doi: 10.3969/j.issn.1005-3026.2015.02.008

一种改进的分布约束优化算法 MULBS$^+$

段沛博, 张长胜, 张 斌
(东北大学 信息科学与工程学院, 辽宁 沈阳 110819)

摘 要: 完备算法虽然能够求得分布式约束优化问题最优解,但要消耗大量资源及时间,相反,非完备算法通过求得次优解来提高效率. MULBS 作为一个有效的非完备算法,虽然在求解质量和时间上有所提高,但在解决赋值冲突时采用的回溯策略及并行搜索方面存在不足. 通过对该算法的深入分析,本文针对上述问题进行了改进,提出其改进算法 MULBS$^+$. 通过在回溯策略中引入最小冲突选择机制,以及在约束图密度较大时采用基于动态子图划分的并行搜索策略,进一步提高了算法的性能. 实验表明,该算法除增加一定的通信信息外,其执行时间及求解质量均优于原算法.

关 键 词: 分布式约束优化;动态子图;图密度;MULBS;MULBS$^+$
中图分类号: TP 311 **文献标志码**: A **文章编号**: 1005-3026(2015)02-0188-06

An Improved Distributed Constraint Optimization Algorithm MULBS$^+$

DUAN Pei-bo, ZHANG Chang-sheng, ZHANG Bin
(School of Information Science & Engineering, Northeastern University, Shenyang 110819, China. Corresponding author: ZHANG Bin, professor, E-mail: zhangbin@ise.neu.edu.cn)

Abstract: Although complete algorithms can be used in finding the optimal solution of a distributed constraint optimization problem, the resource and time consumptions are heavy. On the contrary, the shortcoming can be avoided in incomplete algorithms by obtaining suboptimal solutions. As an effective incomplete algorithm, MULBS can improve the quality of solution and reduce the runtime of solving process. However, there are shortcomings in parallel search and backtracking processes in dealing with conflict assignments. Based on thorough analysis of MULBS, an improved algorithm MULBS$^+$ is proposed, which overcomes the disadvantages of backtrack strategy caused by conflict and parallel search strategy in the original algorithm. In MULBS$^+$, a minimal conflict selection mechanism is introduced in backtracking. On the other hand, MULBS$^+$ adopts global parallel search strategy based on dynamic graph partitioning which can quickly find a better solution in a constraint graph with large density. Experiments show that the proposed algorithm is better than the original one in both execution time and solution quality, except a certain increase of communication information.

Key words: distributed constraint optimization; dynamic sub-graph; graph density; MULBS; MULBS$^+$

分布式约束优化问题(distributed constraint optimization problem, DCOP)作为一个有效的多agent系统(MAS)模型[1],被广泛应用于诸如灾难救援[2]、传感器感知网络[3]、虚拟环境训练[4]、多agent组织[5]、多agent团队协作[6]等实际问题中. 其形式化定义为一个五元组$\langle A, X, \alpha, D, F\rangle$,其中,$A=\{a_1, a_2, \cdots, a_p\}$表示agent集合;$X=\{x_1, x_2, \cdots, x_n\}$表示agent包含的变量集合;$\alpha: X\rightarrow$

收稿日期: 2013-12-24
基金项目: 国家自然科学基金资助项目(61100090);中央高校基本科研业务费专项资金资助项目(N110204006, N120804001);沈阳市科技计划项目(F12-277-1-80);宁夏回族自治区自然科学基金资助项目(NZ13265).
作者简介: 段沛博(1987-),男,辽宁沈阳人,东北大学博士研究生;张 斌(1964-),男,辽宁本溪人,东北大学教授,博士生导师.

A 表示每一个 $x_i \in X$ 有且只属于一个 $a_j \in A$；$D = \{d_1, d_2, \cdots, d_n\}$ 表示每个变量有限取值空间的集合，对于每个 $x_i \in X$ 有且只有一个 $d_i \in D$ 与之对应；$F = \{f_1, f_2, \cdots, f_m\}$ 表示效应函数集合. 一个效应函数 $f_{ij}: d_i \times d_j \rightarrow R^+ \cup \infty$ 表示两个 agent 或两个变量间的关联关系.

由于非完备算法能够在有限的时间内，通过寻找最优解或次优解来解决分布式约束优化问题，因此被广泛应用于大规模、动态环境中的分布式约束优化问题，并且具有更好的稳定性和健壮性[7-8]，而 MULBS[9]（multiple local bounded search）就是该类算法的典型代表. 该算法首先根据贪心原则，至下而上为每个 agent 中的变量赋值，构建初始解；其次，算法至上而下，使每个 agent 中的变量通过局部搜索完善全局解，进而得到最优解或次优解. 由于该算法采用逐级回溯的方式解决变量间取值的冲突，同时又仅在同优先级的节点中进行搜索，因此求解效率受到一定制约.

本文在 MULBS 算法的基础上提出了 MULBS$^+$ 算法，该算法从两方面对 MULBS 进行了改进. 首先，以最小冲突选择机制代替原算法的逐级回溯策略；另一方面，算法根据约束图密度，通过子图的动态划分，进行子图内的局部并行搜索和子图间的全局并行搜索. 实验结果表明，在约束图密度较大的分布式约束优化问题中，除个别情况由于并行搜索的消息传递造成的执行时间延迟，改进算法无论在求解准确度和求解速度方面都优于原算法.

1　MULBS 算法分析

MULBS 算法是一个用以解决大规模分布式约束优化问题，特别是会议排班问题的非完备算法. 该算法的执行过程主要分为两个阶段：一个全局候补解的生成，全局候补解的重定义.

在 MULBS 求解过程中，有两个问题备受关注，第一是不同节点取值发生冲突时的解决策略，第二是算法的并行搜索方式. 这两个问题都出现在算法的第二个阶段中，也是决定 MULBS 算法求解效率和质量的关键，以下是 MULBS 算法针对这两个问题给出的解决方案.

1）冲突解决.

在异步算法中，不同节点的取值冲突是需要解决的一个重要问题. 通过节点间良好的信息交流可以很好地解决该类冲突，但是，一个好的通讯机制依靠节点间传送的大量信息，这也意味着算法对存储空间的需求增大，求解的速度也相应受到限制.

在解决冲突的过程中，为了减小消息传递带来的弊端，MULBS 算法采用了回溯策略. 在回溯中，冲突消息逐次向高级节点传递，直至存在一个高级节点可以解决冲突. 回溯策略避免了在局部冲突解决过程中，由于过多的消息传递带来的负面影响.

2）并行搜索策略.

结合 MULBS 的执行过程可以获知，含有当前解的 STORE_SOLUTION 消息只有在同一优先级的节点结束局部搜索后，才会被传向低优先级的节点，这说明 MULBS 中采用的并行搜索策略严格意义上是一种局部的并行搜索，即伪树中同层节点的并行搜索. 在这种并行搜索策略下，如果约束图的节点不多，算法的执行效率不会受到太大影响；然而，如果是一个密度较大的复杂约束图，执行速度肯定会受到一定制约.

2　MULBS$^+$ 算法

MULBS 的求解目的是更快地找出一个大规模分布式约束优化问题的解. 作为一个非完备算法，通过分析可知，在 MULBS 重定义解的过程中，由于冲突导致的回溯及并行搜索的局限性，算法的求解性能会受到一定影响. 对此，本文给出了 MULBS 的改进算法 MULBS$^+$，该算法采用最小冲突解决机制和基于子图划分的全局并行搜索策略，增强了算法性能，扩大了原算法的求解范围.

2.1　最小冲突选择机制

MULBS 算法利用回溯策略解决冲突的过程包含两种情况. 第一种情况是存在这样一个高级节点，能够选择合适的值来解决子节点冲突；在这种情况下，发生冲突的子节点可能会通过局部搜索改变自身赋值情况，也可能不发生改变. 另一种情况是不存在这样一个能够解决冲突的高级节点，这样，为了避免算法陷入冲突解决的局部循环，就需要一种执行机制代替回溯的策略.

本文提出一种最小冲突选择机制代替回溯策略，能够快速解决子节点间的冲突. 为便于描述该机制，首先对变量及变量间关系进行如下说明.

对于任意的两个变量 x_i 和 x_j，Optcost(x) 表示以变量 x 为视角的局部解 ps(x) 的最优解，val(x_i, x_j) 表示 x_i 与 x_j 在局部最优化情况下的变量取值. Set(x) 表示与 x 具有二维约束关系的变

量集合. 如果冲突发生,则存在变量 $x \in (\mathrm{Set}(x_i) \cap \mathrm{Set}(x_j))$,使 $\mathrm{val}(x,x_i)$ 和 $\mathrm{val}(x,x_j)$ 对 x 的取值不相同.

基于上述变量关系,当任意两个变量 x_i 和 x_j 发生冲突时,算法优先选择具备以下特性的变量:

$x = \min(\mathrm{Optcost}(x_i), \mathrm{Optcost}(x_j))$,即选择该节点局部解的效应值最小;

$x = \min(\mathrm{Set}(x_i), \mathrm{Set}(x_j))$,如果局部解的效应值相同,含有的邻居节点最少;

$x = \mathrm{Random}(x_i, x_j)$,以上两个条件都相同时,则向上回溯,并计算与该节点相关的子图密度,若密度大于原约束图密度,则随机选取节点;否则更改高级节点赋值信息,并不断向上回溯,直到根节点.

2.2 基于动态子图划分的全局并行搜索策略

在解的重定义阶段,MULBS$^+$算法试图在每个节点处,并行传递 STORE_SOLUTION 消息给低优先级的节点以提高搜索速度. 这种搜索策略势必导致一定量的冲突产生. 为了解决冲突,本文在算法中引入 CONFLICT_SOLUTION 消息. 该消息含有当前发生冲突节点的取值信息,由低优先级节点传向高优先级节点. 一个复杂的分布式约束优化问题往往会转化成为一个高约束密度的约束图. 本文假设一个密度为 d 的约束图有 n 个 agent(变量),则图中约束边的数量 e 与两者的关系定义为

$$e = \frac{n \times (n-1) \times d}{2}. \tag{1}$$

过于稠密的约束关系会导致算法在全局并行搜索时产生大量 CONFLICT_SOLUTION 消息,使算法的执行时间受到一定影响. 为了充分发挥全局并行搜索的有效性,避免在高约束密度情况下算法执行时间骤增,本文提出一种基于动态子图划分的全局并行搜索策略,该策略能够有效发挥局部和全局并行搜索的特点. 考虑到一个约束图生成后,它所含有的节点数量、约束数量是确定的,则根据式(1)可知,该约束图的密度也随之确定. 根据这一特点,本策略首先在算法的预处理阶段对约束图的密度进行判断. 将原约束图划分成 n 个满足一定条件的子图,称这样的子图为团.

定义 1 给定一个约束图 $G(V,E)$,以及子图 $G_i(V,E)$,有 $V_{\in G_i} \in V_{\in G}$,且 $E_{\in G_i} \in E_{\in G}$,若 $\cup_{i=1}^{n} G_i = G$,对于 $\forall i,j$ 有 $G_i \cap G_j = \varnothing (i \neq j)$,并且 G_i 的图密度 $\mathrm{density}(G_i) < \mathrm{density}(G)$,则 G_i 就是原约束图 G 的一个团.

在每个团中,算法采用局部的搜索策略,整体上,算法对所有团进行全局并行搜索. 由于每个团之间有约束存在,当约束的两个变量发生取值冲突时,不同子图之间的冲突也随之发生;为了能够解决这一冲突,本文通过在团之间传递 CONFLICT_SOLUTION 消息来解决并行搜索过程中产生的冲突. 团的划分过程在算法的预处理阶段进行,因此并不影响算法的执行时间. 在子图划分的情况下,算法的执行时间由执行时间最长的某一子图决定,为了能进一步加快算法的求解速度,避免初始化的子图划分模式成为制约算法执行的瓶颈,MULBS$^+$允许算法在执行过程中对子图进行动态划分. 当超过当前子图数量一半的子图完成局部搜索时,算法对剩余的子图进行进一步的划分,直到算法找到最优解为止.

2.3 MULBS$^+$算法描述及分析

参照 MULBS 算法,本文也通过功能模块的划分对 MULBS$^+$算法进行阐述. 在 MULBS 算法的基础上,主函数中的预处理模块添加了对 GraphPartition 函数的调用,该函数用来实现对当前约束图或子图的划分. 主函数中的 updateCurrentPartialSolution 函数添加了 solveConflict 函数的调用. solveConflict 函数的作用是利用最小冲突选择机制来解决变量取值的冲突问题;同时,为了提高算法全局搜索的并行性,对 LocalSearch 函数也做了必要的修改.

算法 1 MULBS$^+$——主函数

```
1:  procedure Initialize( )
    // 与原函数相比,该函数添加了对 GraphPartition 函数的调用
2:  GraphPartition(G);
3:  procedure GraphPartition(G)
    // 该函数用于对约束图进行划分
4:      if density(C) < density(G)
5:      C(i) = C;
6:      end if
7:      if V(C) < V(G)
8:      GraphPartition(G);
9:      end if
10: procedure localSearch ( Threshold, NewSolution, NewThreshold,conflict_solution)
    // 该函数添加了动态子图划分时机的判断,并完成并行搜索功能
11: num_subgraph = get_localsearch( );
12: if num_subgraph < num_graph then continue
13: else
14: GraphPartition(G);
15: end if
16: procedure updateCurrentPartialSolution(NewSolution,
```

```
NewCost)
    // 该函数添加了 solveConflict 函数的调用
17： if all children have sent try_local_solution to parent
18：     if try_local_solution conflict
19：      solveConflict(try_local_solution);
20：   else if NewCost < CurrentCost then
21：     CurrentPartialSolution←NewSolution;
22： end if
23：  CurrentCost←NewCost;
24：  CurrentValue←agentValue(NewSolution);
25： else continue
26： end if
27： procedure SolveConflict(try_local_solution)
    // 最小冲突选择机制的实现
28： if exit the smallest cost in try_local_solution ;
29：     node←child with try_local_solution;
30： else if no smallest cost
31：     if the number of association nodes are different
32：       node ← child with least number of nodes in
     part_solution;
33：   else node←backtrack(density)
34：   end if
35：   end if
36： NewSolution←node. NewSolution;
37：     NewCost←node. NewCost;
38：      UpdateCurrentPartialSolution(NewSolution,
     NewCost);
39：     SendConflictSolution(NewSolution, Cost);
40： end if
41： procedure SendConflictSolution(NewSolution, Cost)
    // 并行搜索中冲突消息的传递
42： Solution←NewSolution
43：     while(i < |nodes in low level|do
44： send(store_solution, node[i], Solution, Cost);
45： i←i+1;
46： end while
47： procedure sendPartialSolution(NewSolution)
48： procedure sendTryLocalSolution(Solution, Cost)
49： procedure sendStoreSolution(Solution,Cost)
```

算法 2　MULBS⁺——消息处理

```
1： procedure evaluate(PARTIAL_SOLUTION, Solution)
2：  procedure evaluate(STORE_SOLUTION, NewSolution,
     NewCost)
3： C←CurrentPartialSolution;
4： Cost←CurrentCost;
5： if(Cost < = NewCost) or (C = NewSolution) then
6： Return;
7： else
8： if (Cost > NewCost) then
9： CurrentPartialSolution←NewSolution;
10：  CurrentCost←NewCost;
11：  CurrentValue←agentValue(CurrentPartialSolution);
12：  end if
13：  if subgraph then
    // 全局并行搜索
14：  localSearch(Threshold, NewSolution, NewThreshold,
     conflict_solution);
15：  else
    // 局部并行搜索
16：  localSearch(Threshold, NewSolution, NewThreshold,
     conflict_solution);
17：  end if
18：  if(NewPartialCost < Cost) then
19：  CurrentPartialSolution←∅;
20：  CurrentCost←∅;
21：  send Try Local Solution(New Partial, New Partial
     Cost);
22：  else
23：  send Store Solution(Current Partial Solution, Current
     Cost);
24：  end if
```

MULBS⁺ 的 sendPartialSolution (New Solution), sendTryLocalSolution(Solution, Cost), sendStoreSolution (Solution, Cost) 和 evaluate (PARTIAL _ SOLUTION, Solution) 函数与 MULBS 一致.

与 MULBS 算法相比,在原约束图不被划分的情况下,算法的执行时间与原算法相同;当原约束图被划分,则运行时间最长的子图将决定 MULBS⁺ 算法的执行时间. 假设约束图中节点的数目为 N,MULBS 和 MULBS⁺ 求得的解一致,在 MULBS⁺ 中,原约束图被划分为 n 个部分,每个节点进行局部搜索的平均时间为 t;则在此假设条件下,MULBS 的最长执行时间为 $t \cdot N$(所有节点均处在同一优先级),同等条件下,MULBS⁺ 的执行时间为 $\min(t_i, i = 1, 2, \cdots, N)$,小于 $t \cdot N/n$. 因为 $t \cdot N/n < t \cdot N$,所以 MULBS⁺ 的执行时间一定小于原算法.

3　实验评估

为了更好地估计 MULBS⁺ (MU⁺) 算法的性能,本文将其同 MULBS(MU)算法以及知名的分布式约束优化算法 ADOPT(AD)和 DPOP(DP)算法进行比较. 对于 ADOPT 和 DPOP 算法,当搜索的解不再发生变化时,算法结束;对于 MULBS⁺ 和 MULBS 算法,当解的重定义执行到叶节点并不进行回溯时,算法结束. 所有的算法都

以 FRODO(framework of distributed optimization)随机生成的图染色问题作为测试用例.

3.1 评价标准

随着越来越多的实际问题通过分布式约束优化问题的框架解决,许多具有针对性的算法,如 MULBS 被提出来. 这些算法的目的有的是为解决 agent 之间的通信问题,有的是为提高问题的求解速度. 为了更全面地衡量算法的性能,除了执行时间和非并行性检测,本文采用以下评价标准:①循环,所有 agent 发送和接收消息为一次循环;②信息容量,agent 之间进行通讯的消息容量;③信息数量,agent 之间进行通讯的消息总量;④完整度,设算法求出的解为 SC(solution cost),而问题的最优解为 BC(best cost),因为 ADOPT 和 DPOP 算法是完备算法,所以这两个算法的解被视为 BC. 在此基础上,得到一个解 SC 的完整度估计:

$$\text{completeness(SC)} = 100 - \left(\frac{\text{SC} - \text{BC}}{\frac{n \times d}{2}} \times 100\right).$$

3.2 实验结果分析

本章主要针对 FRODO[10] 生成的测试用例,以非并行性检测以外的其他测试指标作为衡量标准,进行实验结果的分析. 因为非完备算法求解的完整度是定义在完备算法求解基础之上,所以,本文只比较 MULBS$^+$ 与 MULBS 算法求解的完整度. 对于一个分布式约束优化算法,它的性能主要受到以下三个因素的影响:变量的数量,变量取值空间的大小,变量间的约束数量.

根据以上测试指标,本文的实验主要分为两组,结果见表 1 和表 2. 第一组实验,本文分别对 10 个变量,约束图密度为 0.5,变量空间大小分别为 3,5,7,10 的测试用例进行分析.

表 1 不同取值空间下的实验分析

Table 1 Experimental analysis under different domain sizes

取值空间	解的完整度 MU/MU$^+$	循环次数 AD/DP/MU/MU$^+$	执行时间/s AD/DP/MU/MU$^+$	消息容量 AD/DP/MU/MU$^+$	消息数量 AD/DP/MU/MU$^+$
3	0.995 0/0.995 1	1 450/100/98/101	400/15/15/14	100 000/10 563/8 000/8 500	500/55/49/48
5	0.992 0/0.993 0	1 560/110/99/105	408/20/16/15	113 450/22 554/90 00/9 200	500/60/50/51
7	0.996 0/0.995 0	1 680/116/100/104	410/30/18/17	105 630/29 875/9 100/9 500	500/65/51/52
10	0.995 0/0.996 0	1 970/120/101/115	409/68/20/18	110 560/39 877/9 300/10 000	500/70/54/55

表 2 不同图密度下的实验分析

Table 2 Experimental analysis under different graph densities

图密度	解的完整度 MU/MU$^+$	循环次数(取对数) AD/DP/MU/MU$^+$	执行时间/s AD/DP/MU/MU$^+$	消息容量 AD/DP/MU/MU$^+$	消息数量 AD/DP/MU/MU$^+$
0.3	1/1	267/109/101/106	375/10/11/9	5 480/3 680/4 000/4 200	490/55/49/48
0.5	1/1	1 450/100/98/98	400/15/15/14	113 450/11 000/9 000/9 500	500/60/50/53
0.7	0.975 0/0.990 0	80 000/101/94/94	750/46/67/72	900 000/40 459/35 608/45 673	573/65/51/52
0.8	0.951 0/0.951 0	500 000/1 023/92/91	900/68/240/213	3 120 000/98 700/57 895/59 240	602/72/50/49
0.9	0.948 0/0.950 0	1 100 000/102/100/102	1 200/128/367/299	10 120 000/123 809/67 830/68 972	680/79/51/65

从表 1 可以观察到,在解的完整度方面,除了变量的取值空间大小为 7 时,MULBS$^+$ 算法的完整度低于 MULBS 算法;其他情况下的完整度都优于原算法;其次,原算法的完整度在较复杂的约束图中所求解的稳定性也得不到保证. 在循环次数方面 DPOP,MULBS 和 MULBS$^+$ 算法除在变量取值空间大于 10 时,算法的执行时间和循环次数大致相同. 这是因为三个算法都采用了从上至下的搜索策略;在消息数量和容量方面,MULBS$^+$ 算法在执行过程中的消息容量和消息数量相比 MULBS 算法有所增加,这是因为冲突消息被应用在 MULBS$^+$ 算法中,但远小于 ADOPT 算法.

第二组实验,本文采用 5 变量,变量取值空间大小为 3,约束图密度分别为 0.3,0.5,0.7,0.8 和 0.9. 由表 2 可知,即使图密度增大造成 MULBS$^+$ 算法的完整度减小,算法的运行结果也优于原算法,在约束图接近完全连接图的时候,算法求得解的完整性也在 97% 以上. 当图密度小于 0.7 时,MULBS 的执行时间比 MULBS$^+$ 和 DPOP 算法都好,这是因为 MULBS$^+$ 和 DPOP 算法需要更多的交流信息来解决冲突. 由实验结果可以看

出，MULBS$^+$算法更适用于解决大规模的分布式约束优化问题. 这个结果与第一组的实验结果相似，而 ADOPT 算法则显示出了最差的性能.

根据实验结果，ADOPT 算法的性能除在完整度方面，其他性能都是最差的. 这个结果在意料之中，因为分布式约束优化算法的性能受到多方面因素影响. DPOP 算法与 MULBS 和 MULBS$^+$算法在循环次数、执行时间和消息数量方面的区别不是很大，这个结果与文献[9]中的结果相符. 而 DPOP 算法和其他两个算法最大的区别在于其消息容量呈超立方增长.

MULBS 所求解的完整性随着图密度的增加开始下降，但在图密度恒定时，MULBS 和 MULBS$^+$算法求解的完整度差别很小. 在两组实验中，MULBS$^+$算法消息数量要比原算法多，这是因为 CONFLICT_SOLUTION 消息有助于算法进行全局并行搜索.

4　结　　语

本文在 MULBS 算法的基础上，提出了其改进算法 MULBS$^+$. 首先，本文提出了一种最小冲突选择机制代替原算法的回溯策略；其次，为了能够提升算法的并行搜索速度，MULBS$^+$算法根据约束图的密度，采用基于动态子图划分的全局并行搜索策略加快解的重定义.

目前，分布式约束优化算法的研究主要将更多实际问题建模成为 MAS，并用分布式约束优化问题的框架进行求解，其次是改善原有算法，提升算法性能，并根据实际问题的特点提出具有针对性的分布式约束优化算法.

参考文献：

[1] Gutierrez P, Meseguer P, Yeoh W. Generalizing ADOPT and BnB-ADOPT[C]// Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11). Barcelona, 2011: 554 – 559.

[2] Marconi M, Helmut P. Multi-user eco-driving training environment based on distributed constraint optimization [C]//Proceedings of AAMAS. St. Paul, 2013: 925 – 932.

[3] Lesser V, Corkill D. Challenges for multi-agent coordination theory based on empirical observations[C]//Proceedings of AAMAS. Paris, 2014: 1157 – 1160.

[4] Thomas L. Distributed constraint optimization: privacy guarantees and stochastic uncertainty[D]. Lausanne: Swiss Federal Institute of Technology in Lausanne, 2011.

[5] Tambe M. Towards flexible teamwork [J]. *Journal of Artificial Intelligence Research*, 1997, 7(1): 83 – 124.

[6] Scerri P, Johnson L, Pynadath D, et al. A prototype infrastructure for distributed robot, agent, person teams[C]// Proceedings of AAMAS. Melbourne, 2003: 433 – 440.

[7] Yokoo M, Hirayama K. Distributed breakout algorithm for solving distributed constraint satisfaction problems [C]// Proceedings of the Second International Conference on Multiagent Systems. Kyoto, 1996: 401 – 408.

[8] Fitzpatrick S, Meertens L. An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs[C]// Proceedings of the International Symposium on Stochastic Algorithms: Foundations and Applications. Berlin, 2001: 49 – 64.

[9] Enembreck F, Barthès Jean-Paul 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.

[10] Petcu A. FRODO: a framework for open and distributed constraint optimization. technical report. No. 2006/001[R]. Lausanne: Swiss Federal Institute of Technology, 2006.