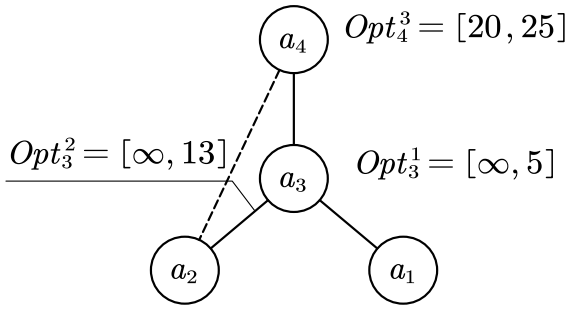


图 4.2 算法结束后各个节点的Opt数据结构中的内容

Fig. 4.2 Opt for each node when the algorithm terminates

4.4 理论分析

4.4.1 完备性证明

为了证明算法的完备性，必须证明以下两点：1) 算法会终止（即不重）；2) 当算法终止时，它能找到全局最优代价（即不漏）。下面分别就上述两个命题进行证明。

引理 4.1：同一个部分解不会重复搜索，即一个节点不可能收到两个相同的部分解消息。

证明：上述命题对于根节点显然成立，因为根节点没有父节点，并因此不会收到部分解消息。当一个节点a_i收到来自父节点的部分解消息后，它（可能）将给它的子节点继续发送若干部分解消息。根据算法的 58 行，对于每一个子节点来说，这些部分解所包含的对a_i的赋值均不相同。因为根节点不可能重复收到部分解，且对于一个部分解，任意一个节点不可能重复发送相同的部分解给其子节点，所以上述命题对于所有节点成立。

引理 4.1 说明了算法不会重复搜索。因此，对于给定的搜索空间，算法一定会终止。下面证明当算法终止时，它能找到全局最优的代价。为了证明这一点，假设存在一个全局代价$cost^*$小于算法所返回的代价$cost$。因为算法没能发现该解，所以该解一定被算法所剪枝。在算法中，剪枝发生在第 21 行、35 行、73 行和 92 行。因为剪枝时，算法始终使用推理的结果作为取值的下界，所以下面将对$Subtree_utils_i$进行讨论。

引理 4.2：对于任意一个部分解Cpa，节点a_i取值为v_i时其任意一个孩子$a_c \in C_i$及其子树（以下称为分支a_c）的任意一个完整赋值的代价均大于该分支的推理结果$Subtree_utils_i^c(v_i, Cpa)$。

证明：下面用数学归纳法证明。因为叶子节点没有子节点，因此不再命题讨论范围内。

归纳基础：当a_i的孩子节点均为叶子节点时，对于其每一个分支a_c，有：

$$\begin{aligned} cost_c &= \sum_{a_p \in PP_c \cup \{a_i\}} f_{cp}(v_c, v_p) + \sum_{a_p \in PP_c \cup \{a_i\}} f_{pc}(v_p, v_c) \\ &\geq \min_{v_c'} \sum_{a_p \in PP_c \cup \{a_i\}} f_{cp}(v_c', v_p) + \sum_{a_p \in PP_c \cup \{a_i\}} f_{pc}(v_p, v_c) \\ &= Subtree\_utils_i^c(v_i, Cpa) + \sum_{a_p \in PP_c \cup \{a_i\}} f_{pc}(v_p, v_c) \\ &> Subtree\_utils_i^c(v_i, Cpa) \end{aligned} \tag{4.1}$$

因此，归纳基础成立。

现假设上述命题对$a_c \in C_i$成立，下面证明该命题对a_i成立。考虑分支a_c，其代价为：

$$cost_c = \sum_{a_p \in PP_c \cup \{a_i\}} f_{cp}(v_c, v_p) + \sum_{a_p \in PP_c \cup \{a_i\}} f_{pc}(v_p, v_c) + \sum_{a_{c'} \in C_c} cost_{c'} \tag{4.2}$$

根据归纳假设，式(4.2)中等号右边最后一项大于每一个推理结果$Subtree_utils_c^{c'}(v_i, Cpa)$，因此有：

$$\begin{aligned} cost_c > & \sum_{a_p \in PP_c \cup \{a_i\}} f_{cp}(v_c, v_p) + \sum_{a_p \in PP_c \cup \{a_i\}} f_{pc}(v_p, v_c) \\ & + \sum_{a'_c \in C_c} Subtree\_utils_c^{c'}(v_c, Cpa) \\ \geq & \min_{v'_c} \left(\sum_{a_p \in PP_c \cup \{a_i\}} f_{cp}(v'_c, v_p) + \sum_{a'_c \in C_c} Subtree\_utils_c^{c'}(v'_c, Cpa) \right) \\ & + \sum_{a_p \in PP_c \cup \{a_i\}} f_{pc}(v_p, v_c) \end{aligned} \tag{4.3}$$

根据推理过程（算法第 6 行及第 16 行），式(4.3)中不等号右边第一项即为推理结果$Subtree_utils_i^c(v_i, Cpa)$，因此有：

$$\begin{aligned} cost_c > & Subtree\_utils_i^c(v_i) + \sum_{a_p \in PP_c \cup \{a_i\}} f_{pc}(v_p, v_c) \\ > & Subtree\_utils_i^c(v_i) \end{aligned} \tag{4.4}$$

因此，上述命题成立。

引理 4.3：对于任意一个部分解，a_i子树的任意一个代价大于lw_bound_i的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子树的赋值为Spa_i，其对应代价为$\text{cost}(Spa_i)$。上述命题等价于证明如果一个Spa_i满足$\text{cost}(Spa_i) > lw_bound_i$，则对于$a_i$的父节点$a_j$一定有$\text{cost}(Spa_j) > lw_bound_j$，其中$Spa_i \subset Spa_j$。

事实上，lw_bound_i的来源有两处：由子节点的搜索结果给出（即 LocalBound）和由父节点给出（即 *rcv_bound*）。当lw_bound_i由a_i的子节点的搜索结果给出时，说明一定存在一个比Spa_i更优的赋值，因此不会向上回溯Spa_i。而当lw_bound_i由父节点a_j的 CPA 消息给出时，根据算法第 34 行和第 72 行，有：

$$\begin{aligned} lw\_bound_j = & lw\_bound_i + \sum_{a_p \in PP_j \cup \{P_j\}} f_{jp}(v_j, v_p) + f_{pj}(v_p, v_j) \\ & + \sum_{a_c \in C_i : Opt_j^c(v_j) \neq null} Opt_j^c(v_j) \end{aligned} \tag{4.5}$$

所以，当$\text{cost}(Spa_i) > lw_bound_i$时，一定有$\text{cost}(Spa_j) > lw_bound_j$。

引理 4.2 说明了算法的下界始终小于任意一个子树的完整赋值代价，因此算法不会错误地将最优解剪枝；引理 4.3 说明了局部次优解一定不会出现在完整的最优解中。综上，PT-ISBB 是完备的。

4.4.2 复杂度分析

算法的复杂度主要体现在四个方面：1) Agent 的存储复杂度；2) Agent 的计算复杂度；3)消息大小；4)消息数量。因为 PT-ISBB 中 Agent 需要保存每一个子树的推理结果，而每一个子树的推理结果又和子树的诱导宽度[20]相关。此外，Agent 还需保存 *Opt*、*Srch_val*、*High_cost* 等数据结构，因此，a_i的整体存储复杂度为

$O(|C_i|d^m+|C_i||D_i|)$。其中，d 为a_i所有与其后代节点相约束的祖先节点的最大值域长度，m 是以a_i为根节点的子树的诱导宽度。

在推理阶段，算法所做的计算等价于 DPOP 的计算。因此处理一条 UTIL 消息的复杂度为$O(d^m)$。在搜索阶段，Agent 收到一条 CPA 消息时会去找第一个满足当前部分解和搜索上界的取值，如果该值存在，则需要向其上层邻居请求私有代价。因此，a_i 最差情况下处理一条 CPA 消息的时间复杂度为$O(|D_i|+|PP_i\cup\{P_i\}|)$。当 Agent 收到代价请求消息后，它直接查表并返回，因此其时间复杂度为$O(1)$。当 Agent 收到代价响应消息后，Agent 需要更新其当前搜索上界，这需要$O(|D_i||C_i|)$的复杂度；而它的其他操作均需要$O(|D_i|)$、$O(|C_i|)$或者$O(|PP_i\cup\{P_i\}|)$的复杂度，因此 Agent 处理一条代价响应的时间复杂度为$O(|D_i||C_i|+|PP_i\cup\{P_i\}|)$。类似地，当 Agent 收到来自子节点的回溯消息后，也需要更新当前搜索上界，因此其时间复杂度为$O(|D_i||C_i|)$。除此之外，Agent 还需要遍历值域为子节点找到下一个可以继续探索的取值或者继续向其上层邻居请求私有代价，因此其处理回溯消息时的整体时间复杂度为$O(|D_i||C_i|+|PP_i\cup\{P_i\}|)$。

与其他基于搜索的完备算法类似，本章所提出的算法在最差情况下需要线性于 Agent 数的效用消息和指数于问题规模（如 Agent 数、值域大小等）的其他消息。因为子树的效用消息包含了与子树有约束关系的所有上层节点的所有赋值组合对应的最优代价，因此一个效用消息的大小为$O(d^m)$。对于 CPA 消息，因为它要存储所有 Agent 的取值，因此它的大小为$O(|A|)$。因为代价请求和响应消息以及回溯消息均只包含所请求代价的赋值对或代价，因此它们的大小均为$O(1)$。

4.5 实验评估

4.5.1 实验目的及配置

本节共有四组实验，实验目的和配置如下：

① 实验 1：不同 Agent 数下各算法的性能比较。该项实验的目的是从 Agent 数的角度来测试在问题规模逐渐增加的情况下，各个算法的性能。该实验采用随机 ADCOP 作为测试问题，其中各 Agent 值域大小为 3，代价选取范围是[1,100]；固定约束图密度为 0.2，Agent 数分别从 10 到 16 变化。

② 实验 2：不同密度下各算法的性能比较。该项实验的目的是从密度的角度来测试在问题规模逐渐增加的情况下，各个算法的性能。该实验采用随机 ADCOP 作为测试问题，其中各 Agent 值域大小为 3，代价选取范围是[1,100]；固定 Agent 数为 8，密度从 0.3 到 0.9 变化。

③ 实验 3：不同紧度下各算法求解满足问题的性能比较。该项实验的目的是从满足问题的紧度的角度来测试在问题规模逐渐增加的情况下，各个算法的性能。

该实验采用随机 ADCSP 作为测试问题，其中各 Agent 值域大小为 10， Agent 数为 10，图密度为 0.4；紧度从 0.1 到 0.9 变化。

④ 实验 4：不同紧度下各算法求解满足问题的隐私性比较。该项实验的目的是测试算法在求解不同紧度的满足问题时所泄露隐私的多少。该实验采用随机 ADCSP 作为测试问题，其中各 Agent 值域大小为 10， Agent 数为 10，图密度为 0.4；紧度从 0.1 到 0.5 变化。

在上述实验中，实验 1~3 所使用的性能指标是消息数和 *NCCCs*，实验 4 所使用的性能指标是隐私损失。实验所比较的算法包括 SyncABB 和 ATWB。因为一阶段的 SyncABB 性能优于两阶段的 SyncABB，所以在下面的实验中我们使用一阶段的 SyncABB。每个算法的求解时限是 10 分钟，超时即终止算法的运行。以下所有实验结果都是求解独立的 25 个问题，且每个问题重复求解 10 遍取平均后的结果。

4.5.2 实验结果及分析

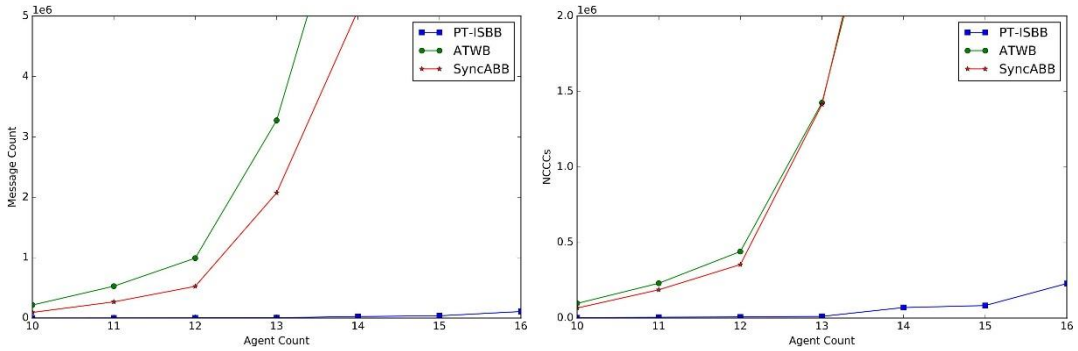


图 4.3 不同 Agent 数下各算法的性能比较

Fig 4.3 Performances under different agent count

图 4.3 给出了各个算法在不同的 Agent 数下的性能。从图中可以发现，随着 Agent 数目的增加，SyncABB 和 ATWB 求解问题所需的消息数和 *NCCCs* 呈指数级增长。这是因为增加 Agent 数会使得整个问题的搜索空间呈指数级增长。此外，不难看出对于相同规模的问题，SyncABB 和 ATWB 所需的消息数和 *NCCCs* 明显多于本章提出的 PT-ISBB，并最终只能求解不超过 14 个 Agent 的问题。而随着问题规模的增加，PT-ISBB 的消息数和 *NCCCs* 增长缓慢，最终可以求解所有的问题。造成这种现象的主要原因有两个：首先，SyncABB 和 ATWB 都是在链式结构中将问题的搜索空间进行有序的展开，每个时刻仅有一个 Agent 在工作，这就使得算法的并行性较差，最终使得其消息数较多。另一方面，链式结构完全忽略了问题本身的拓扑结构，这就导致了一个 Agent 先后收到的两个不同的部分解中可能会包含相同的上层邻居取值，并最终使得其 *NCCCs* 较多。换句话说，链式结构

会导致多次代价的重复计算。而在 PT-ISBB 中，问题在每一个分支节点都被划分成若干个更小的子问题，而这些子问题可以并行地求解。此外，由于伪树结构充分地考虑了问题的拓扑结构，最大限度地避免上述重复计算的问题。

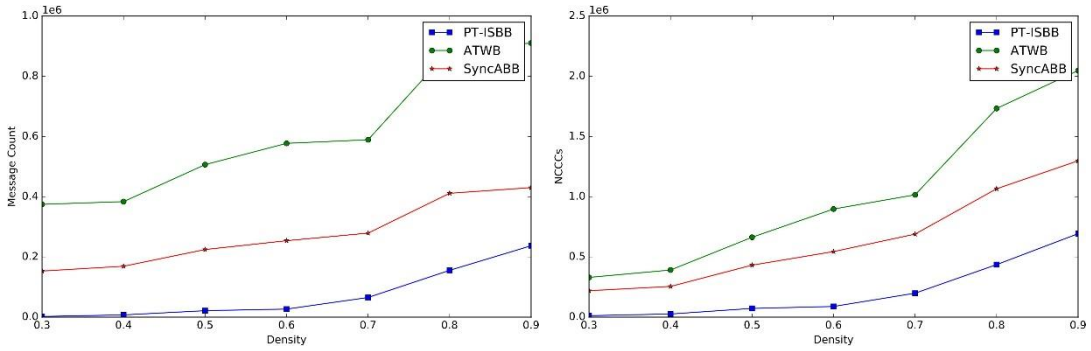


图 4.4 不同密度下各算法性能的比较

Fig. 4.4 Performances under different density

图 4.4 给出了各个算法在不同密度下的性能。与图 4.3 类似，PT-ISBB 在消息数和 *NCCCs* 上均优于 SyncABB 和 ATWB，但随着密度的增加，PT-ISBB 和 SyncABB 的差距在减小。这是因为增加图密度并不会增加问题的搜索空间，只是增加了算法的剪枝难度。因此，SyncABB 在不同密度下的性能变化不十分剧烈。而由于 PT-ISBB 使用伪树作为其通信结构，其并发性依赖于伪树的形态。但是，随着问题密度的增加，所形成的伪树结构趋近于单链结构。这就导致 PT-ISBB 在求解高密度问题时优势不明显。

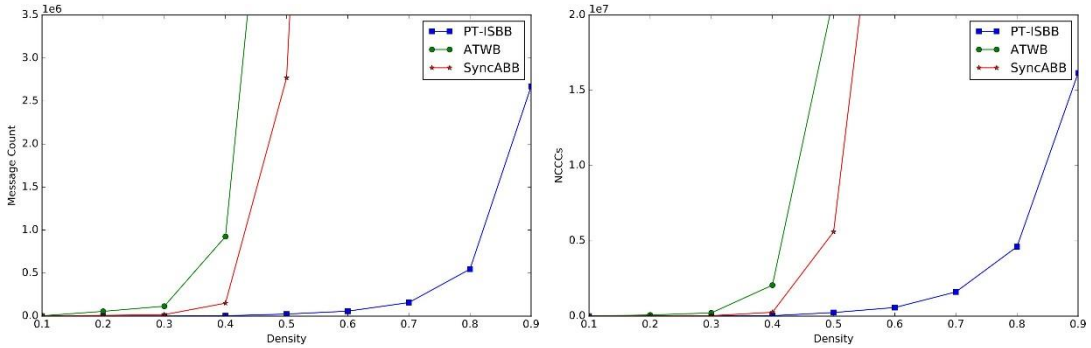


图 4.5 不同紧度下各算法求解随机 ADCSP 问题的性能

Fig. 4.5 Performances under different tightness of Random ADCSP

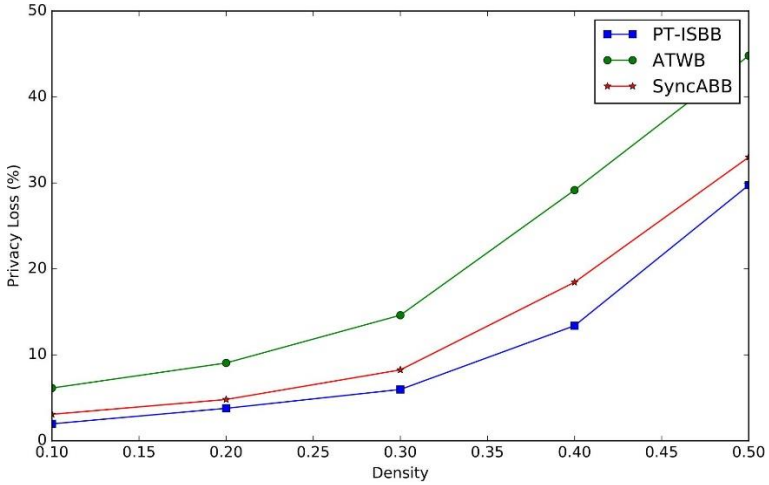


图 4.6 不同紧度下各算法求解随机 ADCSP 问题的隐私损失

Fig. 4.6 Privacy loss under different tightness of Random ADCSP

图 4.5 和图 4.6 分别给出了各个算法在求解不同紧度下的随机 ADCSP 的性能比较和隐私损失的比较。从图 4.5 中可以看出，随着紧度的增加，问题的复杂度越来越高，因此各个算法的剪枝难度也越来越大。特别地，当紧度分别超过 0.3 和 0.4 后，ATWB 和 SyncABB 所需的消息数和 *NCCCs* 急剧增加，并最终无法求解紧度超过 0.5 的问题。而 PT-ISBB 由于使用了并发性较强的伪树结构，且使用了子树的推理结果作为取值的下界来加速剪枝，使得其在求解相同紧度的问题时，所耗费的代价显著低于其他的两个算法，并最终成功求解了全部的问题。此外，从图 4.5 不难看出，当求解紧度较低的问题时，PT-ISBB 的优势并不明显。这是因为在低紧度下，所有算法均能很快找到可行解（即代价为 0 的解），并迅速缩减搜索上界到 0。因此，所有算法均可以较快地对搜索空间进行剪枝。另外，因为紧度较低时可行解广泛存在，PT-ISBB 在推理阶段给出的推理结果中 0 居多，这导致了 PT-ISBB 难以利用推理结果对搜索空间进行剪枝。

在隐私性方面，随着问题紧度的增加，算法剪枝越来越困难，因此所需要暴露的私有代价越来多。其中，值得一提的是，与 ATWB 类似，PT-ISBB 仅会在累加代价时暴露上层邻居的私有代价（通过代价请求消息，算法 24、38、48、66 行）。换句话说，在 PT-ISBB 中，最差情况下仅会泄露问题的一面隐私，因此，其隐私泄露不会超过 50%。而又由于 PT-ISBB 有更好的剪枝性能，所以在相同的紧度下具有更好的隐私性。

4.6 本章小结

本章首先分析了现有的 ADCOP 完备搜索算法泄露隐私的本质，指出提高算法隐私性的重要途径是提高算法的剪枝效率，并说明了 DPOP 算法无法用于直接

求解 ADCOP 的原因。然后创新性地提出了一种基于推理-搜索相结合的完备算法 PT-ISBB，该算法充分利用 DPOP 的快速求解特性，先利用 DPOP 预先求解一面的约束，并将其求解结果存储于各个非叶子节点。在搜索阶段，Agent 利用深度优先伪树中各个分支可以互不干扰地独立搜索这一事实，不断地将整个问题划分为更小的独立子问题来并行求解。在求解过程中，推理阶段所给出的子树的推理结果作为取值的下界来增加算法的剪枝效率。在理论分析中，证明了该算法的完备性并分析了其时空复杂度。此外，实验结果表明，该算法在多种测试问题下均显著优于传统的纯搜索完备算法，并可以求解较大规模的 ADCOP 问题。实验结果同时表明，在求解相同紧度的随机 ADCSP 问题时，PT-ISBB 较传统的完备算法具有更好的隐私性。

5 总结与展望

5.1 本文工作总结

分布式约束优化问题（DCOP）作为多智能体系统（MAS）协作问题的一个重要框架，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际价值。非对称约束优化问题（ADCOP）在 DCOPs 的基础上增加了私有个体偏好，具有更强的建模能力和更广泛的应用前景。其中，推理算法是求解 DCOP 的重要手段。本文从求解 DCOP 的推理算法入手，重点研究了 Max-sum 类算法中的值传播机制对算法性能的影响，并提出了一系列可以有效平衡算法的探索和利用的策略。另外，考虑到 ADCOP 的特性，研究了完备推理算法用于求解 ADCOP 的可能性，并创新性地提出了一种基于搜索-推理混合的完备算法。具体地，本文研究工作总结如下：

① 分析了值传播机制对 Max-sum 类算法的影响。本文从理论上证明了虽然值传播可以极大地提高算法的性能，但是其同时阻碍了 Max-sum 类算法的信念传播。特别地，本文证明了当值传播机制与 Max-sum_AD 算法相结合时，Agent 将完全无法利用全局累加的信念，因此算法将等价于一个顺序的贪心局部搜索算法。由此，提出了在值传播机制下如何有效地平衡探索与利用这一重要的科学问题。

② 针对上述问题，通过细粒度地控制值传播的执行，本文提出了三种可以平衡探索与利用的非连续值传播 Max-sum 算法，即基于单向值传播的 Max-sum_AD 算法（Max-sum_ADSSVP）、基于混合信念/值传播的 Max-sum 算法（Max-sum_HBVP）和基于概率的 Max-sum_ADVP 算法（Max-sum_ADPVP）。在 Max-sum_ADSSVP 中，值传播和信念传播被交叉执行，即当消息传播方向为正方向时，其执行信念传播来探索更有潜力的解，反之则执行值传播来保证解的质量，因此值传播始终沿着一个方向执行。为了加速算法的收敛，本文进一步提出用连续的值传播或者短时间的局部搜索算法来对值传播阶段的解进行优化，即 Max-sum_ADSSVP(t)和 Max-sum_ADSSVP_LS。在 Max-sum_HBVP 中，信念传播和值传播分别从一个有向无环图的两端同时开始执行，且变量节点仅在收到来自所有上游邻居的值传播消息后才开始选值。这极大地加速了算法的优化速度，并克服了 Max-sum_ADSSVP 中，Agent 依据无效取值信息作出决策的问题。在 Max-sum_ADPVP 中，函数节点则是依据某一概率，随机的决定本轮是否执行值传播以达到平衡探索与利用的目的。本文进一步提出了四种自适应值传播概率的方式，使得算法可以随着优化过程自适应地平衡探索与利用。另外，本文从理论上分析了上述提出的算法，指出在这些算法中，Agent 均可以考虑本地函数之外

的累积信念，即不会等价于贪心的局部搜索算法。实验结果表明，上述算法可以显著提高解的质量，且可以克服 Max-sum_ADVP 对取值时机敏感的问题。

③ 充分考虑了 ADCOP 的特点，指出完备推理算法（如 DPOP）不能直接用于求解 ADCOP 的本质。针对 ADCOP 对隐私性的要求，创新性地提出了一种基于搜索-推理混合的完备算法 PT-ISBB。该算法利用完备推理算法速度较快这一优势，预先求解一面的约束，并将推理结果存储；在搜索阶段，利用伪树中不同分支间相互独立这一事实，不断将问题划分为更小的子问题。在每一个节点上，都使用子树的推理结果作为取值的下界，以实现高效率的剪枝。本文同时在理论上证明了其完备性，并分析了其复杂度。实验结果表明，PT-ISBB 在多个测试问题上均优于传统的搜索算法。

5.2 未来工作展望

本文深入研究了基于推理的 DCOP/ADCOP 求解算法，这些算法在多种测试问题上均取得了较好的效果。然而，DCOP 领域仍然有很多开放问题亟待解决。下面给出了未来可能的研究方向。

① Max-sum_AD 类算法在确定消息传递方向时，通常采用 Agent 的编号等作为优先级排序的依据。而这种依据完全忽略了问题本身的拓扑结构。在未来，可以进一步研究当交替执行信念传播和值传播时，各种排序方式对解的质量的影响，并提出适合单向值传播的排序依据。

② Max-sum 类算法由于无效取值假设的存在，无法产生高质量的解，而尽管值传播可以解决这个问题，但同时又会限制 Max-sum 的探索能力。所以，未来的一个可能的工作是找到一个既可以平衡探索与利用，又能有效消除无效取值假设的方法。

③ 现有的所有完备搜索算法都存在不同程度的重复搜索的问题。例如，对于一个叶子节点来说，对于父节点的同一个取值，在任意部分解下其所能达到的最优结果是一样的。因此，如何基于动态规划思想为基于伪树的完备搜索算法提供一个可以保存搜索结果以减少重复搜索的机制仍然是一个有意义并富有挑战性的课题。

致　谢

时光飞逝，三年的研究生学习生活即将结束，毕业论文的工作也迫近尾声。回想这三年的研究生生涯，有刚入学时的迷惘，有看不懂论文时的焦虑，有等待投稿结果的紧张，也有论文接受后的喜悦。这些酸甜苦辣都将是我人生中宝贵的财富和不可磨灭的印记，激励着自己砥砺前行。

首先我要感谢我的导师陈自郁老师在我学习和科研中提供了很多无私的帮助。正是由于陈老师的悉心指导与督促，让我在刚踏入校园时，比其他同学更早地进入了科研的正轨。在陈老师的指导下，我开始大量阅读本领域的相关文献，并培养自己的动手能力和创新能力，这为我后面的科研工作打下了坚实的基础。经过一个学期的训练，我对本领域有了较为深入的认识，也逐渐形成了自己的一些思路，而这些思路往往都是一些很质朴的、不成熟的想法，不能直接成文。每当这时，陈老师总是耐心地和我讨论，从她对本领域认知的角度高屋建瓴地对我的思路给出她的建议。正是这样一次又一次的讨论，使我在科研过程中逐渐积累经验，去粗取精，最终使得我的第一个思路达到了成文的标准。在小论文撰写的过程中，陈老师在论文写作的每个阶段都全程跟踪，悉心指导，手把手地从句、段、篇章入手，纠正我的英文表达、逻辑结构，不厌其烦地帮助我修改论文中的每一个细节。“千淘万漉虽辛苦，吹尽狂沙始到金”。付出总会得到回报。正是在陈老师的悉心指导下，我的论文才得以在高水平国际学术会议上得以发表。她严谨求实的学术精神和认真负责的治学态度，深深地感染和激励着我。在此感谢陈老师，希望陈老师身体健康，阖家欢乐，工作顺利。

感谢何中市教授，何老师有着渊博的知识和独特的人格魅力。在每一次的大组讨论班上，何老师带领我们整个研究组交流学习心得，分享研究成果，让我有机会了解研究组其他同学是如何做研究的。在开题时，何老师也为我提供了很多富有建设性意义的指导意见，为我的研究提供了极大的帮助。在此，衷心地祝愿何老师身体健康，工作顺利。

感谢在校期间的所有的授课老师，感谢实验室的老师和同学。感谢余浙鹏、何振和何琛师兄所搭建起的实验平台以及讨论会中提供的思路和建议，让我顺利地完成实验工作。感谢同门吴腾飞和同学陈定定在我新思路的提出和修正中提供了很多建设性意见和帮助。我还要感谢师妹姜兴琼、张文昕和刘力贞在我论文撰写时所提供的帮助。衷心祝愿大家在以后的工作和生活中称心如意，前程似锦。

我还要感谢我的父母，是你们的辛苦工作，为我提供了一个安心稳定的学习环境。不论我遇到什么样的困难，你们都毫无理由地鼓励和支持我，让我时时刻

刻都满怀乐观和温暖的心一路走过这二十多个春秋，你们是我人生中最重要的亲人。

最后，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！

邓衍晨

二O一八年四月 于重庆

参考文献

[1] 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.

[2] 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.

[3] 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.

[4] Cheng, S., Raja, A., Xie, J. 2014. Dynamic multi-agent load balancing using distributed constraint optimization techniques[J]. Web Intelligence and Agent Systems: An International Journal, 12(2), 111-138.

[5] 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.

[6] 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]. Articial Intelligence,2005, 161(1): 55-87.

[7] Muldoon, C., O'Hare, G. M., O'Grady, M. J., Tynan, R., Trigoni, N. 2013. Distributed constraint optimization for resource limited sensor networks[J].Science of Computer Programming, 78(5), 583-593.

[8] Grinshpoun T, Grubshtein A, Zivan R, et al. Asymmetric distributed constraint optimization problems[J]. Journal of Artificial Intelligence Research, 2013, 47: 613-647.

[9] A. Farinelli, A. Rogers, A. Petcu, N.R. Jennings, Decentralised coordination of low-power embedded devices using the max-sum algorithm[C]. In Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems, 2008: 639–646.

[10] 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.

[11] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. Cambridge: MIT press, 1998.

[12] Grinshpoun T, Meisels A. Completeness and Performance Of The APO Algorithm[J]. J. Artif. Intell. Res.(JAIR), 2008, 33: 223-258.

[13] Hirayama K, Yokoo M. Distributed partial constraint satisfaction problem[C]//Principles and

Practice of Constraint Programming-CP97. 1997: 222-236.

[14] Gershman A, Meisels A, Zivan R. Asynchronous forward bounding for distributed COPs[J]. Journal of Artificial Intelligence Research, 2009, 34(1): 61-88.

[15] 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.

[16] Gutierrez P, Meseguer P. Saving messages in adopt-based algorithms[C]. Proc. 12th DCR workshop in AAMAS. 2010, 10: 53-64.

[17] Gutierrez P, Meseguer P, Yeoh W. Generalizing adopt and bnb-adopt[C]. In Proceedings of International Joint Conference on Artificial Intelligence(IJCAI), 2011: 554.

[18] 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.

[19] Litov O, Meisels A. Forward bounding on pseudo-trees for DCOPs and ADCOPs[J]. Artificial Intelligence, 2017, 252:83-99.

[20] Petcu A, Faltings B. A scalable method for multiagent constraint optimization[C] In IJCAI. 2005, 5: 266-271.

[21] Dechter R. Bucket elimination: A unifying framework for reasoning[J]. Constraints, 2013, 2(1):51-55.

[22] Petcu, Adrian, and Boi Faltings. MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization[C]. In IJCAI. 2007: 1452-1457.

[23] Petcu A, Faltings B. ODPOP: an algorithm for open/distributed constraint optimization[C] In PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, 21(1): 703.

[24] 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 Multiagent Systems(AAMAS), 2009: 1239-1246.

[25] Aji S M, McEliece R J. The generalized distributive law[J]. Information Theory, IEEE Transactions on, 2000, 46(2): 325-343.

[26] 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. Springer US, 2006: 127-146.

[27] Steven, O., Roie, Z., Aviv, N. Distributed breakout: Beyond satisfaction[C]. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016:447–453.

[28] 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.

[29] Hirayama K, Yokoo M. The distributed breakout algorithms[J]. Artificial Intelligence, 2005, 161(1): 89-115.

[30] Chapman A C, Alex R, Jennings N R, et al. A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems1[J]. Knowledge Engineering Review, 2011, 26(4):411-444.

[31] Katagishi, H, Pearce, J. P. KOPT: Distributed DCOP Algorithm for Arbitrary K-optima with Monotonically Increasing Utility. DCR-07, 2007.

[32] Pearce J P, Maheswaran F T, Tambe M. How local is that optimum? k-optimality for dcop[C]. In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. ACM, 2005: 1303-1304.

[33] Zilberstein S. Using anytime algorithms in intelligent systems[J]. AI magazine, 1996, 17(3): 73-83.

[34] Zivan R, Okamoto S, Peled H. Explorative anytime local search for distributed constraint optimization[J]. Artificial Intelligence, 2014, 212: 1-26.

[35] Yu Z, Chen Z, He J, et al. A Partial Decision Scheme for Local Search Algorithms for Distributed Constraint Optimization Problems[C] In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. 2017: 187-194.

[36] Ottens B, Dimitrakakis C, Faltings B. Duct: An upper confidence bound approach to distributed constraint optimization problems[C] In Proceedings of the 26th conference of the AAAI, 2012: 528–533.

[37] Nguyen D T, Yeoh W, Lau H C. Distributed Gibbs: a memory-bounded sampling-based DCOP algorithm[C]. In Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. 2013: 167-174.

[38] A. 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.

[39] Rollon, E., Larrosa, J. Improved bounded max-sum for distributed constraint optimization[C]. In: Principles and Practice of Constraint Programming, 2012:624-632.

[40] Rollon, E., Larrosa, J. Decomposing utility functions in bounded max-sum for distributed constraint optimization[C]. In: International Conference on Principles and Practice of Constraint Programming, 2014:646-654.

[41] Cohen L, Zivan R. Max-sum Revisited: The Real Power of Damping[C] In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. 2017: 1505-1507.

[42] Greenstadt R, Pearce J P, Tambe M. Analysis of privacy loss in distributed constraint

optimization[C] In Proceedings of the 21-st national conference on Artificial Intelligence. 2006: 647-653.

[43] Maheswaran R T, Pearce J P, Varakantham P, et al. Valuations of possible states (VPS): a quantitative framework for analysis of privacy loss among collaborative personal assistant agents[C] In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. 2005: 1030-1037.

[44] Yokoo M, Suzuki K, Hirayama K. Secure distributed constraint satisfaction: Reaching agreement without revealing private information[C] In International Conference on Principles and Practice of Constraint Programming. 2002: 387-401.

[45] 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 Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1. IEEE Computer Society, 2004: 310-317.

[46] Grinshpoun T, Grubshtein A, Zivan R, et al. Asymmetric Distributed Constraint Optimization Problems[J]. Journal of Artificial Intelligence Research, 2013, 47: 613-647.

[47] Zivan R, Parash T, Naveh Y. Applying Max-Sum to Asymmetric Distributed Constraint Optimization[C]. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina, 2015: 432-439.

[48] Barabási A L, Albert R. Emergence of scaling in random networks[J]. science, 1999, 286(5439): 509-512.

[49] Meisels A, Kaplansky E, Razgon I, et al. Comparing performance of distributed constraints processing algorithms[C] In Proc. AAMAS-2002 workshop on distributed constraint reasoning DCR. 2002: 86-93.

[50] Brito I, Meisels A, Meseguer P, et al. Distributed constraint satisfaction with partially known constraints[J]. Constraints, 2009, 14(2): 199-234.

[51] Ezzahir R, Bessiere C, Belaissaoui M, et al. DisChoco: A platform for distributed constraint programming[C] In Proceedings of the IJCAI. 2007, 7: 16-21.

[52] Léauté T, Ottens B, Szymanek R. FRODO 2.0: An open-source framework for distributed constraint optimization[C] In Proceedings of the IJCAI. 2009, 9(134): 160-164.

[53] Sultanik E A, Lass R N, Regli W C. DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms[C] In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers. 2008: 1667-1668.

附　录

A. 作者在攻读学位期间发表的论文目录：

[1] Chen Ziyu, **Deng Yanchen**, Wu Tengfei. An Iterative Refined Max-sum_AD Algorithm via Single-side Value Propagation and Local Search[C]//Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'17). 2017: 195-202. (CCF B 类会议)

[2] Ziyu Chen, **Yanchen Deng**, Tengfei Wu, Zhongshi He. A class of iterative refined Max-sum algorithms via non-consecutive value propagation strategies[J]. Journal of Autonomous Agents and Multi-Agent Systems. (SCI，CCF B 类期刊，大修)

[3] Ziyu Chen, Tengfei Wu, **Yanchen Deng** and Cheng Zhang. An Ant-based Algorithm to Solve Distributed Constraint Optimization Problems[C]//Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18), accepted. (CCF A 类会议)

[4] Yu Zhepeng, Chen Ziyu, He Jinyuan, **Deng Yanchen**. A Partial Decision Scheme for Local Search Algorithms for Distributed Constraint Optimization Problems[C]//Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'17). 2017: 187-194. (CCF B 类会议)

B. 作者在攻读学位期间取得的科研成果目录：

[1] 重庆市前沿与应用基础研究(一般)项目：非对称分布式约束优化问题求解算法及其应用研究，项目编号：cstc2017jcyjAX0030, 2017.7-2020.6.

[2] 重庆市项目博士后资助项目：配电网分布式电源多目标优化配置模型和算法研究，项目编号：Xm201324，2013.10-2015.9.

[3] 重庆市前沿与应用基础研究(一般)项目：面向认知无线电传感器网络节能优化的体系架构与频谱共享方法研究，项目编号：cstc2013jcyjA40049，2013.6-2017.5.