



基于回溯树分解的互斥约束工作流可满足性计数

翟治年¹, 王巍橡¹, 卢亚辉², 吴茗蔚¹, 郑志军¹, 余法红³

(1. 浙江科技学院信息与电子工程学院, 浙江 杭州 310023;

2. 深圳大学计算机与软件学院, 广东 深圳 518060;

3. 嘉兴学院数理与信息工程学院, 浙江 嘉兴 314001)

摘要: 工作流可满足性(WS)研究一定访问控制策略下的资源分配问题, 其计数问题有利于判断工作流对资源异常情况的顽健性。本文研究互斥约束下的 WS 计数问题, 通过多项式计数归约为约束可满足性计数问题, 将经典的回溯树分解方法用于 #WS($\neq$)求解。实验表明, 改进后的算法降低了执行时间, 相对于现有 #WS($\neq$)算法, 提出的算法对低密度约束下的工作流具有一定的综合性能优势。

关键词: 工作流; 授权; 约束; 资源分配; 可满足性

中图分类号: TP309

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2016260

Counting workflow satisfiability with exclusion constraints based on backtracking tree-decomposition

ZHAI Zhinian¹, WANG Weixiang¹, LU Yahui², WU Mingwei¹, ZHENG Zhijun¹, YU Fahong³

1. School of Information and Electronics Engineering, Zhejiang University of Science and Technology,

Hangzhou 310023, China

2. School of Computer and Software, Shenzhen University, Shenzhen 518060, China

3. College of Mathematics Physics and Information Engineering, Jiaxing University, Jiaxing 314001, China

Abstract: Workflow satisfiability (WS) concerns the issue of resource allocation under some access control policies. Counting all its solutions is advantaged to verify the robustness of a workflow to resource exceptions. The counting problem of WS with only exclusion constraints was addressed. The classic backtracking tree-decomposition method was employed to solve #WS ($\neq$) via a polynomial-time counting reduction to a counting constraint satisfiability problem. Experiments show that, the proposed optimized algorithm declined in running time, and has well synthetical performance for workflows with low-density constraints compared to the existing #WS($\neq$) algorithms.

Key words: workflow, authorization, constraint, resource allocation, satisfiability

收稿日期: 2016-07-04; 修回日期: 2016-09-30

通信作者: 余法红, fhyu520@whu.edu.cn

基金项目: 浙江省教育厅科研基金资助项目(No.Y201533771); 浙江省自然科学基金资助项目(No.LY16F020027); 国家自然科学基金资助项目(No.61302112)

Foundation Items: Zhejiang Provincial Education Department Research Foundation of China(No.Y201533771), Zhejiang Provincial Natural Science Foundation of China(No.LY16F020027), The National Natural Science Foundation of China(No.61302112)



1 引言

随着信息通信技术的发展,大量分散资源具备了联网服务能力,并亟待进行集约化的利用。而在互联网和移动经济趋势下,消费需求日益个性化,市场竞争进一步加剧,对业务过程的应变创新和快速组织提出了更高要求。由于这两种趋势的融合,出现了从 Web 服务组合到云制造等一系列以众包为特征,并需要处理复杂或结构化任务的技术模式和产业形态。

在相应的服务平台上,业务过程依靠大量第三方资源来完成,从而面临着复杂多变的资源环境。首先,资源服务过程中可以接触相关的业务数据,而不同资源的提供者之间存在各种关系和纽带,有可能串通推断业务机密和客户隐私。不仅如此,第三方资源的虚拟性、分散性和多源性使其更容易失效,给业务过程执行造成时间和经济损失。资源环境的上述特点对业务过程提出了特殊的安全性问题以及进一步的顽健性要求。

工作流是业务过程规划与组织的核心技术。它将复杂任务分解为有序的步骤,统一分配资源加以执行。每个步骤都匹配了一组具备相应服务能力的资源,形成所谓授权关系,作为资源分配的候选范围。为了保证业务安全性,通常还要求特定步骤的执行资源之间满足一定的约束关系。例如云制造应用中,某产品的若干零部件不能由同一资源或同一提供者的资源来加工,以免核心设计被组合破解。在授权和约束限制下,资源分配存在可行与否的问题,称为工作流可满足性(workflow satisfiability, WS)。WS 不仅是工作流访问控制的基础性问题^[1],而且对于云服务平台上的众包业务过程有很强的现实应用价值。为了保证在若干资源失效时,工作流可满足关系仍然成立,相关研究进一步提出了工作流 k 弹性或顽健性问题^[1,2],有助于避免业务执行过程中的时间和经济损失。

在最基本的互斥约束下^[3],WS 已经是 NP 完全问题^[1],但相关求解研究一直在进展。1999 年, Bertino^[4]在其基于逻辑程序的工作流模型中首次涉及 WS 问题。2008 年, Crampton^[5]在讨论委托对资源分配的影响时,给出了一种 WS 算法。这两个工作本质上都是朴素的穷举搜索,其时间复杂度不低于 $O(|U|^{|\mathcal{S}|})$,实测性能也很差。2010 年, Wang 等^[1]证明资源规模可约简至 $|\mathcal{S}|$ 以内,然后进行穷举搜索,给出了一种时间复杂度为 $O(|\mathcal{S}|^{|\mathcal{S}|+1}|\mathcal{C}|)$ 的 WS($\neq$)算法($\mathcal{S}, \mathcal{C}$ 分别表示步骤和约束集)。2011 年, Basin 等^[6]讨论了考虑互

斥和绑定约束的 WS 决策问题,并给出了多项式时间的近似算法^[2]。他们的模型支持步骤间的循环和选择路由,算法开销低,但是除缺少精确性以外,还依赖于授权资源在步骤之间的均匀分布。2013 年, Crampton 等通过归约为集合最大权划分问题,将 WS($\neq$)求解时间降低至 $O^*(2^{|\mathcal{S}|}(|\mathcal{C}|+|\mathcal{U}|^2))$ ($\mathcal{U}$ 表示资源集)^[7],并对多种约束下的 WS 进行了研究。这些研究判定 WS 成立与否的方式是寻找一个具体的可行资源分配,相应的问题称为 WS 决策。然而,即使一个工作流是决策可满足的,由于执行期间的资源失效,仍可能导致资源分配不再可行。由于工作流顽健性要求任意删除若干资源后仍存在资源分配可行解,因而对解的数量有内在的要求,并且数量越多,越有利于抵抗个别资源失效,统计 WS 解的个数可为工作流顽健性提供重要的参考指标^[8]。与此相应的问题称为 WS 计数,记作 #WS。由于 WS 计数结果可以反映 WS 有解与否,也可将 #WS 作为工作流可满足性判定的一种方式,进而用于工作流顽健性分析^[2]。目前关于 WS 计数的研究还很少,主要是参考文献[2]通过归约为 #SAT,给出了一种 #WS($\neq, =$)算法。尽管归约后变量数目庞大,整体时间复杂度高达 $O^*(2^{|\mathcal{U}| \cdot |\mathcal{S}|})$,该算法借助 sharpSAT 求解器的隐式子句传播、组件缓存及其压缩与清理等优化特性,仍取得了很好的实测时间性能。但是,组件缓存是 sharpSAT 提高时间性能的重要手段。该算法的空间复杂度同样达到 $O^*(2^{|\mathcal{U}| \cdot |\mathcal{S}|})$,且实测空间代价通常处于很高的水平,给算法应用造成了较大的局限。此外,参考文献[8]给出了一种基于集合划分权重和的 #WS($\neq$)算法,具有 $O^*(2^{|\mathcal{S}|}|\mathcal{U}|)$ 时间复杂度,是目前最优的,但其空间复杂度也达到同样级别,且实际空间占用稳定达到理论值,不受随机输入实例特征的影响,这就严重制约了其求解性能和规模。

本文对互斥约束下的 WS 计数问题 #WS($\neq$)进行研究。引入树分解技术来识别互斥约束图的簇集结构,以降低问题规模,进而应用并改进树分解回溯(backtracking tree-decomposition, BTD)方法,得到了一种新型的 #WS($\neq$)算法,其时间复杂度以簇集大小为指数,在低密度约束情况下,相对于现有 #WS($\neq$)算法有一定的时间性能优势,且空间性能有着突出的优势。

2 预备知识

本节介绍约束可满足性问题(constraint satisfiability problem, CSP)、约束网络树分解技术以及树分解回溯(backtracking tree-decomposition, BTD)方法,它们在 #WS($\neq$)

求解中有重要应用。

定义 1^[9] 一个 CSP 问题 P 定义为四元组 $\langle V, D, C, R \rangle$, 其中:

- (1) 变量集 V 为一有限集, 设其大小为 n ;
- (2) $D = \{d_v | v \in V\}$ 为各变量的值域的集合。任取 $v \in V, d_v$ 表示其值域, 为一个有限集;
- (3) $C \subseteq 2^V$ 为约束集, 且任取 $c \in C, |c| \geq 2$, 即每个约束是 V 的一个多元子集;
- (4) $R = \{d_c | c \in C\}$ 为各约束值域的集合。任取 $c = \{v_1, v_2, \dots, v_k\} \in C$, 其值域 $d_c \subseteq d_{v_1} \times d_{v_2} \times \dots \times d_{v_k}$ 。

CSP 关注是否存在 v 中变量的合法赋值, 使每个变量的赋值均在其值域内, 且 C 中每个约束的取值均在其值域内。为了回答约束可满足性问题, 通常只需寻找任意一个合法赋值, 但在某些应用中需要计算所有合法赋值的个数, 此时的问题记为 #CSP。

由于约束的作用, 变量之间可能具有任意复杂的耦合关系, 不利于问题规模压缩和求解效率提高。Robertson 提出的树分解技术有助于克服这一困难。

定义 2^[10] 超图 $\langle V, C \rangle$ 的树分解定义为二元组 $\langle N, T \rangle$, 其中 $T = \langle I, E \rangle$, 是顶点集为 I 、边集为 E 的树, 而 $N = \{N_i | i \in I\}$ 是 V 的一个子集族, 且有:

- (1) $\bigcup_{i \in I} N_i = V$;
- (2) 任取 $c \in C$, 存在 $i \in I$ 使得 $c \subseteq N_i$;
- (3) 任取 $i, j, k \in I$, 若在树 T 中 k 位于从 i 到 j 的路径上, 则 $N_i \cap N_j \subseteq N_k$ 。

对上述定义的树分解, 将 $\max_{i \in I} |N_i| - 1$ 称为其宽度。

树分解对 CSP 的约束网络进行了整理, 将变量集 V 分为若干个 N_i , 使得每个约束都被局限于某个 N_i 内部。由于每个变量和约束的值域都不变, 树分解前后的 CSP 是等价的。CSP 所有树分解的最小宽度称为其树宽^[11]。

树分解的一种生成途径是将约束图三角化, 并取 N_i 为三角化图中各最大团的顶点集, 以诸 N_i 的下标为顶点集, 按照其最大团交集最大的原则在顶点之间生成边, 可在多项式时间内完成^[11]。

将树分解与回溯搜索结合, 得到 BTM 方法。其主要理论依据如下。

定理 1^[9] 设 $T = (I, E)$ 是超图 $\langle V, C \rangle$ 的树分解, 可看作根树, 其中 $I = \{1, 2, \dots, |I|\}$, 并且 $I = 1, 2, \dots, |I|$ 为其先根遍历序列。设 $i, j \in I$, 且 i 是 j 的父节点, $Desc(N_i) = \bigcap_{r \in Desc(j)} N_r$ (其中 $Desc(j)$ 表示 j 及其在 T 中的所有后代节点), 则不存在 $u \in \bigcup_{k=i}^{j-1} N_k - N_i \cap N_j, v \in Desc(N_j) - N_i \cap N_j$ 和 $c \subseteq C$, 使得 $\{u, v\} \subseteq c$ 。

BTM 将按照与 $I = 1, 2, \dots, |I|$ 兼容的顺序对 V 中变量赋值 (设 $p < q$, 即在 I 中 p 先于 q , 则 $N_p - N_p \cap N_q$ 中变量先于 $N_q - N_p \cap N_q$ 中变量获得赋值), 并在此过程中检查约束满足情况。设节点 i 有 k 个儿子, 在 I 中的次序为 $j_1, j_2, \dots, j_k$, 根据定理 1 有如下递归子结构^[9]: 设 $Desc(N_i)$ 代表的问题是求其中所有未赋值变量在相关约束下的合法赋值。若给定 N_i 中变量的合法赋值, 则 $Desc(N_{j_1}), Desc(N_{j_2}), \dots, Desc(N_{j_k})$ 代表的子问题可依次求解, 前面求解的子问题不会对后面的子问题造成干扰。BTM 将对已求解子问题进行缓存, 以免在不同的搜索路径中重复求解。其时间复杂度为 $O(|V| \cdot |C| \cdot d^{w+1})$, 空间复杂度为 $O(|V| \cdot w \cdot d^w)$, 其中 $d = \max_{v \in V} |d_v|, w = \max_{\{i, j\} \in E} |N_i \cap N_j|$ ^[9]。按上述方法对 CSP 的解空间进行系统化搜索, 遍历下去可以找到所有的解, 从而完成 CSP 解的计数, 而时间和空间复杂度不变^[8]。

3 工作流可满足性($\neq$)计数

3.1 问题定义

给出 #WS($\neq$) 相关定义。以下 S 表示步骤集, U 表示资源集。

定义 3 工作流 $\langle S, \leq \rangle$ 是步骤的偏序集。 $\leq$ 为 S 上的偏序关系, 表示步骤间的执行顺序。任取 $s, s' \in S, s < s'$ 表示 s 先于 s' 执行。工作流的每次执行称为一个案例。

定义 4 访问控制 (access control, AC) 策略是一个二元组 $\langle A, X \rangle$, 其中:

(1) 授权 $A \subseteq S \times U$, 表示以 S 为步骤集的工作流在资源集 U 上的执行资格分配。若 $\langle s, u \rangle \in A$, 则资源 u 是步骤 s 的一个候选执行者, 称 u 是 s 的授权资源, s 是 u 的授权步骤;

(2) 互斥约束集 $X \subseteq S \times S$ ($\&$ 为无序笛卡尔积运算), 表示步骤间的二元职责分离关系。若 $\{s, s'\} \in X$, 则在同一案例中, s 和 s' 不能由同一资源执行。互斥禁止某些利益冲突的步骤由同一资源执行, 以防止借机欺诈。例如出纳和会计工作必须由不同的资源负责。

定义 5 从 S 到 U 的资源分配 π 是 $S \rightarrow U$ 函数, 它从 U 中为 S 中每个步骤指派唯一的执行资源。工作流的每个案例都要进行资源分配。

定义 6 一个互斥约束 (下的) WS($\neq$) 问题定义为四元组 $\langle S, U, A, X \rangle$ (若存在默认的 S 和 U , 可仅用 AC 策略 $\langle A, X \rangle$ 来表示), 要求判定是否存在从 S 到 U 的资源分配 π , 使其满足:



- (1) 基于 A : 任取 $s \in S, \langle s, \pi(s) \rangle \in A$;
- (2) 不违反 X : 任取 $\{s, s'\} \in X, \pi(s) \neq \pi(s')$ 。

如果要求统计满足上述条件的所有资源分配的个数, 则称为 $WS(\neq)$ 计数问题, 记为 $\#WS(\neq)$ 。若强调给出满足条件的一个资源分配, 则称为 $WS(\neq)$ 决策。

3.2 问题求解

$WS(\neq)$ 问题 $\langle S, U, A, X \rangle$ 可归约为 $CSP \langle V, D, C, R \rangle$, 只须令 $V=S, D=\{d_s | s \in S\}$, 其中 $d_s=U_A(s)=\{u \in U | \langle s, u \rangle \in A\}, C=X, R=\{d_{\{s_1, s_2\}} | \{s_1, s_2\} \in X\}$, 其中 $d_{\{s_1, s_2\}}=U_A(s_1) \& U_A(s_2) - I_U, I_U$ 为 U 上的恒等关系。所得 CSP 的一个解必为原 $WS(\neq)$ 的一个解。反过来, 令 $S=V, U=\bigcup_{s \in V} d_s, A=\{\langle s, u \rangle | \exists d \in D (u \in d)\}, X=C$, 可以将该 CSP 归约为一个 $WS(\neq)$, 并且恰为原来的 $WS(\neq)$ 问题, 由此可知原 $WS(\neq)$ 的一个解必为该 CSP 的一个解。所以, 归约前后的问题等价, 它们的解一一对应。进一步分析前述从 $WS(\neq)$ 到 CSP 的归约。遍历 A 一次, 将每个 $\langle s, u \rangle$ 中的 u 加入相应的 $U_A(s)$ 。即可求出所有 $U_A(s)$, 所需时间不超过 $O(|A|)=O(|S| \cdot |U|)$ 。进一步对 X 中每个约束 $\{s_1, s_2\}$, 求 $U_A(s_1) \& U_A(s_2) - I_U$, 所需总时间不超过 $O(|X| \cdot (\max_{s \in S} |U_A(s)|)^2) = O(|U|^4)$ 。因此, 这一归约可在多项式时间内完成。由于归约后的 CSP 目前只有指数时间算法, 归约过程不影响整体时间复杂度。且由于归约前后问题解集的基数相等, 此归约为一多项式计数归约^[12], 因此 $\#WS(\neq)$ 可以通过完全相同的方法归约为 $\#CSP$ 进行求解。

$\#CSP$ 改变了 CSP 的求解目标, 在能够找到一个解的情况下需要无重复和遗漏地完成解的计数, 并可能存在不触及任何具体解而给出计数结果的方法, 是一类与 CSP 紧密联系又很不相同的问题。目前关于 $\#CSP$ 精确求解的研究尚不多见, 主要有变量值域切分和 BTD 两类方法^[13], 本质上都是基于解空间搜索。前者主要通过对变量值域进行切分, 将问题转化为一组小规模 $\#SAT$ 问题。例如参考文献[14]将变量值域分为大小为 2 的若干子域, 使问题转化为一组 $\#2SAT$ 问题, 而参考文献[15]将值域大小切分到低于 5, 转化为一组赋权 $\#2SAT$ 问题。后者注意到约束图中的局部簇集关系, 利用树分解技术来识别这些簇集并通过缓存解除它们之间的耦合, 使回溯搜索局部化。本文选择 BTD 方法来求解 $\#CSP$ 进而 $\#WS(\neq)$ 问题, 主要是因为:

(1) 在工作流背景中, 约束仅用来描述关键性的业务规则, 相应的约束图通常比较稀疏, 易于形成松散的簇集结构和较小的树分解宽度。

采用 BTD 方法有助于利用原始问题的这一特征, 在

相应的条件下获得较好的理论和实际性能。不妨结合这一应用特征, 将 BTD 的性能特点与几种相关算法进行对比。解 $\#CSP$ 的变量值域切分方法只能将时间复杂度 (以穷举搜索为基准) 的底数压缩至常数比例。根据参考文献[14], 其时间复杂度在各种情形下均不低于 $\#2SAT$ 的 $O((d/2)^n)$ 倍。设 $\#2SAT$ 需要 $O(\alpha^n)$ 时间, 尽管已降低至 $1.256 \cdot 1^{[16,17]}$, 该方法的整体时间复杂度仍不低于 $O((0.628 \cdot 1d)^n)$ 。而参考文献[15]在不同情形下, 时间复杂度随着 d 的增大一致趋向于 $O((\alpha^4 \cdot d/4)^n) \approx O((0.622 \cdot 2d)^n)$, 其中 $\alpha \approx 1.256 \cdot 1$ 。两种方法的计算时间均以变量集规模为指数。而 BTD 可以实现指数部分的压缩, 具有 $O(n \cdot |C| \cdot d_{\max}^{W+1})$ 的时间复杂度^[9,18], 其中 W 为树分解宽度, $d_{\max}$ 是各变量值域大小的最大值。若 $(W+1)/n$ 足够小, 则 BTD 方法的理论时间将更具优势。后文的实验表明, 在低于 0.15 的约束密度下, 指数 $W+1$ 的值往往还不到 n 的 $1/2$ (相当于底数部分降低至原来的平方根), 足以保证 BTD 的这一优势。在解 $\#WS(\neq)$ 的方法中, 参考文献[8]具有目前最优的时间复杂度, 但其基于规整的组合计数方法, 对约束图的特征缺乏敏感 (其时间效率随约束数量增加呈略微提高趋势, 但变化不大)。相对于该算法, BTD 有可能取得更好的执行时间, 后文将通过实验进行对比测试。现有两种 $\#WS(\neq)$ 算法^[7,2] 为保证理论或实测时间性能需要付出很高的空间代价, 其空间和时间复杂度均处于同一级别。而 BTD 的空间复杂度优于时间复杂度, 由于 $1 \leq w \leq W, w$ 平均可以按 $(W+1)/2$ 来估计, 相当于空间复杂度的指数部分比时间复杂度降低了一半。在较低的约束密度下, 割集大小 w 将随树分解宽度 W 一起受到限制, 对于降低空间代价同样是重要的有利因素。后文将进一步对 BTD 的实际空间占用进行测试, 并与现有 $\#WS(\neq)$ 算法对比。

(2) $\#WS(\neq)$ 每个步骤的授权资源集可以不同, 而解 $\#CSP$ 的变量值域切分方法要求所有变量的值域相同。事实上, 参考文献[14,15]解决的 $\#CSP$ 是一类记为 $\#(d,2)$ - CSP 的特殊情形, 其中 d 表示变量值域大小, 2 表示约束元数。一般的 $\#WS(\neq)$ 很难归约为 $\#(d,2)$ - CSP 借助此类方法求解。

根据上面的分析, 将每个 $WS(\neq)$ 多项式归约为等价的 CSP 问题后, 可用 BTD 方法计算后者解的个数, 并映射为原问题解的个数。不过, 在 $\#CSP$ 的 BTD 方法中, 每次约束检查是对所有约束进行的, 且验证每个约束的方法是将相关变量的当前赋值与该约束值域中的所有值逐一比较。每次约束检查的开销为 $O(|C| \cdot (\max_{c \in C} d_c \cdot \max_{c \in C} |c|))$, 其中 $|c|$ 表示约束 c 包含的变量数。本文将根据 $\#WS(\neq)$ 的特征压缩

BTD 的约束检查范围。在搜索前计算每个步骤 s 的互斥步骤集 $S_X(s)=\{s' \in S | \{s, s'\} \in X\}$, 此过程只需多项式时间和空间, 不影响整体的时间和空间复杂度。搜索中每次对 s 赋值后, 只需检查与 $|S_X(s)|$ 中的已赋值步骤是否冲突, 从而每次约束检查的开销降低至 $O(x)$, 其中 $x=\max_{s \in S} \{|S_X(s)|\}$, 不超过约束集的大小, 而 BTD 的时间复杂度也相应降低。此外, 本文的 BTD 算法也适用于约束图非连通的情况。

以上述思路为基础, 给出下面的算法 1, 其核心部分以算法 2 另外描述。

算法 1 #WS($\neq$)

输入: S, U, A, X

输出: $\langle S, U, A, X \rangle \in \#WS(\neq)$ 的解

(1) 求互斥约束图 $G=\langle S, X \rangle$ 的所有连通片;

(2) $num \leftarrow 1$;

(3) for (G 的每个连通片 C) // 每个连通片构成一个独立子问题

(4) 求 C 的树分解 $\langle S_C, T_C \rangle$, 其中 $T_C=\langle I_C, E_C \rangle$ 是以 $p \in I$ 为根的树, $S_C=\{S_i | i \in I\}$;

(5) $num \leftarrow num \times \#WS\text{-}BTD(\emptyset, N_p, N_p)$; // 从 T_C 的根开始, 以空资源分配调用算法 2 求 C 的资源分配数

(6) end for

(7) return num ;

该算法第 5 步调用的算法 2 描述如下。

算法 2 #WS-BTD($\neq$)

输入: α, S_i, E_i , 其中 α 表示可行的部分资源分配, S_i 为分解树某节点 i 所对应的步骤集 (后文将不再严格区分 S_i 与 i), E_i 表示在 α 之下 S_i 中尚未分配执行资源的步骤。

输出: 与 α 兼容的关于 $Desc(S_i)$ 的可行资源分配数目。

(1) if ($E_i=\emptyset$) // 此时 α 已为整个 S_i 分配资源

(2) if (S_i 对应叶节点) return 1; // α 已经为 $Desc(S_i)$ 中所有步骤分配了资源, 与之兼容的资源分配唯一

(3) else

(4) $num \leftarrow 1$;

(5) $M \leftarrow Sons(i)$; // S_i 的子节点;

(6) for ($M \neq \emptyset$)

(7) 按与 Γ 兼容的顺序取下一个 $S_j \in M$; // 排序已经事先确定

(8) $M \leftarrow M - \{S_j\}$;

(9) if (存在缓存项 $(\alpha[S_i \cap S_j], nb)$) // $\alpha[S_i \cap S_j]$ 表示 α 在 $S_i \cap S_j$ 上的投影

(10) $num \leftarrow num \times nb$; // nb 表示在固定 $\alpha[S_i \cap S_j]$ 时, $Desc(S_j)$ 中剩余步骤的可行资源分配数

(11) else // $\alpha[S_i \cap S_j]$ 不存在, 说明 $Desc(S_j)$ 尚未完全分配资源, 需要进行递归调用

(12) $nb \leftarrow \#WS\text{-}BTD(\alpha, S_j, S_j - S_i)$; // $S_j - S_i = S_j - S_i \cap S_j$

(13) $num \leftarrow num \times nb$; // 为 S_i 分配资源后, 不同 $Desc(S_j)$ 的资源分配互相独立, 按乘法原理计数

(14) 缓存 $(\alpha[S_i \cap S_j], nb)$; // 缓存避免重复计算

(15) end if

(16) end for

(17) return num ; // 此时, num 表示在固定 $\alpha(S_i)$ 时, $Desc(S_i)$ 中剩余步骤的可行资源分配数

(18) end if

(19) else // $E_i=\emptyset$, 需要继续 S_i 的资源分配, 以确定每个 $\alpha[S_i \cap S_j]$, 使相应的 $Desc(S_j)$ 成为独立子问题

(20) $num \leftarrow 0$;

(21) 取 E_i 中的步骤 s ;

(22) $E_i \leftarrow E_i - \{s\}$;

(23) $d_s \leftarrow U_A(s)$; // 为 s 分配的资源必须是其授权资源。 $U_A(s)=\{u \in U | \langle s, u \rangle \in A\}$, 预先一次性计算即可

(24) for ($d_s \neq \emptyset$)

(25) 任取 $u \in d_s$;

(26) $d_s \leftarrow d_s - \{u\}$;

(27) if (存在 $s' \in S_X(s)$ 且 $\alpha(s')=u$) continue; // $S_X(s)=\{s' \in S | \{s, s'\} \in X\}$, 预先一次性计算即可

(28) $num \leftarrow num + \#WS\text{-}BTD(\alpha \cup \{s \leftarrow u\}, S_i, E_i - \{s\})$; // s 的可行资源分配之间可替代, 按加法原理计数

(29) end for

(30) return num ;

(31) end if

根据 BTD 本身的时间和空间复杂度以及本文对其约束检查方式的修改不难导出, 算法 2 的时间复杂度为 $O(|S| \cdot x \cdot d^{W+1})$, 其中 $x=\max_{s \in S} \{|S_X(s)|\}$, 即各步骤的互斥步骤集的最大基数, $d=\max_{s \in S} \{|U_A(s)|\}$, 即各步骤授权资源集的最大基数, 而 W 是树分解的宽度; 算法 2 的空间复杂度为 $O(|S| \cdot w \cdot d^w)$, 其中 $w=\max_{\{i,j\} \in E} \{|S_i \cap S_j| \}$, 即树分解中父子节点交集的最大基数。又由于树分解的时间复杂度为 $O(|S|^3)^{[1]}$, 图的连通片划分也只需多项式时间, 算法 1 的时间复杂度仍为 $O(|S| \cdot x \cdot d^{W+1})$, 树分解和求连通片均可以通过多项式空



间完成,故算法 1 的空间复杂度仍为 $O(|S| \cdot w \cdot d^w)$ 。

4 实验研究

现在对算法 1 进行实验研究,并与现有两个 #WS 算法对比。测试环境为 3.4 GHz Core i3 CPU、分配了 8 GB RAM 的 RedHat Enterprise Linux 7 64 bit 虚拟机(宿主系统为 Windows 7)、GNU C++, 并使用 GMP6.1.1 高精度算术库。由于工作流访问控制数据涉及企业业务机密,无法通过技术手段搜集真实数据,现有 WS 决策和计数研究均以一定规则进行仿真,本文也将用类似规则来准备数据。

4.1 实验 1:4 种 #WS(≠)算法的性能对比

本实验将算法 1 与原始 BTD、参考文献[8]和参考文献[2]的算法对比,测试本文算法对 BTD 的改进效果,及其在低密度约束下相对于 #WS(≠)的现有算法是否具有优势。数据集生成规则是:在 $[10,30]$ 之间随机生成 20 个 $|S|$ 值,令 $S = \{1, 2, \dots, |S|\}$; 对每个 $|S|$, 在 $[25, 75]$ 之间随机取整数值 μ , 取 $|U| = 0.01\mu|S|$, 并类似地确定 U ; 对每组 S 和 U , 首先在范围集合 $\{[1, 33], [34, 66], [67, 100], [1, 100]\}$ 之中随机取 $[m, M]$, 然后对 S 中每个步骤 s , 从 $[m, M]$ 中随机取整数值 k , 从 U 中取 $0.01k|U|$ 个不同的资源作为 $U_k(S)$, 从而确定授权 A ; 对

每一组 S, U, A , 在 $[5, 15]$ 之间随机生成一个整数值 ω , 然后对 S 中每一对步骤, 以概率 0.01ω 决定在两者之间是否生成互斥约束, 由此得到一个约束方案 X , 为了抵消生成约束时的偶然性, 对每个 ω 重复生成 5 个约束方案, 测试结果取 5 组数据的平均值。4 种算法测得的执行时间和峰值空间见表 1, 每个实例配置记为 $|S|_{\mu-m-M}_{\omega}$, 峰值空间超出可用内存时则记为 8 000+, 相应的执行时间记为 ∞ , 执行时间超出半小时则记为 1 800+, 并以这些值参与相应的平均值计算。当一组数据发生时间超限时, 将无法测得峰值空间, 因而在空间代价平均值计算时, 将忽略该组数据。

首先将算法 1 与原始 BTD 算法比较。可以看到两者需要的兆字节数相同, 统计表 1 中相关数据可知, 在两者均未超时限的 15 个实例上, 算法 1 将 BTD 的求解时间平均降低了 23.1%, 最高达到 66.8%, 且在其中 12 个实例上具有优势, 表明约束检查范围的优化可以稳定有效地提高算法效率。两种算法采用相同的树分解策略, 对所有 20 个实例, $(W+1)/|S|$ 平均为 0.186, 最大不超过 0.413, 可以验证前述的 BTD 在低约束密度下相对于值域划分法的时间复杂度优势。不过, $W+1$ 达到或超过 5.4 的 5 个实例中, 两种算法均在 3 个实例(17、18、19)上出现了严重超时, 表明

表 1 4 种 #WS(≠)算法的时间和空间代价

序号	实例配置	参考文献[2]		参考文献[8]		BTD		算法 1		
		时间/s	空间/MB	时间/s	空间/MB	时间/s	空间/MB	时间/s	空间/MB	$W+1$
1	13_70_1-33_6	0.009 35	5 458	0.422	21	0.000 320	17	0.000 239	17	2.4
2	15_41_1-33_7	0.010 7	2 729	0.922	30	0.000 242	17	0.000 203	17	2.2
3	16_40_34-66_9	0.011 7	6 822	3.39	44	0.001 89	17	0.000 841	17	3.2
4	16_74_67-100_6	0.015 0	6 821	14.5	71	0.030 4	17	0.010 4	17	2.6
5	17_43_1-100_5	0.013 4	6 822	12.7	83	0.000 742	17	0.000 426	17	2
6	18_26_34-66_15	0.014 5	6 822	8.74	116	0.003 01	17	0.001 14	17	4.6
7	18_27_1-33_8	0.009 88	0	5.68	116	0.000 267	17	0.000 228	17	2.8
8	18_45_34-66_9	0.013 7	6 821	28.2	175	0.036 8	17	0.022 7	17	3.4
9	18_46_67-100_5	0.009 34	6 821	40.4	175	0.002 01	17	0.000 810	17	2
10	21_50_67-100_6	0.065 9	6 821	601	2011	0.006 59	17	0.002 22	17	2
11	21_73_1-100_8	1.07	6 822	822	2732	360+	14	360+	14	3.6
12	23_28_34-66_13	0.014 2	6 825	679	4777	0.128	18	0.214	18	6.2
13	27_39_34-66_8	0.278	6 824	∞	8 000+	0.143	18	0.241	18	4.4
14	27_50_1-33_7	0.012 3	6 824	∞	8 000+	0.008 26	17	0.002 74	17	5.4
15	28_43_1-100_7	6.08	6 824	∞	8 000+	2.50	19	4.82	19	4.4
16	29_38_67-100_7	∞	7 057+	∞	8 000+	360+	17	360+	17	3.6
17	30_29_1-100_15	1.79	7 339	∞	8 000+	1 098+	29	1 096+	29	12.4
18	30_35_34-66_13	∞	7 745+	∞	8 000+	1 444+	24	1 444+	24	9.8
19	30_73_67-100_8	∞	7 884+	∞	8 000+	1 133+	22	1 131+	22	5.4
20	30_74_1-33_7	0.002 47	7 388	∞	8 000+	0.017 0	17	0.006 52	17	3.2

BTD 很难承受较大的树分解宽度,后文将控制其他条件相同,对时间性能随约束密度的变化做进一步实验。

将算法 1 与参考文献[8]比较。在所有实例上,后者的空间代价平均为前者的 194 倍以上。可以看到由于空间代价的迅速增长,后者只能求解前 12 个步骤数不超过 23 的实例,而前者在所有实例上都未出现空间超限,在 15 个实例配置上均未出现时间超限的情况。在两者均未出现时间和空间超限的 11 个实例上,算法 1 比对方的求解时间至少降低了 99.9%,实测时间性能具有极大优势。

将算法 1 与参考文献[2]比较。在 19 个实例上,后者的空间代价(使用 sharpSAT 本身记录的缓存大小,作为空间代价是偏保守的值,并可能为 0 值)达到前者的 253 倍以上。仅实例 7 因 5 组数据均无解未占用缓存(可能在预处理阶段即发现了冲突)。在共同未超限的 15 个实例上,算法 1 的求解时间比参考文献[2]平均增加了 48.8%。但是在其中 12 个实例上,算法 1 耗时更少,且在其余 3 个实例(8、12、20)上,绝对耗时增加不超过 0.2 s,只是相对耗时增加达到 65.7%~1 400%。这 3 个实例的 $W+1$ 在 3.2~6.2 之间,而优势的 12 个实例在 2~5.4 之间。在不超限的情况下,后者可求解 17 个实例,比算法 1 多 2 个。在实例 11 和 17 的各组数据处,后者均能够快速求解,而算法 1 在部分数据上出现了性能恶化。其中实例 17 的 $W+1=12.4$,是所有数据中最大的。实例 11 的 5 个随机数据中也出现了 1 个超时的情况,该数据上的 $W+1=6$,也属于偏高的情况。整体来看,算法 1 在多数实例上的时间性能都有一定优势,但是整体性能表现不够稳定,在少量实例上相对劣势非常明显。同时算法 1 的实测性能表现不能完全参照树分解宽度来评判,仅当 $W+1$ 在 3.2 以下的低区间时,算法 1 相对于参考文献[2]才始终具有优势,或在 6 以上区间均处于劣势

(若对方空间未超限),而在 3.2~5.4 区间,互有优劣,且其分布不完全有规律。

总的来说,算法 1 相对现有两个 #WS(≠)大幅度降低了空间占用,在树分解宽度足够小的情况下,求解时间与其中最快的参考文献[2]算法相当,综合实际性能具有明显的优势。并且实测性能和时间复杂度的反差小于参考文献[2]的算法。

4.2 实验 2:算法 1 时间性能随约束密度的变化

本实验将通过改变约束密度,考查算法 1 执行时间的变化,以便进一步确定其适用范围。数据集生成规则是:分别取 $|S|=8, 12, 14$, 相应确定 S ; 对每个 $|S|$, 在 $0.25|S| \sim 0.75|S|$ 之间随机生成一个 $|U|$ 的值, 并相应确定 U ; 取 $A=S \times U$ 以形成最大的负载; 对每一组 S, U 和 A , 令约束密度 0.01ω 从 0.05 到 0.35, 以步长 0.05 变化, 按实验 1 所述方法根据每个 ω 值生成约束方案 X , 并且对每一个实例配置, 重复生成 3 个 X , 测试结果取 3 组数据的平均值。对每一组配置, 记录算法 1 的执行时间、约束的数量和 $W+1$ 的值, 见表 2。可以看到, 除了个别点有例外, $|X|$ 和 $W+1$ 大体上随着 ω 的增长而增长, 执行时间随着 $W+1$ 的增长基本保持不减。在表 2 中, $|S|=14, \omega=20, 25$ 时, $W+1=4.7, 5$, 而执行时间为 175.0, 32.4, 显著偏离前述规律。主要是因为 $\omega=20$ 的 3 组数据, 有一组数据耗时 516.5 s, 拉高了平均值。该组数据的 $W+1=6$, 是比较高的。 $\omega=35$ 时, 平均时间达 19 772.0 s, 主要因为有一组数据的处理时间高达 58 816.2 s。其 $W+1=8$, 相对性能情况来说并非特别大, 但是该实例数据的峰值空间达到 251.9 MB, 显著大于其他数据, 因而其性能恶化原因很可能是 BTD 缓存的存取效率偏低(本文的实现并未对此进行优化)。

总结表 2 数据可知, 当约束密度达到 0.2 而 $W+1$ 超过

表 2 算法 1 执行时间随约束密度的变化

算法 1	ω	5	10	15	20	25	30	35
$ S =8, U =4$	$W+1$	2	2	2.3	2	2.7	3.3	3.3
	$ X $	1.3	2.7	5.7	4	7.3	10	10.3
	时间/s	0.000 338	0.000 211	0.000 314	0.000 249	0.000 601	0.000 738	0.001 43
$ S =12, U =5$	$W+1$	2	2.3	2.7	3	4	6	5.7
	$ X $	4.7	6.7	10.3	11.3	17.7	24.7	26.3
	时间/s	0.000 898	0.000 532	0.001 09	0.003 57	0.011 1	0.767	0.291
$ S =14, U =8$	$W+1$	2	3.3	6	4.7	5	6.3	6.7
	$ X $	3.3	13	15.3	21	19.3	30	32.3
	时间/s	0.000 937	0.034 6	14.4	175.0	32.4	1 009.5	19 772.0



4 时,即使步骤数量不超过 14 个,算法 1 也不能保证总具有很强的时间性能。

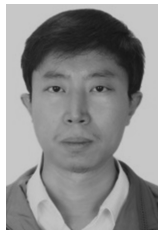
5 结束语

#WS 问题对于安全约束工作流的资源顽健性分析有重要意义。本文对现有 BTD 方法进行一定的改进,压缩其约束检查范围,然后用于 #WS($\neq$)求解,得到了一种时间复杂度为 $O(|S| \cdot x \cdot d^{W+1})$ 的算法。若确定分解树宽 W 不超过 W_0 ,则该时间复杂度简化至 $O(|S|^3 \cdot |U|^{W_0+1})$,可认为是多项式阶,是一个非常优良的理论特性。而该算法的空间复杂度为 $O(|S| \cdot w \cdot d^w)$,较时间复杂度更低。仿真实验表明,该算法在实际的约束密度足够低时,实测时间性能优于目前最快的 #WS($\neq$)算法,尽管继续提高约束密度会导致其求解效率的稳定性变差,该算法的实测空间性能始终大大优于现有两种 #WS($\neq$)算法,综合实测性能具有明显的优势。以后的工作将考虑更多约束情形下的 #WS 问题,对 BTD 技术进行研究改进,并探讨其他 CSP 求解技术在 #WS 场景中的适用性。

参考文献:

- [1] WANG Q, LI N. Satisfiability and resiliency in workflow authorization systems [J]. ACM Transactions on Information and System Security, 2010, 13(4): 747-759
- [2] 翟治年, 王刚, 郑志军, 等. 基于可满足性计数的($\neq, =$)约束工作流顽健性验证[J]. 电子学报, 2015, 43(11): 2298-2234.
ZHAI Z N, WANG G, ZHENG Z J, et al. Verification of ($\neq, =$) constrained workflow robustness based on satisfiability counting[J]. Acta Electronica, 2015, 43(11): 2298-2234.
- [3] AHMED T, TRIPATHI A R. Security policies in distributed CSCW and workflow systems[J]. IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans, 2010, 40(6): 1220-1231.
- [4] BERTINO E, FERRARI E, ALTURI V. The specification and enforcement of authorization constraints in workflow management systems [J]. ACM Transactions on Information System Security, 1999, 2(1): 65-104.
- [5] CRAMPTON J, KHAMBHAMMETTU H. Delegation and satisfiability in workflow systems [C]//13th ACM Symposium on Access Control Models and Technologies, June 11, 2008, Colorado, USA, New York: ACM Press, 2008: 31-40.
- [6] BASIN D, BURRI S J, KARJOTH G. Obstruction-free authorization enforcement: aligning security and business objectives[C]//The 24th Computer Security Foundations Symposium, June 27, 2011, Cernay, France. New Jersey: IEEE Press, 2011: 99-113.
- [7] CRAMPTON J, GUTIN G, YEO A. On the parameterized complexity and Kernelization of the workflow satisfiability problem [J]. ACM Transactions on Information and System Security, 2012, 16(1): 1518-1527.
- [8] 翟治年, 卢亚辉, 周武杰, 等. 工作流可满足性($\neq$)计数的固定参数线性算法[J]. 计算机学报, 2016(39).
ZHAI Z N, LU Y H, ZHOU W J, et al. Fixed-parameter linear algorithm for counting workflow satisfiability ($\neq$) [J]. Chinese Journal of Computers, 2016(39).
- [9] JEGOU P, TERRIOUX C. Hybrid backtracking bounded by tree-decomposition of constraint networks[J]. Artificial Intelligence, 2003, 146(1): 43-75.
- [10] ROBERTSON N, SEYMOUR P D, GRAPH MINORS. II. Algorithmic aspects of tree-width[J]. Journal of Algorithms, 1986, 7(86): 309-322.
- [11] DECHTER R, PEARL J. Tree clustering for constraint networks[J]. Artificial Intelligence, 1989, 38(4): 353-366.
- [12] BULATOV A A. The complexity of the counting constraint satisfaction problem [J]. Physical Review D Particles & Fields, 1993, 49(10): 5349-5363.
- [13] 谷文祥, 朱磊, 黄平, 等. 可满足问题中的模型计数 [J]. 智能系统学报, 2012, 7(1): 33-39.
GU W X, ZHUL, HUANG P, et al. The model counting of a satisfiability problem[J]. CAAI Transactions on Intelligent Systems, 2012, 7(1): 33-39.
- [14] ANGEL SMARK O, JONSSON P, LINUSSON S, et al. Determining the number of solutions to binary CSP instances[C]//8th International Conference on Principles and Practice of Constraint Programming, September 9, 2002, New York, USA. New Jersey: IEEE Press, 2002: 181-186.
- [15] ANGELSMARK O, JONSSON P. Improved algorithms for counting solutions in constraint satisfaction problems[J]. Lecture Notes in Computer Science, 2003(2833): 81-95.
- [16] DAHLÖF V, JONSSON P, WAHLSTRÖM M. Counting models for 2SAT and 3SAT formulae [J]. Theoretical Computer Science, 2005, 332(1-3): 265-291.
- [17] MARCIAL-ROMERO J R, LUNA G D I, HERNÁNDEZ J A, et al. A parametric polynomial deterministic algorithm for #2SAT[M]. Advances in Artificial Intelligence and Soft Computing. New York: Springer International Publishing, 2015.
- [18] FAVIER A, GIVRY S D, JÉGOU P. Exploiting problem structure for solution counting [C]//International Conference on Principles and Practice of Constraint Programming, September 20, 2009, Lisbon, Portugal. New Jersey: IEEE Press, 2009: 335-343.

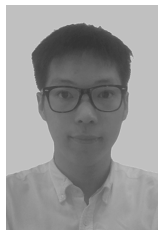
[作者简介]



翟治年(1977-),男,博士,浙江科技学院信息与电子工程学院讲师,主要从事访问控制、服务组合与 workflow 调度方面的研究工作。



吴茗蔚(1977-),女,博士,浙江科技学院信息与电子工程学院副教授,主要从事无线通信技术研究工作。



王巍橡(1995-),男,浙江科技学院信息与电子工程学院本科在读。



郑志军(1967-),男,博士,浙江科技学院信息与电子工程学院副教授,主要从事智能优化与并行计算研究工作。



卢亚辉(1976-),男,博士,深圳大学计算机与软件学院副教授,主要从事 workflow 理论与过程挖掘研究工作。



余法红(1977-),男,博士,嘉兴学院数理与信息工程学院讲师,主要从事智能计算、复杂网络与隐私保护研究工作。