doi:10.3969/j.issn.1000-436x.2013.06.012

结合 look-ahead 值排序的自适应分支求解算法

王海燕[1,2,3]，欧阳丹彤[1,2]，张永刚[1,2]，张良[1,2]

（1. 吉林大学 计算机科学与技术学院，吉林 长春 130012；2. 吉林大学 符号计算与知识工程教育部重点实验室，吉林 长春 130012；
3. 吉林师范大学 计算机学院，吉林 四平 136000）

摘 要：基于新近提出的自适应分支约束求解框架，结合 look-ahead 值启发式，提出一种新的约束求解算法 AdaptBranch^{LVO}。为验证算法效率，在标准测试库上进行了充分对比实验。结果表明，新提出算法在效率上明显优于已有的自适应分支求解算法。

关键词：约束满足问题；约束求解；自适应分支；look-ahead 值启发式

中图分类号：TP311.5 **文献标识码**：A **文章编号**：1000-436X(2013)06-0102-06

Novel adaptive branching constraint solving algorithm with look-ahead strategy

WANG Hai-yan[1,2,3], OUYANG Dan-tong[1,2], ZHANG Yong-gang[1,2], ZHANG Liang[1,2]

(1. College of Computer Science and Technology, Jilin University, Changchun 130012,China;
2. Key Laboratory of Symbolic Computation and Knowledge Engineering for Ministry of Education, Jilin University, Changchun 130012, China;
3. College of Computer, Jilin Normal University, Siping 136000, China)

Abstract: Based on the state-of-the-art scheme of adaptive branching constraint solving, a novel algorithm named AdaptBranch^{LVO} was proposed, combined with the look-ahead value ordering heuristics. To demonstrate the efficiency of AdaptBranch^{LVO}, sufficient experiments on the wide range of the problem instances in Benchmark were carried out, and the experiment results show that AdaptBranch^{LVO} outperforms the existing adaptive branching constraint algorithm by a large margin.

Key words: constraint satisfaction problems; constraint solving; adaptive branching; look-ahead value ordering heuristics

1 引言

约束满足问题(CSP, constraint satisfaction problems)是人工智能领域研究的热点问题。而 CSP 的经典求解方法是将回溯搜索与约束传播相结合。搜索过程是基于某种分支策略，并由变量和值排序启发式引导。当前，在 CSP 的回溯搜索中，搜索的分支选择有多种策略，其中 2-way 和 d-way[1]是 2 种最标准的分支策略。在 2-way 分支策略下，在选择了变量 x 和 x 域中的某个值 a_i 后，建立 2 个分支，一个分支是将 $x=a_i$ 加入到问题中进行传播，另一个分支是将 $x<>a_i$ 加入到问题中进行传播。在 2 个分支都失败时，算法回溯。在传播 $x<>a_i$ 时，可以选择不同于 x 的变量 y 的某个值继续传播，也可以选择变量 x 的不同于 a_i 的其他值进行传播。如果将问题限定为后一种情况，则称其为受限的 2-way 分支[2]。而在 d-way 分支策略下，变量 x 被选择之后，建立 d（d>2）个分支，每个分支对应 x 的一个值分配。在第一个分支中，x 分配值 a_1，并引发约束传播，如果这个分支失败了，则从 x 论域中移去 a_1，然后给 x 分配 a_2，以此类推，即，在 $x=a_i$ 失败后，算法必须选择 x 的下一个可用值。如果 d 个分支都失败，

收稿日期：2012-05-18；**修回日期**：2012-08-15

基金项目：国家自然科学基金资助项目(61170314,61133011,60973089,61003101,61170092,60973088,41172294);吉林省科技发展计划基金资助项目(20101501,20100185,201101039)；国家教育部博士点专项基金资助项目(20100061110031)

Foundation Items: The National Natural Science Foundation of China (61170314, 61133011, 60973089, 61003101, 61170092, 60973088, 41172294); Jilin Province Science and Technology Development Plan (20101501, 20100185, 201101039); Doctoral Fund of Ministry of Education of China (20100061110031)

算法回溯。受限的 2-way 在某种程度上近似于 d-way 分支。

Kostas 通过实验证实[2]，不同的变量排序启发式对 2-way 分支策略和 d-way 分支策略的影响是不同的。当从简单的变量排序启发式 dom[3]一直测试到更经典的启发式如 dom/ddeg[4]时，2-way 分支策略要优于 d-way 分支策略。但采用当前公认最好的 dom/wdeg[4]变量排序启发式时，d-way 分支却比 2-way 分支更有效。而且实验结果表明，受限的 2-way 分支策略与 d-way 分支策略效果很接近。基于不同分支策略的不同表现，Kostas 提出 2 个启发式，在搜索中的某些点应用它们，根据启发式的决定在完全的 2-way 分支和受限的 2-way 之间进行选择。从根本上实现自适应分支选择。

本文首先在 dom/wdeg 变量启发式下，对受限的 2-way 分支和完全的 2-way 分支策略进行了详细的对比实验。结果表明，在不同实例上，受限的 2-way 和完全 2-way 表现出不同的性能。然后，将 4 种自适应分支方法与上面 2 种分支方法在标准测试库 Benchmark 中的 composed、bqwh、domino、graph 进行了求解效率的对比，实验结果充分展示了自适应分支策略的优势。同时，由于相关研究中没有充分考虑值启发式对自适应分支策略的影响，而 look-ahead[5]值启发式能够有效引导搜索到更可能成功的分支。为进一步提高算法效率，将自适应分支策略与 look-ahead 两者结合起来，提出一种新算法 AdaptBranch^{LVO}，该算法在自适应分支的基础上加入 look-ahead 值启发式，在有效避免误用不合适分支策略的基础上，考虑到每个值对未来情况的影响，选择最有可能成为解一部分的值优先实例化，以更快求出最终解。在 sat 和 unsat 两大类问题的多个标准类实例上进行实验，结果表明，新提出算法在效率上明显优于已有的自适应分支方法，即加入 look-ahead 值启发式的自适应分支求解效率有更大幅度的提高。

2　背景知识

一个 CSP 表示为一个三元组（X,D,C），其中，$X=\{x_1,\cdots,x_n\}$，是一组 n 个变量的集合；$D=\{D(x_1),\cdots,D(x_n)\}$是对应于每个变量的一组论域；$C=\{c_1,\cdots,c_e\}$是 e 个约束的集合。每个约束 c 表示为有序对（var(c),rel(c)），其中，var(c)=$\{x_1,\cdots,x_k\}$是 X 的一个有序子集，而 rel(c)是 $D(x_1)\times\cdots\times D(x_k)$的子集[7]。验证一个元组是否满足约束 c 的过程称为一次约束检查。

回溯搜索与约束传播的结合方法是求解 CSP 的经典方法。搜索过程是基于某种分支策略，并由变量和值排序启发式引导。

在经典的变量排序启发式（VOH, variable ordering heuristic）中，dom/wdeg 以其普适性及高效性受到研究者的广泛重视。它是基于 fail-first 原则，为每个约束分配一个权值，初始设置为 1。每次约束引发一个冲突（例如一次 DWO），它的权值就加 1。每个变量都与一个加权度相关，它是包含此变量和至少一个未实例化变量的所有约束的权的总和。Dom 是现有论域的大小，dom/wdeg 启发式选择现有域大小与带权的度比值最小的变量。在后文中，都默认使用 dom/wdeg 变量排序启发式。

值实例化的顺序对约束求解效率有着深刻的影响。而搜索中向前看（look-ahead）是提高求解效率的关键技术，它能在搜索中尽早导致失败节点产生，从而为变量排序提供有价值信息。当前，look-ahead 技术已成功用于值排序启发式，特别是针对难解 CSP。典型 look-ahead 值排序(LVO, look-ahead value ordering)启发式[5]有：最小冲突(MC, min-conflicts)启发式，最大论域(MD, max-domain-size)启发式，加权最大论域(WMD, weighted-max-domain-size)启发式和论域大小分值(PDS, point-domain-size)启发式等 4 种。其中，MC 是针对当前变量论域中每个值，考虑与当前变量相关的未实例化变量的论域中与这个值不相容值的个数，并按此个数的升序对当前变量的值排序。即总是优先选择冲突值最少的值；MD 是针对当前变量论域中每个值，考虑所有与当前变量相关的未实例化变量移去不相容值的剩余子论域，优先选择在未实例化变量中产生最大的最小剩余子论域的值；WMD 是 MD 的一个改进版本，目的是解决第 3 种启发式中，当前变量论域中可能有几个值产生相同大小的剩余子域集合而导致“结”的产生的问题。WMD 优先选择剩下更大子论域的值，这点和 MD 相似，只是打破“结”的方法不同。它是根据具有给定剩余子论域大小的未实例化变量的数目来打破“结”的；PDS 给当前变量论域中每个值打分。依据是所有与当前变量相关的未实例化变量移去不相容值的剩余子论域。每个大小为 1 的子论域给 8 分，每个大小为 2 的子论域给 4 分，每个大小为 3 的子论域给

2 分，每个大小为 4 的子论域给 1 分。优先选择具有最小总分的值。已有实验表明[5]，MC 启发式是 LVO 启发式中效果最好的，所以选择它进一步实验。后文中提到的 LVO 均指 MC。

3 分支策略性能对比

2-way 和 d-way[1]这 2 种标准的分支策略在不同实例上的表现不同。时而 2-way 分支策略胜出，时而受限的 2-way 分支策略胜出，时而两者持平。研究者考虑在搜索的不同位置选择更合适的分支策略，即自适应分支策略。实现的基本思想是根据具体问题的结构将不同分支组合到一起，借助于启发式和传播函数，达到压缩搜索空间的目的，进而提高求解效率。代表工作为 2010 年 Kostas 提出的自适应分支策略[2]，其中提到 2 种启发式策略：$H_{sdiff}(e)$和 $H_{cadv}(VOH_2)$。前者是基于变量排序启发式分值差异的自适应分支启发式，其工作原理是：假如当前变量是 x，且 VOH 建议去选择一个其他变量 y，当 $|score(y)-score(x)|>e$ 时，接受这个建议。其中，score(x)和 score(y)是 VOH 分配给变量 x 和 y 的值，而 e 是一个可变的阈值参数。后者为辅助顾问启发式中，其主要思想为：假如当前变量是 x，且算法用到的 VOH（VOH_1）建议去选择另一个变量 y，仅当辅助 VOH（VOH_2）也选择 y 时，接受这个建议，即当 $scoreVOH_2(y)>scoreVOH_2(x)$时，其中，$scoreVOH_2(x)$和 $scoreVOH_2(y)$是 VOH_2[8]分配给变量 x 和 y 的值。受文献[2]中实验结果启发，在 $H_{sdiff}(e)$中，VOH 采用 dom/wdeg，e 取值 0.1；$H_{cadv}(VOH_2)$中辅助启发式为 wdeg。2 个启发式可合取或析取应用。

本文在文献[2]基础上进行实验，选取不同的 Benchmark 测试问题类，以进一步验证自适应分支策略的优势。在实验中将 $H_{sdiff}(0.1)$和 $H_{cadv}(wdeg)$简记为 H_1 和 H_2，它们的合取和析取记为 $H_{12}^{\wedge}$和 $H_{12}^{\vee}$。CPU 运行时间实验结果如表 1 所示。表中每个实例 CPU 运行时间的最好情况都用黑体显示。从表中可以看出，自适应分支策略总是趋于选择效果好的分支方式，有些情况甚至比两者中的优胜者更好。

4 AdaptBranch^{LVO} 算法

为进一步提高约束求解效率，本文在自适应分支的基础上加入 look-ahead 值启发式。考虑到求解难解 CSP 时，多数时间花费在探查搜索空间不可能产生问题解的分支上。为减少回溯，应首先尝试那些更可能导致相容解的值。即使被选择的值是某个解的一部分可能性的微小增大都能对找到解的时间开销有着本质上的影响。本文利用在自适应约束传播 look-ahead 阶段中收集到的信息改进值排序启发式。在自适应分支框架上，加入文献[5]中的 LVO 启发式，它根据 look-ahead 得到的信息为当前论域中的值划分等级。当前变量则用最高级别的值实例化。得到的算法是 AdaptBranch^{LVO}，其维持弧相容的框架描述如图 1 所示。

由于需要区分完全的 2-way 策略和受限的 2-way 策略，而这 2 种策略的区别仅在于下一个实例化的变量改变与否。所以用布尔变量 *Restricted_or_Not*（第 2 行）作为是否需要判断限定分支的标识变量，其值为真，表示下一次需要判断是否限定分支，反之不需要。*Free_variables*（第 4 行）是未实例化的变量。*Soulution_Stack*（第 5 行）为存储解的动态堆栈。AdaptBranch^{LVO} 算法的 MAC 过程为，只要还有未实例化的变量，就根据 SELECT_VAR 函数（下面详述）选择出一个变量（第 7 行），并按 LVO 为其选择一个值（第 9 行）。自适应分支的具体实现在 SELECT_VAR 函数中，总体上实现了自适应分支和 LVO 结合的方式。

表 1 **自适应分支策略与标准分支策略的比较（单位：ms）**

问题实例	2-way	受限的 2-way	H_1	H_2	$H_{12}^{\wedge}$	$H_{12}^{\vee}$
composed-25-10-20-5	47	63	**31**	32	46	32
composed-25-10-20-7	31	47	32	32	47	**31**
bqwh-15-106-3	1188	1031	**938**	969	**938**	953
bqwh-15-106-4	188	78	78	78	**62**	93
bqwh-15-106-7	156	47	**31**	**31**	**31**	47
domino-100-100	47	63	**46**	47	47	62
graph14	1 000	3828	985	1 000	985	**984**

```
MAC (P(X,D,C))
1) begin
2) Restricted_or_Not←FALSE;
3) if (not AC_consistency(P)) then return no_solution;
4)   Free_variables←X;
5)   an empty stack Soulution_Stack for solution;
6)   while(Free_variables not empty) do
7)                 Xi←SELECT_VAR(Free_variables, Re-
stricted_or_Not);
8)     Restricted_or_Not←FALSE;
9)     select a value ai from current domain of Xi using LVO;
10)      if( AC_consistency(Xi=ai)) then
11)            Soulution_Stack.push(Xi,ai);
12)            delete Xi From Free_variables;
13)      else
14)            while(not AC_consistency(Xi≠ai)) then
15)                  if(Soulution_Stack is not empty)then
16)                        (Xi,ai)← Soulution_Stack.pop();
17)                  else
18)                        return no_solution;
19)                  Backtrack(Xi,ai);
20)                  Free_variables←Free_variables∪Xi;
21)            end
22)            Restricted_or_Not←TRUE;
23)            cur_var←Xi;
24)    end
25)    return solution;
26) end
```

图 1　MAC 框架描述

SELECT_VAR 函数在整个 MAC 的过程中尤为重要（如图 2 所示）。在选择变量的时候，主要考虑的是 *Restricted_or_Not* 的值。在其值为真的前提下，需要判定是否限定分支，判定的依据是 H_1 和 H_2 2 种自适应分支启发式的满足情况，并根据判定结果选择合适的变量作为下一个实例化对象。Switch 的 4 个分支（第 17）~25)行）对应着 4 种启发式运用的方式。如果使用 dom/wdeg 启发式筛选出变量恰为当前变量 *cur_var*，则不执行启发式 H_1、H_2。

```
SELECT_VAR(Free_variables,Restricted_or_Not)
1) begin
2)   If(not Restricted_or_Not)
3)     return dom_wdeg(Free_variables);
4)   else
5)    Xi←dom_wdeg(Free_variables);
6)    If(Xi == cur_var)
7)       return Xi;
8)   else
9)     if( score(Xi)-score(cur_var)>0.1)
10)          h1←TRUE;
11)    else
12)          h1←FALSE;
13)    if(wdeg(Xi) > wdeg(cur_var))
14)          h2←TRUE;
15)    else
16)          h2←FALSE;
17)    switch   Heuristic_user_choice：
18)     case H1：if( h1 ) return Xi;
19)                 else   return cur_var;
20)     case H2:   if(h2)   return Xi
21)                 else   return cur_var;
22)     case H1∧H2: if (h1 & h2)   return Xi;
23)                 else return cur_var;
24)     case H1∨H2: if(h1 | h2)   return Xi;
25)                 else   return cur_var;
26) end
```

图 2　选择变量过程

5　实验结果

为验证 AdaptBranch^{LVO} 算法优势，本文利用标准测试库 Benchmark 中的多类问题实例对算法进行测试。实验是在 AMD Athlon(tm) 64 X2 双核处理器 3600 的 DELL 计算机上完成的，主频为 1.90 GHz，内存为 1.00 GB，操作系统为 Microsoft Windows XP Professional。测试环境为 Microsoft Visual Studio 2008。将 AdaptBranch^{LVO} 和已有自适应分支算法进行比较，考查 CPU 运行时间、约束检查次数和搜索树生成节点数 3 项技术指标。CPU 时间（单位：ms）记为 cpu，约束检查次数记为#ccks，搜索树生成节点数记为#nodes。将 AdaptBranch^{LVO} 算法应用于搜索中，得到与原自适应分支算法的实验对比结果，如表 2 所示，最好的情况均用黑体标记。实验结果表明：新提出算法在时间开销、约束检查次数及搜索树生成节点数方面均明显优于已有自适应分支算法。

为检验新提出算法的高效性及普适性，本文在可满足问题（简记为 sat）和不可满足问题（简记为 unsat）两大类问题上进行实验，共选取 composed、bqwh、driver、frb、rlfap、geom、ehi、QCP 等 8 类问题，并从每个分类中选出 5~10 个实例进行测试，取时间测试结果的平均值作为此分类实例的实验数据。

在 sat 类中，选出部分测试结果如表 3 所示。很清楚看出，加入 LVO 的自适应分支策略在平均时间上明显优于未加入的情况，尤其在 composed-25-10-20 和 geom 两类问题上，总体效率提高了 2 倍左右。有些更是提高了 3 倍左右，如 frb30-15 在 H_1 上的改进。总之，AdaptBranch^{LVO} 综合性能明显优于已有自适应分支算法。

在 unsat 问题类上，本文给出 composed-25-1-2、ehi-85、QCP-10 和 rlfapModScens 4 类子问题的测试结果如表 4 所示。可以看出，在不可满足问题类上，加入 LVO 的自适应分支求解效率整体上也优于已有自适应分支算法。需要指出的是，对于 rlfapModScens 问题实例，加入 LVO 策略的基于 H_2 和 H_{12}^ 自适应分支求解算法在平均性能上比已有自适应分支约束算法差，考虑这和 rlfapModScens 问题实例的特殊结构有关，未来工作将深入探讨和问题结构密切相关的自适应约束求解算法。

表 2　　AdaptBranch^{LVO} 与已有自适应分支算法对比结果

问题实例	技术指标	H_1	H_2	$H_{12}^{\wedge}$	$H_{12}^{\vee}$	H_1^{LVO}	H_2^{LVO}	$H_{12}^{\wedge LVO}$	$H_{12}^{\vee LVO}$
composed-25-10-20-1	CPU	78	78	94	93	**15**	31	15	16
	#ccks	75 714	78 512	78 948	78 721	20 465	20 773	**19 924**	20 465
	#nodes	688	670	657	646	129	130	**123**	129
composed-25-10-20-3	CPU	93	125	110	78	**15**	32	16	16
	#ccks	85 757	85 212	87 722	80 270	23 259	22 723	**22 494**	23 259
	#nodes	813	765	800	727	147	142	**138**	147
bqwh-15-106-3	CPU	938	969	938	953	**47**	**47**	63	47
	#ccks	575 169	565 715	601 714	575 169	**22 895**	22 897	22 897	**22 895**
	#nodes	6 533	6 784	6 492	6 533	394	394	**388**	394
driverlogw-08c-sat_ext	CPU	20 078	19 453	20 266	20 750	18 828	**18 188**	18 828	19 109
	#ccks	570 795	570 795	**570 767**	570 795	586 193	586 193	586 165	586 193
	#nodes	3 749	3 749	**3 728**	3 749	3 901	3 901	3 881	3 901
scen10_w1_f3	CPU	203	250	266	203	**172**	**172**	203	188
	#ccks	**80 739**	86 889	87 405	**80 739**	83 032	83 715	84 458	83 032
	#nodes	142	206	209	142	**129**	131	136	**129**
scen9_w1_f3	CPU	203	250	265	234	**187**	172	203	188
	#ccks	**80 739**	86 889	87 405	**80 739**	83 032	83 715	84 458	83 032
	#nodes	142	206	209	142	**129**	131	136	**129**

表 3　　sat 类上平均求解时间对比（单位：ms）

问题实例	H_1	H_2	$H_{12}^{\wedge}$	$H_{12}^{\vee}$	H_1^{LVO}	H_2^{LVO}	$H_{12}^{\wedge LVO}$	$H_{12}^{\vee LVO}$
composed-25-10-20	57.9	70.6	78.1	79.6	26.6	32.9	25.1	26.5
bqwh15_106	315.7	311.1	279.6	315.9	287.5	270.1	297	299.8
driver	10 060.3	9 493.29	9 736.71	10 123	9 276.71	9 051.29	9 111.71	9 263.29
frb30-15	965.8	959.4	912.4	981	343.8	447	712.6	674.8
frb35-17	4 506.2	5 500	4 293.4	4 328.2	3 787.6	3 806.2	3 394	4 084.6
rlfapGraphs	1 369.5	1 046.75	1 476.625	1 250	777.5	654.375	640.625	648.375
geom	4 248.125	4 173.75	3 949.25	3 218.75	1 781.25	1 609.375	1 730.625	1 837.875

表 4　　unsat 类上平均求解时间对比（单位：ms）

问题实例	H_1	H_2	$H_{12}^{\wedge}$	$H_{12}^{\vee}$	H_1^{LVO}	H_2^{LVO}	$H_{12}^{\wedge LVO}$	$H_{12}^{\vee LVO}$
composed-25-1-2	12.5	12.5	15.7	12.5	12.3	12.5	15.9	9.4
ehi-85	5 175.2	11 603.2	5 732.6	11 417.2	4 021.7	9 714	2 393.7	3 551.4
QCP-10	35.25	43	39	34.75	35.25	31.25	35.25	35.5
rlfapModScens	10 795.11	2 449.56	4 395.8	10 767.4	10 119.7	3 531.11	16 131.7	7 043.56

6　结束语

不同分支策略在不同实例上有不同效率表现。自适应分支策略以选择最合适分支为最终目标，是自适应约束求解方法的重要研究方向。分支策略性能对比实验充分证明：引入自适应分支可以有效提高约束求解效率。本文基于新近提出的自适应分支约束求解框架，结合 look-ahead 值启发式，提出一种新的约束求解算法 AdaptBranch^{LVO}。并针对标准测试库 Benchmark 中的典型 sat 和 unsat 问题进行比较实验。实验结果表明，加入 look-ahead 值启发式的自适应分支约束求解方法能更有效地提高

约束求解的效率。未来工作考虑将学习型值启发式嵌入到自适应分支框架中，以进一步提高约束求解效率。

参考文献：

[1] BALAFOUTIS T, PAPARRIZOU A, STERGIOU K. Experimental evaluation of branching schemes for the CSP[A]. CoRR[C]. 2010.

[2] BALAFOUTIS T, STERGIOU S. Adaptive branching for constraint satisfaction problems[A]. Proceedings of ECAI[C]. Lisbon, Portugal, 2010. 855-860.

[3] HARALICK R M, ELLIOTT G L. Increasing tree search efficiency for constraint satisfaction problems[J]. Artificial Intelligence, 1980, 14(3): 263-313.

[4] BOUSSEMART F, HEREMY F, LECOUTRE C, *et al*. Boosting systematic search by weighting constraints[A]. Proceedings of ECAI[C]. Valencia, Spain, 2004. 482-486.

[5] DANIEL FROST, RINA DECHTER. look-ahead value ordering for constraint satisfaction problems[A]. Proceedings of IJCAI [C]. Montréal, Québec, Canada, 1995. 572-578.

[6] LIKITVIVATANAVONG C, ZHANG Y L, BOWEN J, *et al*. Arc consistency during search[A]. Proceedings of IJCAI[C]. Hyderabad, India, 2007. 137-142.

[7] STAMATATOS E, STERGIOU K. Learning how to propagate using random probing[A]. Proceedings of CPAIOR[C]. Pittsburgh, USA, 2009. 263-278.

[8] GRIMES D, WALLACE R J. Sampling strategies and variable selection in weighted degree heuristics[A]. Proceedings of CP[C]. Providence, USA, 2007. 831-838.

作者简介：

王海燕（1980-），女，吉林白城人，吉林大学博士生，吉林师范大学讲师，主要研究方向为约束求解与约束优化。

欧阳丹彤（1968-），女，吉林长春人，吉林大学教授、博士生导师，主要研究方向为基于模型诊断、自动推理、模型检测等。

张永刚[通信作者]（1975-），男，辽宁沈阳人，吉林大学副教授、硕士生导师，主要研究方向为约束求解与约束优化。E-mail:zhangyg@jlu.edu.cn。

张良（1990-），男，河北故城人，吉林大学硕士生，主要研究方向为约束求解与约束优化。