软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(4):625−639 [doi: 10.3724/SP.J.1001.2011.03765]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

低约束密度分布式约束优化问题的求解算法*

丁 博[1+], 王怀民[1,2], 史殿习[1], 唐扬斌[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](国防科学技术大学 计算机学院 并行与分布处理国家重点实验室,湖南 长沙 410073)

Algorithm for Distributed Constraint Optimization Problems with Low Constraint Density

DING Bo[1+], WANG Huai-Min[1,2], SHI Dian-Xi[1], TANG Yang-Bin[1]

[1](School of Computer, National University of Defense Technology, Changsha 410073, China)

[2](National Key Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: dingbo@nudt.edu.cn

Ding B, Wang HM, Shi DX, Tang YB. Algorithm for distributed constraint optimization problems with low constraint density. *Journal of Software*, 2011,22(4):625−639. http://www.jos.org.cn/1000-9825/3765.htm

Abstract: Many challenges in multi-agent coordination can be modeled as Distributed Constraint Optimization Problems (DCOPs). Aiming at DCOPs with low constraint density, this paper proposes a distributed algorithm based on the idea of greed and backjumping. In this algorithm, each agent makes decisions according to the greedy principle that the most assignment combinations in the problems with low constraint density occur at a zero cost, and the backjumping mechanism among the agents ensures the success of this algorithm, even when this greedy principle leads to a local optimum. In contrast with the existing mainstream DCOP algorithms, this algorithm can solve problems with low constraint density with fewer messages while keeping the polynomial message length and space complexity. The correctness of the key mechanisms in this algorithm has been proved, and those advantages in performance have been verified by experiments.

Key words: distributed constraint optimization problem; multi-agent; algorithm

摘 要: 多 Agent 协作过程中的许多挑战都可以建模为分布式约束优化问题.针对低约束密度的分布式约束优化问题,提出了一种基于贪婪和回跳思想的求解算法.在该算法中,各 Agent 基于贪婪原则进行决策,能够利用低约束密度问题中大量赋值组合代价为 0 这一特点来加快求解速度.同时,Agent 间的回跳机制可以在贪婪原则陷入局部最优时保证算法的完全性.相对于已有主流算法,该算法可以在保持多项式级别的消息长度/空间复杂度的前提下,以较少的消息数目求解低约束密度的分布式约束优化问题.给出了算法关键机制的正确性证明,并通过实验验证了算法的上述性能优势.

关键词: 分布式约束优化问题;多 Agent;算法

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 国家自然科学基金(90818028); 国家重点基础研究发展计划(973)(2011CB302600); 国家杰出青年科学基金(60625203)

收稿时间: 2009-06-16; 定稿时间: 2009-10-10

在多 Agent 协作过程中,资源和任务分配、协同决策等许多具体问题都可被抽象为分布式约束优化问题(distributed constraint optimization problem,简称 DCOP)[1].这一问题是 NP 难问题,设计其求解算法时的重要挑战之一是如何尽可能地利用应用域特点来降低复杂度[2].

在分布式约束优化问题中,约束密度[3]代表了结点间存在约束关系的平均概率.在许多情况下,Agent 物理邻接时才会产生直接的制约\冲突关系.例如:在传感器分配问题的约束图中,结点是被监控的目标,边则是可能被共用的传感器[4];在服务组合 QoS 问题约束求解模型中,约束关系只存在于可能发生调用关系的结点之间[5].另一方面,近期研究指出,大量真实网络中结点物理邻接的平均概率相当低——文献[6]所给出的 880 个结点的对等网络中平均度为 1.47,即两个结点邻接的平均概率仅为 0.003;物理世界中铁路网的这一概率为 0.113;其他许多网络的这一概率在 10^{-4} 级别或更低.因此,以真实网络为基础的许多问题具备约束密度较低的特点,我们称此类问题为低约束密度的分布式约束优化问题.

针对低约束密度的分布式约束优化问题,本文提出了一种基于贪婪和回跳思想的算法 HEDA(high efficient density-aware algorithm).在该算法中,每个 Agent 基于贪婪原则进行决策,利用低约束密度问题中大多数赋值组合不存在冲突这一特点来加快求解速度,并在必要时通过贪婪求解过程的回跳来保证算法完全性,克服贪婪原则可能会陷入局部最优的弊端.HEDA 算法能够以较少消息数目求解低约束密度问题,同时将消息长度和空间复杂度控制在多项式级别.这一性能优势已在实验中得到验证:在求解随机生成和第三方测试集的低约束密度问题时,HEDA 算法的消息数目在大多数情况下远少于基于搜索/回溯的 NCBB[7],Adopt[1]等算法.同时,算法消息长度和单个结点空间需求也远少于基于动态规划的 DPOP[8]算法.

本文第 1 节给出分布式约束优化问题定义.第 2 节分析相关工作及其不足.第 3 节概述 HEDA 算法的基本思想.第 4 节给出 HEDA 算法的关键机制及其正确性证明.第 5 节讨论 HEDA 算法的实现细节.第 6 节给出算法复杂度分析和实验对比测试结果.

1 分布式约束优化问题

分布式约束优化问题的定义存在若干形式上有细微差异的版本[1,4,7,9],在本文中,我们将其定义如下:

定义 1(分布式约束优化问题). 分布式约束优化问题是一个五元组$\langle X,D,F,A,\delta\rangle$,其中,

$X=\{x_1,\ldots,x_n\}$是变量集合;

$D=\{D_1,\ldots,D_n\}$是论域集合.其中,D_i是变量 x_i的论域,包括有限个离散的元素;

$F=\{f_1,\ldots,f_s\}$是一组约束函数,$f_i:D_{i1}\times\ldots\times D_{ik}\to N\cup\{+\infty\}$定义了变量 $x_{i1},\ldots,x_{in}$的赋值组合代价,当代价为$+\infty$时,该组合被禁止.若某个赋值组合代价非 0,则称相应变量间存在约束关系;

$A=\{a_1,\ldots,a_t\}$是 Agent 集合,每个 Agent 管理若干变量,负责为这些变量赋值;

δ指明了 Agent 与变量的管理关系,它是 $A\to\boldsymbol{X}$ 的双射,其中,$\boldsymbol{X}$ 是 X 的一个划分.

在上述定义中,δ同时也隐含了约束函数的分布性:Agent a_i 初始了解且仅了解与$\delta(a_i)$中变量相关的约束函数,它们在后文中被称为 Agent a_i的本地约束.

定义 2(分布式约束优化问题的解及其代价). 对于 $X'=\{x'_1,\ldots,x'_u\}\subseteq X, D'_1\times\ldots\times D'_u$ 中的一个元素被称为分布式约束优化问题在 X'上的一个部分解(partial solution).如果 $X'=X$,则称该部分解为问题的一个解.部分解的代价是其所违反约束的代价总和,即

$$Cost(PS)=\sum_{f\in F} f(PS|_{dom(f)}) \tag{1}$$

其中,符号|表示 PS 在 f 定义域上的投影.

定义 3(分布式约束优化问题的约束密度). 给定一个分布式约束优化问题$\langle X,D,F,A,\delta\rangle$,其约束图是一个无向图 $G=\langle X,E\rangle$,如果 x_{j1},x_{j2} 存在约束关系,则边$(x_{j1},x_{j2})\in E$.约束密度 p_d 是 G 中边数与$|X|$阶完全图边数的比值,也即

$$p_d=2|E|\Big/|X|\times(|X|-1) \tag{2}$$

定义 3 仅考虑了二元约束的情况,在存在更高元约束时需引入超边(HyperEdge)等概念[2].

分布式约束优化问题求解算法的目标是通过 Agent 协同得到一个最小代价的解,使每一个 Agent 可以为其所管理的变量赋值.本文工作针对的是低约束密度的分布式约束优化问题,一个简单实例是较大规模的地图着色问题:我们可以指定地图每个区块由单独的一个 Agent 负责,Agent 基于本地视图进行协同,对其负责的区块着色,要求最终整个地图上被违反的约束(即相邻区块不能着相同颜色)总数最少[10].由于约束关系存在且仅存在于负责物理相邻区块的 Agent 之间,由平面图边数和结点数$|E|\leqslant 3|X|-6$ 的关系[11],可得该问题约束密度小于等于$(6|X|-12)/(|X|\times(|X|-1))$,即 20 个结点时,约束密度小于等于 0.28,30 个结点时,小于等于 0.19,并且随着结点数的增加,约束密度趋向于 0.

本文将使用如下一些标记法:

- 直观起见,$\{x'_1,...,x'_u\}$ 上的部分解 $\langle d'_1,...,d'_u\rangle$ 直接记为赋值序列“$x'_1=d'_1,...,x'_u=d'_u$”;
- 若 PS 是变量集合 X' 上的一个部分解,则记 $var(PS)=X'$,且对任意 $x_i\in X'$,使用符号 $PS(x_i)$ 表示变量 x_i 在 PS 中所赋的值;
- 使用符号∪来表示两个部分解的合并操作,例如,“$x_1=a$”∪“$x_2=b$”=“$x_1=a,x_2=b$”;
- 符号|表示部分解在指定变量集合上的投影,例如,“$x_1=a,x_2=b,x_3=c$”$|_{\{x_1,x_2\}}$=“$x_1=a,x_2=b$”.

此外,本文后续内容基于对分布约束优化问题的如下一些简化:1) 每个 Agent 只拥有一个变量,在措辞上也将不再区分 Agent、结点和变量;2) 所有约束均是一元或二元的,即 F 中的成员均为一元或二元函数的.这些简化只是为了行文简单起见,也是本领域通常的做法[4,7],但本文算法略作改造后即可处理单 Agent 多变量及多元约束的情况.

2 相关研究工作

分布式约束优化问题可对多 Agent 领域许多真实问题建模,例如,Pectu 在文献[4]中给出了多 Agent 组合拍卖、覆盖网优化等的分布式约束优化问题模型,Maheswaran 在文献[12]中将分布式多事件调度建模为分布式约束优化问题,Sultanik 等人实现了多 Agent 任务调度的语言 C-TÆMS 到分布式约束优化问题的自动转换[13],Zhang 等人给出了移动传感器网络中分布式约束优化问题的应用等[14].该问题求解算法近年来引起了较为广泛的关注,其中,与本文工作相关的是具备完全性的非近似算法,它们可被分为两大类:基于搜索/回溯的算法和基于动态规划的算法.

(1) 基于搜索/回溯的算法

SBB(synchronous branch and bound)[15]算法是约束优化领域经典分枝定界(branch and bound)算法[2]的简单分布化,它基于搜索/回溯的思想遍历解空间,通过记录上界/下界来剪枝以提高性能.但与集中式分枝定界算法不同,它将全局状态封装为在 Agent 间传递的令牌,从而不必维护一个集中的状态控制器.以 SBB 算法为基础衍生出了 dAOBB[4],NCBB[7]等算法,这些算法仍基于搜索/回溯的思想,主要改进在于采用不同的 Agent 组织方式、进一步重用搜索过程中产生的知识等.

Adopt 算法[1]是具备完全性的异步算法,它基于回溯和上界/下界传播的思想:各个结点不需要等待其他结点,而是基于其当前所能获得的信息直接做出决策,并在发现可能获得更优解时重新决策.但这种完全基于局部视图的行为方式有可能导致已被搜索过的空间被重新搜索,从而造成大量不必要的消息[4].OptAPO[16]是集中-分布混合的异步算法,将问题分解成子问题,子问题采用集中式求解,可大幅减少消息数目,但在 Agent 隐私保护、自主性等方面存在缺陷[3].

(2) 基于动态规划的算法

DPOP(distributed pseudotree optimization procedure)[8]是基于动态规划的求解算法,其基本思想与集中式的 Bucket Elimination 算法[2]类似:每次从问题中剔除一个结点,该结点对问题求解所有可能的“影响”被保存下来.该算法的消息数目是线性的,但消息长度和空间复杂度的上界是呈指数级的.例如,在 10 个变量、论域大小为 10 时,该上界可能达到 10^9(约 953M),从而限制了其应用范围.MB-DPOP[17]和 O-DPOP[18]是该算法针对消息长度的两个改进:前者通过必要时的暴力枚举来控制消息长度/空间复杂度的上界;后者将消息长度降到了多项式级

别,但消息缓存机制的存在使其结点空间需求的上界仍呈指数级.

上述算法在求解低约束密度问题时的不足主要表现在:在基于搜索/回溯的算法中,消息数目呈指数级,实际消息数目依赖于剪枝效率,而约束密度仅仅是问题特征之一,约束密度低并不一定意味着剪枝效率就高;另一方面,将消息数目降到了多项式级别的动态规划算法虽然有可能利用低约束密度问题的特点,但其消息长度/空间复杂度的上界仍呈指数级.

3 HEDA 算法概述

HEDA 算法利用了低约束密度问题中大量赋值不存在冲突(即组合代价为 0)的特点,在保持多项式级别的消息长度/空间复杂度的同时,能够以较少的消息数目求解此类问题.本节首先介绍 HEDA 算法的设计思想,然后结合一个简单实例说明 HEDA 算法的基本执行过程.

3.1 HEDA算法基本设计思想

HEDA 算法的基本设计思想包括:

(1) 基于伪树的结点组织.与 Adopt,DPOP 等算法相同,HEDA 算法中结点被组织成伪树[2](pseudo tree).伪树是约束图的一种生成树,其特点是不同分支的结点间不存在约束关系,也即在原约束图中不邻接.伪树组织方式使得不同分支在 HEDA 算法中可独立实施贪婪求解过程,降低问题规模,提高并行性;

(2) 自底向上的贪婪求解.在 HEDA 算法中,每个结点向其父结点报告一个符合特定条件的部分解集合(后文称其为最优部分解集),父结点在子结点报告内容的基础上,基于贪婪原则尝试计算自身的最优部分解集.如此层层上报,直至到达根结点,根结点最优部分解集中代价最小者即为问题的解;

(3) 必要时自顶向下的回跳.贪婪原则可能无法求出当前结点的最优部分解集,此时结点会启动回跳机制,也即通知其子树中某些已计算完毕的结点重新开始一次贪婪求解过程.此过程终止于当前结点并得到一个确定的最优部分解,从而使前一贪婪求解过程得以继续.回跳机制保证了 HEDA 算法的完全性.

与第 2 节所介绍算法的设计思想相比,HEDA 算法最显著的特点是各个结点会首先尝试基于贪婪原则进行决策,而贪婪原则成立的概率与代价为 0 的赋值组合的数目密切相关(见第 4.2 节定理 1),因此,HEDA 算法有可能快速求解低约束密度的问题,甚至在某些情况下只需要线性消息数目.HEDA 算法结合了基于动态规划与基于搜索/回溯的算法的某些设计:一方面,算法基于贪婪原则自底向上计算最优部分解集,子结点的同一条消息可以在父结点生成多个最优部分解时复用,从而减少消息数量,这一特点与基于动态规划的算法类似;另一方面,回跳有可能利用前一次贪婪求解过程中的某些历史信息,例如本文第 5.2 节将搜索/回溯算法中定界剪枝机制应用到了本算法中.

3.2 使用HEDA算法求解TriVar问题

下面通过一个名为 TriVar 的简单分布式约束优化问题说明 HEDA 算法的基本执行过程,其中某些细节将在第 4 节和第 5 节给出.

(1) TriVar 问题定义

TriVar 问题定义如下:

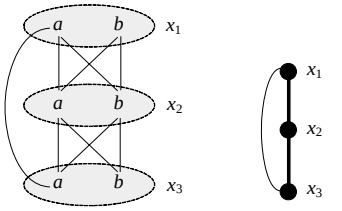
- $X=\{x_1,x_2,x_3\}$;
- $D=\{D_1,D_2,D_3\}$,$D_1=D_2=D_3=\{a,b\}$;
- $F=\{f_{12},f_{13},f_{23}\}$,其中,$f_{ij}$ 是定义域为 $D_i\times D_j$ 的约束函数.约束函数所建立的具体映射关系如图 1(a)所示:两个论域元素之间存在边代表相应赋值组合的代价为 1,否则,代价为 0.例如,对于 $f_{13}\in F$,有 $f_{13}(a,a)=1$,$f_{13}(a,b)=f_{13}(b,a)=f_{13}(b,b)=0$.由 F 得到的 TriVar 问题约束图如图 1(b)所示.

X 中每个变量由一个对应的 Agent 负责,因此不再对 A 和 δ 作单独定义.

(2) TriVar 问题求解过程

HEDA 算法首先通过预处理将结点组织成伪树.我们假定 x_1 是伪树根结点,x_3 是叶结点(如图 1(b)中粗线所

示).在结点组织完毕后,每个结点都将成为一棵子树的根结点,叶结点子树则仅包含其自身.



(a) Constraint function (b) Constraint graph
(a) 约束函数 (b) 约束图

Fig.1 TriVar problem
图 1 TriVar 问题

此后,算法将进入贪婪/回跳求解过程.在这一过程中,每个结点将为自己求一个最优部分解集.该解集由本结点论域中每个元素所对应的一个部分解组成,此部分解覆盖了本结点子树上所有结点,且是本结点赋予该论域元素的所有部分解中代价最小者.最优部分解集基于贪婪和回跳机制,从叶结点到根结点自底向上生成.我们通过 TriVar 问题的求解来说明该过程.

Step 1. 由于叶结点 x_3 的子树仅包含其自身,因此,它向 x_2 报告的最优部分解集显然是{⟨"x_3=a",0⟩,⟨"x_3=b",0⟩},其中,两个部分解分别对应论域元素 a 和 b,0 则是相应部分解的代价;

Step 2. x_2 在 x_3 报告内容的基础上,根据如下贪婪原则生成一个部分解集:对于 x_2 论域中的每个元素,将其与 x_3 报告的解集中每个部分解合并后得到一组新的、覆盖了本结点的部分解,选择其中代价最小者放到解集中.具体地,x_2 首先将"x_2=a"与"x_3=a","x_3=b"合并得到两个新的部分解"x_2=a,x_3=a"和"x_2=a,x_3=b",取其中代价最小者作为论域元素 a 所对应的部分解,然后再为论域元素 b 重复上述计算过程,得到部分解集{⟨"x_2=a,x_3=a",1⟩,⟨"x_2=b,x_3=a",1⟩}(假定代价相同时取第 1 个).虽然是基于贪婪原则计算出来的,但因为事实上已考虑了 x_2 子树中所有可能的赋值组合,这一解集显然是最优部分解集,因此,x_2 将其报告给 x_1;

Step 3. x_1 在 x_2 报告内容的基础上,依据相同的贪婪原则计算出一个部分解集{⟨"x_1=a,x_2=a,x_3=a",3⟩,⟨"x_1=b,x_2=b,x_3=a",2⟩}.此时,结点 x_1 发现,由于 $f_{13}(a,a)$=1,导致其无法确认第 1 个部分解是否真的是以 x_1=a 为首的所有部分解中代价最小者.此时,x_1 将启动回跳机制,通知 x_3 在假定"x_1=a"已成立的情况下开始新的一轮贪婪求解过程;

Step 4. x_3 计算出假定"x_1=a"已成立情况下的最优部分解集{⟨"x_3=a",1⟩,⟨"x_3=b",1⟩},并报告给 x_2;

Step 5. x_2 计算出假定"x_1=a"已成立情况下的最优部分解集{⟨"x_2=a,x_3=b",2⟩,⟨"x_2=b,x_3=b",2⟩},并报告给 x_1;

Step 6. 因为 Step 4 和 Step 5 都是在假定"x_1=a"已成立情况下执行的,因此,根结点 x_1 收到 x_2 的报告后,可以确认以 x_1=a 为首、代价最小的部分解是"x_1=a,x_2=a,x_3=b"(代价为 2).结点 x_1 使用这一信息更新第 3 步解集中的第 1 个元素,得到自身的最优部分解集.

由最优部分解集含义可知,x_1 最优部分解集中代价最小者"x_1=a,x_2=a,x_3=b"即为 TriVar 问题的最优解.

(2) 算法性能的初步分析和对比

在求解 TriVar 问题时,HEDA 算法一共进行了两次贪婪求解过程,发送 4 条消息后得到了正确的解.如果使用基于搜索/回溯的 SBB 算法,按 $x_1\rightarrow x_2\rightarrow x_3$ 的搜索顺序来求解本问题,那么由于客观上无法剪枝,需要在 3 个结点间评估 8 个完整的解,因此无论是总消息数目还是消息总长度都将远大于 HEDA 算法.

HEDA 算法求解时需要在每个结点上得到一个最优部分解集合,因此表面上看似乎 HEDA 算法会产生额外开销,但如前所述,在低约束密度问题中,许多情况下,贪婪原则即可获得最优部分解.本例 Step 3 中,由于 $f_{13}(b,a)$=0,因而可以确定基于贪婪原则得到的第 2 个部分解"x_1=b,x_2=b,x_3=a"是论域元素 b 所对应的最优部分解,不需要为其回跳(参见第 4.2 节定理 1).甚至在某些情况下,HEDA 算法只需要线性消息数目即可求得最优解.例如,若 TriVar 问题中 $f_{13}(a,a)$=0(即删掉图 1(a)中的弧线),则第 3 步之后算法即可结束.而此时,基于搜索/回溯的

SBB 算法仍然需要评估 8 个完整的解.

4 HEDA 算法核心机制

第 3 节仅阐述了 HEDA 算法的基本思想,并未触及如下问题:

(1) 算法的完全性,也即第 3.2 节 Step 6 结束后是否得到了真正的最优解;

(2) 算法的分布性,也即如何保证第 3.2 节所示过程不需要一个全局且集中的状态控制器;

(3) 其他一些有挑战性的问题,包括在伪树中如何将多个子树的结果汇聚、如何判断一个结点基于贪婪原则所求出的部分解集是否需要回跳、回跳起点如何确定、在回跳过程中又需要回跳如何处理等.

本节将详述 HEDA 算法核心机制,并回答上述问题.

HEDA 算法的执行可分为结点组织、贪婪\回跳、解传播这 3 个阶段,其中,贪婪\回跳是算法最主要的阶段.

4.1 结点组织

HEDA 算法首先将结点组织成具备伪树特征的深度优先搜索树(DFS 树).这一阶段可被视作算法的预处理阶段,主要完成如下工作:

(1) DFS 树生成,并填充每个结点上的如下数据结构:父结点 *Parent*;子结点集合 *Childs*;伪父结点集合 *PseudoParent*,伪子结点集合 *PseudoChilds*.其中,伪父结点和伪子结点分别是指在 DFS 树中不邻接但存在约束关系的祖先结点和子孙结点[4].在 DFS 树生成以后,每个结点都将成为一个子树的根结点,后文记该子树中所有结点为 *subtree*(*x*);

(2) 为每个结点生成具体分离子(detailed separator).分离子(separator)是约束图及其伪树中的概念[2],它是伪树祖先结点的一个子集,将它们删除后,本结点子树将不再与原约束图连通.HEDA 算法中的具体分离子是这一概念的扩展,其区别在于,它还携带了这些结点的论域元素信息,即它们中的哪些赋值导致本结点子树与原约束图连通.例如,第 3 节 TriVar 问题的 DFS 树中,x_3 分离子是$\{x_1,x_2\}$,而具体分离子还包含了具体的赋值,即 *DetailedSeparators*(x_i)={"x_1=*a*","x_2=*a*","x_2=*b*"}.具体分离子可以为贪婪/回跳阶段回跳起点的确定提供依据(见第 4.2 节中的定理 3).

文献[4]给出了将结点组织成 DFS 树的多项式复杂度的分布式算法.具体分离子的分布式生成算法可以对分离子生成算法[17]作简单扩展而得,只需要线性消息数目.

4.2 贪婪\回跳

在贪婪\回跳阶段,算法沿 DFS 树自底向上求解各结点的最优部分解集,最终得到根结点的最优部分解集.在这一过程中,可能需要贪婪求解过程的多次回跳,也即要求某些已经完成最优部分解集计算的结点重新计算,并在重新计算时假定某些被称为上下文的赋值组合已成立.

下面首先给出上下文和最优部分解集的严格定义.

定义 4(上下文和上下文相关代价). 上下文τ是一个部分解.给定另一个部分解 *PS* 且 *var*(*PS*)∩*var*(τ)=∅,*PS* 的上下文τ相关代价被定义为

$$CC(PS,\tau)=Cost(PS)+\sum_{x_i\in \mathrm{var}(PS),\, x_j\in \mathrm{var}(\tau)} f_{ij}(PS(x_i),\tau(x_j)) \tag{3}$$

其中,$f_{ij}\in F$ 是以 $D_i\times D_j$ 为定义域的二元约束函数.直观地说,上下文相关代价除了定义 2 所给代价外,还额外考虑了与当前上下文中赋值的冲突.例如,在第 3.2 节的 Step 3~Step 6 中,"x_1=*a*"就是上下文,此时,"x_3=*a*"的上下文相关代价是 1.

上下文是贪婪求解过程的基本参数,可以用来唯一地标识一个贪婪求解过程.后述定义均是上下文相关的,因此,为简单起见,后文所称代价均为上下文相关代价,若不存在上下文时,记上下文为∅.

定义 5(最优部分解和最优部分解集). 给定 $x_i\in X$ 和上下文τ,对于任意 $d\in D_i$,x_i 在 d 上的最优部分解 $OptPS_{x_i}(d,\tau)$ 被定义为:(1) *subtree*(x_i)上的部分解;(2) 在x_i处的赋值为 *d*;(3) 满足以上两点的所有部分解中上下

文τ相关代价最小者.x_i的最优部分解集被定义为

$$OptPSSet_{x_i}(\tau)=\{OptPS_{x_i}(d,\tau)\mid d\in D_i\} \tag{4}$$

显然,根结点的最优部分解集 $OptPSSet_{root}(\varnothing)$中代价最小者即为问题最优解,且依据最优部分解的定义,叶结点 x_{leaf} 最优部分解集必定是{"x_{leaf}=d"|$d\in D_{leaf}$}.因此,要得到根结点的最优部分解集,只需回答如下问题:在给出所有子结点最优部分解集时,父结点如何构造其最优部分解集?

为回答上述问题,下面首先引入贪婪部分解的概念.简言之,贪婪部分解是当前结点在其子结点报告的最优部分解集(及代价信息)基础上找到的满足定义 5 中条件(1)和条件(2)、代价尽可能小的部分解.

定义 6(贪婪部分解). 给定上下文τ,$x_i\in X$ 和 $x_j\in Childs(x_i)$,$d\in D_i$的一个贪婪基点是 x_j 论域 D_j 中的某个元素

$$GreedyBase_{x_i}(d,x_j,\tau)=\arg\min_{v\in D_j}(CC(OptPS_{x_j}(v,\tau),\tau)+f_{ij}(d,v)) \tag{5}$$

x_i 在其论域元素 d 上的贪婪部分解可被定义为

$$GreedyPS_{x_i}(d,\tau)=\text{"}x_i=d\text{"}\cup\left(\bigcup_{x_j\in Childs(x_i)}OptPS_{x_j}(GeedyBase(d,x_j,\tau),\tau)\right) \tag{6}$$

上述定义中的∪是第 1 节中所给的部分解合并运行符.公式(5)和公式(6)给出了 HEDA 算法中的贪婪原则,即构造贪婪部分解的方法.在子结点已经报告了最优部分解集及其中每个元素代价的前提下,这一方法只需用到本地约束函数 f_{ij},因此,可以分布式实现.

贪婪部分解有可能是最优部分解,但也有可能不是.定理 1 将给出判断一个贪婪部分解是否为最优部分解的方法,定理 2 将给出当无法判断时,如何获得最优部分解的方法.

定理 1. 给定一个贪婪部分解 $P=GreedyPS_{x_i}(d,\tau)$,如果对$\forall x_k\in subtree(x_i)-child(x_i)-\{x_i\}$都有 $f_{ik}(d,P(x_k))=0$,则 P 是 x_i 在其论域元素 d 上的最优部分解.

证明:若 x_i 是叶结点,由公式(5)计算得到的 $GreedyPS_{x_j}(d,\tau)$ 即为"x_i=d",它显然是符合定义 5 中条件的最优部分解.若 x_i 是非叶结点,由于 DFS 树是伪树,不同分支间不存在约束关系,因此只需证明 x_i 只存在一个子结点 x_j 的情况即可.

用反证法.假设存在符合定义 5 中条件(1)和条件(2)的一个部分解 P',且其代价小于 P.由本定理条件易知:

$$CC(P,\tau)=CC(P|_{subtree(x_j)},\tau)+f_{ij}(d,P(x_j))+\sum_{x_k\in \mathrm{var}(\tau)}f_{ik}(d,\tau(x_k))+f_i(d) \tag{7}$$

$$CC(P',\tau)\geqslant CC(P'|_{subtree(x_j)},\tau)+f_{ij}(d,P'(x_j))+\sum_{x_k\in \mathrm{var}(\tau)}f_{ik}(d,\tau(x_k))+f_i(d) \tag{8}$$

由 $CC(P',\tau)<CC(P,\tau)$的假设及公式(7)和公式(8)可得:

$$CC(P'|_{subtree(x_j)},\tau)+f_{ij}(d,P'(x_j))<CC(P|_{subtree(xj)},\tau)+f_{ij}(d,P(x_j)) \tag{9}$$

可以分两种情况加以讨论:

(1) $P(x_j)=P'(x_j)$:此时有 $f_{ij}(d,P(x_j))=f_{ij}(d,P'(x_j))$,且由贪婪部分解的构造方法可知,$P|_{subtree(x_j)}$ 是 x_j 在 $P(x_j)$上的最优部分解,即 $CC(P'|_{subtree(x_j)},\tau)\geqslant CC(P|_{subtree(x_j)},\tau)$.公式(9)不成立.矛盾;

(2) $P(x_j)!=P'(x_j)$:设子结点 x_j 在其论域元素 $P'(x_j)$上的最优部分解为 Q,由最优部分解定义可知:

$$CC(P'|_{subtree(x_j)},\tau)\geqslant CC(Q,\tau),$$

代入公式(9)可得 $CC(Q,\tau)+f_{ij}(d,P'(x_j))<CC(P|_{subtree(x_j)},\tau)+f_{ij}(d,P(x_j))$,由贪婪部分解生成方法中公式(5)可知,$P$ 不是 x_i 在 d 上的贪婪部分解,"x_i=d"∪Q 才是.矛盾. □

定理 2. 给定上下文τ和 $x_i\notin var(\tau)$,任取 $d\in D_i$,有

$$OptPS_{x_i}(d,\tau)=\text{"}x_i=d\text{"}\cup\left(\bigcup_{x_j\in Childs(x_i)}\mathrm{Min}CC(OptPSSet_{x_j}(\tau\cup\text{"}x_i=d\text{"}))\right) \tag{10}$$

其中,$\mathrm{Min}CC(OptPSSet_{x_j}(\tau\cup\text{"}x_i=d\text{"}))$是 $OptPSSet_{x_j}(\tau\cup\text{"}x_i=d\text{"})$中代价最小的元素.

证明:x_i 是叶结点时显然成立.若 x_i 是非叶结点,与定理 1 的证明一样,我们证明 x_i 只存在一个子结点 x_j 的

情况.对任意符合定义 5 中条件(1)和条件(2)的部分解 P,由于 $P=$“$x_i=d$” $\cup P|_{subtree(x_j)}$,因此,

$$CC(P,\tau)=CC(P|_{subtree(x_j)},\tau)+\sum_{x_k\in subtree(x_j)}f_{ik}(d,P(x_k))+\sum_{x_k\in \mathrm{var}(\tau)}f_{ik}(d,\tau(x_k))+f_i(d) \tag{11}$$

公式(11)右侧前两项即为 $CC(P|_{subtree(x_j)},\tau\cup$“$x_i=d$”),因此可得

$$CC(P,\tau)=CC(P|_{subtree(x_j)},\tau\cup\text{“}x_i=d\text{”})+\sum_{x_k\in \mathrm{var}(\tau)}f_{ik}(d,\tau(x_k))+f_i(d) \tag{12}$$

公式(12)右侧后两项在本定理条件给定后是固定的,因此,若 $CC(P,\tau)$最小,则 $CC(P|_{subtree(x_j)},\tau\cup\text{“}x_i=d\text{”})$ 最小.定理 2 成立. □

需要指出的是,公式(11)同时给出了仅利用本地约束和子结点所报告信息计算部分解代价的方法,因此,结点在 HEDA 算法中有能力向父结点报告最优部分解集中元素的代价.

定理 1 和定理 2 的意义如下:

(1) 定理 1 是 HEDA 算法在低约束密度问题中性能优势的基础.直观上说,由于这类问题存在大量代价为 0 的赋值组合,使得定理 1 中条件成立的概率相当高,此时可直接从贪婪部分解得到最优部分解;

(2) 定理 2 是 HEDA 算法完全性保证的手段:当结点无法通过定理 1 确认贪婪部分解是最优部分解时,它可以通知其子树上结点启动一个新的、终止于本节点且扩展了上下文的子贪婪求解过程,从而得到本结点在相应论域元素上的最优部分解,使得父贪婪求解过程得以继续.由于这一机制要求已完成求解过程的结点重新计算,因此在 HEDA 算法中被称为回跳.

HEDA 算法中的回跳必须解决如下两个关键问题:

(1) 如何确定回跳的上下文?

由定理 2 可知,回跳所产生的子贪婪求解过程的上下文是对父贪婪求解过程上下文的扩充,也即公式(10)中的 $\tau\cup$“$x_i=d$”.回跳过程中可能仍需要回跳,相应上下文也逐层增加.由于每次仅增加一个赋值,且由上下文是部分解、部分解最多包含$|X|$个赋值,可得 HEDA 算法中回跳嵌套的深度至多为$|X|$.

(2) 如何确定回跳的起点?

回跳的另外一个问题是需要确定起点,也就是从哪些结点开始计算新上下文条件下的最优解.依据贪婪部分解自底向上的构造方法,回跳起点应是 $subtree(x_i)$中所有叶结点,但下述定理说明大多数情况下无需如此.

定理 3. 对于$\forall x_j\in subtree(x_i)$,如果对$\forall x_k\in Childs(x_j)$均有“$x_i=d$”$\notin DetailedSeparator(x_k)$,则有

$$OptPSSet_{x_j}(\tau\cup\text{“}x_i=d\text{”})=OptPSSet_{x_j}(\tau) \tag{13}$$

证明:由 $DetailedSeparator$ 的含义和本定理条件可知,$subtree(x_j)-\{x_j\}$中任意元素的任意赋值与 $x_i=d$ 的组合代价均为 0.因此,对任意 $d'\in D_j$,有

$$CC(OptPS_{x_j}(d',\tau),\tau))+f_{ij}(d,d')=CC(OptPS_{x_j}(d',\tau\cup\text{“}x_i=d\text{”}),\tau\cup\text{“}x_i=d\text{”})) \tag{14}$$

由于 $f_{ij}(d,d')$是确定的,可得 $OptPS_{x_j}(d',\tau)=OptPS_{x_j}(d',\tau\cup$“$x_i=d$”),定理 3 成立. □

依据定理 3,为求结点 x_i 在其论域元素 d 上的最优部分解,回跳起点集合 $Culprit$(“$x_i=d$”)可被定义为

$$\{x_j|\text{“}x_i=d\text{”}\in DetailedSeparator(x_k)\text{且}\forall x_k\in Childs(x_j),\text{“}x_i=d\text{”}\notin DetailedSeparator(x_k)\} \tag{15}$$

综上所述,HEDA 算法贪婪/回跳阶段基本流程如下:所有叶结点在本阶段开始时启动上下文为∅的贪婪求解过程;当前结点在收到子结点消息后,依据公式(4)和公式(5)中的贪婪原则生成贪婪部分解,依据定理 1 判断其是否为最优部分解,若无法确定,则实施回跳,回跳所产生的新的子贪婪求解过程的上下文由定理 2 确定,其起点则由公式(15)确定;回跳在当前结点处结束,由定理 2 可知,必定得到了相应论域元素上的最优部分解,然后父贪婪求解过程得以继续;叶结点所启动的上下文为∅的贪婪求解过程最终将终止于根结点,并得到 $OptPSSet_{root}(\varnothing)$.

4.3 解传播

在得到 $OptPSSet_{root}(\varnothing)$之后,HEDA 算法将进入解传播阶段.根结点从该最优部分解集中取出代价最小的部分解,由定义 5 中的条件可知,它就是问题的最优解.根结点通过 DFS 树向下传播这一最优解,各结点依据最优解

为其变量赋值.当解传播消息到达所有叶结点之后,算法结束.显然,这一阶段只需线性消息数目.

5 HEDA 算法具体实现

本节首先给出 HEDA 算法核心的贪婪/回跳阶段的伪代码描述,然后讨论如何在 HEDA 算法中应用基于搜索/回溯的约束优化算法中已有的一些技术.

5.1 贪婪/回跳阶段的伪代码描述

图 2 是 HEDA 算法贪婪/回跳阶段的伪代码描述.在该阶段,每个结点使用 *Context* 数据结构来记录当前贪婪求解过程的上下文;使用 *ChildCache* 来缓存子结点所报告的最优部分解集;使用 *BackjumpingStack* 来缓存哪些论域元素的贪婪部分解可能不是最优部分解,需要回跳;使用 *PSSetStack* 来保存当前贪婪求解过程所有祖先过程的结果,从而使得在应用公式(13)时可以取到父贪婪求解过程的解(如图 2 中的第 1 行~第 4 行所示).

```
Procedure Initialize
1   Context←empty;
2   ChildCache←empty;
3   PSSetStack←empty;
4   BackjumpingStack←empty;
5   if I am a leaf of DFS tree
6       Generate Leaf's OptPSSet;
7       PSSetStack.push(⟨Context,MyOptPSSet⟩);
8       SendOptPSSet(OptPSSet,Context,MyParent);

Procedure OnOptPSSet(child_opt_psset,child_context,sender)
9   ChildCache[sender]←child_opt_psset
10  Target←child_context.back()
11  if (Target.Id!=MyId) and (child_context!=Context)
12      Context←child_context.context;
13      MaintainPSSetStack();
14      SendAskSolution(Context,MyChilds);
15  if ChildCache is full
16      if (Target.Id==MyId) and (ChildCache is full)
17          OptPSSet[Target.Value]←GenerateOptPS();
18      else
19          GenerateMyOptPSSet();
20      ChildCache←empty
21      if BackjumpingStack is empty
22          if I am the root of DFS tree
23              start the value propagation stage;
24          else
25              PSSetStack.push(⟨Context,OptPSSet⟩);
26              SendOptPSSet(OptPSSet,Context,MyParent);
27      else
28          NewContext←Context;
29          NewContext.push_back(BackjumpingStack.pop());
30          SendBackjump(NewContext,MyChilds∪MyPseudoChilds);

Procedure OnBackjumping(new_context,sender)
31  Target←new_context.back();
32  if Target not in ChildsDetailedSeparator
        and Target in MyDetailedSeparator
33      Context←context
34      MaintainSessionSoltuionStack();
35      OptPSSet=PSSetStack.top();
36      OptPSSet.Context←Context;
37      add local cost of Target to each PS in OptPSSet;
38      PSSetStack.push(⟨Context,OptPSSet⟩);
39      SendOptPSSet(OptPSSet,Context,Parent);

Procedure OnAskSoltuionMsg(context,sender)
40  if Target not in MyDetailedSeparator
41      Context←context
42      MaintainPSSetStack();
43      SendOptPSSet(PSSetStack.top(),Context,Parent);

Procedure MaintainPSSetStack
44  while PSSetStack.top().Context is not Context's ancestor
45      PSSetStack.pop();

Procedure GenerateMyOptPSSet
46  for each value in MyDomain
47      PS←GenerateGreedyPS(value);
48      if PS is optimal
49          OptPSSet[Target.Value]←PS;
50      else
51          JumpingBackStack.push_back(⟨MyId,value⟩);
```

Fig.2 Actions of each node in the Greedy/Backjumping stage

图 2 贪婪/回跳阶段单个结点上的动作

结点动作是消息驱动的,存在 3 类消息:*OptPSSetMsg*,*BackjumpMsg* 和 *AskSolutionMsg*.*OptPSSetMsg* 是携带了代价信息的最优部分解集,*BackjumpMsg* 通知子孙结点开始一个新的贪婪求解过程,*AskSolutionMsg* 则强制子结点报告一个最优部分解.结点上各消息的处理操作如下:

- *OnOptPSSet* 定义了收到子结点最优部分解集时的动作.结点首先判断子结点的 *Context* 是否与本身 *Context* 一致,若不一致则说明是一个新的贪婪求解过程,此时,结点会实施 *PSSetStack* 维护、发送 *AskSolution* 消息等动作(图 2 中第 12 行~第 14 行).其中,*PSSetStack* 维护保证 *PSSetStack* 中只保存了祖先过程的结果(图 2 中第 44 行、第 45 行).在收到了所有子结点的最优部分解之后,若结点是当前贪婪求解过程的发起者(终点),则依据公式(10)为父贪婪求解过程生成一个确定的最优部分解(图 2 中第 17

行);否则,依据公式(6)为论域中所有元素生成贪婪部分解,并使用定理 1 进行判断,将那些无法确认最优解的论域元素放到 *BackjumpingStack* 当中,等待回跳(图 2 中第 19 行、第 46 行~第 51 行).如果 *BackjumpingStack* 为空,则说明所有论域中元素都已得到最优部分解,可以将最优部分解集发送给父结点,或者进入解传播阶段(图 2 中第 21 行~第 26 行);否则,从 *BackjumpingStack* 取出一个元素实施回跳.由于公式(15)所给结点必定在当前结点的子结点集和伪子结点集中,回跳消息被发给所有这些结点(图 2 中第 28 行~第 30 行);

- *OnBackjumping* 定义了结点收到回跳消息时的动作.结点只有判断其满足公式(15)的条件后,才会处理该消息(图 2 中第 32 行).此时,依据公式(13),回跳起点的最优部分解集可以沿用父贪婪求解过程的结果(图 2 中第 35 行),但其中最优部分解的代价可能会发生了变化(图 2 中第 36 行);
- *OnAskSolution* 定义了结点收到 *AskSolutionMsg* 时的动作.由于定理 3 确定的回跳起点并非所有叶结点,因此,某些子树可能不在当前贪婪求解过程中.为了避免无限等待某个分支报告的情况,父结点会主动向子结点发送 *AskSolutionMsg* 消息(图 2 中第 14 行),结点只有在确认其子树不会主动报告最优部分解时才会处理该消息(图 2 中第 40 行).此时,由定理 3 可知,父贪婪求解过程的结果就是本次贪婪求解过程的结果(图 2 中第 43 行).

5.2 使用约束优化算法中的已有技术

约束优化算法已有的弧一致性维护、定界剪枝等技术可以应用到 HEDA 算法之中.

(1) 弧一致性维护

弧一致性是约束满足算法中的概念,文献[19]等研究工作将其扩展到约束优化问题之中.HEDA 算法的性能与问题中代价为 0 的二元赋值组合个数密切相关,而弧一致性维护有可能减少此类赋值组合的个数[19],从而提高 HEDA 算法的性能.例如,图 3 左侧问题中的约束函数定义如下:$f_i(a)=1$,$f_{ij}(a,a)=f_{ij}(a,c)=1$,$f_{ij}(a,b)=2$.经过弧一致性维护之后,它变成了右侧的问题,也即将二元约束的代价“公共”部分加到了一元约束上.这两个问题是等价的,但后者只剩下了一个代价为 1 的二元赋值组合.

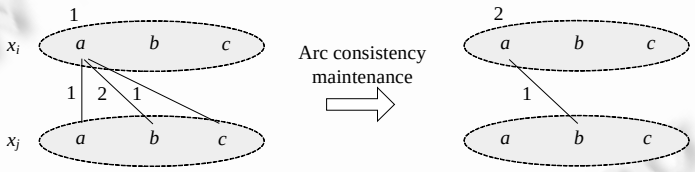


Fig.3 Example of arc consistency maintenance

图 3 弧一致性维护示例

弧一致性维护可在 HEDA 算法的结点组织阶段以分布式方式实施:如果某个结点通过检查本地约束,发现其论域内所有元素均与另一结点某个赋值的组合代价非 0,则可以执行弧一致性维护动作.

(2) 利用历史信息

搜索/回溯算法中可利用历史信息维护一个上界,并基于该值来进行剪枝[2].这一机制可以应用到 HEDA 算法的回跳过程中:由于回跳时我们实际上已经得到了一个部分解,只是不能确定其是否最优,因此可以将这一部分解的代价作为回跳过程中的上界.在回跳过程中,如果某个结点发现其解集中某个元素的代价已超过该上界,则它可以不向父结点报告这一元素,从而减少最优部分解集中元素的个数,避免后续过程中一些不必要的回跳.

6 HEDA 算法性能评估

本节首先对 HEDA 算法的复杂性作简单的定量分析,然后在实验环节验证算法在低约束密度问题中的性能优势.

6.1 算法复杂性

在 HEDA 算法中,最长的消息是携带最优部分解集的 *OptPSSetMsg*.设分布式约束优化问题中变量数为 n,且所有变量论域大小均为 m,则最优部分解集最多包括 m 个部分解,每个部分解最多包括 n 个赋值,即 *OptPSSetMsg* 的长度为 $O(mn)$.HEDA 算法的空间复杂度(单个结点上的空间需求)主要取决于缓存所有祖先过程最优部分解集的 *PSSetStack* 的大小,由第 4.2 节讨论可知,回跳嵌套深度不会超过 n,因此可得空间复杂度是 $O(mn^2)$.

HEDA 算法在如下情况下达到消息数目的上界:1) 所有结点组织成一条链;2) 在求任意非叶结点任意论域元素上的最优部分解时,贪婪原则均不成立,且需要回跳到叶结点.直观地说,第 1 个条件使得问题无法通过将结点组织成伪树来分解,第 2 个条件使得算法中任何“投机”策略都不成立.可用归纳法证明,此时,HEDA 算法的消息数目为

$$(m+1)^{n-2}+\frac{((m+1)^{n-1}-(m+1))}{m} \tag{16}$$

因此,HEDA 算法消息长度和空间复杂度上界是多项式级别的,消息数目上界是指数级别的.

6.2 实验结果及其分析

分布式约束优化问题求解算法的性能可采用消息数目、最长消息序列、约束检查总数、周期数、非并发约束检查总数等单位来衡量[4,20,21],其中,包括消息数目、长度等在内的网络负载是衡量算法性能的重要指标[4],也是本文实验的主要对比对象.我们在随机 MaxCSP 问题、基于无标度网络的问题和第三方测试集上进行了测试,对比了 HEDA 算法与 NCBB,Adopt 和 DPOP 这 3 种主流算法的性能.

以下实验除 Adopt 算法外均在同一模拟器(http://cyberdb.googlepages.com/ dawn2)上实施,模拟器中每个 Agent 由一个线程模拟,Agent 间只能通过消息通信,伪树生成采用 MCS(maximum cardinality set)启发式[4].由于异步算法性能与具体实现密切相关,如消息处理延迟等都可能影响消息数目[22],因此,我们直接使用 Adopt 算法作者所提供的参考实现[23]来进行对比.实验环境为 Core2Duo 6600,4GB RAM, Windows7 RC,VS2008,Java RE 6.13.

(1) 随机 MaxCSP 问题

MaxCSP 问题[24]中所有约束都具有相同代价 1,也即其最小代价解违反的约束数量最少.它被广泛用来衡量约束优化算法的性能[25,26].随机 MaxCSP 问题可以由 4 个参数生成[3]:结点个数 n、论域大小 m、约束密度 p_d、约束紧度 p_t.其中,约束紧度是存在约束关系的结点间赋值组合代价非 0 的概率.由于对消息数目的比较只有在消息长度位于同一级别时才有意义[4],我们只与 NCBB 和 Adopt 算法进行了对比.

- 第 1 组实验:约束紧度可变的随机 MaxCSP 问题

在约束密度固定的情况下,约束紧度越大,结点间互相制约的关系就越紧密,问题直观上也就越“难”.第 1 组实验用来测试 HEDA 算法在问题难易程度发生变化时的性能表现.实验参数被设定为 n=15,m=5,p_d=0.25,p_t 位于[0.5,1.0]区间内并以 0.025 为单位步进递增,设定 p_t 大于 0.5 是为了产生优化问题(即最优解代价大于 0).对于每一组参数随机生成 30 个问题,并求相应指标的平均值.

图 4(a)给出了各种算法单个问题所需的平均消息数目.可以看出,HEDA 算法较 Adopt 和 NCBB 算法有明显优势,且由于弧一致性维护机制的存在,消息数目表现出了类似于某些分布式约束满足算法的相变(phase transition)特性[22],即约束紧度增长到一定阈值后所需消息数目反而开始减少.

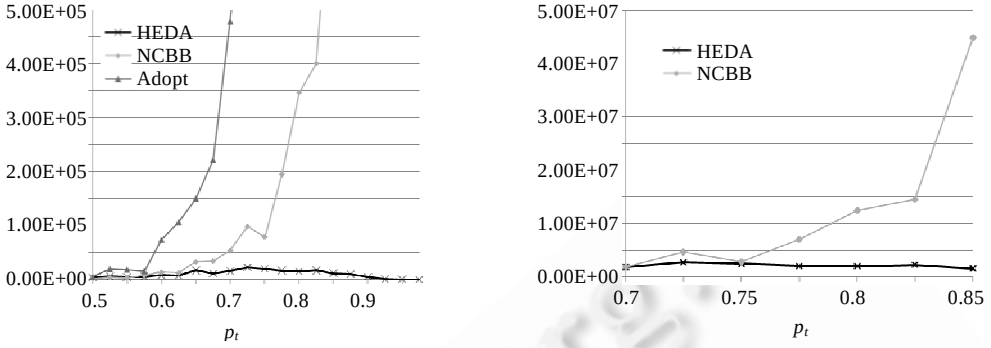
由于在 HEDA 算法中,*OptPSSetMsg* 消息同时携带多个部分解,因此,单条消息长度要大于 NCBB 算法.为了更准确地反映网络负载,我们进一步对约束紧度在 0.7~0.85 之间的单个问题总消息长度的平均值进行了对比(如图 4(b)所示).由于 Adopt 算法采用了不同的模拟器,消息编码格式等存在差异,因此未在此项上进行对比.

- 第 2 组实验:结点数可变的随机 MaxCSP 问题

第 2 组实验被用来测试 HEDA 算法的可伸缩性.我们将问题结点数 n 设定在[10,30]区间内并以 2 为单位步进递增,m=5,结点在约束图中的平均度维持在恒定的 3(即 p_d 位于 0.33~0.10 之间),p_t=0.8.对于每一组参数随机

生成 30 个问题,并求相应指标的平均值.

图 5(a)给出了各种算法单个问题所需的平均消息数目.可以看出,NCBB 和 Adopt 算法消息数目在 20 个结点以后均已超过 120 万条,而 HDEA 算法在 28 个结点时的 30 个问题实测平均值为约 32 万条.图 5(b)进一步给出了 10~22 区间内 HEDA 和 NCBB 算法单个问题总消息的长度平均值.

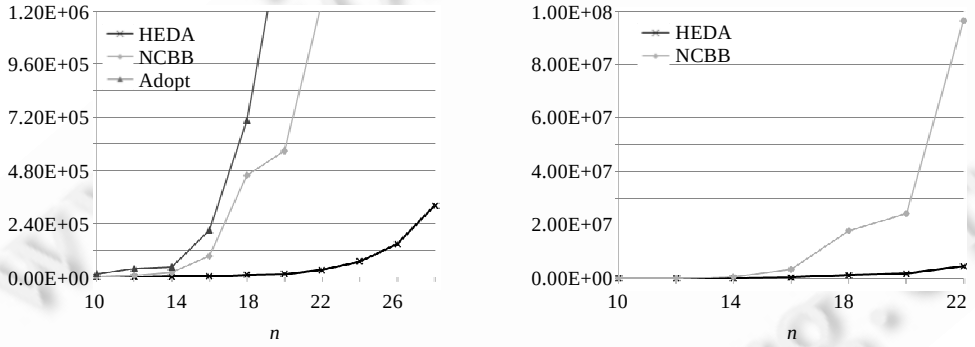


(a) Message count (a) 消息数目

(b) Total message length (bytes) (b) 消息总长度(字节)

Fig.4 Experiment results of the first set of random MaxCSP

图 4 随机 MaxCSP 问题第 1 组实验结果



(a) Message count (a) 消息数目

(b) Total message length (bytes) (b) 消息总长度(字节)

Fig.5 Experiment results of the second set of random MaxCSP

图 5 随机 MaxCSP 问题第 2 组实验结果

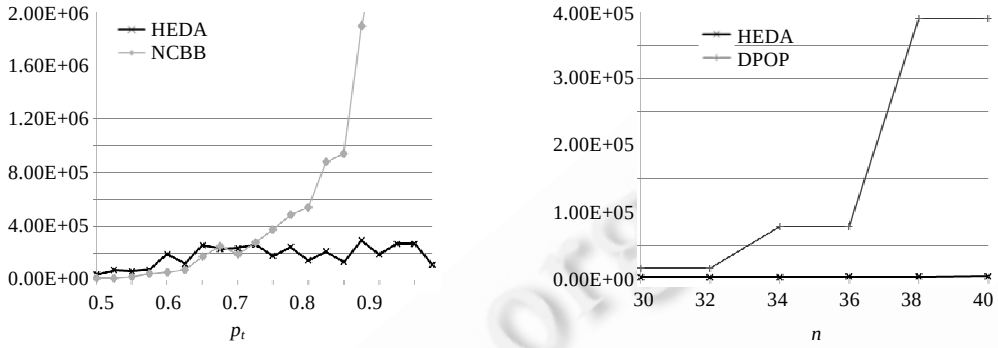
(2) 基于无标度网络的问题

许多真实网络结点度的分布表现出无标度特征,而非随机分布[27],以这些真实网络为基础的分布式约束优化问题约束图同样具备无标度特征.在本实验中,我们使用 T_{ANG} 模型[28]生成具备无标度特征的约束图,然后进一步通过指定论域大小、约束代价等来生成分布式约束优化问题.实验所使用的 T_{ANG} 模型的参数如下:结点数 n=30,起始有 4 个节点,然后每步增加 1 个结点和 2 条边(即约束密度约为 0.07),ε=0.在所生成的约束图的基础上,指定每个结点论域大小为 5,约束紧度 p_t 位于[0.5,1.0)区间内并以 0.025 为单位步进递增,约束代价则是[1,5]区间内的随机整数.

图 6(a)是按上述模型生成 30 个问题后得到的单个问题平均消息数目.由于 Adopt 算法已无法在合理时间内求解大多数问题,因此,我们仅与 NCBB 算法进行了对比.可以看到,HEDA 算法在 p_t>0.75 时具备明显性能优势,在 p_t<0.7 时算法性能则略差于 NCBB 算法.后者是因为:一方面,当约束紧度较小时,最优解代价也较小,因

此,NCBB 算法可以快速剪枝的概率较高;另一方面,此问题中约束代价随机分布,而非 MaxCSP 问题中统一的 1,因此弧一致性维护机制对算法性能的贡献将有所减少.

HEDA 算法通过贪婪原则将消息长度控制在多项式级别,我们通过与 DPOP 算法的对比实验验证了这一特点(如图 6(b)所示).实验结点数 n 位于[30,40]区间内并以 2 为单位步进递增,约束紧度为 0.8,其他参数同前.此时,在实测的 30 个问题中,HEDA 算法的最大单条消息长度保持在多项式级别(<4k 字节),而 DPOP 算法中最大单条消息长度超过了 5^8,即论域大小的 8 次方.



(a) Message count
(a) 消息数目

(b) Total message length (byte)
(b) 消息总长度(字节)

Fig.6 Experiment results of DCOP based on scale-free constraint graph

图 6 基于无标度网络的分布式约束优化问题实验结果

(3) 第三方测试集中的真实问题

前述实验中的问题均由模拟器根据参数和模型随机生成,本实验使用与 Adopt 算法参考实现一起提供的地图着色问题测试集[23],并对比了 HEDA 算法与 Adopt 算法的性能.该测试集兼具低约束密度和低约束紧度两个特点:约束密度分布在 0.36~0.05 之间;约束紧度则为恒定的 0.33.由于地图着色问题的特点,HEDA 算法将无法实施任何弧一致性维护动作,因此可反映无弧一致性维护的 HEDA 算法性能.我们对该测试集中结点数 10~16 之间、平均度为 3 的前 5 个问题进行了测试,每个问题所需消息数目见表 1,平均消息数目如图 7 所示.

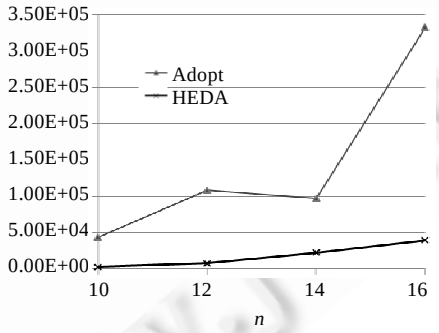


Fig.7 Average message count of graph coloring problems

图 7 图着色问题平均消息数目

Table 1 Detailed message count of graph coloring problems

表 1 图着色问题具体消息数目

Name	n=10		n=12		n=14		n=16	
	Adopt	HEDA	Adopt	HEDA	Adopt	HEDA	Adopt	HEDA
Problem-GraphColor-⟨n⟩_3_3_0.4_r0	36 941	2 183	109 562	6 758	69 300	10 835	712 581	46 732
Problem-GraphColor-⟨n⟩_3_3_0.4_r1	29 596	2 821	152 841	11 837	67 607	12 857	162 991	26 885
Problem-GraphColor-⟨n⟩_3_3_0.4_r2	60 833	3 182	14 674	2 783	81 424	7 426	315 790	52 668
Problem-GraphColor-⟨n⟩_3_3_0.4_r3	28 984	2 544	185 490	8 709	169 528	26 596	204 942	14 271
Problem-GraphColor-⟨n⟩_3_3_0.4_r4	63 513	3 127	79 838	9 068	94 981	53 564	265 005	57 090

7 结束语

本文针对低约束密度的分布式约束优化问题,提出了一种基于贪婪和回跳思想的分布式约束优化算法HEDA.其特点在于,在保持多项式级别的消息长度和空间复杂度的同时,可以使用较少的消息数来求解此类问题.本文给出了算法的基本思想、关键机制及其正确性证明、主要阶段伪代码实现、简要的算法复杂度分析,并通过多个实验场景验证了 HEDA 算法的上述特点.由于 HEDA 算法在每一个结点上均缓存了一个最优部分解集,在父结点论域发生变化或者结点动态加入/退出情况下有可能复用此解集,因此,我们下一步的工作将探讨 HEDA 算法在开放的分布式约束优化问题[29]及动态和可演化问题[4]中的应用.

References:

[1] Modi PJ, Shen WM, Tambe M, Yokoo M. Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence, 2005,161(1-2):149−180. [doi: 10.1016/j.artint.2004.09.003]

[2] Dechter R. Constraint Processing. San Fransisco: Morgan Kaufmann Publishers, 2003.

[3] Meisels A. Distributed Search By Constrained Agents. Heidelberg: Springer-Verlag, 2007.

[4] Petcu A. A class of algorithms for distributed constraint optimization [Ph.D. Thesis]. Lausanne: Ecole Polytechnique Federale de Lausanne, 2007.

[5] Nguyen XT, Kowalczyk R, Phan MT. Modelling and solving QoS composition problem using fuzzy DisCSP. In: Proc. of the IEEE Int'l Conf. on Web Services. 2006. 55−62. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4032012 [doi: 10.1109/ICWS.2006.93]

[6] Newman MEJ. The structure and function of complex networks. SIAM Review, 2003,45(2):167−256.

[7] Chechetka A, Sycara K. No-Commitment branch and bound search for distributed constraint optimization. In: Proc. of the Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2006). 2006. 1427−1429. http://portal.acm.org/citation.cfm?id=1160900 [doi: 10.1145/1160633.1160900]

[8] Petcu A, Faltings B. A scalable method for multiagent constraint optimization. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence (IJCAI 2005). 2005. 266. http://portal.acm.org/citation.cfm?id=1642336

[9] He LJ, Zhang W. An agent organization structure for solving DCOP based on the partitions of constraint graph. Journal of Computer Research and Development, 2007,44(3):434−438 (in Chinese with English abstract). [doi: 10.1360/crad20070310]

[10] Pearce JP, Maheswaran RT, Tambe M. Dcop games for multi-agent coordination. In: Proc. of the Int'l Workshop on Distributed Constraint Reasoning. 2005. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.1370

[11] Buckley F, Lewinter M. A Friendly Introduction to Graph Theory. Prentice Hall, 2002.

[12] Maheswaran RT, Tambe M, Bowring E, Pearce JP, Varakantham P. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: Proc. of the Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2004). 2004. 310−317. http://portal.acm.org/citation.cfm?id=1018762 [doi: 10.1109/AAMAS.2004.257]

[13] Sultanik E, Modi PJ, Regli WC. On modeling multiagent task scheduling as a distributed constraint optimization problem. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence (IJCAI 2007). 2007. 1531−1536. http://portal.acm.org/citation.cfm?id=1625523

[14] Zhang W, Xing Z, Wang G, Wittenburg L. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In: Proc. of the Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2003). 2003. 185−192. http://portal.acm.org/citation.cfm?id=860605 [doi: 10.1145/860575.860605]

[15] Hirayama K, Yokoo M. Distributed partial constraint satisfaction problem. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. 1997. http://www.springerlink.com/index/w0g12371455j61x5.pdf [doi: 10.1007/BFb0017442]

[16] Davin J, Modi PJ. Impact of problem centralization in distributed constraint optimization algorithms. In: Proc. of the Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2005). 2005. 1057−1063. http://portal.acm.org/citation.cfm?id=1082633 [doi: 10.1145/1082473.1082633]

[17] Petcu A, Faltings B. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence (IJCAI 2007). 2007. http://www.aaai.org/Library/IJCAI/2007/ijcai07-234.php

[18] Petcu A, Faltings B. ODPOP: An algorithm for open/distributed constraint optimization. In: Proc. of the National Conf. on Artificial Intelligence (AAAI 2006). 2006. http://www.aaai.org/Library/AAAI/2006/aaai06-112.php

[19] Larrosa J, Schiex T. Solving weighted CSP by maintaining arc consistency. Artificial Intelligence, 2004,159(1):1−26. [doi: 10.1016/j.artint.2004.05.004]

[20] Junges R, Bazzan A. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In: Proc. of the Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008). 2008. 599−606. http://portal.acm.org/citation.cfm?id=1402308

[21] Gershman A, Zivan R, Grinshpou T, Meisels A. Measuring distributed constraint optimization algorithms. In: Proc. of the Int'l Workshop on Distributed Constraint Reasoning. 2008. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.8032

[22] Bejar R, Domshlak C, Fernández C, Gomes C, Krishnamachari B, Selman B, Valls M. Sensor networks and distributed CSP: Communication, computation and complexity. Artificial Intelligence, 2005,161(1-2):117−148. [doi: 10.1016/j.artint.2004.10.001]

[23] USC distributed constraint optimization problem (DCOP) repository. 2009. http://teamcore.usc.edu/dcop/

[24] Galinier P, Hao JK. Tabu search for maximal constraint satisfaction problems. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. 1997. http://www.springerlink.com/content/d147151702x758w3/ [doi: 10.1007/BFb0017440]

[25] Gershman A, Meisels A, Zivan R. Asynchronous forward-bounding for distributed constraints optimization. In: Proc. of the European Conf. on Artificial Intelligence (ECAI 2006). 2006. http://portal.acm.org/citation.cfm?id=1567044

[26] Ezzahir R, Bessiere C, Benelallam I. Dynamic backtracking for distributed constraint optimization. In: Proc. of the European Conf. on Artificial Intelligence (ECAI 2008). 2008. http://portal.acm.org/citation.cfm?id=1567527

[27] Wang XF, Li X, Chen GR. Complex Network Theory and its Application. Beijing: Tsinghua University Press, 2006 (in Chinese).

[28] Bar S, Gonen M, Wool A. An incremental super-linear preferential Internet topology model. In: Proc. of the Annual Passive & Active Measurement Workshop (PAM). 2004. http://www.springerlink.com/content/8pwttx4wkxwr7lvl/ [doi: 10.1007/978-3-540-24668-8_6]

[29] Faltings B, Macho-Gonzalez S. Open constraint programming. Artificial Intelligence, 2005,161(1-2):181−208. [doi: 10.1016/j.artint.2004.10.001]

附中文参考文献:

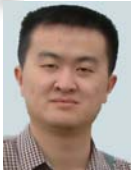
[9] 贺利坚,张伟.基于约束图分片求解 DCOP 的 Agent 组织结构.计算机研究与发展,2007,44(3):434−438. [doi: 10.1360/crad20070310]

[27] 汪小帆,李翔,陈关荣.复杂网络理论及其应用.北京:清华大学出版社,2006.

丁博(1978－),男,湖南攸县人,博士,助理研究员,CCF 会员,主要研究领域为分布计算,软件自适应.

史殿习(1966－),男,博士,副教授,CCF 会员,主要研究领域为分布计算,普适计算.

王怀民(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布计算,网络安全.



唐扬斌(1975－),男,博士,讲师,CCF 会员,主要研究领域为可信计算,分布式算法.