工作流可满足性(≠)计数的固定参数线性算法

翟治年[1)] 卢亚辉[2)] 周武杰[1)] 陈志豪[3)] 王中鹏[1)] 林 江[1)]

[1)](浙江科技学院信息与电子工程学院 杭州 310023)

[2)](深圳大学计算机与软件学院 广东 深圳 518060)

[3)](南加州大学软件工程中心 美国 洛杉矶 CA90089)

摘 要 工作流可满足性(Workflow Satisfiability,WS)(≠)判定给定授权和互斥约束下的资源分配是否存在,是工作流访问控制中的基本问题.目前可以通过寻找一个具体的解来完成该判定,相应问题称为 WS(≠)决策,现有的最低时间复杂度为 $O^*(2^{|S|}(|C|+|U|^2))$(S,C,U 分别为步骤集、约束集、用户集).然而,仅 WS(≠)有解时,工作流授权规划未必合理,对资源异常可能缺乏鲁棒性.若能统计所有解的个数,不仅可判定 WS(≠)有解与否,还能为授权规划提供重要的参考,相应的问题称为 WS(≠)计数.该文提出 WS(≠)计数问题,并根据 Bjorclund 关于集合划分权重和的结果证明其时间复杂度为 $O^*(2^{|S|}|U|)$,即其以 $|S|$ 为固定参数,关于 $|U|$ 线性时间可解,由此降低了当前的 WS(≠)判定时间复杂度.进而,该文提出了一种快速的动态规划递推式,并全面优化 Bjorclund 方法的空间利用方式,使该文算法的实际性能随之提高,而 O^* 时间复杂度不变.随机合成数据集上的实验表明,该文最终的计数算法相对前述决策算法,执行时间平均降低了 93%,峰值空间平均降低了 87%,而求解规模提高了 44%.

关键词 工作流;访问控制;资源分配;可满足性;职责分离

中图法分类号 TP309 **DOI 号** 10.11897/SP.J.1016.2016.02291

Fixed-Parameter Linear Algorithm for Counting Workflow Satisfiability (≠)

ZHAI Zhi-Nian[1)] LU Ya-Hui[2)] ZHOU Wu-Jie[1)]

CHEN Zhi-Hao[3)] WANG Zhong-Peng[1)] LIN Jiang[1)]

[1)](*School of Information and Electronics Engineering, Zhejiang University of Science and Technology, Hangzhou* 310023)

[2)](*School of Computer and Software, Shenzhen University, Shenzhen, Guangdong* 518060)

[3)](*Center for Systems and Software Engineering, University of Southern California, Los Angeles CA* 90089, *USA*)

Abstract Workflow Satisfiability (≠) is an elementary problem in workflow access control, determining whether or not there exists a resource allocation which satisfies given authorizations and exclusion constraints. In the existing research, a concrete solution is found to show the satisfiability, the problem of which is named as WS(≠) decision with the best-known time complexity $O^*(2^{|S|}(|C|+|U|^2))$ (S, C, U denote the sets of steps, constraints and users respectively). However, in a merely satisfiable workflow, the authorization planning might be unreasonable, with low robustness to resource exception. If the number of all solutions is counted, not only the satisfiability can be determined, but also the authorization planning can be referred to it. In this paper, the counting problem of WS(≠) is proposed. Based on Bjorclund's result of sum weighted partitions, its time complexity is proved to be $O^*(2^{|S|}|U|)$, that means the problem is

收稿日期:2015-06-18;在线出版日期:2016-03-10.本课题得到国家自然科学基金(61502429,51376162,61302112)资助.**翟治年**,男,1977 年生,博士,讲师,主要研究方向为访问控制、服务组合与工作流调度.E-mail:zhaizhinian@gmail.com.**卢亚辉**,男,1976 年生,博士,副教授,主要研究方向为工作流管理、信息系统安全、Petri 网和 Pi 演算.**周武杰**,男,1983 年生,博士,讲师,主要研究方向为信息安全、图像处理.**陈志豪**,男,1973 年生,博士,主要研究方向为软件过程.**王中鹏**,男,1966 年生,博士,副教授,主要研究方向为无线通信.**林 江**,女,1963 年生,博士,教授,主要研究领域为数值计算.

solvable within the linear time of $|U|$ with $|S|$ as the fixed parameter. Thus our counting WS($\neq$) algorithm can determine WS($\neq$) in lower time complexity. Further, a fast dynamic programming recursion formula is proposed in this paper, and the space utilization of Bjorclund's method is optimized roundly. As a result, the practical performance of our algorithm is improved with the same O^* time complexity. Experiments on randomly created dataset show that: compared to the aforementioned decision algorithm, our final counting algorithm achieves averagely 93% decrease in executing time, 87% decrease in space usage, while the solvable input scale is increased by 44%.

Keywords workflow; access control; resource allocation; satisfiability; separation of duty

1 引 言

工作流管理是企业过程再造的核心技术[1],也是网格、云服务等计算体系结构的关键组成.它关注业务过程规划组织、执行控制和分析验证的自动化,由此高效调动人员、应用和数据,实现充分的商业价值.在工作流管理系统中,业务过程被划分为一组有序的步骤(任务),并统一调度执行.为了保障业务安全,防止非法访问与职权滥用,必须建立适当的访问控制机制.

工作流访问控制贯穿业务过程的规划与执行,分为授权、约束和资源分配 3 个部分.在业务规划阶段,将每个步骤的执行权限分配给一组候选用户,称为授权.授权可以采取基于角色的访问控制[2]、基于任务的授权控制[3]等方式,最终将建立步骤与用户的二元关系.用户(或应用、Web 服务等执行资源)必须具备其授权步骤的执行能力,但尚不能在工作流的任何案例中执行这些步骤.在业务执行阶段,为每个步骤指派唯一的用户,负责该步骤在当前案例中的执行,称为资源分配[1].资源分配使用户在特定案例中切实获得步骤的执行权限.如果步骤之间存在冲突或敏感关系,那么在同一案例中,它们的执行用户之间也必须满足一定的关系,称为约束.根据步骤关系的不同及其对用户关系的要求,可以定义多种类型的约束.例如职责分离[4]禁止某些利益冲突的步骤由同一用户执行,以免出现借机欺诈.若职责分离特指其二元情形,则也称为互斥约束,记为$\neq$.例如报销和审核、编程和测试等都具有互斥关系.

给定授权和约束,是否存在可行的资源分配,使得任何步骤的执行用户均满足预先授权,又不违反任何约束?这就是工作流的可满足性(Workflow Satisfiability,WS).WS 是工作流访问控制的基础性问题[5],因为它决定着受限于访问控制策略的业务过程能否得以实施.WS 有多方面的含义,若问题目标是判定可行资源分配是否存在,则称为 WS 判定(Determining),记作?WS.此问题关系到访问控制规划是否正确.进入案例执行时,还须具体给出一个可行的资源分配[5-6],相应的问题称为 WS 决策(Decision),记作 $*$WS.求解 $*$WS 也是回答?WS 的一条途径.WS 研究通常针对特定的约束类型(在括号中标注,例如互斥约束 WS 记为 WS($\neq$)),目前主要集中于各类约束下的 $*$WS.

2010 年,Wang 等人[5]指出由于职责分离是必不可少的约束类型,$*$WS 问题本质上是 NP 完全的.他们给出了一种 $O(|S|^{|S|+1}|C|)$时间的 $*$WS($\neq$)算法(S,C 分别为步骤和约束集).2013 年,Crampton 等人[6]将 $*$WS($\neq$)的时间复杂度降低至 $O^*(2^{|S|}(|C|+|U|^2))$(U 表示用户集),并对多种约束下的 $*$WS 进行了研究.然而,一个工作流即使是决策可满足的,仍可能因为执行期间的资源失效,导致案例处理中断甚至资源分配无解.第 2 节给出的引例说明,通过合理的授权规划来增加资源分配解的数量,将能有效提高工作流在资源异常情况下的鲁棒性.

给定约束,统计在某种授权下资源分配可行解的个数,称为 WS 计数(Counting),记作 $\sharp$WS.它与工作流的资源鲁棒性有密切联系.其一,引例表明 $\sharp$WS 为资源鲁棒性的度量提供了参考性指标.其二,为精确度量这种鲁棒性,Wang 等人已经提出了工作流 k 弹性(Workflow k Resiliency, WR$^{(k)}$)概念,即当任意 k 个资源失效时 WS 总保持成立.WR$^{(k)}$ 验证可分解为 $C(|U|,k)$个?WS 问题,而 $\sharp$WS 为?WS 的回答提供了一条非 $*$WS 的途径.其三,授权规划通常存在多次试误,相应的 WR$^{(k)}$ 验证只是完成排除,有可能用更简单的条件检查来代替.由于 WR$^{(k)}$ 对 WS 解的数量有一定要求,例如只有 1 个 WS 解不可能保证 WR$^{(1)}$ 成立,$\sharp$WS 为这种条件的构造提供了

潜在的途径.此外,合取范式可满足性(Conjunctive Normal Form Satisfiability,SAT)、约束可满足性(Constraint Satisfiability,CS)等经典问题均包括决策和计数两个方面.决策强调给出具体可行的解,而计数侧重整体和统计特征,其结果互为补充.它们从不同角度进行可满足性判定,有利于深入理解其内在逻辑,拓宽求解途径.#SAT 和#CS 的研究已有多年的历史,出现了很多重要的算法与应用[7-8].目前,WS 研究主要集中于决策.然而,无论作为资源鲁棒性的参考指标,还是 $WR^{(k)}$ 排除条件的构造途径,或回答?WS 的另一途径和 WS 本身的重要侧面,#WS 研究都是不可或缺的.

由于长期实践证明职责分离是业务安全的基础性原则[4,9],而其最典型的情况是二元互斥.本文将对相应的#WS(≠)问题进行研究.在此之前,Wang 等人[5]已在 *WS 研究中引入了参数化复杂性[10],本文将从这一角度进行#WS 的研究.对于 *WS(≠),Crampton 等人[6]给出了 $O^*(2^{|S|}(|C|+|U|^2))$ 的参数化时间,是其已知最低的时间复杂度,而他们的求解方法依赖于 Bjorclund[11]关于集合最大权划分的有力结果.Bjorclund 为解决集合划分问题发展了一种赋权集容斥原理和快速 ζ 变换结合的方法.本文将基于这一方法来建立#WS(≠)的算法.

本文的主要贡献在于,基于前述思路给出一种 $O^*(2^{|S|}|U|)$ 时间的算法,表明#WS(≠)固定参数线性时间可解.本文算法时间复杂度优于前述 Crampton 的 *WS(≠)算法,因而给出了目前最低的?WS(≠)时间复杂度.不仅如此,本文还提出了一种快速的动态规划递推式,并对 Bjorclund 方法的空间利用进行优化,使本文#WS(≠)算法在保持时间复杂度的前提下,进一步提高实际性能.

本文第 2 节通过实例来引入 WS 计数问题;第 3 节介绍预备知识,包括参数化算法、赋权集容斥原理、快速 ζ 变换、划分权重和问题及其求解方法;第 4 节给出各类 WS 问题的定义;第 5 节根据 Bjorclund 关于划分权重和的结果证明#WS(≠)的时间复杂度,事实上也给出了一种#WS(≠)算法;第 6 节改进 Bjorclund 的划分权重和求解方法,由此得到一种性能更为优化的#WS(≠)算法,并表明其保持第 5 节所得时间复杂度;第 7 节对本文两种算法进行实验研究,并与现有代表性 *WS(≠)算法就 WS 判定性能进行对比;第 8 节介绍和分析相关工作;第 9 节总结全文.

2　引　例

一个物料采购工作流如图 1 所示,包括 5 个步骤:申请采购(s_0)、创建订单(s_1)、验收货物(s_2)、准备支付(s_3)和同意支付(s_4).在 s_0 中,根据生产需求拟定物料请购单,提交订货;在 s_1 中,审核采购需求、选择供货商,创建并发出订单;在 s_2 中,对供货商发来的物料进行检验,确认符合订货要求;在 s_3 中,若订货价格基本合理则创建支付申请;在 s_4 中,根据货物验收单等凭据对支付申请进行审批.

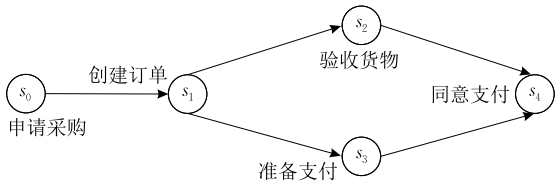


图 1　物料采购工作流

由于支付涉及资金流出,必须通过一定的规则来保证业务安全.为了控制不合理订货,订货人不能准备支付.为了分离实物验收和价格检查两项控制职能,验收货物和准备支付不能由同一人执行.为了防止申请者自我审批,一个人不能既申请采购,又创建订单,也不能既准备支付,又同意支付.将这些规则表达为互斥约束,如图 2 所示.图中顶点表示步骤,而边表示步骤间的互斥关系.

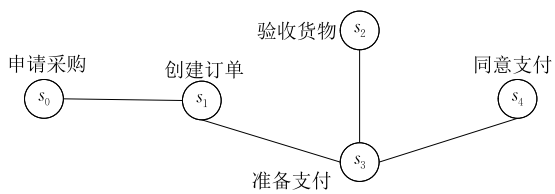


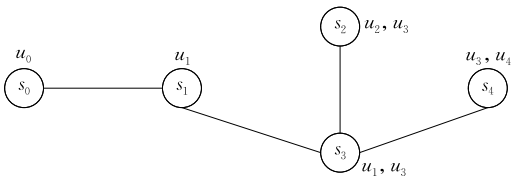
图 2　本例的互斥约束

假设有 5 个用户,记为 u_0,u_1,u_2,u_3 和 u_4,负责该工作流的处理.如图 3 给出了两种可能的授权方案,在每个步骤旁边列出了其所有授权用户.

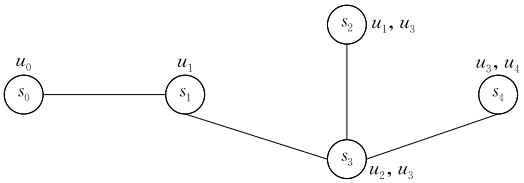
容易看出,授权 1 只有一种可行的资源分配.如果用从步骤集到用户集的函数 π 来描述,则

$$\pi(s_0)=u_0,\pi(s_1)=u_1,\pi(s_2)=u_2,$$
$$\pi(s_3)=u_3,\pi(s_4)=u_4$$

表示同一案例中 s_0,s_1,s_2,s_3 和 s_4 分别由 u_0,u_1,u_2,u_3 和 u_4 执行.该资源分配依赖于每个用户,任何用户作为执行资源失效(如离职)都会使相关案例处理中断,新案例的资源分配无解.采用该授权方案时,工作流是决策可满足的,但是缺乏必要的资源鲁棒



(a) 授权1



(b) 授权2

图 3 本例的两种授权方案

性,即在资源异常情况下维持运转的能力,无法在缺少某些资源时仍然给出资源分配可行解.将授权 1 中步骤 s_2,s_3 的授权用户 u_2,u_1 进行对调,即得授权 2.它有如下 5 种可行的资源分配:

(1) $\pi(s_0)=u_0,\pi(s_1)=u_1,\pi(s_2)=u_1,\pi(s_3)=u_2,\pi(s_4)=u_3$;

(2) $\pi(s_0)=u_0,\pi(s_1)=u_1,\pi(s_2)=u_1,\pi(s_3)=u_2,\pi(s_4)=u_4$;

(3) $\pi(s_0)=u_0,\pi(s_1)=u_1,\pi(s_2)=u_3,\pi(s_3)=u_2,\pi(s_4)=u_3$;

(4) $\pi(s_0)=u_0,\pi(s_1)=u_1,\pi(s_2)=u_3,\pi(s_3)=u_2,\pi(s_4)=u_4$;

(5) $\pi(s_0)=u_0,\pi(s_1)=u_1,\pi(s_2)=u_1,\pi(s_3)=u_3,\pi(s_4)=u_4$.

其中第(1)、(3)种资源分配不依赖于 u_4,第(2)种不依赖于 u_3,第(5)种不依赖于 u_2,只有第(4)种依赖所有用户.采用授权 2 时,不论单独缺少用户 u_2、u_3 还是 u_4,对新到达案例,总存在资源分配可行解.受影响的原有案例,通过适当调整也可以迁移到相近的解,继续其处理.该授权方案同样决策可满足,而且比前一方案更能抵抗资源异常的干扰,在此意义上也更为合理.

由本例可见,工作流的资源鲁棒性与可行资源分配的数量有密切关系.解的数量是其多样性的集中体现.解集越大,替代的资源分配方式越多,工作流对单一资源的依赖性也越低,从而越少受资源异常的干扰.解集大小对资源鲁棒性的影响不是绝对的,但是为其提供了一个重要的参考指标.

3 预备知识

本节介绍必要的预备知识.约定 Z^+ 表示非负整数集,并记 $N=\{1,2,\cdots,n\}$.

3.1 参数复杂性与 O^* 分析

经典复杂性理论表明,NP 难问题不太可能存在多项式时间的有效算法.在实际应用中,多采用近似或智能算法求解,无法获得可重复的精确解.20 世纪 90 年代,Downey 和 Li 等人[10,12]提出了参数化复杂性理论,为求解此类问题开辟了一条新途径.他们发现很多问题具有 $f(k)p(n)$ 形式的复杂度,其中 k 和 n 反映不同输入的规模,$f(k)$ 是关于 k 的任意函数,而 $p(n)$ 是关于 n 的多项式函数.在 k 不超过某个固定值的条件下,问题复杂度是 n 的多项式,从而可以有效求解.

当问题复杂度很高时,通常采用 O^* 记法进行分析.它是 O 表示法的简化,可以反映算法的主要开销[13].设 c 为任意正常数,若 $p(n)$ 表示关于 n 的多项式,则 $O(c^n p(n))$ 可以简化表示为 $O^*(c^n)$.

3.2 赋权集容斥原理

容斥原理是一种基本的组合计数方法,在有限集元素计数问题中有广泛的应用.它可以推广到赋权有限集上[11],如下面的定理.

定理 1. 设 S 为有限集,$w:2^S\to Z$ 是其子集格上的权函数,即 2^S 为 w 赋权集.又设 $\{S_i\mid i\in I\}$ 是 S 的一组子集,其中 I 为整数下标集.若将 S 记为 $\bigcap_{i\in\varnothing}S_i$,则有

$$w\left(\bigcap_{i\in I}\overline{S}_i\right)=\sum_{J\subseteq I}(-1)^{|J|}w\left(\bigcap_{j\in J}S_j\right).$$

3.3 快速 ζ 变换

ζ 变换也称 Möbius 变换,在证据理论等研究中有着重要应用[14].下面介绍它的一种形式及快速计算方法.

定义 1. 设 $S=\{s_1,s_2,\cdots,s_n\}$,2^S 为 w 赋权集,定义函数 $\hat{g}:2^S\to Z^+$,使得对任何 $S'\subseteq S$,$\hat{g}(S')$ 等于 S' 所有子集的 w 权重之和,即 $\hat{g}(S')=\sum_{S''\subseteq S'}w(S'')$.根据 w 计算 $\hat{g}$ 的过程称为 ζ 变换.

通过定义直接进行 ζ 变换,需要 $O(3^n)$ 时间[11].下面的定理给出了一种根据 w 计算 $\hat{g}$ 的递推式.

定理 2. 设 $g_0,g_1,\cdots,g_n$ 是一组从 2^S 到 Z^+ 的函数,且对任何 $S'\subseteq S$,$g_i(S')$ 定义为

$$g_i(S')=\begin{cases}g_{i-1}(S')+g_{i-1}(S'-\{s_i\}), & s_i\in S'\\ g_{i-1}(S'), & s_i\notin S'\end{cases},\quad 1\le i\le n,$$

则当 $g_0=w$ 时,$g_n=\hat{g}$.

若以 $O^*(2^n)$ 空间存储所有中间的 $\hat{g}_i$,则 $\hat{g}$ 可通过 $O(2^n n)$ 次加法求得,称为快速 ζ 变换[11,14].

3.4 集合划分权重和

本文提出的#WS(≠)问题可归结为集合划分权重和问题，其定义如下.

定义 2. 设 $w_1,w_2,\cdots,w_k:2^S\to[-M,M]\subseteq Z$ 是集合 $S=\{s_1,s_2,\cdots,s_n\}$ 子集格上的 k 个权函数. 若 $S_i\subseteq S(1\leq i\leq k)$，则称 $t_k=(S_1,S_2,\cdots,S_k)$ 为 S 上的 k 元组，将其集合记为 T_k. 令权函数 $w:T_k\to Z$，使对任意 $t_k\in T_k$，$w(t_k)=\prod_{1\leq i\leq k} w_i(S_i)$，则 T_k 为 w 赋权集. 若 $\bigcup_{1\leq i\leq k} S_i=S$ 且 $S_i\cap S_j=\varnothing(i\neq j)$，则称 $t_k=(S_1,S_2,\cdots,S_k)$ 为 S 的 k 划分(划分块可为空). 集合划分权重和，即 S 所有 k 划分的 w 权值之和.

文献[11]利用赋权集容斥原理与快速 ζ 变换结合的方法求解该问题，得到了 $O^*(2^n k\log M)$ 的时间复杂度，其求解方法介绍如下.

将 T_k 中所有 k 划分的集合记为 P_k，对其应用赋权集容斥原理可得

$$w(P_k)=\sum_{J\subseteq\{1,2,\cdots,n\}}(-1)^{n-|J|}w(T_k^{(n)}(S_J)) \quad (1)$$

式中 $T_k^{(n)}(S_J)$ 为 $S_J=\{s_j\in S\mid j\in J\subseteq N\}$ 上满足 $|S_1|+|S_2|+\cdots+|S_k|=n$ 的 k 元组 $t_k=(S_1,S_2,\cdots,S_k)$ 的集合. 由此可得

$$w(T_k^{(n)}(S_J))=\sum_{m_1+m_2+\cdots+m_k=n}\prod_{i=1}^{k}\hat{g}_i^{(m_i)}(S_J) \quad (2)$$

式中 $\hat{g}_i^{(m)}:2^S\to Z(1\leq i\leq k,0\leq m\leq n)$. 设 $w_i^{(m)}:2^S\to[-M,M]$，使得对 $S'\subseteq S$，

$$w_i^{(m)}(S')=\begin{cases}w_i(S'), & |S'|=m,\\ 0, & \text{否则}\end{cases} \quad (3)$$

则任取 $S'\subseteq S$，

$$\hat{g}_i^{(m)}(S')=\sum_{S''\subseteq S'}w_i^{(m)}(S'') \quad (4)$$

因此，从 $w_i^{(m)}$ 到 $\hat{g}_i^{(m)}$ 是一个 ζ 变换. 利用快速 ζ 变换求出所有 $\hat{g}_i^{(m)}$，需要 $O^*(kn2^n\log M)$ 时间(文献[11]引理 6).

为了进一步利用 $\hat{g}_i^{(m)}$ 计算式(2)的右端，根据如下递推式进行动态规划

$$g(p,q,r,S_J)=\begin{cases}\sum_{m=0}^{r}\{g(p,\lfloor(p+q)/2\rfloor,m,S_J)\cdot \\ \quad g(\lfloor(p+q)/2\rfloor+1,q,r-m,S_J)\}, & p<q\\ \hat{g}_p^{(r)}(S_J), & p=q\end{cases} \quad (5)$$

由该式求得 $g(1,k,n,S_J)$ 即为 $w(T_k^{(n)}(S_J))$. 动态规划缓存中间所有 $g(p,q,r,S_J)$，避免重复计算，每个 $w(T_k^{(n)}(S_J))$ 均可在 $O^*(kn^3+k(\log k)n^2\log M)$ 时间内完成计算(文献[11]引理 6).

4 问题定义

本节定义 WS(≠)相关问题. 设 $U=\{u_1,u_2,\cdots,u_k\}$ 表示用户集，$S=\{s_1,s_2,\cdots,s_n\}$ 表示步骤集.

定义 3. 工作流 $(S,\leq)$ 是步骤的偏序集，其中 $\leq$ 表示步骤间的执行顺序. 任取 s,s' 两个步骤，若 $s<s'$，则 s 先于 s' 执行. 将工作流的一次执行称为一个案例.

定义 4. 工作流授权 $A\subseteq S\times U$ 是二元组的集合，表示以 S 为步骤集的工作流在用户集 U 上的权限分配情况. 若 $(s,u)\in A$，则用户 u 是步骤 s 的一个候选执行者.

定义 5. 互斥约束 $X\subseteq S\times S$ 是无序二元组的集合，表示步骤间的职责分离关系. 若 $(s,s')\in X$，则在同一案例中，s 和 s' 不能由同一用户执行.

定义 6. 设 $S'\subseteq S$，函数 $\pi:S'\to U$ 称为 S' 上的资源分配. 所有 S' 上的资源分配的集合记为 $\prod_{S'}$. 在不引起混淆时，将 S 上的资源分配简称为资源分配. 对 S' 上的资源分配 π：

(1) 若任取 $s\in S'$ 均有 $(s,\pi(s))\in A$，则称 π 是基于 A 的.

(2) 若存在 $(s,s')\in X$ 使得 $\pi(s)=\pi(s')$，则称 π 违反 X.

定义 7. 互斥约束工作流可满足性，记为 WS(≠)，是一个二元组 $\langle A,X\rangle$，从中可以引出以下 3 个问题：

(1) WS(≠)判定问题，记为?WS(≠)，要求判定基于 A 且不违反 X 的资源分配是否有解. 当?WS(≠)的回答为 Yes/No 时，称 WS(≠)成立/不成立.

(2) WS(≠)决策问题，记为 $*$WS(≠)，要求给出一个基于 A 且不违反 X 的资源分配，或指出其无解.

(3) WS(≠)计数问题，记为#WS(≠)，要求统计基于 A 且不违反 X 的所有资源分配解的个数，即 $*$WS(≠)所有可能解的个数.

其中 $*$WS(≠)属于 NP 问题，其解可在多项式时间内验证.

5 #WS(≠)的固定参数线性复杂度

本节将通过归结为集合划分权重和问题，证明#WS(≠)具有 $O^*(2^{|S|}|U|)$ 时间复杂度.

一个资源分配给出了步骤集 S 上的一个 $|U|$-划分，第 $1\leq i\leq|U|$ 个划分块由该资源分配指派用

户 u_i 执行的所有步骤构成，若 u_i 未被指派执行任何步骤，相应的划分块为空. 反过来，S 上的任何 $|U|$-划分也给出了一个如此含义的资源分配. 该资源分配是可行的，当且仅当每个划分块满足以下要求：

(1) 第 i 个划分块中，任何步骤的授权用户中均含有 u_i.

(2) 对每个划分块，当其中所有步骤由同一用户执行时，不违反 X 中任何约束.

由于 $\sharp\mathrm{WS}(\neq)$ 对所有可行资源分配计数，可将其转化为对符合条件(1)、(2)的 $|U|$-划分计数. 根据这两个条件选取适当的权函数，可进一步将问题转化为求集合划分权重和，相应有以下定理.

定理 3. $\sharp\mathrm{WS}(\neq)=(S,U,A,X)$ 可在 $O^*(2^n k)=O^*(2^{|S|}|U|)$ 时间内求解.

证明. 记 $S_A(u_i)=\{s\in S|(s,u_i)\in A\}(1\leq i\leq|U|)$，表示用户 u_i 根据 A 有权执行的所有步骤. 取权函数 $w_i:2^S\to\{0,1\}\subseteq[-2,2]$，使对任意 $S'\subseteq S$

$$w_i(S')=\begin{cases}1, & S'\subseteq S_A(u_i)\wedge\neg\exists(s,s')\in X(\{s,s'\}\subseteq S')\\ 0, & \text{否则}\end{cases}\tag{6}$$

任取 S 上的 $|U|$-划分 $p_k=(S_1,S_2,\cdots,S_k)$，划分块 S_i 符合前述条件(1)、(2)当且仅当 $w_i(S_i)=1$. 令 $w(p_k)=\prod_{1\leq i\leq k}w_i(S_i)$ 可知，p_k 构成可行资源分配当且仅当 $w(p_k)=1$，于是可行资源分配的数量即为 S 上所有 $|U|$-划分的 w 权重和. 由 3.4 节可知，此划分权重和可在 $O^*(2^n k\log 2)=O^*(2^n k)$ 时间内求得，这也就是 $\sharp\mathrm{WS}(\neq)=(S,U,A,X)$ 的时间复杂度. 证毕.

上述结论表明 $\sharp\mathrm{WS}(\neq)$ 以 $|S|$ 为固定参数，关于 $|U|$ 线性时间可解. 此结果优于 $*\mathrm{WS}(\neq)$ 目前的最低时间复杂度 $O^*(2^{|S|}(|C|+|U|^2))$[6]，从而也降低了 $?\mathrm{WS}(\neq)$ 的时间复杂度.

文献[11]未分析集合划分权重和问题的空间复杂度，但其求解过程(参见 3.4 节)中需要在步骤集 S 的子集格上计算 $|U|\cdot|S|$ 个快速 ζ 变换，使得集合划分权重和与 $\sharp\mathrm{WS}(\neq)$ 的空间复杂度至少为 $O^*(2^{|S|}|U|)$.

空间复杂度高是上述方法固有的特点，然而文献[11]的空间利用方式缺乏深入规划，造成了进一步的浪费. 特别地，即使空间充裕，算法的实测时间性能仍不够理想.

6 优化的 $\sharp\mathrm{WS}(\neq)$ 固定参数线性算法

本节将在赋权集容斥原理与快速 ζ 变换结合框架下，通过更有效地动态规划和空间利用来求解集合划分权重和问题，由此给出一种时间和空间复杂度均为 $O^*(2^{|S|}|U|)$ 的 $\sharp\mathrm{WS}(\neq)$ 算法，而其实际的时间和空间开销明显降低. 为了简化公式表述，约定两个记法 $m_{p,q}=\lfloor(p+q)/2\rfloor$，$l_{p,q}=q-p+1$. 当 p,q 的形式比较复杂时，也分别写成 $m(p,q)$ 和 $l(p,q)$.

下面首先在 1.4 节给出的集合划分权重和求解框架中，引入一种更有效的动态规划递推式.

在该框架中，由容斥原理将赋权集上所有 k 划分的权重和转化为一组独立 $w(T_k^{(n)}(S_J))$，也即 $g(1,k,n,S_J)(J\subseteq N)$，的计算；并利用快速 ζ 变换根据权函数 $w_i(1\leq i\leq k)$ 计算 $\hat{g}_i^{(m)}(0\leq m\leq n)$；为了沟通容斥原理和 ζ 变换之间的联系，由 $\hat{g}_i^{(m)}$ 高效计算 $g(1,k,n,S_J)$，关键在于适当的动态规划. 文献[11]采用式(5)，将 $g(p,q,r,S_J)$ 递推分解为 $g(p,\lfloor(p+q)/2\rfloor,m,S_J)$，$g(\lfloor(p+q)/2\rfloor+1,q,r-m,S_J)$ 的一组乘积之和，递推终止时分解为单独的 $\hat{g}_i^{(m)}$. 不过，若 $\hat{g}_i^{(m)}=0$，则其参与的所有乘积均为 0，对 $g(p,q,r,S_J)$ 的值没有贡献. 而采用式(5)，递推终止前，只能假设所有 $\hat{g}_i^{(m)}(p\leq i\leq q)$ 对 $g(p,q,r,S_J)$ 都有贡献，无法及时进行剪枝，这就影响了计算性能的进一步提高. 基于上述分析，本文提出一种有利于快速剪枝的递推式，如下面的定理 4.

定理 4. 设 $\hat{g}_i^{(m)}:2^S\to Z^+(1\leq i\leq k,0\leq m\leq n)$，由式(4)定义，式中 $w_i^{(m)}$ 由式(3)定义，其中 w_i 又由式(6)定义，任取 $S_J\subseteq S,1\leq p,q\leq k,0\leq r_{p,q}\leq n$，定义 $g(p,q,r_{p,q},S_J)$ 如式(7). 则当 $p<q$ 且 $r_{p,q}=r$ 时，$g(p,q,r_{p,q},S_J)$ 等于式(5)给出的 $g(p,q,r,S_J)$.

$$g(p,q,r_{p,q},S_J)=\begin{cases}\sum\limits_{r(m_{p,q},m_{p,q})=0}^{r_{p,q}}\{g(q,p,r_{p,q}-r(m_{p,q},m_{p,q}),S_J)\cdot \hat{g}_{m_{p,q}}^{(r(m_{p,q},m_{p,q}))}(S_J)\}, & p<q\\ \sum\limits_{r_{p,q}=0}^{r_{q,p}}\{g(q,m_{p,q}-1,r_{p,q},S_J)\cdot g(m_{p,q}+1,p,r_{q,p}-r_{p,q},S_J)\}, & q<p\\ \hat{g}_p^{(r_{p,p})}(S_J), & p=q\end{cases}\tag{7}$$

证明. 当 $p<q$ 且 $r_{p,q}=r$ 时，将式(7)变形得

$$g(p,q,r,S_J)=\begin{cases}\sum\limits_{m=0}^{r}\{\hat{g}_{m_{p,q}}^{(m)}(S_J)\cdot\sum\limits_{t=0}^{r-m}\{g(m_{p,q}+1,q,r-m-t,S_J)\cdot g(p,m_{p,q}-1,t,S_J)\}\}, & p<q\\ \hat{g}_p^{(r)}(S_J), & p=q\end{cases}\cdot$$

该式与式(5)以不同方式计算同样的 $g(p,q,r,S_J)$，也就是式(7)中的 $g(p,q,r_{p,q},S_J)(p<q)$. 证毕.

根据这一定理，可以用式(7)取代式(5)来计算 $g(1,k,n,S_J)$，即 $w(T_k^{(n)}(S_J))$. 当 $p<q$ 时，式(7)的每次递推都要代入 $\hat{g}_i^{(m)}(S_J)$ 的多个值，形成对 $g(q,p,r-m,S_J)$ 的多次调用，若 $\hat{g}_i^{(m)}(S_J)=0$，则相应的调用分枝可以剪去. 分析定义可知，若 $|S_J|<m$，或者 S_J 中基数为 m 的子集 w_i 权值为 0，均有 $\hat{g}_i^{(m)}(S_J)=0$，因此对任意 $1\le i\le k$，在有 m 和 S_J 两个维度的 $\hat{g}_i^{(m)}(S_J)$ 矩阵中，至少有近一半的 0 值. 这就为前述剪枝的效果提供了一定的保证.

在本文定理 3(将 ♯WS(≠)归结为集合划分权重和，从而可以利用其求解框架)和定理 4(提出一种新的动态规划递推式)的基础上，进一步对空间利用进行优化，给出 ♯WS(≠)算法如下.

算法 1.　Counting Workflow Satisfiability(≠).

输入：(S,U,A,X)，其中 $A\subseteq S\times U, X\subseteq S\times S$

输出：符合 A 且不违反 X 的资源分配的个数

1. 对每个 $1\le i\le k$，根据式(6)计算 $w_i:2^S\to\{0,1\}$ 并保存结果. 分为以下 3 步：

1.1 遍历 X 判定每个 $S'\subseteq S$ 是否违反约束，保存结果.

1.2 遍历 A，求每个 $u_i(1\le i\le k)$ 的授权步骤集，记为 $S_A(u_i)$，并保存结果.

1.3 对每个 $1\le i\le k$ 和 $S'\subseteq S$，若 $S'\subseteq S_A(u_i)$ 且 S' 不违反约束，$w_i(S')\leftarrow 1$，否则 $w_i(S')\leftarrow 0$. 所有 w_i 计算完成后保存结果，每个 w_i 占用 n 位二进制以备重复利用，释放 1.1、1.2 两步保存的结果.

2. 求 $\hat{g}_i^{(m)}:2^S\to[0,C(n,\lfloor n/2\rfloor)](1\le i\le k,0\le m\le n)$ 并保存. 分为以下 2 步：

2.1 对每个 $1\le i\le k,0\le m\le n$，根据式(3)由 w_i 计算 $w_i^{(m)}:2^S\to\{0,1\}$ 并保存.

2.2 对每个 $1\le i\le k,0\le m\le n$，根据式(4)由 $w_i^{(m)}$ 计算 $\hat{g}_i^{(m)}$. 采用快速 ζ 变换，令 $g_0=w_i^{(m)}$，按照定理 2 的 ζ 变换递推式求 $g_n=\hat{g}_i^{(m)}$. 先在 $g_0=w_i^{(m)}$ 之外开辟一块辅助空间作为 g_1 的存储，然后令 $j\leftarrow 1$，根据 g_{j-1} 对所有 $S'\subseteq S$ 计算 $g_j(S')$，并写入其存储空间，再根据 g_j 对所有 $S'\subseteq S$ 计算 $g_{j+1}(S')$，并清空整个 g_j 占据的存储，用来存放 g_{j+1}(相当于 $j\leftarrow j+1$，重复上一步操作)，如此交替(循环)，直至求出 g_n. 此时若 g_n 位于辅助空间中，将其复制到最初 $g_0=w_i^{(m)}$ 所占空间. 所有快速 ζ 变换均如此计算，并共用前述辅助空间. 所有 $\hat{g}_i^{(m)}$ 计算完成后，释放辅助空间.

3. 根据式(1)，采用部分求和方式计算 $w(P_k)$. 先将部分和置为 0，依次对每个 $J\subseteq N$ 计算 $(-1)^{n-|J|}w(T_k^{(n)}(S_J))$ 并加入部分和，最终结果即为 $w(P_k)$. 其中每个 $w(T_k^{(n)}(S_J))$，即 $g(1,k,n,S_J)$，根据式(7)由 $\hat{g}_i^{(m)}$ 计算，共有 2^n 个动态规划，令其共用同一块缓存.

以上算法描述中，$\hat{g}_i^{(m)}$，$g(p,q,r,S_J)$ 和 $w(P_k)$ 的结果均可能为大整数，其可能值的长度随问题规模增长，实现时应保证其空间分配. $\hat{g}_i^{(m)}$ 的最大可能值为 $C(n,\lfloor n/2\rfloor)$(n 元集中 $\lfloor n/2\rfloor$ 组合的个数，也就是 n 元集中具有相同基数的子集最大可能的数量)，小于 2^n，所占据的二进制位数不超过 n. 关于 $g(1,k,n,S_J)$ 的估值，有如下不等式.

定理 5.　设 $\hat{g}_i^{(m)}:2^S\to Z^+(1\le i\le k,0\le m\le n)$ 由式(4)定义，式中 $w_i^{(m)}$ 由式(3)定义，其中 w_i 又由式(6)定义，则任取 $S_J\subseteq S,1\le p,q\le k,0\le r_{p,q}\le n$ 有

$$g(p,q,r_{p,q},S_J)\le g(p,q,r_{p,q},S)\le [(r_{p,q}+1)2^n]^{|q-p|+1}\le[(n+1)2^n]^{|q-p|+1}\quad(8)$$

证明.　见附录.

因此，$g(p,q,r,S_J)$ 不超过 $l_{p,q}[n+\log(n+1)]$ 位二进制. 由于 $w(P_k)\le\sum_{J\subseteq N}g(1,k,n,S_J)\le(n+1)^k2^{n(k+1)}$. $w(P_k)$ 的二进制位数不超过 $k\log(n+1)+n(k+1)$ 位.

上述大整数的运算时间也可能超出常量范围. 设某个整数二进制位数为 $p(n)$，则其加减和比较运算需 $O(p(n))$ 时间，乘法运算需 $O^*(p(n))$ 时间[15].

下面通过逐步分析，确定算法 1 的时间和空间复杂度.

1.1 由于 $x=|X|=O(n^2)$，$|2^S|=2^n$，判定 S' 是否违反某个约束需要进行 $O(n)$ 次比较，本步共需 $O(2^nnx)=O(2^nn^3)=O^*(2^n)$ 时间，保存结果需要 $O(2^n)$ 空间.

1.2 由于 $|A|=O(nk)$，本步需 $O(nk)$ 时间，而 $|S_A(u_i)|=O(n)$，保存所有 $S_A(u_i)$ 需要 $O(nk)$ 空间.

1.3 判定 $S'\subseteq S_A(u_i)$，需要 $O(n^2)$ 时间，再利用 1.1 步保存的结果，可在常数时间内求出 $w_i(S')$. 本步共需 $O(2^nn^2k)=O^*(2^nk)$ 时间，保存结果需要 $O(2^nnk)=O^*(2^nk)$ 空间.

第 1 步共需 $O(2^nnx+nk+2^nn^2k)=O^*(2^nk)$ 时间，最大空间占用 $O(2^n+nk+2^nnk)=O^*(2^nk)$，出现在第 1.3 步.

2.1 用 $O(n)$ 时间求出 S' 的基数后，利用 1.3 步保存的 w_i，只需常数时间即可确定 $w_i^{(m)}(S')$ 的值，本步共需 $O(2^nn^2k)=O^*(2^nk)$ 时间，保存结果需要 $O(2^nnk)=O^*(2^nk)$ 空间.

2.2 本步共有 nk 个快速 ζ 变换，完成计算需要 $O(2^nn^2k)$ 次 $O(n)$ 位加法，其时间复杂度为 $O(2^nn^3k)=O^*(2^nk)$. 在第 2.2 步 $w_i^{(m)}$ 的存储之外，只需 $O(2^n)$ 个 $O(n)$ 位整数，即 $O(2^nn)=O^*(2^n)$ 大小的辅助空间，较之粗放的利用方式，每次快速 ζ 变换降低了 $(n-1)/(n+1)$ 的空间占用.

第 2 步共需 $O(2^nn^2k+2^nn^3k)=O^*(2^nk)$ 时间，最大空间占用为 $O(2^nnk+2^nn)=O^*(2^nk)$，出现在第 2.2 步.

3. 先对 $g(1,k,n,S_J)$ 的计算进行分析. 根据式(7)计算

$g(1,k,n,S_J)$，递推完全终止时，将分解为若干 $\hat{g}_p^{(r)}(S_J)=g(p,p,r,S_J)$，其下标 $1\le p\le k$，有 k 个不同的值. 不妨设 $k=2^{h+1}-1$，根据式(7)计算 $g(1,k,n,S_J)$ 的过程可以表示为一棵有 $k=2^{h+1}-1$ 个叶结点的递推树，如图 4 所示. 树的深度为 $2h$，其根结点位于第 0 层. 此递推树是以一棵深度为 h 的正则完全二叉树为基础，将每个内结点(图中矩形框)扩展为 2 个内结点(以垂直边相连的一对黑色结点)和一个叶结点(阴影结点)而得. 基础二叉树有 $2^{h+1}-1$ 个结点，其中 2^h-1 个内结点，故递推树有 $2^{h+2}-3$ 个结点，其中 $2^{h+1}-2$ 个内结点(偶数层和奇数层各有 2^h-1 个)，$k=2^{h+1}-1$ 个叶结点(阴影结点 2^h-1 个，白色结点 2^h 个). 给定 $S_J\subseteq S$，结点 $g(p,q,r_{p,q},S_J)$ $(1\le p,q\le k)$ 对任意 $0\le r_{p,q}\le n$ 计算式(7)给出的函数 $g(p,q,r_{p,q},S_J)$，不同 $r_{p,q}$(若 p,q 的形式比较复杂，可记为 $r(p,q)$)对应不同任务，根结点 $g(1,k,r_{1,k},S_J)$ 上只有一个 $r_{p,q}=n$ 的任务. 递推树中的边表示调用关系，父结点根据式(7)将自身的每个任务分解为若干对子任务，每一对子任务分别调用两个子结点进行计算，然后将其结果相乘，所有乘积的和，即为父结点任务的计算结果.

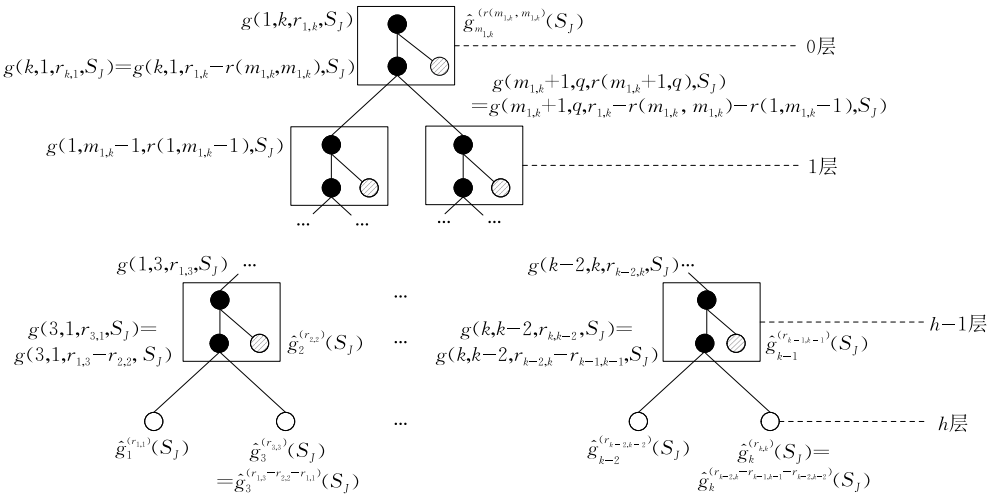


图 4 $g(1,k,r_{1,k},S_J)$ 的递推树

根据式(7)的终止条件，任何叶结点 $g(i,i,r_{i,i},S_J)$ 的值即 $\hat{g}_i^{(r_{i,i})}(S_J)$，算法第 2 步已求得，本步没有开销. 对所有内结点的计算开销，按图 4 中的矩形框分组来分析. 每个矩形框包含 $g(p,q,r_{p,q},S_J)$ $(p<q)$ 和 $g(q,p,r_{q,p},S_J)$ 两个内结点，根据式(7)，其递推分解关系如图 5 所示.

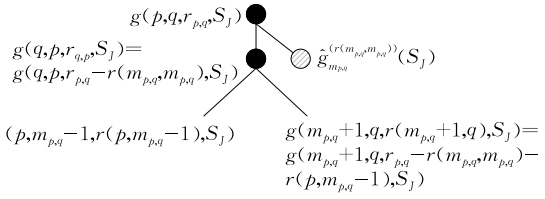


图 5 $g(p,q,r_{p,q},S_J)$ 和 $g(q,p,r_{q,p},S_J)$ 的递推分解

因 $0\le r_{p,q}\le n$，结点 $g(p,q,r_{p,q},S_J)$ 可能有 $O(n)$ 个不同任务，最大为 $r_{p,q}=n$. $0\le r(m_{p,q},m_{p,q})\le r_{p,q}\le n$，故结点 $g(q,p,r_{q,p},S_J)$ 上也可能有 $O(n)$ 个不同任务，最大为 $r_{q,p}=r_{p,q}-r(m_{p,q},m_{p,q})=n-0=n$. 由于动态规划缓存任务的计算结果，每个不同的任务即使被多次调用，也只需计算一次. 因此，该矩形框的递推计算开销(不含子结点的计算开销)包括

$$O\Big(n\cdot\sum_{r(m_{p,q},m_{p,q})=0}^{n}1+n\cdot\sum_{r(p,m_{p,q}-1)=0}^{n}1\Big)=O(n^3)$$

次乘法和等量的加法. 加数和乘数均不超过 $g(p,q,r_{p,q},S_J)$ 的最大可能值，故一次加法或乘法分别需 $O(l_{p,q}[n+\log(n+1)])$ 或 $O^*(l_{p,q}[n+\log(n+1)])$ 时间. 因此，该矩形框的递推计算总时间为

$$O^*(n^3 l_{p,q}[n+\log(n+1)])=O^*(l_{p,q}[n^4+n^3\log(n+1)]).$$

如前面分析，单个 $g(p,q,r_{p,q},S_J)$ 或 $g(q,p,r_{q,p},S_J)$ 占据 $O(l_{p,q}[n+\log(n+1)])$ 位二进制，而矩形框中有 $O(2n)$ 个不同任务，存储它们的计算结果，所需空间开销为

$$O(l_{p,q}[n^2+n\log(n+1)]).$$

在进一步分析之前，给出递推树及其基础二叉树的如下性质.

定理 6. 对递推树的任意结点 $g(p,q,r_{p,q},S_J)$，称 $l_{p,q}=|q-p|+1$ 为其宽度，从而同一矩形框中两个内结点宽度相同，称其为矩形框的宽度. 此时有

(1) 第 $0\le t\le h-1$ 层任何矩形框的宽度均为 $l_t=2^{h-t+1}-1$.

(2) 第 $0\le t\le h-1$ 层所有矩形框的宽度之和 $L_t=2^t l_t=2^{h+1}-2^t$.

证明见附录.

现在，第 $0\le t\le h-1$ 层所有 2^t 个矩形框的递推

计算时间之和为

$$O^*(2^t l_t[n^4+n^3\log(n+1)])=$$
$$O^*((2^{h+1}-2^t)[n^4+n^3\log(n+1)])=$$
$$O^*(k[n^4+n^3\log(n+1)]).$$

第 t 层所有 2^t 个矩形框的空间开销之和为 $O(2^t l_t[n^2+n\log(n+1)])=O(k[n^2+n\log(n+1)])$.

所有 $h=\log(k+1)-1$ 层矩形框的递推计算时间之和

$$O^*(k\log(k+1)[n^4+n^3\log(n+1)])$$

即为总的计算时间. 由于根结点只有 $r_{1,k}=n$ 的任务,而此结果计入了该结点 $0\leq r_{1,k}\leq n$ 所有任务的递推计算时间开销,因而偏于保守. 所有 $h=\log(k+1)-1$ 层矩形框的空间开销之和为

$$O(k\log(k+1)[n^2+n\log(n+1)]).$$

以上分析了对特定的 $S_J\subseteq S$,计算 $g(1,k,n,S_J)$所需的时间和空间开销. 因为有 2^n 个 $J\subseteq N$,计算所有的 $g(1,k,n,S_J)$需要的时间之和为

$$O^*(k\log(k+1)2^n[n^4+n^3\log(n+1)])=O^*(2^n k).$$

由于根据 $g(1,k,n,S_J)$计算 $w(P_k)$时,逐一计算每个 $g(1,k,n,S_J)$并将其加入部分和,不必事先全部计算,特别是各 $g(1,k,n,S_J)$重复利用动态规划缓存($O(k\log(k+1)[n^2+n\log(n+1)])$),而部分和只占据 $O(k\log(n+1)+n(k+1))$空间,故第 3 步的最大空间占用为 $O(k\log(n+1)+n(k+1)+k\log(k+1)[n^2+n\log(n+1)])$即 $O^*(kn^2)$,出现于计算 $g(1,k,n,S_J)$的动态规划.

综合 1～3 步的分析,整个算法的时间和空间复杂度均为 $O^*(2^n k)$,即 $O^*(2^{|S|}|U|)$. 其中最大空间占用出现在第 2 步.

7　实验研究

本节将对相关 WS 算法进行实验研究,所有算法均以 C++实现,涉及大整数运算均使用 GMP 算术库. 实验环境为 3.4GHz Core i3 CPU、16GB RAM、Red Hat Enterprise Linux 7 64 bit 虚拟机(宿主系统为 Windows 7). 由于缺少相关的标准数据集,将通过一定规则随机生成测试数据. 在所有数据中均取 $A=S\times U$,这是 S 和 U 可提供的最大负载(可能的资源分配数量达到最大).

实验 1.　WS(≠)判定性能对比

WS(≠)判定有 *WS(≠)和 ♯WS(≠)等多种途径. 本文给出了两种 ♯WS(≠)算法(定理 3 和算法 1). 文献[6]给出了目前时间复杂度最低的 *WS(≠)算法. 在此之前,有时间复杂度分析结果的 WS 确定性算法主要有文献[5,16-17],均基于穷举搜索. 不同在于文献[16]枚举所有的资源分配解,文献[5]排除了授权用户数$|U_A(s)|>|S|$的步骤,当$|U|\geq|S|$时与文献[17]等价,发现一个解则返回,比枚举所有解更快. 因此选择文献[17]的 *WS 版本作为穷举搜索算法的代表. 本实验将对 4 种算法的实际性能进行比较.

按以下规则随机生成 17 组数据,并在测试结果转折处,按同样的规则补充了 12 组数据:将$|S|$的值指定为 8～24;在 $0.25|S|$～$0.75|S|$之间随机生成$|U|$的值($|U|\geq|S|$时,WS(≠)必然成立,只需令所有步骤的执行用户均不相同);在任意两个步骤之间,以 0.15 的概率决定是否生成互斥约束(每个步骤已有各自的授权要求,而约束是根据步骤间关键联系施加的进一步访问控制,其密度通常不太高). 在所得数据上,4 种算法的执行时间和峰值空间如表 1 所示(—表示执行时间超过 2 h,×表示进程内存分配失败被杀死),比较如下.

穷举搜索算法空间开销最低,并且稳定. 当步骤数为 8～13 时,其执行时间也是最少的,均在 0.25 s 以内,主要原因是算法的初始开销极低,且输入规模足够小,能够以很高的概率快速找到一个解. 但是由于时间复杂度高,当步骤数达到 14 以后,执行时间增长很快,仅在 3 组数据上取得了优于多数算法的效率. 该算法在 2 h 时间和 16GB 内存限制下可计算的输入规模为 19 个步骤和 5 个用户,但小于此规模时,已出现时间性能急剧恶化的情况.

文献[6]的算法空间开销最大,并且随输入规模的增大而快速增长,导致其可计算的数据规模最小,为 16 个步骤和 9 个用户. 相对于穷举搜索算法,当步骤数为 14～16 时,在 4 组数据上降低了执行时间,最大降幅达到 98%或以上,开始体现出时间复杂度上的优势. 随着输入规模的增加,该算法的执行时间以一种较为稳定的方式增长,没有剧烈的随机波动.

对于定理 3 的方法,在前述限制下可计算的最大规模为 19 个步骤和 5 个用户,与穷举搜索算法相同,在小于此规模时,已经出现了空间性能恶化的情况. 在步骤数为 8～13 时,时间性能弱于穷举搜索算法,但其绝对时间差不超过 0.93 s. 在步骤数达到 14 以后,相对于穷举搜索算法,在共同可计算的 12 组数据上,有 8 组数据将执行时间降低了 58%～99.8%,仅 4 组数据执行时间增加,但绝对时间差在 45.6 s

之内,较明显体现了低阶指数时间复杂度的优势.相对于文献[6]的算法,在共同可计算的数据上,执行时间降低了61%～95%,体现了相同固定参数下线性时间算法的优势,同时空间占用也降低了25%～78%.

对于算法1,可计算的最大规模为23个步骤和7个用户,明显优于其它算法.仅按步骤数规模计算,相对于穷举搜索算法和定理3的算法提高了21%,相对于文献[6]的算法提高了44%.在时间和空间性能上,相对于穷举搜索算法,在步骤数为8～13时,时间性能虽然较弱,但绝对时间差不超过0.41 s,在步骤数为14～24时,对共同可计算的12组数据,有9组数据的执行时间降低达50%～99.9%,仅2组数据执行时间增加,但绝对时间差不超过17.1 s.相对于文献[6]的算法,在共同可计算的14组数据上,执行时间降低了81%～99%,平均为93%,空间占用降低了32%～99.6%,平均为87%,时间和空间性能均取得更大的优势.相对于定理3的方法,执行时间降低了25%～80%,平均为56%,表明本文提出的递推式收到了良好的剪枝效果.对步骤数为11～24的15组(共同可计算)数据,空间占用降低了81%～99%,对步骤数为8～10的3组数据,空间占用降低了7%～39%,这是因为输入规模较小时,初始开销占主导作用,整体差别不大.空间占用平均降低了84%,算法1采取的空间优化措施,作用非常明显.

表1 4种WS(≠)判定算法的时间/s和空间/MB代价

	$\|S\|.\|U\|.\|X\|$	穷举搜索决策		文献[6]		定理3		算法1	
		时间	空间	时间	空间	时间	空间	时间	空间
1	8.5.3	0.1E-3	12.7	0.119	20.8	0.047	15.5	0.023	12.9
2	9.2.5	0.8E-5	12.7	0.093	19.0	0.005	13.8	0.001	12.9
3	10.4.6	0.8E-4	12.7	0.394	45.9	0.073	21.4	0.029	13.0
4	11.7.5	0.233	12.7	1.62	171.2	0.291	70.0	0.219	13.6
5	12.6.13	0.082	12.7	3.42	306.6	0.466	103.9	0.323	14.2
6	13.5.8	0.002	12.7	6.91	541.4	0.926	150.7	0.39	15.1
7	14.4.14	0.025	12.7	12.4	917.0	1.26	202.0	0.431	16.8
8	14.9.14	18.190	12.7	34.1	2574.9	4.78	959.5	2.14	21.6
9	15.7.19	769.6	12.7	55.2	4079.1	7.20	1235.0	3.60	27.8
10	15.7.23	23.2	12.7	58.2	4079.0	6.65	1235.0	3.50	27.8
11	15.8.19	4310.7	12.7	81.8	4867.6	8.19	1606.7	3.81	29.9
12	15.8.23	87.1	12.7	67.7	4867.8	8.54	1606.7	3.87	29.9
13	16.6.33	32.6	12.7	123.6	7408.0	13.6	1922.0	4.61	39.7
14	16.9.24	—	12.7	239.0	12557.0	21.6	4288.3	9.45	52.9
15	17.7.12	34.7	12.7	×	×	45.3	5500.4	17.2	78.4
16	18.5.15	0.501	12.7	×	×	46.1	5949.4	17.6	111.8
17	18.7.22	1701.2	12.7	×	×	75.7	11593.7	36.7	150.9
18	18.9.32	—	12.7	×	×	×	×	43.4	190.0
19	18.11.35	—	12.7	×	×	×	×	59.3	229.0
20	19.4.32	21.4	12.7	×	×	52.8	8043.1	20.1	179.6
21	19.5.31	6014.5	12.7	×	×	103.729	12508.5	31.4	220.6
22	19.6.21	—	12.7	×	×	×	×	56.4	261.7
23	19.7.26	—	12.7	×	×	×	×	74.4	302.7
24	20.11.27	—	12.7	×	×	×	×	280.8	965.1
25	21.10.26	—	12.7	×	×	×	×	548.6	1826.2
26	22.14.33	—	12.7	×	×	×	×	1980.3	5310.3
27	23.7.26	—	12.7	×	×	×	×	1768.6	5559.5
28	24.6.19	—	12.7	×	×	×	×	—	9925.8
29	24.13.29	—	12.7	×	×	×	×	×	×

实验2. 算法1的时间复杂度.

本文给出两种♯WS(≠)算法均具有$O^*(2^{|S|}|U|)$时间复杂度.由于定理3已将其时间复杂度归结为集合划分权重和,而算法1经过了较为复杂的分析,这里仅对后者进行实验,以验证分析结果.本实验首先对算法1的执行时间仅随步骤数量的变化趋势进行实验.在0约束的情况下,取$|U|=5,8,11,14$和17,令$|S|$从1变化到21,测试执行时间的变化.取约束数量为0的原因是:

(1)根据随后实验3的结果,在$|S|$和$|U|$相同时,约束越少,负载越大,所需执行时间越长.

(2)便于步骤数量自由变化,例如$|S|=1,2,3,$

4 和 5 时，约束的数量不能超过 0，1，3，6 和 10 个，约束数量越多，步骤数量的取值范围越窄.

(3)不需要考虑约束拓扑的影响. 同样数量的约束在同样数量的步骤之间有多种分布方式，所形成的负载并不相同. 当步骤数量变化时，即使控制约束数量不变，仍然存在约束拓扑的变化. 而在 0 约束情况下，约束拓扑总是相同的.

测试结果如图 6 所示，执行时间比函数 $2^{|S|}$ 的增长略快. 因算法 1 的时间复杂度为 $O(2^{|S|}|S|^4|U|)$，实验结果与理论分析是相吻合的.

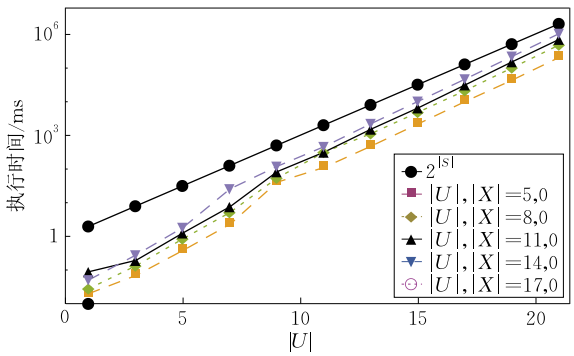


图 6　算法 1 执行时间随步骤数量的变化

接下来让用户数量变化，对算法 1 的执行时间进行测试. 取实验 1 的第 3，6，13，22 和 26 组数据，其|S|分别为 10，13，16，19 和 22，保持|S|和 X 不变，让|U|从 2 增加到 20，测试算法 1 的执行时间如何变化，结果如图 7 所示. 令|S|=22，忽略一个常数因子的差异，所有执行时间曲线均以关于|U|的线性函数 $2^{|S|}|U|$ 为上界，这与算法 1 的最坏时间复杂度 $O^*(2^{|S|}|U|)$ 是吻合的.

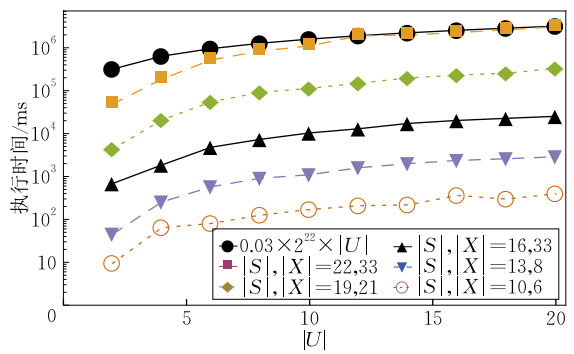


图 7　算法 1 执行时间随用户数量的变化

实验 3.　约束规模对本文算法时间性能的影响.

本实验对本文两种算法的时间性能做进一步研究，测试它们与约束规模之间的相关性. 按以下规则生成数据：将|S|的值指定为 9，12，15，18 和 21；在 0.3|S|～0.7|S|之间随机生成一个|U|的值；对每一组|S|和|U|，在任意两个步骤之间，以 0.9 的概率决定是否生成互斥约束. 对前面所得每一组数据，逐渐删除约束，测试算法执行时间如何变化，结果如图 8 所示.

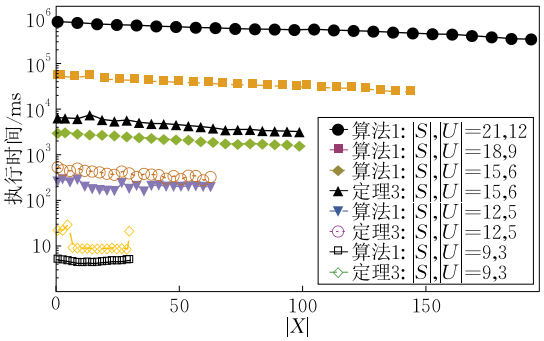


图 8　算法 1 执行时间随约束数量的变化

可以看到，当约束逐渐增加时，两个算法的执行时间均呈降低趋势. 这主要是因为约束密度增加使得权函数 $w_i(1\leq i\leq k)$ 和 $w_i^{(m)}$ 中的 0 值增加，$\hat{g}_i^{(m)}(0\leq m\leq n)$ 的数值变小，从而递推式中加法和乘法运算的速度变快，动态规划的效率提高. 对|S|为 9，12 的小规模数据，在上述趋势之外，某些点的函数值出现了较大波动，可能有以下原因.

(1) 权函数计算需要遍历约束集，随约束规模增长耗时增加. 这部分计算的时间复杂度为 $O(2^{|S|}|S|^2)$，而动态规划部分的时间复杂度至少为 $O(2^{|S|}|S|^3|U|)$，这两部分开销随约束密度增长的变化趋势相反. 尽管后者对整体执行时间起决定作用，然而当|S|较小时，前者的影响也相对明显，因而造成了一定的波动.

(2) 在对数纵坐标下，当函数值较小时，其波动也更为明显.

8　相关工作

8.1　工作流访问控制

从 Thomas 等人[3]提出基于任务授权的概念开始，工作流访问控制研究已有超过 20 年的历史. 有关研究主要集中在授权机制设计[3,18-20]、约束表达能力与描述语言[18,22]、约束冲突与冗余分析[16,23]等方面，较少考虑访问控制对资源分配的影响. Bertino 的模型最早涉及 WS 相关问题，后经更多研究者工作的推动，近来 Crampton 在 *WS 问题上取得了显著的成果，下面择要进行介绍.

1999 年，Bertino 等人[16]提出了一种基于规则的约束描述语言，包含来自用户、步骤和角色等集合

的常/变量,以及反映它们之间关系的多种谓词,可以描述规划、执行阶段分别检测的静态、动态约束以及需要两个阶段共同检测的混合约束.在该语言中,通常的授权和约束都可以表达为规则.利用形如 $user(u,s)\leftarrow$ 的规则可以描述授权;$cannot\_do_u(u,s')\leftarrow must\_execute_u(u,s)$ 可以描述互斥约束;$must\_execute_u(u,s)\leftarrow$ 可以描述资源分配.将 WS($\neq$)相应的约束和授权表达为一组规则之后,所谓用户规划过程将生成与规则基一致的所有资源分配,并利用启发式规则优化其排序.根据其规则一致性标准,这些资源分配就是 WS($\neq$)的所有解.用户规划对 $O((|R|\cdot|U_{\max}(R)|\cdot|N_{\text{act}}(S)|)^{|S|})$ 种可能进行一致性验证,本质上是一个穷举搜索过程,其中 R 是角色集,$U_{\max}(R)$ 是单个角色的最大用户数,$N_{\text{act}}(S)$ 是单个步骤在工作流结构中的出现数.通常用户有一到多种角色,$|R|\cdot|U_{\max}(R)|\geq|U|$,而每个步骤只有一次出现(不同出现看作不同步骤),$|N_{\text{act}}(S)|=1$.因此,用他们的算法求 WS($\neq$)的所有解,进而统计其个数,所需时间不低于 $O(|U|^{|S|})$,高于本文的结果.

2003 年,Atluri 等人[24]将上述方法用于工作流委托授权中的可满足性问题.他们增加了 delegate 和 volunteer 两组谓词,将委托和受托关系也表达为规则.通过角色和用户规划,生成与规则基一致的所有资源分配.然而,他们只是扩展了 Bertino 方法的应用范围,并未改进规划算法及其性能.

2008 年,Crampton 等人[17]在讨论委托对资源分配的影响时,给出了工作流可满足性的判定算法.该?WS 算法穷举搜索整个解空间,对于只考虑互斥约束的情形,时间为 $O(|U|^{|S|}|S|^2)$,高于本文结果.

2010 年,Wang 等人[5]证明只考虑职责分离的 $*$WS 可以在 $O(|S|^{|S|+1}|C|)$ 时间内解决.由于对授权用户数超过 $|S|$ 的步骤,总可以指派恰当的用户避免与其它步骤冲突,他们只对剩余的步骤进行穷举求解.如果只考虑互斥约束,其时间复杂度为 $O(|S|^{|S|}|C|)$,高于本文的结果.值得注意的是,Wang 等人进一步讨论了工作流 k-弹性问题,也就是删除任意 k 个用户后的工作流可满足性.与本文提出的 $\sharp$WS 相比,k-弹性可以更精确地衡量工作流在资源失效情况下的鲁棒性.他们证明此问题是 NP 难的,属于 coNP^{NP}(其补问题可以为带喻示 NP 的非确定性图灵机所判定),但未进行求解.

2011 年,Basin 等人[26]在支持循环和选择路由的工作流中讨论了考虑互斥和绑定约束的可满足性决策问题,并给出了多项式时间的近似算法.绑定约束要求两个步骤必须由同一用户完成,考虑这一约束并不复杂,并且有利于降低问题规模.他们的算法除了缺乏精确性,还依赖于授权用户在步骤之间的均匀分布.

2013 年,Crampton 等人[6]对第 4 节所述的 $*$WS($\neq$)问题进行了研究,这是目前与本文最为相关的工作.他们通过判定当每个步骤子集由每个用户执行时是否导致违反约束的情况,将 $*$WS($\neq$)归约为集合最大权划分问题.归约过程需要 $O^*(2^{|S|}(|C|+|U|))$ 时间,而集合最大权划分可以利用 $O^*(2^{|S|}p(|U|))$ 空间,在 $O^*(2^{|S|}|U|^2)$ 时间内解决[12],整体所需时间为 $O^*(2^{|S|}(|C|+|U|^2))$.他们进一步证明,根据颇有影响的指数时间假设[26],该时间的指数阶很难获得明显改进[6].这是迄今关于 $*$WS($\neq$)的最好理论结果.相比他们的工作,通过计数途径进行 WS($\neq$)判定,取得了更好的时间复杂度.本文的 $\sharp$WS($\neq$)和 Crampton 的 $*$WS($\neq$)求解方法均需 $O^*(2^{|S|}p(|U|))$ 空间,这是它们的主要缺陷.Crampton 的方法依赖于 Bjorclund 有关集合最大权划分的结果,只给出了归约方法.本文不仅基于 Bjorclund 对集合划分权重和的结果确定了 $\sharp$WS($\neq$)的时间复杂度,而且通过挖掘问题特征,在集合划分权重和求解框架中提出了新的动态规划递推式,在保持时间复杂度的前提下明显优化了实际执行时间,并通过优化空间利用方式,大幅度降低了动态规划和快速变换部分的空间开销.Crampton 使用的归约方法不限制约束类型,且在一定条件下保持时间复杂度,但他们以此为基础,为更多约束类型设计了有针对性的算法,取得了丰富的结果.本文的 $\sharp$WS 算法同样可对适度推广的约束类型保持现有时间复杂度.不过,针对多种约束情形的优化,还有待继续研究.

8.2 工作流资源分配

资源分配为工作流各步骤指派执行资源,若进一步给出对执行起/止时间的计划,则称为资源调度.随着网格和云工作流的兴起,步骤的候选资源主要由自动化服务来提供,资源分配问题也称为服务组合.不论以何种形式提出,资源分配始终是工作流管理的核心主题之一.

相关研究主要关注成本约束下的性能目标,据此给出优化的资源分配策略,或者给定策略进行性能与成本的分析.成本约束将资源分配映射为某种

业务成本，例如工资、处理时间等等，并对其加以限制. 而性能是面向应用或系统的优化目标. 应用性能为用户所关注，在面向服务体系中也称为服务质量(Quality of Service，QoS)，例如平均或总处理时间、处理时间波动性、执行成功率等. 系统性能聚焦于内部机制的合理性，例如资源利用率、系统吞吐量、能量耗费等. 成本约束和性能目标统称为分配准则，两者并无绝对的分别. 不同的成本和性能指标选择与组合方式，形成了不同的资源分配需求. 分配准则从需求角度提出，而体系结构可能对其计算方式产生影响. 例如在云环境中，服务价格计费可以基于虚拟机实例数量、虚拟机执行时间单元数或一定的计费周期，步骤执行、数据传输、数据存储的计费方式及其对工作流单次执行价格的影响程度也不相同，并且公有云、私有云和混合云有着不同的计费模型[27].

在应用性能的优化方面，例如 Zeng 等人[28]对状态图表示的工作流，将处理时间、价格、可用性和声誉等 QoS 指标加权归一作为优化目标，并建立关于截至期限、价格、声誉、执行成功率和可用性等指标的全局性约束，基于混合整数规划方法进行求解，不但为工作流每个步骤选定执行服务，而且可以求出其预期开始时间；Jia 等人[29]对有向无环图(Directed Acyclic Graph，DAG)表示的工作流，以截止时间约束下执行成本为优化目标进行资源分配，方法是通过步骤聚合简化过程结构，并将截至期限约束分解到各聚合步骤，然后在聚合步骤上进行 Markov 决策，求出其中每个步骤的执行服务及其时间槽；于炯等人[30]以 DAG 过程结构为基础，对关键和非关键路径总执行时间的差异情形进行分类，据此设计资源分配算法，平衡了案例执行成功率与算法自身的效率；Alrifai 等人[31]改进了 Zeng 的方法，将每个步骤的候选服务按其 QoS 值划分为若干等级，利用混合整数规划将全局性约束分解到每个步骤，然后在不违反局部约束的前提下根据局部 QoS 进行服务选择，可以快速找到全局次优解；Abrishami 等人[32]对云环境中的 DAG 表示的科学工作流，以截至期限约束下的价格最小为目标进行优化资源分配，主要方法是通过关键路径分析将截至期限分解到每个步骤，在此前提下为每个步骤选择价格最低的服务，其特点是考虑了服务切换引起的数据传递延迟以及云服务计费按一定时间单位取整造成的影响；等等. 在系统性能的优化方面，例如 Bittencourt 等人[33]对共享同一组资源的多工作流调度问题进行研究，实现了最小完成时间下的最大公平性；田国忠等人[34]研究了多个具有截止期限约束的工作流共享资源调度的问题，在最大化多工作流吞吐量的同时，尽可能降低了执行费用；等等. 在给定策略的性能与成本分析方面，例如 Eder 等人[35]通过对路由选择可能性的分析得出案例可能执行路径的分布，并根据步骤的执行时间计算总体执行时间的分布；刘胜等人[36]对基于 4 种模式组合的过程结构，在案例到达间隔和资源服务时间服从负指数分布的情况下，对先来先服务的资源调度策略进行性能分析，得出了案例执行时间的分布；等等. 这些研究所考虑的分配准则首先是时序相关的，例如总执行时间或截止期限是占据支配地位的指标，通常还需要根据价格、可用性、执行成功率、声誉等其它指标进行整体优化. 在不同的研究中，这些指标可能被建模为柔性的优化目标，或者刚性的约束[27]. 然而，作为一种具有刚性特点的资源分配准则，安全因素所受到的重视还很不够[27,37]. 现有少量研究涉及域间信任，例如文献[38]定义了处理不可移动数据的不可移动步骤，要求其只能在特定数据中心边界内执行，文献[39]允许有一定安全级要求的步骤在公有云上执行，或者消息传递安全性，例如文献[40]研究了服务组合的消息签名与加密完整性，等等. 但是，未见考虑执行资源访问控制约束的工作.

9　结论与未来工作

WS 是工作流访问控制的基本问题，现有工作主要局限于其决策问题，本文提出了其计数问题，分析了该问题的研究意义和应用价值. 对于只考虑互斥约束的 WS 计数问题，即 ♯WS(≠)，本文基于集合划分权重和的结果证明其存在 $O^*(2^{|S|}|U|)$ 时间的算法，表明此问题固定参数可解，对任意固定的 $|S|$，可在关于 $|U|$ 的线性时间内解决. 随后，本文在基于赋权集容斥原理与快速 ζ 变换的集合划分权重和求解方法中提出了一种有利于快速剪枝的动态规划递推式，并对此方法的空间利用情况进行系统优化，由此提出另一种 $O^*(2^{|S|}|U|)$ 时间的 ♯WS(≠)算法. 实验表明，相对于改进前，执行时间平均降低了 56%，空间占用平均降低了 84%，而可求解的问题规模提高了 21%. 由于对应的决策问题，即 ∗WS(≠)，时间复杂度为 $O^*(2^{|S|}(|C|+|U|^2))$，所以本文算法也降低了 WS(≠)判定的时间复杂度. 实验表明，相对于该 ∗WS(≠)算法，本文算法的执行时间平均降

低了 93%,空间占用平均降低了 87%,而可求解的问题规模提高了 44%.除为 WS(≠)判定提供了当前时间复杂度最低的算法,本文的 #WS(≠)算法可为工作流抗资源失效鲁棒性的验证提供新的手段.作为下一步工作,我们将对更多约束类型的 #WS 进行研究.

致 谢 感谢匿名评审人对初稿提出了严格而富有帮助的意见,使本文在研究程度上得以深化,组织表达也更为清晰!

参 考 文 献

[1] Aalst W, Hee K. Workflow Management: Models, Methods and Systems. 2nd Edition. London, UK: MIT Press, 2004

[2] Sandhu R, Coyne E J, Feinstein H L, et al. Role-based access control models. IEEE Computer, 1996, 29(2): 38-47

[3] Thomas R K, Sandhu R S. Towards a task-based paradigm for flexible and adaptable access control in distributed applications//Proceedings of the 2nd New Security Paradigms Workshop. Rhode Island, USA, 1993: 138-142

[4] Clark D D, Wilson D R. A comparison of commercial and military computer security policies//Proceedings of the 8th IEEE Symposium on Security and Privacy. Oakland, USA, 1987: 184-194

[5] Wang Q, Li N. Satisfiability and resiliency in workflow authorization systems. ACM Transactions on Information and System Security, 2010, 13(4): 4.1-4.31

[6] Crampton J, Gutin G, Yeo A. On the parameterized complexity and kernelization of the workflow satisfiability problem. ACM Transactions on Information and System Security, 2013, 16(1): 40.1-40.31

[7] Valiant L G. The complexity of computing the permanent. Theoretical Computer Science, 1979, 8(2): 189-201

[8] Gu Wen-Xiang, Zhu Lei, Huang Ping, Yin Ming-Hao. The model counting of a satisfiability problem. CAAI Transactions on Intelligent Systems, 2012, 7(1): 33-39(in Chinese)
(谷文祥, 朱磊, 黄平, 殷明浩. 可满足问题中的模型计数. 智能系统学报, 2012, 7(1): 33-39)

[9] Ahmed T, Tripathi A R. Security policies in distributed CSCW and workflow systems. IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans, 2010, 40(6): 1220-1231

[10] Downey R, Fellows M. Parameterized Complexity. Berlin, Germany: Springer, 1999

[11] Björclund A, Husfeldt T, Koivisto M. Set partitioning via inclusion-exclusion. SIAM Journal on Computing, 2009, 39(2): 546-563

[12] Li Shao-Hua, Wang Jian-Xin, Feng Qi-Long, Chen Jian-Er. Kernelization techniques and its applications to parameterized computation. Journal of Software, 2009, 20(9): 2307-2319 (in Chinese)
(李绍华, 王建新, 冯启龙, 陈建二. 参数计算中核心化技术及其应用. 软件学报, 2009, 20(9): 2307-2319)

[13] Woeginger G. Exact algorithms for NP-hard problems: A survey. Combinatorial Optimization, 2003, 2570(1): 185-207

[14] Kennes R. Computational aspects of the mobius transformation of graphs. IEEE Transactions on Systems, Man and Cybernetics, 1992, 22(2): 201-223

[15] Schönhage A, Strassen V. Schnelle multiplikation grosser zahlen. Computing, 1971, 7(3-4): 281-292

[16] Bertino E, Ferrari E, Atluri V. The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information System Security, 1999, 2(1): 65-104

[17] Crampton J, Khambhammettu H. Delegation and satisfiability in workflow systems//Proceedings of the 13th ACM Symposium on Access Control Models and Technologies. Estes Park, USA, 2008: 31-40

[18] Liu D, Wu M, Lee S. Role-based authorizations for workflow systems in support of task-based separation of duty. Journal of Systems and Software, 2004, 73(3): 375-387

[19] Oh S, Park S. Task-role-based access control model. Information Systems, 2003, 28(6): 533-562

[20] Zhai Z, Lu Y, Zhang P, et al. Association-based active access control models with balanced scalability and flexibility. Computers in Industry, 2014, 65(1): 116-123

[21] Bertino E, Ferrari E, Atluri V. The specification and enforcement of authorization constraints in workflow management systems. ACM Transactions on Information System Security, 1999, 2(1): 65-104

[22] Botha R A, Eloff J H P. Separation of duties for access control enforcement in workflow environments. IBM Systems Journal, 2001, 40(3): 666-682

[23] Zhai Zhi-Nian, Lu Ya-Hui, Xi Jian-Qing, et al. Enterprise-level business process oriented framework for separation of duties with its redundancy analysis. Acta Electronica Sinica, 2013, 41(10): 2087-2093(in Chinese)
(翟治年, 卢亚辉, 奚建清等. 面向企业级流程的职责分离框架及其冗余分析. 电子学报, 2013, 41(10): 2087-2093)

[24] Atluri V, Bertino E, Ferrari E, et al. Supporting delegation in secure workflow management systems//Proceedings of the 17th Annual IFIP WG11.3 Conference on Data and Application of Security. Estes Park, USA, 2003: 190-202

[25] Basin D, Burri S J, Karjoth G. Obstruction-free authorization enforcement: aligning security and business objectives//Proceedings of the 24th Computer Security Foundations Symposium. Cernay-la-Ville, France, 2011: 99-113

[26] Impagliazzo R, Paturi R, Zane F. Which problems have strongly exponential complexity? Journal of Computer and System Sciences, 2001, 63(4): 512-530

[27] Smanchat S, Viriyapant K. Taxonomies of workflow scheduling problem and techniques in the cloud. Future Generation Computer Systems, 2015, 52: 1-12

[28] Zeng L Z, Benatallah B, Ngu A, et al. QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering, 2004, 30(5): 311-327

[29] Jia Y, Buyya R, Chen K T. Cost-based scheduling of scientific workflow applications on utility grids//Proceedings of the 1st International Conference on e-Science and Grid Computing. Melbourne, Australia, 2005: 147-154

[30] Yu Jiong, Tian Guo-Zhong, Cao Yuan-Da, Sun Xian-He. A resource allocating algorithm in grid workflow based on critical regions reliability. Journal of Computer Research and Development, 2009, 46(11): 1821-1829(in Chinese)
(于炯,田国忠,曹元大,孙贤和. 基于关键区间可靠度的网格工作流资源分配算法. 计算机研究与发展, 2009, 46(11): 1821-1829)

[31] Alrifai M, Risse T, Nejdl W. A hybrid approach for efficient web service composition with end-to-end QoS constraints. ACM Transactions on the Web, 2012, 6(2): 7.1-7.31

[32] Abrishami S, Naghibzadeh M, Epema D H. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Generation Computer Systems, 2013, 29(1): 158-169

[33] Bittencourt L F, Madeira E R. Towards the scheduling of multiple workflows on computational grids. Journal of Grid Computing, 2010, 8(3): 419-441

[34] Tian Guo-Zhong, Xiao Chuang-Bai, Xie Jun-Qi. Scheduling and fair cos-optimizing methods for concurrent multiple DAGs with deadline share resources. Chinese Journal of Computers, 2014, 37(7): 1607-1619(in Chinese)
(田国忠, 肖创柏, 谢军奇. 有期限约束的多 DAG 共享资源的调度及公平费用优化方法. 计算机学报, 2014, 37(7): 1607-1619)

[35] Eder J, Pichler H, Vielgut S. Avoidance of deadline-violations for inter-organizational business processes//Proceedings of the 7th International Baltic Conference on Databases and Information Systems. Paris, France, 2006: 33-40

[36] Liu Sheng, Fan Yu-Shun. Method for analyzing staying-time of instances in workflow models with resources constraints. Acta Electronica Sinica, 2005, 31(10): 141-145(in Chinese)
(刘胜, 范玉顺. 资源约束下实例在工作流中停留时间分析方法. 电子学报, 2005, 31(10): 141-145)

[37] Jula A, Sundararajan E, Othman Z. Cloud computing service composition: A systematic literature review. Expert Systems with Applications, 2014, 41(8): 3809-3824

[38] Zeng L, Veeravallia B, Li X. SABA: A security-aware and budget-aware workflow scheduling strategy in clouds. Journal of Parallel and Distributed Computing, 2015, 75: 141-151

[39] Watson P. A multi-level security model for partitioning workflows over federated clouds. Journal of Cloud Computing, 2012, (1): 1-15

[40] Satoh F, Tokuda T. Security policy composition for composite web services. IEEE Transactions on Services Computing, 2011, 4(4): 314-327

附　录.

定理 5 证明.　(1) $p\leq q$ 时,由于 $g(p,q,r_{p,q},S_J)$ 表示 S_J 中各分量 $w_i^{(m)}$ 权值均为 1 的 $q-p+1$ 元组的数量,$S_J=S$ 取最大值,故 $g(p,q,r_{p,q},S_J)\leq g(p,q,r_{p,q},S)$. 由式(2),

$$g(p,q,r_{p,q},S)=\sum_{m_p+m_{p+1}+\cdots+m_q=ri=p}\prod_{i=p}^{q}\hat{g}_i^{(m_i)}(S)(0\leq m_i\leq r_{p,q}),$$

故满足 $\sum_{i=p}^{q}m_i=r_{p,q}$ 的基数列 m_p, m_{p+1}, …, m_q 不超过 $(r_{p,q}+1)^{q-p+1}$ 个. 对其中每个基数列,由于 $\hat{g}_i^{(m_i)}(S)\leq 2^n$, 故 $\prod_{i=p}^{q}\hat{g}_i^{(m_i)}(S)\leq(2^n)^{q-p+1}$. 于是

$$g(p,q,r_{p,q},S)\leq[(r_{p,q}+1)2^n]^{q-p+1}\leq[(n+1)2^n]^{q-p+1}.$$

综合可知式(8)成立.

(2) $p>q$ 时,根据式(7)有(注意 p,q 的大小关系)

$$\begin{aligned}g(q,p,r_{q,p},S_J)&=\sum_{r(m_{p,q},m_{p,q})=0}^{r_{q,p}}\{\hat{g}_{m_{p,q}}^{(r(m_{p,q},m_{p,q}))}(S_J)\cdot\\&\quad g(p,q,r_{q,p}-r(m_{p,q},m_{p,q}),S_J)\}\\&=\sum_{r_{p,q}=0}^{r_{q,p}}\{\hat{g}_{m_{p,q}}^{(r_{q,p}-r_{p,q})}(S_J)\cdot g(p,q,r_{p,q},S_J)\}.\end{aligned}$$

由于 $r_{p,q}=r_{q,p}$ 时

$$\hat{g}_{m_{p,q}}^{(r_{q,p}-r_{p,q})}(S_J)=\hat{g}_{m_{p,q}}^{(0)}(S_J)=\sum_{S'\subseteq S_J}w_{m_{p,q}}^{(0)}(S')=1.$$

而 $r_{p,q}<r_{q,p}$ 时任何 $\hat{g}_{m_{p,q}}^{(r_{q,p}-r_{p,q})}(S_J)\cdot g(p,q,r_{p,q},S_J)$ 非负,故必有

$$g(p,q,r_{q,p},S_J)\leq g(q,p,r_{q,p},S_J).$$

由于 $0\leq r_{q,p}\leq n$,与 $0\leq r_{p,q}\leq n$ 取值范围相同,简单地进行变量重命名即得

$$g(p,q,r_{p,q},S_J)\leq g(q,p,r_{p,q},S_J).$$

由于 $q<p$,根据(1)有

$$\begin{aligned}g(q,p,r_{p,q},S_J)&\leq g(q,p,r_{p,q},S)\leq\\&[(r_{p,q}+1)2^n]^{|q-p|+1}\leq[(n+1)2^n]^{|q-p|+1}.\end{aligned}$$

综合可知式(8)也成立.　　证毕.

定理 6 证明.　(1) 对矩形框层数 $0\leq t\leq h-1$ 做归纳.

基始. $t=0$ 时只有一个矩形框,包含内结点 $g(1,k,r_{1,k},S_J)$ 和 $g(k,1,r_{k,1},S_J)$,其宽度为 $k=2^{h+1}-1$,命题成立.

归纳步骤. 设层数为 $0\leq t<h-1$ 时结论成立,即 $l_t=2^{h-t+1}-1$,证明层数为 $t+1$ 时也成立. 从第 t 层任取一个矩形框,设其所包含内结点为 $g(p,q,r_{p,q},S_J)(p<q)$ 和 $g(q,p,r_{q,p},S_J)$,取 $g(p,q,r_{p,q},S_J)$ 为其代表结点. 该矩形框在第 $t+1$ 层有两个子矩形框,由式(7)知,其代表结点分别为

$g(p,m_{p,q}-1,r(p,m_{p,q}-1),S_J)$, $g(m_{p,q}+1,q,r(m_{p,q}+1,q),S_J)$(如图 5 所示). 由于

$$m_{p,q}=\lfloor(p+q)/2\rfloor=\lfloor p+|p-q|/2\rfloor=\lfloor p+l_t/2\rfloor$$
$$=p+\lfloor l_t/2\rfloor=p+\lfloor(2^{h-t+1}-1)/2\rfloor=p+2^{h-t}-1.$$

第 1 个子矩形框的宽度为

$$|(m_{p,q}-1)-p|+1=|2^{h-t}-1|+1=2^{h-t}=2^{h-(t+1)+1}.$$

类似可证第 2 个子矩形框具有同样的宽度. 由于第 $t+1$ 层任一矩形框都是第 t 层某个矩形框的子矩形框,故第 $t+1$ 层任何矩形框的宽度均为 $2^{h-(t+1)+1}$,即 $l_{t+1}=2^{h-(t+1)+1}-1$,层数为 $t+1$ 时结论成立.

综合基始与归纳步骤,命题得证.

(2) 由于第 $0\leq t\leq h-1$ 层有 2^t 个矩形框,由(1)的结论易证本命题成立. 证毕.

ZHAI Zhi-Nian, born in 1977, Ph.D., lecturer. His research interests include access control, service composition and workflow scheduling.

LU Ya-Hui, born in 1976, Ph. D., associate professor. His research interests include workflow management, information systems security, Petri net and Pi calculus.

ZHOU Wu-Jie, born in 1983, Ph. D., lecturer. His research interests focus on information security and image processing.

CHEN Zhi-Hao, born in 1973, Ph. D. His research interests focus on software process.

WANG Zhong-Peng, born in 1966, Ph. D., associate professor. His research interests focus on wireless communication.

LIN Jiang, born in 1963, Ph.D., professor. Her research interests focus on numerical calculation.

Background

In the issue of Workflow Satisfiability, the consistency between authorizations, constraints and resource allocation is analyzed. The issue is mainly concerned in the field of workflow access control and forms a base for the feasibility of workflows. Since 1999, many WS algorithms have been proposed which address many types of constraints. Separation of Duty (SoD) has been proved the most essential one of these constraint types, and the corresponding WS is noted as WS($\neq$). In 2013, Crampton studied WS systematically and proposed a WS($\neq$) algorithm with $O^*(2^{|S|}(|C|+|U|^2))$ (S, C, U denote the sets of steps, constraints and users respectively) time, which achieves the best theoretical result so far. However, most existing researches focus on the WS decision which finds any one WS solution to determine whether or not a workflow is satisfiable.

In this paper, the issue of counting WS is exploited to determine WS, in which the number of all WS solutions is counted. We propose a counting WS($\neq$) algorithm with the time complexity $O^*(2^{|S|}|U|)$ which exceeds the result of Crampton. Further, counting WS has more applications to verify the robustness of a workflow when some users are out of service.

The work and its early preparation have been partly supported by the National Natural Science Foundation of China (Nos. 61502429, 51376162, and 61302112).

In the field of workflow access control, we had ever modeled some interesting and useful features, such as the simplification of authorization, the analysis of constraint redundancy, the Petri-net representation of SoD constraints, etc.