软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(9):2769−2782 [doi: 10.13328/j.cnki.jos.005989]

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

基于多核 CPU 的表约束并行传播模式研究*

陈佳楠[1,3], 李 哲[2,3], 李占山[1,2,3]

[1](吉林大学 软件学院,吉林 长春 130012)

[2](吉林大学 计算机科学与技术学院,吉林 长春 130012)

[3](符号计算与知识工程教育部重点实验室(吉林大学),吉林 长春 130012)

通讯作者: 李占山, E-mail: zslizsli@163.com

摘 要: 并行传播是并行约束程序领域中的一个研究方向,其研究内容是如何并行执行在约束上的过滤算法.根据维持表约束网络广义弧相容(generalized arc consistency,简称 GAC)的串行传播模式,提出了维持表约束网络临时广义弧相容(temporary generalized arc consistency,简称 TGAC)的并行传播模式,该模式基于多核 CPU,由并行传播算法和并行过滤算法两部分组成;之后,给出了并行传播模式的可靠性证明,而且通过对并行传播模式的最坏时间复杂度分析,可以认为并行传播模式在平均过滤时间较长的实例上要快于串行传播模式;最终的实验结果也验证了上述结论,并行传播模式在多数实例集上取得了从 1.4~3.4 不等的加速比.

关键词: 约束满足问题;并行传播;广义弧相容;表约束;简单表缩减

中图法分类号: TP18

中文引用格式: 陈佳楠,李哲,李占山.基于多核 CPU 的表约束并行传播模式研究.软件学报,2021,32(9):2769−2782. http://www.jos.org.cn/1000-9825/5989.htm

英文引用格式: Du RZ, Li MY, Tian JF, Wu WQ. Research on parallel propagation mode of table constraint based on multi-core CPU. Ruan Jian Xue Bao/Journal of Software, 2021,32(9):2769−2782 (in Chinese). http://www.jos.org.cn/1000-9825/5989.htm

Research on Parallel Propagation Mode of Table Constraint Based on Multi-core CPU

CHEN Jia-Nan[1,3], LI Zhe[2,3], LI Zhan-Shan[1,2,3]

[1](College of Software, Jilin University, Changchun 130012, China)

[2](College of Computer Science and Technology, Jilin University, Changchun 130012, China)

[3](Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China)

Abstract: Parallel propagation is a research direction in the field of parallel constraint programming, and its research content is how to implement filtering algorithms on constraints in parallel. According to the serial propagation mode which enforces generalized arc consistency (GAC) on table constraint network, this study proposes a parallel propagation mode to enforce temporary generalized arc consistency (TGAC) on table constraint network. This mode is based on multi-core CPU and consists of parallel propagation algorithm and parallel filtering algorithm. After that, the reliability of the parallel propagation mode is proved, and through the analysis of the worst case time complexity of the parallel propagation mode, it is also demonstrated that the parallel propagation mode is faster than the serial propagation mode in instances of which the average filtering time is longer. Finally, the experimental results also confirm the above conclusion, and the parallel propagation mode achieves a speed-up ratio ranging from 1.4 to 3.4 on most series.

* 基金项目: 国家自然科学基金(61802056); 吉林省自然科学基金(20180101143JC); 吉林省发展和改革委员会产业技术研究与开发项目(2019C053-9); 中国科学院太空应用重点实验室开放基金课题(LSU-KFJJ-2019-08)

Foundation item: National Natural Science Foundation of China (61802056); Natural Science Foundation of Jilin Province (20180101143JC); Industrial Technology Research and Development Project of Jilin Development and Reform Commission (2019C053-9); Open Research Fund of Key Laboratory of Space Utilization, Chinese Academy of Sciences(LSU-KFJJ-2019-08)

收稿时间: 2019-07-22; 修改时间: 2019-09-11; 采用时间: 2019-12-03

Key words: constraint satisfaction problem; parallel propagation; generalized arc consistency; table constraint; simple tabular reduction

约束程序(constraint programming,简称 CP)是一种求解组合优化问题的有效方法,在人工智能、运筹学、图论等诸多领域都有广泛的应用.并行计算是在不改变预期结果的情况下,同时对问题的多个部分进行处理的能力.随着计算机并行处理能力的发展,约束程序与并行计算的结合研究已成为被关注的领域,即并行约束程序.

针对并行约束程序的研究大致可分为以下几类:并行传播、搜索空间分割、组合算法和问题分解等[1].其中,并行传播是指并行执行在约束上的过滤算法,其研究意义在于利用并行计算提高约束传播的效率.但这种并行化操作并不简单,会出现变量论域的同步问题,具有一定的挑战性[2],因此只有少数有关并行传播的研究成果.比如:Rolf 等人[3]提出了维持约束网络并行相容性的两种传播模型,分别是使用共享中间论域的传播模型和使用局部中间论域的传播模型;李哲等人[4]给出了一种适合 GPU 运算的并行弧相容算法 $AC4^{GPU}$.

由于缺少相应的表约束并行过滤算法,Rolf 等人提出的传播模型不能直接应用在对表约束的并行传播上.根据我们掌握的资料,目前还没有针对表约束并行传播的研究成果被公布.表约束是约束程序中最常用的约束类型之一,也被称为外延约束,每个约束在包含的变量集上显式地列举出所有支持或禁止的取值组合.表约束理论上可以编码任何类型的约束,在许多应用程序领域中,当对组合问题进行建模时,常常需要使用它们.

维持表约束网络广义弧相容(generalized arc consistency,简称 GAC)的串行传播模式由串行传播算法和串行过滤算法两部分组成,串行传播算法串行执行维持表约束 GAC 的串行过滤算法.过去的十多年中,维持表约束 GAC 的串行过滤算法研究已经公布了大量的成果,其中具有代表性的系列成果是基于简单表缩减的串行过滤算法(若无特殊说明,后文的过滤算法均指基于简单表缩减的过滤算法).比如:STR(simple tabular reduction)[5]能够动态地维持表约束的有效元组;在 STR 的基础上,STR2[6]在检查元组的有效性和收集被支持的取值时仅考虑相关的变量子集;STR3[7]通过使用 dual table 等数据结构,避免了不必要的表遍历;STRbit[8]采用位向量编码 dual table,大幅度地提升了过滤效率;CT(compact table)[9]使用以位向量为基础的 sparse bit-sets,进一步加快了过滤速度;STR-N[10]能够直接作用于负表约束,其求解效率具有明显的优势;AdaptiveSTR[11]通过自适应地检测并删除无效元组,改进了 STR3,使得算法速度显著提升;STR2*[12]采用了一种动态维持元组集有效部分的全新方法,拥有比 STR2 和 STR3 更高的效率.

根据维持表约束网络 GAC 的串行传播模式,我们提出了维持表约束网络临时广义弧相容(temporary generalized arc consistency,简称 TGAC)的并行传播模式.该模式基于多核 CPU,由并行传播算法和并行过滤算法两部分组成,并行传播算法利用线程池并行执行维持表约束 TGAC 的并行过滤算法.之后,我们给出了并行传播模式的可靠性证明,即并行传播模式也维持表约束网络 GAC.接下来,我们分析了并行传播模式的最坏时间复杂度,并将其与串行传播模式的最坏时间复杂度进行对比;经过分析对比,我们认为,并行传播模式在平均过滤时间较长的实例上要快于串行传播模式.最终的实验结果也验证了上述结论,而且并行传播模式在多数实例集上取得了从 1.4~3.4 不等的加速比.

本文第 1 节介绍相关的背景知识.第 2 节回顾维持表约束网络 GAC 的串行传播模式.第 3 节给出维持表约束网络 TGAC 的并行传播模式.第 4 节展示实验结果.第 5 节做出工作总结和未来展望.

1 背景知识

1.1 约束满足问题

一个约束网络 N 由一个变量集 X 和一个约束集 C 组成,其中:X 包含 n 个变量,C 包含 e 个约束.每个变量 x 有一个初始论域 $D(x)$,$D(x)$是 x 的初始取值集合;$dom(x)$表示在回溯搜索中变量 x 的当前论域(若无特殊说明,后文的论域均指当前论域),(x,a)表示变量 x 在 $dom(x)$中的取值 a.每个约束 c 都包含一个有序的变量集合 $scp(c)$,$scp(c)$被称为 c 的作用域,是变量集 X 的子集.相应地,每个变量 x 有一个约束订阅集 $sbp(x)=\{c|x\in scp(c)\}$.约束 c 的元数 r 为$|scp(c)|$,即 c 所包含的变量个数.约束 c 在语义上由一个关系 $rel(c)$定义,$rel(c)$包含 c 所允许的元组集合.(正)表约束 c 通过列出所允许的元组来表示 $rel(c)$.约束网络 N 的解是对所有变量的一组赋值,这组赋值满足

所有的约束.约束满足问题(constraint satisfaction problem,简称 CSP)确定一个约束网络 N 是否有解.

例 1:变量 x 和 y 的初始论域分别为 $D(x)=\{3,4,5\}$和 $D(y)=\{3,4\}$,约束 $x>y$ 可用表约束 c 表示,即 $scp(c)=\{x,y\}$, $rel(c)=\{(4,3),(5,3),(5,4)\}$.

$\tau=(a_1,a_2,...,a_r)$是约束 c 中的一个元组,其第 i 个值被表示为 $\tau[x_i]$.元组 $\tau\in rel(c)$是有效的当且仅当$\forall i\in\{1,...,r\}$, $\tau[x_i]\in dom(x_i)$;否则,τ是无效的.如果元组 τ是有效的,则称 τ是约束 c 的一个支持.进一步地,如果元组 τ是约束 c 的一个支持且 $\tau[x]=a$,则称 τ是 c 中对(x,a)的一个支持.

定义 1(广义弧相容(GAC)).

- 取值(x,a)是 GAC 的当且仅当$\forall c\in sbp(x)$,c 中至少存在一个对(x,a)的支持;
- 约束 c 是 GAC 的当且仅当$\forall x\in scp(c),\forall a\in dom(x)$,$c$ 中至少存在一个对(x,a)的支持;
- 约束网络 N 是 GAC 的当且仅当$\forall c\in C$,c 是 GAC 的.

为了降低并行传播过程中不同线程同时访问同一变量论域的频率,Rolf 等人[3]提出了局部中间论域,对于变量 $x\in scp(c)$,局部中间论域 $dom^c(x)$是 $dom(x)$在约束 c 中的副本.我们称元组 $\tau\in rel(c)$是临时有效的当且仅当$\forall i\in\{1,...,r\},\tau[x_i]\in dom^c(x_i)$;否则,$\tau$是无效的.如果元组 τ是临时有效的,则称 τ是约束 c 的临时支持.进一步地,如果元组 τ是约束 c 的临时支持且 $\tau[x]=a$,则称 τ是 c 中对(x,a)的临时支持.

接下来,我们给出临时广义弧相容的定义和性质.

定义 2(临时广义弧相容(TGAC)).

- 取值(x,a)是 TGAC 的当且仅当$\forall c\in sbp(x)$,c 中至少存在一个对(x,a)的临时支持;
- 约束 c 是 TGAC 的当且仅当$\forall x\in scp(c),\forall a\in dom^c(x)$,$c$ 中至少存在一个对(x,a)的临时支持;
- 约束网络 N 是 TGAC 的当且仅当$\forall c\in C$,c 是 TGAC 的.

性质 1. 取值(x,a)是 GAC 的,且$\forall c\in sbp(x),\forall y\in scp(c),dom^c(y)=dom(y)$,那么$(x,a)$是 TGAC 的.

证明:因为取值(x,a)是 GAC 的,所以$\forall c\in sbp(x)$,c 中存在对(x,a)的支持 τ_c.又因为$\forall c\in sbp(x),\forall y\in scp(c)$, $dom^c(y)=dom(y)$,所以$\forall c\in sbp(x)$,τ_c是 c 中对(x,a)的临时支持,故(x,a)是 TGAC 的.证毕. □

性质 2. 约束 c 是 GAC 的,且$\forall x\in scp(c),dom^c(x)=dom(x)$,那么 c 是 TGAC 的.

证明:因为约束 c 是 GAC 的,所以$\forall x\in scp(c),\forall a\in dom(x)$,$c$ 中存在对(x,a)的支持 τ_c.又因为$\forall x\in scp(c),dom^c(x)=dom(x)$,所以$\forall x\in scp(c),\forall a\in dom^c(x)$,$\tau_c$是 c 中对(x,a)的临时支持,故 c 是 TGAC 的.证毕. □

性质 3. 约束网络 N 是 GAC 的,且$\forall c\in C,\forall x\in scp(c),dom^c(x)=dom(x)$,那么 N 是 TGAC 的.

证明:因为约束网络 N 是 GAC 的,所以$\forall c\in C$,c 是 GAC 的.又因为$\forall c\in C,\forall x\in scp(c),dom^c(x)=dom(x)$,由性质 2 可知$\forall c\in C$,$c$ 是 TGAC 的,故 N 是 TGAC 的.证毕. □

1.2 线程池

线程池是一种多线程处理形式,管理一组线程,以便并行处理大量任务.在需要多次使用线程的并行程序中,单独创建与销毁线程会增加运行开销,从而影响程序的整体性能.而线程池通过限制线程的数量和重用线程,能够避免创建与销毁线程的代价,还可以提高程序的稳定性.

本文使用的是工作窃取线程池,其中每个线程都有一个独立的任务队列,主线程将任务分配到各个线程的任务队列中.工作窃取是指如果某个线程完成了其队列中的所有任务,而其他线程的队列中还有未处理任务,那么空闲线程可以窃取其他线程的未处理任务,从而改善了并行程序的负载均衡.

1.3 位向量

位向量是由若干个位(bit)组成的向量,具有存储空间占用少和处理速度快的特点.在构建一个 CP 求解器时,位向量是变量论域的重要表示方式之一[13].另外,维持表约束 GAC 的串行过滤算法 STRbit[8]和 CT[9]正是通过使用位向量才得以大幅度地加快过滤速度.

为了便于说明位向量的数据结构及其操作,我们用 v 表示位向量.位向量 v 由 m 个位组成,其第 i 个位由 $v[i]$ 表示.特别地,0b 和 1b 分别表示位的二进制取值.

本文共涉及以下 6 种有关位向量的基本操作.

(1) 位赋值操作:将位向量的某个位赋值为 0b 或 1b,如 $v[i]$:=0b;

(2) 位判等操作:判断位向量的某个位是否为 0b 或 1b,如 $v[i]$=0b;

(3) 位向量赋值操作:对位向量的整体进行赋值,如 v:=0 是将位向量 v 的每个位均赋值为 0b;又如,v:=v'是将位向量 v'整体赋值给位向量 v;

(4) 位向量判等操作:判断位向量的整体取值,如 v=0 是判断位向量 v 中每个位是否均为 0b;

(5) 位向量按位或操作:将两个位向量的对应位进行或运算,如 $v|v'$;

(6) 位向量按位与操作:将两个位向量的对应位进行与运算,如 $v\&v'$.

例 2:位向量 v 和 v'均由 8 个位组成,它们的取值及经过 v:=$v\&v'$操作的结果如图 1 所示.

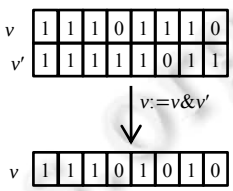

Fig.1 Bit vectors and their operations

图 1 位向量及其操作

在本文的相关算法中,我们用位向量 $bitdom(x)$和 $bitdom^c(x)$分别表示变量 x 的论域 $dom(x)$和 x 在约束 c 中的局部中间论域 $dom^c(x)$.设 x 的初始论域 $D(x)\subseteq[1,M]$,那么 $bitdom(x)$和 $bitdom^c(x)$均由 M 个位组成.$bitdom(x)[a]$为 1b 当且仅当 $a\in dom(x)$,$bitdom(x)$为 0 当且仅当 $dom(x)$为空.同理,$bitdom^c(x)[a]$为 1b 当且仅当 $a\in dom^c(x)$,$bitdom^c(x)$为 0 当且仅当 $dom^c(x)$为空.

2 串行传播模式

维持表约束网络 GAC 的串行传播模式由串行传播算法和串行过滤算法两部分组成,本节将依次介绍这两部分算法,并分析串行传播模式的相关性质.

2.1 串行传播算法

求解 CSP 实例的常用方法是回溯搜索,回溯搜索是一种完整方法,它通过系统地探索实例的搜索空间,可找到全部的解或者证明无解存在.MAC(maintaining arc consistency)算法[14]是一种在搜索过程中维持约束网络 GAC 的回溯搜索算法,目前被认为是求解 CSP 实例最有效的完整方法.在 MAC 算法中,当初始化约束网络、一个变量被赋值或者一个变量被删值时,串行传播算法会被调用执行,以维持约束网络 GAC.

常用的串行传播算法之一是面向变量的串行传播算法[15,16],面向变量是指传播集合中保存引发传播的变量.针对不同类型的过滤算法,面向变量的串行传播算法会有所不同.我们使用的串行传播算法[17]具体见算法 1 和算法 2.

算法 1. *serialPropagate*(X_{evt}:set of variables):Boolean.

```
begin
1    Q:=∅
2    for each variable x∈X_evt do
3        insert(Q,x)
4    inconsistent:=false
5    while Q≠∅ do
6        pick and delete x from Q
7        for each constraint c∈sbp(x)∧stamp(x)>stamp(c) do
```

```
8                $Y_{evt}$:=*enforceGAC*(*c*)
9                if *inconsistent* then
10                   return false
11               for each variable $y \in Y_{evt}$ do
12                   *insert*(*Q*,*y*)
13               *time*:=*time*+1
14               *stamp*(*c*):=*time*
15   return true
end
```

算法 2. *insert*(*Q*:set of variables,*x*:variable).

```
begin
1    Q:=Q∪{x}
2    time:=time+1
3    stamp(x):=time
end
```

在算法 1 中,*Q* 是一个变量集合,保存引发传播的变量,即:当一个变量的论域被删值时,该变量会被添加至 *Q* 中.串行传播算法通过使用时间戳来避免不必要的工作,时间戳是一个值,表示某个事件发生的时间.变量 *x* 的时间戳 *stamp*(*x*)表示最近一次从 *dom*(*x*)中删除值的时间,约束 *c* 的时间戳 *stamp*(*c*)表示最近一次在 *c* 上维持 GAC 的时间.时间通过计数器 *time* 来模拟.*time*,*stamp*(*x*)和 *stamp*(*c*)均被初始化为 0.

算法 1 的传入参数 X_{evt} 是引发初始传播的变量集合,其中的变量会被添加至 *Q* 中.算法 1 第 6 行从 *Q* 中选出一个变量 *x* 后,对包含 *x* 的每个约束 *c*,如果 *stamp*(*x*)大于 *stamp*(*c*),算法 1 第 8 行会执行在 *c* 上维持 GAC 的串行过滤算法.串行过滤算法会返回由被删值的变量组成的集合,并且如果有一个变量的论域被删空,串行过滤算法会将不相容标识 *inconsistent* 置为 true,之后,*inconsistent* 会在算法 1 第 9 行被识别出,这表明传播失败,当前表约束网络无解,MAC 算法发生回溯.当 *Q* 中的所有变量均被选出传播后,并且传播期间没有变量的论域被删空,这表明传播成功,当前表约束网络是 GAC 的,MAC 算法继续搜索.

2.2 串行过滤算法

如前文所述,维持表约束 GAC 的串行过滤算法研究已经公布了大量成果,它们的设计实现虽然有所不同,但主要由更新表(算法 4)和过滤论域(算法 5)两部分组成.我们总结了它们的共同之处,具体见算法 3~算法 5.

算法 3. *enforceGAC*(*c*):set of variables.

```
begin
1    updateTable(c)
2    return filterDomains(c)
end
```

算法 4. *updateTable*(*c*).

```
begin
1    for each τ∈rel(c) do
2        for each x∈scp(c) do
3            if bitdom(x)[τ[x]]=0b then
4                rel(c):=rel(c)\{τ}
5                break
end
```

算法 5. *filterDomains*(*c*):set of variables.

```
begin
1    V:=∅
2    for each x∈scp(c) do
3        for each a of which bitdom(x)[a]=1b do
4            if a doesn't have a support then
5                bitdom(x)[a]:=0b
6        if bitdom(x) has been filtered then
7            if bitdom(x)=0 then
8                inconsistent:=true
9                return ∅
10           V:=V∪{x}
11   return V
end
```

算法 3 中,串行过滤算法首先更新表,即移除无效的元组(算法 4 第 4 行);进而过滤论域,即删除在该约束上不存在支持的值(算法 5 第 5 行);最后返回由被删值的变量所组成的集合.如果有一个变量 x 的论域被删空,即 *bitdom*(x)的每个位均变为 0b,不相容标识 *inconsistent* 将被置为 true(算法 5 第 8 行).

2.3 性质分析

因为不同的串行过滤算法有不同的最坏时间复杂度,所以它们的最坏时间复杂度被统一设为 $O(T)$.另外,我们用 $r_{\max}$ 和 $d_{\max}$ 分别表示最大的约束元数和最大的初始论域大小,即 $r_{\max}=\max_{c\in C}|scp(c)|$,$d_{\max}=\max_{x\in X}|D(x)|$.

性质 4. 维持表约束网络 GAC 的串行传播模式的最坏时间复杂度为 $O(er_{\max}d_{\max}T)$[17].

证明:串行传播算法中,初始化变量集合 Q 的时间代价为 $O(n)$(算法 1 第 2 行、第 3 行).对于 $x\in scp(c)$,每当一个值(x,a)被删除后,约束 c 会被执行一次维持 GAC 的串行过滤算法,那么约束集 C 中的所有约束被执行串行过滤算法的总次数最多为 $er_{\max}d_{\max}$(算法 1 第 7 行).而执行一次串行过滤算法的时间代价为 $O(T)$(算法 1 第 8 行),将串行过滤算法返回的变量集并入 Q 的时间代价为 $O(r_{\max})$(算法 1 第 11 行、第 12 行),并且其他所有语句的时间代价均为 $O(1)$,所以串行传播模式的总体时间代价为 $O(n+er_{\max}d_{\max}(T+r_{\max}))$.又因为 $n=O(er_{\max})$,且 $T>>r_{\max}$,故维持表约束网络 GAC 的串行传播模式的最坏时间复杂度为 $O(er_{\max}d_{\max}T)$.证毕. □

3 并行传播模式

在本节中,我们提出了维持表约束网络 TGAC 的并行传播模式,它同样由并行传播算法和并行过滤算法两部分组成.之后,我们分析了并行传播模式的相关性质.

3.1 并行传播算法

在并行传播模式中,我们引入了一些新的数据结构.

- 约束位标识 *tableMask*:由 e 个位组成的位向量,用以标记需要执行并行过滤算法的表约束,在并行传播模式中是被动态更新的.其第 i 个位用 *tableMask*[c_i]表示,*tableMask*[c_i]为 1b 当且仅当约束 c_i 需要被执行并行过滤算法;
- 位订阅 *bitSbp*(x):由 e 个位组成的位向量,与变量 x 的约束订阅集 *sbp*(x)相对应,第 i 个位用 *bitSbp*(x)[c_i]表示,*bitSbp*(x)[c_i]为 1b 当且仅当 $x\in scp(c_i)$.

通过 *tableMask*:=*tableMask*|*bitSbp*(x)可将 *bitSbp*(x)提交至 *tableMask* 上.

例 3:约束网络 N 的约束集 $C=\{c_1,c_2,...,c_8\}$,变量 x 和 y 的约束订阅集分别为 $sbp(x)=\{c_4,c_8\}$和 $sbp(y)=\{c_1,c_8\}$,相应的位订阅 *bitSbp*(x)和 *bitSbp*(y)如图 2 所示.约束位标识 *tableMask* 初始化为 0(全部位均为 0b),将 *bitSbp*(x)

提交至 *tableMask* 的操作及结果如图 2 所示.

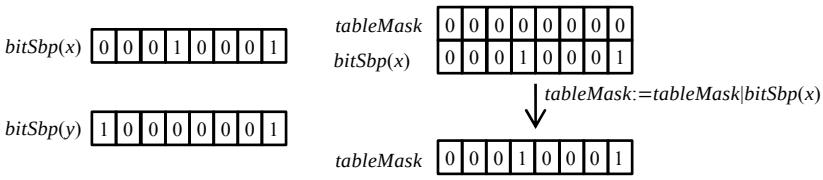

Fig.2 Bit subscriptions and their submission operations

图 2 位订阅及其提交操作

如前文所述,串行传播算法根据变量传播集合顺次执行对相关约束的过滤任务(算法 1 第 7 行、第 8 行),我们将这一操作并行化,提出了并行传播算法.并行传播算法使用约束位标识 *tableMask* 标记相关的表约束,对这些表约束的过滤任务被动态提交至线程池中并行执行,直到发生了论域删空事件或者没有新的过滤任务.具体见算法 6.

算法 6. *parallelPropagate*(X_{evt}:set of variables):Boolean.

```
begin
1    tableMask:=0
2    for each variable x∈X_evt do
3        tableMask:=tableMask|bitSbp(x)
4    inconsistent:=false
5    do
6        submitMask:=tableMask
7        tableMask:=0
8        varChanged:=false
9        for each constraint c of which submitMask[c]=1b do
10           pool.submit(enforeTGAC(c))
11       pool.wait(·)
12       if inconsistent then
13           return false
14   while varChanged
15   return true
end
```

算法 6 的传入参数 X_{evt} 同样是引发初始传播的变量集合,其中的变量被用于初始化约束位标识 *tableMask*(第 2 行、第 3 行).*submitMask* 是一个保存 *tableMask* 的局部临时标识(第 6 行),因为 *tableMask* 会被动态更新,所以算法 6 根据 *submitMask* 向线程池 *pool* 提交在约束 *c* 上维持 TGAC 的并行过滤任务(第 9 行、第 10 行),之后需要等待它们完成(第 11 行).在并行过滤任务的执行期间,当一个变量的论域被删值后,论域改变标识 *varChanged* 被置为 true,并且 *tableMask* 被更新,以在算法 6 的下次循环中向线程池 *pool* 提交新的并行过滤任务;如果有一个变量的论域被删空,不相容标识 *inconsistent* 被置为 true,之后会被识别出(第 12 行),这表明传播失败(第 13 行),当前表约束网络无解,MAC 算法发生回溯.如果所有的并行过滤任务被执行完成后,*varChanged* 仍为 flase,即没有发生删值事件,那么循环结束,这表明传播成功(第 15 行),当前表约束网络是 TGAC 的,MAC 算法继续搜索.

3.2 并行过滤算法

我们对维持表约束 GAC 的串行过滤算法进行扩展,得到了维持表约束 TGAC 的并行过滤算法(算法 7),如

PSTR2,PSTR3,PSTRbit 和 PCT,并进行了一般化总结,它们同样主要由更新表(算法 8)和过滤论域(算法 9)两部分组成.

算法 7. *enforceTGAC*(*c*).

```
begin
1    if inconsistent then
2        return
3    for each x∈scp(c) do
4        bitdom^c(x):=bitdom(x)
5    updateTTable(c)
6    filterTDomains(c)
end
```

算法 8. *updateTTable*(*c*).

```
begin
1    for each τ∈rel(c) do
2        for each x∈scp(c) do
3            if bitdom^c(x)[τ[x]]=0b then
4                rel(c):=rel(c)\{τ}
5                break
end
```

算法 9. *filterTDomains*(*c*).

```
begin
1    for each x∈scp(c) do
2        for each a of which bitdom^c(x)[a]=1b do
3            if a doesn't have a temporary support then
4                bitdom^c(x)[a]:=0b
5        if bitdom^c(x) has been filtered then
6            bitdom(x):=bitdom(x)&bitdom^c(x)        //non-blocking synchronization
7            if bitdom(x)=0 then
8                inconsistent:=true
9                return
10           varChanged:=true
11           tableMask:=tableMask|bitSbp(x)          //non-blocking synchronization
end
```

与算法 3 不同的是,算法 7 首先判断不相容标识 *inconsistent* 是否为真:如果为真,算法 7 返回,即所有的线程能够及时终止并行过滤任务;如果为假,算法 7 将每个变量 $x\in scp(c)$在 c 中的局部中间论域 $bitdom^c(x)$更新至最新的 $bitdom(x)$;然后,算法 7 更新表,即移除无效的元组(算法 8 第 4 行);最后,算法 7 过滤论域,即先从局部中间论域 $bitdom^c(x)$中删除在该约束上不存在临时支持的值(算法 9 第 4 行).若局部中间论域 $bitdom^c(x)$被删值,算法 9 便将 $bitdom(x)$与 $bitdom^c(x)$的按位与结果赋值给 $bitdom(x)$,即从 $bitdom(x)$中删除不存在于 $bitdom^c(x)$中的值(算法 9 第 6 行).进一步地,如果 $bitdom(x)$被删空,即 $bitdom(x)$的每个位均为 0b,全局不相容标识 *inconsistent* 被置为 true(算法 9 第 8 行),并行过滤算法结束;否则,论域改变标识 *varChanged* 被置为 true(算法 9 第 10 行),并且变量 x 的位订阅 $bitSbp(x)$被提交至约束位标识 *tableMask* 上(算法 9 第 11 行),即与 x 有关的约束 c 需要参与下一次并行传播.

多个线程在并行执行对不同表约束的并行过滤算法时,可能会同时过滤同一个变量 x 的论域 $bitdom(x)$(算法 9 第 6 行),也可能会同时更新约束位标识 $tableMask$(算法 9 第 11 行),所以我们对这两种操作采用了非阻塞同步方式,通过比较并替换(compare and swap,简称 CAS)实现,以确保并行过滤算法的正确性.

例 4:表约束 $c_1,c_2\in sbp(x)$被两个线程并行执行维持 TGAC 的并行过滤算法,两个线程同时过滤变量 x 的论域 $bitdom(x)$,过程及结果如图 3 所示.

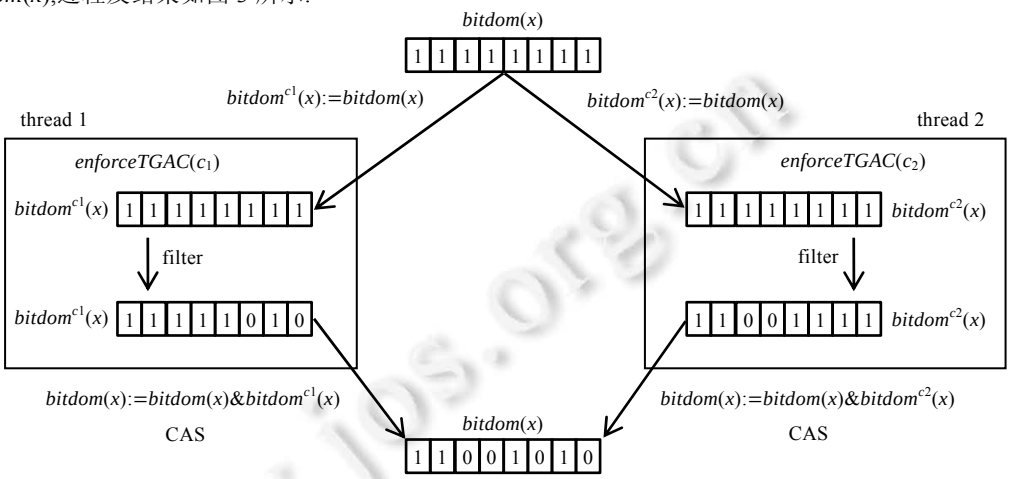

Fig.3 Parallel filtering algorithm execution instance

图 3 并行过滤算法执行实例

3.3 性质分析

首先,我们给出并行传播模式的可靠性证明.

性质 5. 维持表约束网络 TGAC 的并行传播模式也维持表约束网络 GAC.

证明:一方面,如果取值(x,a)不是 GAC 的,即$\exists c_0\in sbp(x)$,c_0 中没有对(x,a)的支持,那么当 c_0 被执行维持 TGAC 的并行过滤算法时,$scp(c_0)$内各变量的局部中间论域均被更新(算法 7 第 3 行、第 4 行),因此 c_0 中也没有对(x,a)的临时支持,(x,a)不是 TGAC 的,会被删除(算法 9 第 4 行、第 6 行).

另一方面,如果取值(x,a)是 GAC 的,那么$\forall c\in sbp(x)$,当 $scp(c)$内各变量的局部中间论域均被更新后(算法 7 第 3 行、第 4 行),有$\forall y\in scp(c)$,$dom^c(y)=dom(y)$.由性质 1 可知,(x,a)是 TGAC 的,会被保留.

综上所述,维持表约束网络 TGAC 的并行传播模式也维持表约束网络 GAC.证毕. □

接下来,我们对并行传播模式的最坏时间复杂度进行分析.通过对比串行过滤算法,因为并行过滤算法中所增加的操作语句均有着 $O(r_{\max})$(算法 7 第 3 行、第 4 行)或 $O(1)$的时间代价,并且 $T>>r_{\max}$,所以并行过滤算法的最坏时间复杂度也为 $O(T)$.线程池的并行度用 p 表示,即线程池内存在 p 个线程.另外,每个并行过滤任务被提交至线程池之后,在线程池中还会有被调度的时间代价,我们将其设为 $O(S_p)$,S_p 随着并行度 p 的增大而增大.

性质 6. 维持表约束网络 TGAC 的并行传播模式的最坏时间复杂度为 $O(er_{\max}d_{\max}(S_p+T)/p)$.

证明:并行传播算法中初始化约束位标识 $tableMask$ 的时间代价为 $O(n)$(算法 6 第 2 行、第 3 行)且 $n=O(er_{\max})$.对于 $x\in scp(c)$,每当一个值(x,a)被删除后,约束 c 会被执行一次维持 TGAC 的并行过滤算法,那么约束集 C 中的所有约束被执行并行过滤算法的总次数最多为 $er_{\max}d_{\max}$(算法 6 第 9 行).而每个并行过滤任务在线程池中被调度的时间代价为 $O(S_p)$,执行一次并行过滤算法的时间代价为 $O(T)$(算法 6 第 10 行),并且其他所有语句的时间代价均为 $O(1)$,所以并行过滤任务的总体时间代价为 $O(er_{\max}d_{\max}(S_p+T))$.又因为并行过滤任务可被 p 个线程并行执行,所以维持表约束网络 TGAC 的并行传播模式的最坏时间复杂度为 $O(er_{\max}d_{\max}(S_p+T)/p)$.证毕. □

当并行度 $p>1$ 时,通过对比串行传播模式的最坏时间复杂度 $O(er_{\max}d_{\max}T)$和并行传播模式的最坏时间复杂度 $O(er_{\max}d_{\max}(S_p+T)/p)$,我们认为:若 $T>>S_p$,那么 $O(er_{\max}d_{\max}(S_p+T)/p)\approx O(er_{\max}d_{\max}T/p)$,即并行传播模式在平均

过滤时间较长的实例上相对串行传播模式能够取得较为明显的加速;若 $T<<S_p$,那么 $O(er_{\max}d_{\max}(S_p+T)/p)\approx O(er_{\max}d_{\max}S_p/p)$,即并行传播模式在平均过滤时间较短的实例上会慢于串行传播模式.

4 实验结果

我们在 22 组 CSP 实例集上使用 MAC 算法分别测试了串行传播模式和并行传播模式.在串行传播模式下,串行过滤算法分别使用 CT 和 STRbit;在并行传播模式下,并行过滤算法分别使用 PCT 和 PSTRbit,线程池的并行度 p 依次被置为 2,4 和 6.MAC 算法采用 *dom*/*ddeg* 变量启发式和 min 值启发式,*dom*/*ddeg* 是稳定的变量启发式,能让两种传播模式在求解同一实例时拥有相同的变量选取顺序,进而实现公平的效率对比.每个实例的搜索时间上限为 900 秒.实验环境为 Intel Core i7 3.40GHz(4 cores) 8GB RAM Windows10 64bit OS,程序编写语言是 Java11.0.3 和 Scala2.12.8.所有的实例集均来自 http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html,我们的测试代码可在 https://github.com/amtfjlsj/lotus 上获取.

表 1 和表 2 给出了不同实例集的平均实验结果,实例集中,未超时的实例个数用#表示.time 表示求解实例所需的平均传播时间(单位 s),每个实例集对应的最短平均传播时间被加粗;filt 表示在求解实例的过程中过滤算法被调用的平均次数(单位 10 万);avg 表示执行一次串行过滤算法的平均时间(单位μs),即实例集的平均过滤时间,我们使用串行传播模式下 time 与 filt 的比值作为 avg 的近似值;speed 是并行传播模式相对于串行传播模式的平均传播时间加速比,即串行平均传播时间与并行平均传播时间的比值.

首先,我们比较一下串行传播模式和并行传播模式的实验结果.在 17 组实例集上,并行传播模式要快于串行传播模式,而且在多数实例集上取得了从 1.4~3.4 不等的加速比;在另外 5 组实例集上,并行传播模式的表现不如串行传播模式.从各实例集在串行传播模式下的平均过滤时间 avg 可以看出:取得较好加速比的实例集,其 avg 普遍较大,如 rand-10-60,ogd,uk 和 MDD0.9 等;并行传播效果不好的实例集,其 avg 普遍较小,如 aim-50,aim-100 和 dubois 等.这与第 3.3 节性质 6 的分析相一致.因此,我们提出的并行传播模式适用于平均过滤时间 avg 较大的实例.

Table 1 Average results to solve instances from different series (CT and PCT)

表 1 不同实例集的平均实验结果(CT 和 PCT)

Series	#	Serial (CT)			Parallel (PCT)								
					p=2			p=4			p=6		
		Time	Filt	Avg.	Time	Filt	Speed	Time	Filt	Speed	Time	Filt	Speed
rand-3-20-fcd	50	14.06	121.86	1.15	10.39	171.78	1.35	**9.65**	176.26	1.46	12.03	176.77	1.17
rand-3-20	50	29.58	244.96	1.21	21.50	344.79	1.38	**19.51**	353.97	1.52	24.09	355.33	1.23
rand-5-12	50	0.85	2.41	3.52	0.43	3.33	1.98	0.27	3.38	3.15	**0.25**	3.41	3.40
rand-8-20	20	4.67	30.34	1.54	**4.11**	42.37	1.14	4.50	43.38	1.04	6.03	43.04	0.77
rand-10-60	32	9.10	4.89	18.60	5.40	7.10	1.69	**3.70**	7.59	2.46	3.84	8.03	2.37
half	19	95.64	605.16	1.58	65.93	833.86	1.45	**54.85**	845.03	1.74	67.60	849.79	1.41
MDD0.7	8	31.22	139.41	2.24	20.27	194.18	1.54	**15.55**	197.26	2.01	17.75	198.91	1.76
MDD0.9	9	1.47	7.91	1.86	1.01	11.08	1.46	**0.80**	11.26	1.84	0.94	11.34	1.56
bddSmall	35	2.66	10.29	2.58	1.55	15.20	1.72	0.95	15.26	2.80	**0.90**	15.31	2.96
uk	42	10.35	8.56	12.09	5.53	11.35	1.87	**4.01**	11.56	2.58	4.07	11.77	2.54
ogd	43	5.80	2.25	25.83	3.28	3.07	1.77	2.18	3.13	2.66	**2.17**	3.20	2.67
lex	62	1.56	4.03	3.87	1.03	5.63	1.51	**1.01**	5.81	1.54	1.32	5.96	1.18
words	65	4.44	10.36	4.29	2.93	14.48	1.52	**2.88**	14.91	1.54	3.56	15.38	1.25
tsp-20	15	2.49	35.92	0.69	2.17	46.19	1.15	**2.07**	46.53	1.20	2.63	46.45	0.95
tsp-25	15	44.08	649.17	0.68	37.34	819.74	1.18	**33.58**	824.59	1.31	41.29	823.70	1.07
jnhSat	16	0.53	8.70	0.61	0.35	10.92	1.51	**0.28**	10.88	1.89	0.31	10.92	1.71
jnhUnsat	34	0.21	3.39	0.62	0.14	4.27	1.50	**0.11**	4.24	1.91	0.12	4.26	1.75
modR	25	**6.48**	75.80	0.85	8.18	102.57	0.79	12.14	104.88	0.53	15.95	104.03	0.41
aim-50	24	**0.03**	0.73	0.41	0.08	0.95	0.38	0.14	0.95	0.21	0.17	0.94	0.18
aim-100	16	**0.15**	4.44	0.34	0.34	5.83	0.44	0.61	5.81	0.25	0.83	5.81	0.18
aim-200	8	**15.51**	355.72	0.44	16.22	465.04	0.96	19.58	465.14	0.79	27.32	465.69	0.57
dubois	3	**22.49**	591.96	0.38	112.77	1 007.99	0.20	275.14	1 008.04	0.08	310.58	1 010.59	0.07

Table 2 Average results to solve instances from different series (STRbit and PSTRbit)

表 2 不同实例集的平均实验结果(STRbit 和 PSTRbit)

Series	#	Serial (STRbit)			Parallel (PSTRbit)								
					p=2			p=4			p=6		
		Time	Filt	Avg.	Time	Filt	Speed	Time	Filt	Speed	Time	Filt	Speed
rand-3-20-fcd	50	17.57	121.86	1.44	13.79	171.90	1.27	**11.66**	176.44	1.51	14.23	177.60	1.23
rand-3-20	50	36.96	244.96	1.51	26.90	345.07	1.37	**22.74**	354.55	1.63	27.70	356.85	1.33
rand-5-12	50	1.98	2.41	8.21	1.18	3.33	1.68	0.86	3.38	2.30	**0.80**	3.41	2.47
rand-8-20	20	19.30	30.34	6.36	15.08	42.67	1.28	**12.83**	44.31	1.50	14.62	44.85	1.32
rand-10-60	32	58.77	4.89	120.14	32.31	7.24	1.82	28.89	8.01	2.03	**28.38**	8.65	2.07
half	19	240.38	605.16	3.97	181.89	836.02	1.32	**123.37**	852.54	1.95	133.33	864.64	1.80
MDD0.7	8	94.14	139.41	6.75	57.99	194.72	1.62	39.71	199.09	2.37	**38.73**	202.30	2.43
MDD0.9	9	8.14	7.91	10.29	4.79	11.11	1.70	3.05	11.35	2.67	**2.73**	11.53	2.98
bddSmall	35	8.69	10.29	8.44	6.15	15.21	1.41	4.18	15.28	2.08	**3.83**	15.34	2.27
uk	42	15.21	8.56	17.77	10.09	11.34	1.51	7.24	11.59	2.10	**7.18**	11.85	2.12
ogd	43	6.79	2.24	30.25	4.80	3.08	1.41	3.54	3.17	1.92	**3.22**	3.27	2.11
lex	62	1.69	4.03	4.20	1.35	5.62	1.25	**1.21**	5.76	1.40	1.49	5.92	1.13
words	65	5.06	10.36	4.88	3.85	14.44	1.31	**3.31**	14.80	1.53	4.16	15.30	1.22
tsp-20	15	3.26	35.92	0.91	2.74	46.24	1.19	**2.52**	46.58	1.29	3.21	46.57	1.02
tsp-25	15	52.22	649.16	0.80	47.99	820.89	1.09	**40.42**	825.40	1.29	47.91	824.96	1.09
jnhSat	16	0.50	8.69	0.58	0.42	10.92	1.19	**0.31**	10.88	1.61	0.36	10.91	1.39
jnhUnsat	34	0.23	3.38	0.68	0.21	4.28	1.10	**0.15**	4.26	1.53	**0.15**	4.27	1.53
modR	25	**8.06**	75.80	1.06	9.81	101.58	0.82	13.56	104.95	0.59	17.48	104.58	0.46
aim-50	24	**0.02**	0.73	0.27	0.08	0.95	0.25	0.14	0.95	0.14	0.18	0.95	0.11
aim-100	16	**0.15**	4.44	0.34	0.39	5.83	0.38	0.69	5.81	0.22	0.92	5.82	0.16
aim-200	8	**12.57**	355.71	0.35	18.66	465.44	0.67	20.77	465.68	0.61	28.77	465.94	0.44
dubois	3	**20.08**	591.96	0.34	138.03	1 008.12	0.15	286.92	1 008.00	0.07	330.55	1 008.59	0.06

并行传播模式在多数实例集上取得了较好的加速比,但是没有实现线性加速(加速比与并行度相等),原因主要有两点.

- 在求解实例的过程中,并行过滤算法被调用的平均次数 filt 多于串行过滤算法被调用的平均次数 filt.

虽然并行传播模式有多个线程并行执行过滤算法,但其工作总量并不与串行传播模式的工作总量相等,而是随着线程的增多而增加.这是因为在并行传播模式下,线程之间不能相互通信.对表约束 c 执行并行过滤算法的线程只在初始时更新变量 $x\in scp(c)$在 c 中的局部中间论域 $bitdom^c(x)$,到算法结束前不再获取最新的 $bitdom(x)$.那么在此过程中,如果其他线程删除了 $bitdom(x)$中的值,c 需要被重新执行一次并行过滤算法以获取最新的 $bitdom(x)$.这种现象在过滤算法被顺次执行的串行传播模式中是不存在的.

例 5:现有两个表约束 c_1 和 c_2,其中,$scp(c_1)=\{x,y\}$,$scp(c_2)=\{x,z,u\}$.假设取值(y,a)和(z,a)被其他表约束删除,这导致 c_1 和 c_2 被执行过滤算法.在串行传播模式下,首先线程 t 对 c_1 执行串行过滤算法,t 从 $bitdom(x)$中删除了(x,a);之后,在 t 对 c_2 执行串行过滤算法时,(x,a)的移除导致 t 从 $bitdom(u)$中删除了(u,a).在并行传播模式下,c_1 和 c_2 同时被两个线程 t_1 和 t_2 分别执行并行过滤算法:首先,$bitdom^{c1}(x)$和 $bitdom^{c2}(x)$被更新至最新的 $bitdom(x)$;之后,t_1 从 $bitdom(x)$中删除了(x,a),但(x,a)仍存在于 $bitdom^{c2}(x)$中,因此 t_2 没有从 $bitdom(u)$中删除(u,a);因为(x,a)的删除,c_1和c_2需要再次被 t_1和t_2分别执行并行过滤算法,其中,t_2将 $bitdom^{c2}(x)$更新至最新的 $bitdom(x)$,进而从 $bitdom(u)$中删除了(u,a).过程如图 4 所示,同样的情况下,串行过滤算法被执行了 2 次,而并行过滤算法被执行了 4 次.

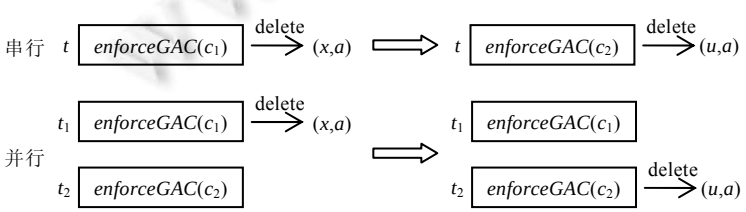

Fig.4 An example of serial propagation mode versus parallel propagation mode

图 4 串行传播模式与并行传播模式的对比实例

- 并行传播进程中线程利用率并不总是 100%.

线程利用率是活跃线程数与并行度的比率,活跃线程是指正在运行任务的线程.我们在求解一个实例的并行传播(并行度 p 为 4)进程中获取了活跃线程的数目,其随时间点的关系如图 5 所示:一方面,并行传播模式也存在串行分量,比如算法 6 除第 9 行~第 11 行之外的部分,在执行这部分串行分量时,活跃线程只有 1 个,线程利用率为 25%;另一方面,当线程池中的并行过滤任务数小于并行度时,活跃线程数将等于任务数,线程利用率小于 100%.

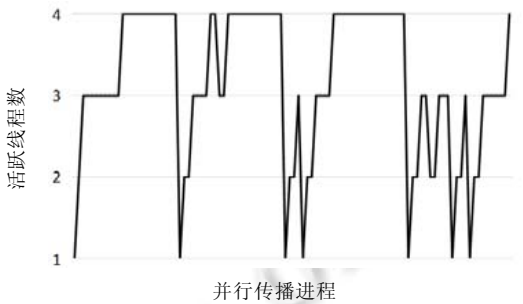

Fig.5 Number of active threads in a parallel propagation process

图 5 并行传播进程中的活跃线程数

最后,我们对比一下并行传播模式在不同并行度 p 下的平均传播时间.p 为 4 的并行传播模式在大多数实例集上均快于 p 为 2 的并行传播模式,但 p 为 6 的并行传播模式在多数实例集上却不如 p 为 4 的并行传播模式.我们认为:实验环境的 CPU 为 4 核处理器,因此当并行度高于核数时,并行传播进程中会出现多个线程抢占同一个核的情况,从而导致平均传播时间变长.

5 总 结

在本文中,首先,我们提出了维持表约束网络 TGAC 的并行传播模式,该模式基于多核 CPU,由并行传播算法和并行过滤算法两部分组成,并行传播算法利用线程池并行执行维持表约束 TGAC 的并行过滤算法;之后,我们给出了并行传播模式的可靠性证明,即并行传播模式也维持表约束网络 GAC;另外,通过对并行传播模式的最坏时间复杂度分析,我们认为并行传播模式在平均过滤时间较长的实例上要快于串行传播模式,最终的实验结果也验证了这一结论.

基于本文的思路,我们认为未来的研究可从以下 3 个方面继续开展.

1) 我们在实验分析中给出了两点关于并行传播模式没有实现线性加速的原因,因此未来的研究可尝试通过减少并行传播模式的工作总量或提高线程利用率,来进一步提升并行传播的效率;
2) 维持表约束 GAC 的串行过滤算法研究除了基于简单表缩减,还有很多基于其他设计思想的成果,比如泛型过滤算法系列(GAC2001[18],GAC3^{rm}[19],GAC4R[20]等)和压缩表示过滤算法系列(MDDc[21],MDD4R[20],AdaptiveMDDc[11]等),这些串行过滤算法均可转化成维持表约束 TGAC 的并行过滤算法;
3) 在相容性理论方面,删值能力强于 GAC 的相容性被称为高阶相容性,如最大限制成对相容性(max restricted pairwise consistency,简称 maxRPWC)和成对相容性(pairwise consistency,简称 PWC).维持表约束网络高阶相容性的串行传播模式设计受到了很多研究工作的关注[22−24],因此未来的研究可尝试设计实现维持表约束网络高阶相容性的并行传播模式.

References:

[1] Hamadi Y, Sais L. Parallel Constraint Programming. Handbook of Parallel Constraint Reasoning, Chapter 9. Springer-Verlag, 2018. 337−379.

[2] Kasif S. On the parallel complexity of discrete relaxation in constraint satisfaction networks. Artificial Intelligence, 1990,45(3): 275−286.

[3] Rolf CC, Kuchcinski K. Parallel consistency in constraint programming. In: Proc. of the 2009 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2009). 2009. 638−644.

[4] Li Z, Li ZS, Li Y. A constraint network model and parallel arc consistency algorithms based on GPU. Ji Suan Ji Yan Jiu Yu Fa Zhan/Journal of Computer Research and Development, 2017,54(3):514−528 (in Chinese with English abstract).

[5] Ullmann JR. Partition search for non-binary constraint satisfaction. Information Sciences, 2007,177(18):3639−3678.

[6] Lecoutre C. STR2: Optimized simple tabular reduction for table constraints. Constraints, 2011,16(4):341−371.

[7] Lecoutre C, Likitvivatanavong C, Yap RHC. STR3: A path-optimal filtering algorithm for table constraints. Artificial Intelligence, 2015,220:1−27.

[8] Wang R, Xia W, Yap RHC, *et al*. Optimizing simple tabular reduction with a bitwise representation. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. AAAI, 2016. 787−795.

[9] Demeulenaere J, Hartert R, Lecoutre C, *et al*. Compact-table: Efficiently filtering table constraints with reversible sparse bit-sets. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. Cham: Springer-Verlag, 2016. 207−223.

[10] Li HB, Liang YC, Li ZS. Simple tabular reduction for generalized arc consistency on negative table constraints. Ruan Jian Xue Bao/Journal of Software, 2016,27(11):2701−2711 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4874.htm [doi: 10.13328/j.cnki.jos.004874]

[11] Yang MQ, Li ZS, Li Z. Optimizing MDDc and STR3 for solving constraint satisfaction problem. Ruan Jian Xue Bao/Journal of Software, 2017,28(12):3156−3166 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5242.htm [doi: 10.13328/j.cnki.jos.005242]

[12] Yang MQ, Li ZS, Zhang JC. Time-stamp based simple tabular reduction algorithm. Ruan Jian Xue Bao/Journal of Software, 2019,30(11):3355−3363 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5559.htm [doi: 10.13328/j.cnki.jos.005559]

[13] Schulte C, Carlsson M. Finite Domain Constraint Programming Systems. Handbook of Constraint Programming, Chapter 14. Elsevier, 2006. 493−524.

[14] Sabin D, Freuder EC. Contradicting conventional wisdom in constraint satisfaction. In: Proc. of the Int'l Workshop on Principles and Practice of Constraint Programming. Berlin, Heidelberg: Springer-Verlag, 1994. 10−20.

[15] Mcgregor JJ. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. Information Sciences, 1979,19(3):229−250.

[16] Li Z, Yang M, Li Z. A new variable-oriented propagation scheme for constraint satisfaction problem. In: Proc. of the Int'l Conf. on Knowledge Science, Engineering and Management. Cham: Springer-Verlag, 2018. 59−68.

[17] Lecoutre C. Generic GAC Algorithms. Constraint Networks: Techniques and Algorithms, Chapter 4. John Wiley & Sons, 2010. 185−237.

[18] Bessière C, Régin JC, Yap RHC, *et al*. An optimal coarse-grained arc consistency algorithm. Artificial Intelligence, 2005,165(2): 165−185.

[19] Lecoutre C, Hemery F. A study of residual supports in arc consistency. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence, Vol.7. AAAI, 2007. 125−130.

[20] Perez G, Régin JC. Improving GAC-4 for table and MDD constraints. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. Cham: Springer-Verlag, 2014. 606−621.

[21] Cheng KCK, Yap RHC. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. Constraints, 2010,15(2):265−304.

[22] Paparrizou A, Stergiou K. An efficient higher-order consistency algorithm for table constraints. In: Proc. of the 26th AAAI Conf. on Artificial Intelligence. 2012.

[23] Lecoutre C, Paparrizou A, Stergiou K. Extending STR to a higher-order consistency. In: Proc. of the 27th AAAI Conf. on Artificial Intelligence. 2013.

[24] Schneider A, Choueiry BY. PW-CT: Extending compact-table to enforce pairwise consistency on table constraints. In: Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming. Cham: Springer-Verlag, 2018. 345−361.

附中文参考文献:

[4] 李哲,李占山,李颖.基于 GPU 的约束网络模型和并行弧相容算法.计算机研究与发展,2017,54(3):514−528.

[10] 李宏博,梁艳春,李占山.负表约束的简单表缩减广泛弧相容算法.软件学报,2016,27(11):2701−2711. http://www.jos.org.cn/1000-9825/4874.htm [doi: 10.13328/j.cnki.jos.004874]

[11] 杨明奇,李占山,李哲.优化求解约束满足问题的 MDDc 和 STR3 算法.软件学报,2017,28(12):3156−3166. http://www.jos.org.cn/1000-9825/5242.htm [doi: 10.13328/j.cnki.jos.005242]

[12] 杨明奇,李占山,张家晨.一种基于时间戳的简单表缩减弧相容算法.软件学报,2019,30(11):3355−3363. http://www.jos.org.cn/1000-9825/5559.htm [doi: 10.13328/j.cnki.jos.005559]

陈佳楠(1996－),男,硕士,主要研究领域为约束规划,并行计算.

李哲(1990－),男,博士,主要研究领域为约束规划,并行计算.

李占山(1966－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为机器学习,约束推理.