Journal of Artificial Intelligence Research 51 (2014) 555-577 Submitted 05/14; published 11/14

Iterative Plan Construction for the Workflow Satisfiability Problem

David Cohen D.COHEN@RHUL.AC.UK
Jason Crampton JASON.CRAMPTON@RHUL.AC.UK
Andrei Gagarin ANDREI.GAGARIN@RHUL.AC.UK
Gregory Gutin G.GUTIN@RHUL.AC.UK
Mark Jones MARK.JONES@RHUL.AC.UK
Royal Holloway, University of London, UK

Abstract

The *Workflow Satisfiability Problem (WSP)* is a problem of practical interest that arises whenever tasks need to be performed by authorized users, subject to constraints defined by business rules. We are required to decide whether there exists a *plan* – an assignment of tasks to authorized users – such that all constraints are satisfied. It is natural to see the WSP as a subclass of the *Constraint Satisfaction Problem (CSP)* in which the variables are tasks and the domain is the set of users. What makes the WSP distinctive is that the number of tasks is usually very small compared to the number of users, so it is appropriate to ask for which constraint languages the WSP is fixed-parameter tractable (FPT), parameterized by the number of tasks.

This novel approach to the WSP, using techniques from CSP, has enabled us to design a generic algorithm which is FPT for several families of workflow constraints considered in the literature. Furthermore, we prove that the union of FPT languages remains FPT if they satisfy a simple compatibility condition. Lastly, we identify a new FPT constraint language, *user-independent* constraints, that includes many of the constraints of interest in business processing systems. We demonstrate that our generic algorithm has provably optimal running time $O^*(2^{k \log k})$, for this language, where k is the number of tasks.

1. Introduction

A *workflow* formalises a business process. It is a collection of interrelated *tasks* that are performed by *users* in order to achieve some objective. In many situations, we wish to restrict the users that can perform certain tasks. In particular, we may wish to specify lists of users who are authorized to perform each of the workflow tasks. Additionally, we may wish – either because of the particular requirements of the business logic or security requirements – to prevent certain combinations of users from performing particular combinations of tasks (Crampton, 2005). Such constraints include *separation-of-duty* (also known as the "two-man" rule), which may be used to prevent sensitive combinations of tasks being performed by a single user, and *binding-of-duty*, which requires that a particular combination of tasks is executed by the same user. The use of constraints in workflow management systems to enforce security policies has been studied extensively in the last fifteen years; for example see the work of Bertino, Ferrari, and Atluri (1999), Crampton (2005) or Wang and Li (2010).

1.1 The Workflow Satisfiability Problem

It is possible that the combination of constraints and authorization lists is "unsatisfiable", in the sense that there does not exist an assignment of users to tasks such that all contraints are satisfied and every task is performed by an authorized user; perhaps the minimal example being a requirement that two tasks are performed by the same user but the intersection of the authorization lists for these tasks is empty. A plan that satisfies all constraints and allocates an authorized user to each task is said to be "valid". The *Workflow Satisfiability Problem (WSP)* takes a workflow specification as input and returns a valid plan if one exists and a null value otherwise. It is important to determine whether a business process is satisfiable or not, since an unsatisfiable one can never be completed without violating the security policy encoded by the constraints and authorization lists. Wang and Li (2010) have shown, by a reduction from GRAPH COLORING, that the WSP is an NP-hard subclass of the CSP, even when we only consider binary separation-of-duty constraints. Nevertheless, for practical applications of the WSP, we require a solving algorithm that is as efficient as possible (Crampton & Gutin, 2013, §2.2).

Many hard problems become less complex if some natural parameter of the instance is bounded. Hence, we say a problem with input size n and parameter k is *fixed-parameter tractable* (FPT) if it admits an algorithm with running time $O(f(k)n^d)$, where d is a constant independent of n and k, and f is a computable function depending only on k.[1]

Wang and Li (2010) were the first to observe that fixed-parameter algorithmics is an appropriate way to study the WSP, because the number of tasks is usually small and often much smaller than the number of users. (The literature does not directly support this assumption, although a widely-cited study, Schaad, Moffett, & Jacob, 2001, found that the number of users exceeds that of job functions, or roles, by a multiplicative factor of around 25; this finding has been confirmed by a recent follow-up study, Jayaraman, Ganesh, Tripunitara, Rinard, & Chapin, 2011. A workflow specification will usually be concerned with a particular business objective and involve a small number of roles. Taking roles as a proxy for tasks, it seems reasonable to assume that the number of users will be an order of magnitude greater than the number of tasks.) We believe, therefore, that it is appropriate to extend the work initiated by Wang and Li on the use of fixed parameter algorithms for solving the WSP parameterized by the number of tasks, and, in particular, to ask which constraint languages are fixed parameter tractable.

Wang and Li (2010) proved that, in general, the WSP is W[1]-hard and thus is highly unlikely to admit a fixed-parameter algorithm. They also showed that the WSP is FPT if we consider only separation-of-duty and binding-of-duty constraints. Crampton, Gutin, and Yeo (2013) obtained significantly faster fixed-parameter algorithms that were applicable to "regular" constraints, thereby including the cases shown to be FPT by Wang and Li. This work, and other recent research, has demonstrated the existence of fixed-parameter algorithms for the WSP in the presence of other constraint types (Crampton, Crowston, Gutin, Jones, & Ramanujan, 2013; Crampton & Gutin, 2013). We define the WSP formally and introduce a number of different constraint types, including regular constraints, in Section 2.

We will use the O^* notation, which suppresses polynomial factors. That is, $g(n, k, m) = O^*(h(n, k, m))$ if there exists a polynomial $q(n, k, m)$ such that $g(n, k, m) = O(q(n, k, m)h(n, k, m))$. In particular, an FPT algorithm is one that runs in time $O^*(f(k))$ for some computable function f depending only on k.

1. An introduction to fixed-parameter algorithms and complexity can be found in, for example, the books by Downey and Fellows (2013), or Niedermeier (Niedermeier, 2006).

1.2 The Relation Between the WSP and the CSP

The *Constraint Satisfaction Problem (CSP)* is a general paradigm for expressing, in a declarative format, problems where variables are to be assigned values from some domain. The assignments are constrained by restricting the allowed simultaneous assignments to some sets of variables. This model is useful in many application areas including planning, scheduling, frequency assignment and circuit verification (Rossi, van Beek, & Walsh, 2006). The CSP community is a well-established research community dedicated to finding effective solution techniques for the CSP (Dechter, 2003).

The CSP is NP-hard, even when only binary not-equals constraints are allowed and the domain has three elements, as we can reduce GRAPH 3-COLORING to the CSP. [2] Hence, a considerable effort has been made to understand the effect of restricting the type of allowed constraints. Recently there has been significant progress towards the completion of this research program and there is now strong evidence to support the algebraic dichotomy conjecture of Bulatov, Jeavons and Krokhin (2005), characterising precisely which kinds of constraint language lead to polynomial solvability.

It is worth noting that the WSP is a subclass of the CSP where for each variable s (called a task in WSP terminology) we have an arbitrary unary constraint (called an authorization) that assigns possible values (called users) for s; this is called the conservative CSP. Note, however, that while usually in CSP the number of variables is much larger than the number of values, for the WSP the number of tasks is usually much smaller than the number of users. It is important to remember that for the WSP we do not use the term 'constraint' for authorizations and so when we define special types of constraints, we do not extend these types to authorizations, which remain arbitrary.

1.3 Outline of the Paper

Our novel approach to the WSP using techniques for the CSP, characterising types of constraints as constraint languages with particular characteristics, enables us to generalise and unify existing algorithms. So, in this paper, for the first time, rather than considering algorithms for specific constraints, we design a generic algorithm which is a fixed-parameter algorithm for several families of workflow constraints considered in the literature. In particular we introduce the notions of *user-independent constraints*, which subsume a number of well-studied constraint types from the WSP literature, including the regular constraints studied by Crampton et al. (2013).

Our generic algorithm builds plans incrementally, discarding partial plans that can never satisfy the constraints. It is based on a naive algorithm, presented in Section 2.2. This naive algorithm stores far more information than is required to solve the WSP, so its running time is no better than exhaustively searching for a valid plan.

Our generic algorithm uses a general and classic paradigm: retain as little information as possible in every step of the algorithm. This paradigm is used in such classical polynomial-time algorithms as Gaussian elimination for solving systems of linear equations and constraint propagation algorithms (used, for example, to solve 2SAT in polynomial time). Our generic algorithm uses this paradigm in a problem-specific way, based on the concepts of extension-equivalence, plan-indistinguishability and patterns, enabling us to retain a single pattern for each equivalence class of indistinguishable plans. Extension-equivalence and plan encodings are described in Section 3. The way the solution is constructed by our algorithm is quite unusual because the accumulation of the

2. Wang and Li's NP-hardness result for the WSP is thus a restatement of this well-known result for CSP.

(representatives of) set of solutions goes along the users (i.e., values for CSP), not along the tasks (i.e., variables for CSP).

To analyze the running time of our algorithm we introduce the notion of diversity (see Definition 6). This notion is reminiscent of pathwidth (measures are taken over all prefixes and the largest outcome is the diversity) with the difference that the diversity is based on the number of equivalence classes, hiding the actual structure behind the scenes. This approach might also be useful for structural analysis of hypergraphs.

In Section 4, we describe our pattern-based algorithm and demonstrate that it is a fixed-parameter algorithm for the WSP with user-independent constraints. We show the running time of our algorithm is $O^*(2^{k \log k})$ for the WSP with user-independent constraints and that there is no algorithm of running time $O^*(2^{o(k \log k)})$ for the WSP with user-independent constraints unless the Exponential Time Hypothesis[3] (ETH) fails. Thus, unlike the WSP with regular constraints (and problems studied by Bodlaender, Cygan, Kratsch, & Nederlof, 2013; Fomin, Lokshtanov, & Saurabh, 2014), the WSP with user-independent constraints is highly unlikely to admit an algorithm of running time $O^*(2^{O(k)})$. To show that our generic algorithm is of interest for constraints other than user-independent, we prove that the generic algorithm is a single-exponential algorithm for a constraint language obtained by an equivalence relation on the set of users.

In Section 5 we show how our generic algorithm can deal with unions of constraint languages. This leads to a generalisation of our result for user-independent constraints. In Section 6 we discuss the results of computational experiments using an implementation of our algorithm (discussed in full detail in the work of Cohen, Crampton, Gagarin, Gutin, & Jones, 2014). A brief conclusion is given in in Section 7.

2. Background

We define a *workflow schema* to be a tuple (S, U, A, C), where S is the set of tasks in the workflow, U is the set of users, $A = \{A(s) : s \in S\}$, where $A(s) \subseteq U$ is the *authorization list* for task s, and C is a set of workflow constraints. A *workflow constraint* is a pair $c = (L, \Theta)$, where $L \subseteq S$ and Θ is a set of functions from L to U: L is the *scope* of the constraint; Θ specifies those assignments of elements of U to elements of L that *satisfy* the constraint c.

Given $T \subseteq S$ and $X \subseteq U$, a *plan* is a function $\pi : T \to X$. Given a workflow constraint (L, Θ), $T \subseteq S$ and $X \subseteq U$, a plan $\pi : T \to X$ *satisfies* (L, Θ) if either $L \setminus T \neq \emptyset$ or $\pi|_L = \theta$ for some $\theta \in \Theta$. A plan $\pi : T \to X$ is *eligible* if π satisfies every constraint in C. A plan $\pi : T \to X$ is *authorized* if $\pi(s) \in A(s)$ for all $s \in T$. A plan is *valid* if it is both authorized and eligible. A plan $\pi : S \to U$ is called a *complete plan*. An algorithm to solve the WSP takes a workflow schema (S, U, A, C) as input and outputs a valid, complete plan, if one exists (and null, otherwise).

As a running example, consider the following instance of the WSP.

Instance 1. *The task set $S = \{s_1, \ldots, s_4\}$ and the user set $U = \{u_1, \ldots, u_6\}$. The authorization lists are as follows (where a tick indicates that the given user is authorized for the given task):*

3. The Exponential Time Hypothesis claims there is no algorithm of running time $O^*(2^{o(n)})$ for 3SAT on n variables (Impagliazzo, Paturi, & Zane, 2001).

	u_1	u_2	u_3	u_4	u_5	u_6
s_1	✔	✔				
s_2		✔	✔			
s_3		✔		✔	✔	✔
s_4		✔		✔	✔	✔

The constraints are as follows: s_1 and s_2 must be assigned to the same user; s_2 and s_3 must be assigned to different users; s_3 and s_4 must be assigned to different users; s_1 and s_4 must be assigned to different users.

Example 1 illustrates the meanings of eligible, complete and authorised plans in the context of Instance 1.

Example 1. *The following table gives assignments for four plans, $\pi_1, \pi_2, \pi_3, \pi_4$:*

	s_1	s_2	s_3	s_4	*Authorized*	*Eligible*	*Complete*
π_1	u_1	u_2	u_4	u_5	✔		✔
π_2	u_1	u_1	u_4	u_5		✔	✔
π_3	u_1	-	u_4	u_5	✔	✔	
π_4	u_2	u_2	u_4	u_5	✔	✔	✔

- *π_1 is a complete plan which is authorized but not eligible, as s_1 and s_2 are assigned to different users.*
- *π_2 is a complete plan which is eligible but not authorized, as u_1 is not authorized for s_2.*
- *π_3 is a plan which is authorized and eligible, and therefore valid. However, π_3 is not a complete plan as there is no assignment for s_2.*
- *π_4 is a complete plan which is eligible and authorized. Thus π_4 is a valid complete plan, and is therefore a solution.*

For an algorithm that runs on an instance (S, U, A, C) of the WSP, we will measure the running time in terms of $n = |U|, k = |S|$, and $m = |C|$. (The set A of authorization lists consists of k lists each of size at most n, so we do not need to consider the size of A separately when measuring the running time.) We will say an algorithm runs in polynomial time if it has running time at most $p(n, k, m)$, where $p(n, k, m)$ is polynomial in n, k and m.

2.1 WSP Constraints

In this paper we are interested in the complexity of the WSP when the workflow constraint language (the set of permissible workflow constraints) is restricted. In this section we introduce the constraint types of interest. All of them have practical applications for real world workflows.

We assume that all constraints and authorizations can be checked in polynomial time. This means that it takes polynomial time to check whether any plan is authorized, eligible and valid. The correctness of our algorithm is unaffected by this assumption, but choosing constraints not checkable in polynomial time would naturally affect the running time.

2.1.1 CONSTRAINTS DEFINED BY A BINARY RELATION

Constraints on two tasks, s and s', can be represented in the form (s, s', ρ), where ρ is a binary relation on U (Crampton, 2005). A plan π satisfies such a constraint if $\pi(s)\ \rho\ \pi(s')$. Writing $=$ to denote the relation $\{(u, u) : u \in U\}$ and $\neq$ to denote the relation $\{(u, v) : u, v \in U, u \neq v\}$, separation-of-duty and binding-of-duty constraints may be represented in the form $(s, s', \neq)$ and $(s, s', =)$, respectively. Crampton et al. (2013) considered constraints for which ρ is $\sim$ or $\nsim$, where $\sim$ is an *equivalence relation* defined on U. A practical example of such workflow constraints is when the equivalence relation partitions the users into different departments: constraints could then enforce that two tasks be performed by members of the same department. Constraints that are not restricted to singleton tasks have also been considered (Crampton et al., 2013; Wang & Li, 2010): a plan π satisfies a constraint of the form (S', S'', ρ) if there are tasks $s' \in S'$ and $s'' \in S''$ such that $\pi(s')\ \rho\ \pi(s'')$.

2.1.2 CARDINALITY CONSTRAINTS

A *tasks-per-user counting constraint* has the form (t_ℓ, t_r, T), where $1 \leqslant t_\ell \leqslant t_r \leqslant k$ and $T \subseteq S$. A plan π satisfies (t_ℓ, t_r, T) if a user performs either no tasks in T or between t_ℓ and t_r tasks. Tasks-per-user counting constraints generalize the cardinality constraints which have been widely adopted by the WSP community (American National Standards Institute, 2004; Bertino, Bonatti, & Ferrari, 2001; Joshi, Bertino, Latif, & Ghafoor, 2005; Sandhu, Coyne, Feinstein, & Youman, 1996).

2.1.3 REGULAR CONSTRAINTS

We say that C is *regular* if it satisfies the following condition: If a partition $S_1, \ldots, S_p$ of S is such that for every $i \in [p]$ there exists an eligible complete plan π and user u with $\pi^{-1}(u) = S_i$, then the plan $\bigcup_{i=1}^{p}(S_i \to u_i)$, where all u_i's are distinct, is eligible. Regular constraints extend the set of constraints considered by Wang and Li (2010). Crampton et al. (2013) show that the following constraints are regular: $(S', S'', \neq)$; $(S', S'', =)$, where at least one of the sets S', S'' is a singleton; tasks-per-user counting constraints of the form (t_ℓ, t_r, T), where $t_\ell = 1$.

2.1.4 USER-INDEPENDENT CONSTRAINTS

Many business rules are not concerned with the identities of the users that complete a set of tasks; they are only concerned with the *relationships* between those users. Accordingly, we say a constraint (L, Θ) is user-independent if whenever $\theta \in \Theta$ and $\psi : U \to U$ is a permutation, then $\psi \circ \theta \in \Theta$. The most obvious example of a user-independent constraint is the requirement that two tasks are performed by different users (separation-of-duty). As a more complex example suppose that at most/at least/exactly p users are required to complete some sensitive set of tasks (cardinality constraints), where p is usually small, i.e., 1, 2, 3 or so. There is a substantial literature on constraints as a method for specifying and enforcing business rules (for example, Gligor, Gavrila, & Ferraiolo, 1998; Simon & Zurko, 1997), including work by researchers at SAP and IBM (for example, Basin, Burri, & Karjoth, 2014; Wolter & Schaad, 2007). The most widely studied constraints are cardinality constraints and separation-of-duty, which form part of the ANSI standard on role-based access control (American National Standards Institute, 2004), developed by the US National Institute of Standards and Technology (NIST). In short, the literature and relevant standards suggest that user-independent constraints are those of most interest in business processes and workflow management systems. Our definition of user-independent includes all the constraints defined in the ANSI RBAC standard and many more.

Every regular constraint is user-independent, but many user-independent constraints are not regular. Indeed, constraints of the type $(S', S'', =)$ are user-independent, but not necessarily regular (Crampton et al., 2013). Many counting constraints in the Global Constraint Catalogue (Beldiceanu, Carlsson, & Rampon, 2012) are user-independent, but not regular. In particular, the constraint NVALUE, which bounds from above the number of users performing a set of tasks, is user-independent but not regular. Note, however, that constraints of the form $(s', s'', \sim)$ and $(s', s'', \not\sim)$ are not user-independent in general.

It is important to note that authorization lists, which are fundamental to any access control system, when viewed as unary constraints, are certainly not user-independent. It is the presence of both user-independent constraints and authorization lists in a workflow specification that makes the WSP challenging.

2.2 A Naive Algorithm

The main aim of this section is to present a simple algorithm (Algorithm 1) which will solve any instance of the WSP. The running time of the algorithm is slightly worse than a brute-force algorithm, but the algorithm's basic structure provides a starting point from which to develop a more efficient algorithm.

We need to introduce some additional notation and terminology.

Let $\pi : T \to X$ be a plan for some $T \subseteq S, X \subseteq U$. Then let $\textsc{Task}(\pi) = T$ and $\textsc{User}(\pi) = X$. It is important for our generic algorithm that $\textsc{Task}(\pi)$ and $\textsc{User}(\pi)$ are given as explicit parts of π. In particular, the set $\textsc{User}(\pi)$ may be different from the set of users assigned to a task by π. That is, a user u can be in $\textsc{User}(\pi)$ without there being a task s such that $\pi(s) = u$. It is worth observing that $\textsc{Task}(\pi)$ may be empty (because π may not allocate any tasks to users in X). For any $T \subseteq S$ and $u \in U$, $(T \to u)$ denotes the plan $\pi : T \to \{u\}$ such that $\pi(s) = u$ for all $s \in T$.

Two functions $f_1 : D_1 \to E_1$ and $f_2 : D_2 \to E_2$ are *disjoint* if $D_1 \cap D_2 = E_1 \cap E_2 = \emptyset$. The *union* of two disjoint functions $f_1 : D_1 \to E_1, f_2 : D_2 \to E_2$ is a function $f = f_1 \cup f_2$ such that $f : D_1 \cup D_2 \to E_1 \cup E_2$ and $f(d) = f_i(d)$ for each $d \in D_i$, where $i \in \{1, 2\}$. Let $g : D \to E$ and $h : E \to F$ be functions. Then $h \circ g$ denotes the composite function from D to F such that $h \circ g(d) = h(g(d))$ for each $d \in D$. For an integer $p > 0$, the set $[p] = \{1, 2, \ldots, p\}$.

Proposition 1. *Let (S, U, A, C) be an instance of the* WSP*, with $n = |U|$, $k = |S|$ and $m = |C|$. Then (S, U, A, C) can be solved in time $O^*((n + 1)^k)$ by Algorithm 1.*

Proof. Let $u_1, \ldots, u_n$ be an ordering of U, and let $U_i = \{u_1, \ldots, u_i\}$ for each $i \in [n]$. For each $i \in [n]$ in turn, we will construct the set Π_i of all plans π such that $\textsc{User}(\pi) = U_i$ and π is valid. If the set Π_n contains no plan π with $\textsc{Task}(\pi) = S$, then (S, U, A, C) has no solution; otherwise, any such plan is a solution for (S, U, A, C).

Algorithm 1 shows how to construct the sets Π_i. It is not hard to verify that Π_i contains exactly every valid plan π with $\textsc{User}(\pi) = U_i$, for each i. This implies the correctness of our algorithm. It remains to analyse the running time.

For each $i \in [n]$ and each $T \subseteq S$, there are at most $i^{|T|}$ valid plans π with $\textsc{User}(\pi) = U_i, \textsc{Task}(\pi) = T$. To construct Π_1, we need to consider all plans π with $\textsc{User}(\pi) = U_1$, and there are exactly 2^k such plans. For each plan we can decide in polynomial time whether to add it to Π_1.

Algorithm 1: Naive solution procedure for the WSP

```
input : An instance $(S, U, A, C)$ of the WSP
Construct an ordering $u_1, \dots, u_n$ of $U$;
Set $\Pi_1 = \emptyset$;
foreach $T \subseteq S$ do
    Set $\pi = (T \to u_1)$;
    if $\pi$ is eligible and $u_1 \in A(s)$ for all $s \in T$ then
        Set $\Pi_1 = \Pi_1 \cup \{\pi\}$;
    end
end
Set $i = 1$;
while $i < n$ do
    Set $\Pi_{i+1} = \emptyset$;
    foreach $\pi' \in \Pi_i$ do
        foreach $T \subseteq S \setminus \textsc{Task}(\pi')$ do
            if $u_{i+1} \in A(s)$ for all $s \in T$ then
                Set $\pi = \pi' \cup (T \to u_{i+1})$;
                if $\pi$ is eligible then
                    Set $\Pi_{i+1} = \Pi_{i+1} \cup \{\pi\}$;
                end
            end
        end
    end
    Set $i = i + 1$;
end
foreach $\pi \in \Pi_n$ do
    if $\textsc{Task}(\pi) = S$ then
        return $\pi$;
    end
end
return NULL;
```

To construct Π_{i+1} for each $i \in [n-1]$, we need to consider every pair (π', T) where $\pi' \in \Pi_i$ and $T \subseteq S \setminus \textsc{Task}(\pi')$. Consider the pair (π', T), where π' is an (S', U_i)-plan for some $S' \subseteq S$, and $T \subseteq S \setminus S'$. Thus there are $i^{|S'|}$ possibilities for π', and there are $2^{|S|-|S'|}$ choices for T. Thus, the total number of pairs is given by $\sum_{S' \subseteq S} i^{|S'|} 2^{|S|-|S'|} = \sum_{j=0}^{k} \binom{k}{j} i^j 2^{k-j} = (i+2)^k$. For each pair (π', T) we can decide whether to add $\pi' \cup (T \to u_{i+1})$ to Π_{i+1} in polynomial time. Thus, to construct all Π_i takes time $O^*(\sum_{i=0}^{n-1} (i+2)^k) = O^*(n(n+1)^k) = O^*((n+1)^k)$. $\square$

Algorithm 1 is inefficient even for small k, due to the fact that each Π_i contains all valid plans π' with $\textsc{User}(\pi') = \{u_1, \dots, u_i\}$. We show in the next section that it is not necessary to store so much information to solve the WSP.

3. Plan-Indistinguishability Relations

We first introduce the notion of extension-equivalence, defined by an equivalence relation on the set of all plans. Informally, the relation enables us to keep a single member of each equivalence class when building plans incrementally.

Definition 1. *Given an instance (S, U, A, C) of the* WSP*, and two eligible plans π_1 and π_2, define $\pi_1 \approx \pi_2$ if the following conditions hold:*

1. $\text{USER}(\pi_1) = \text{USER}(\pi_2)$ *and* $\text{TASK}(\pi_1) = \text{TASK}(\pi_2)$*;*
2. $\pi_1 \cup \pi'$ *is eligible if and only if* $\pi_2 \cup \pi'$ *is eligible, for any plan* π' *disjoint from* π_1 *and* π_2.

Then $\approx$ is an equivalence relation on the set of eligible plans, and we say π_1 and π_2 are extension-equivalent *if $\pi_1 \approx \pi_2$.*

Example 2. *Consider Instance 1.*

Let $\pi_1 : \{s_3, s_4\} \to \{u_2, u_4\}$ be the function such that $\pi_1(s_3) = u_2$ and $\pi_1(s_4) = u_4$. Let $\pi_2 : \{s_3, s_4\} \to \{u_2, u_4\}$ be the function such that $\pi_2(s_3) = u_4$ and $\pi_2(s_4) = u_2$.

The plans π_1 and π_2 are both eligible, and $\text{USER}(\pi_1) = \text{USER}(\pi_2)$ and $\text{TASK}(\pi_1) = \text{TASK}(\pi_2)$. For any plan π' disjoint from π_1 and π_2, the plan $\pi_1 \cup \pi'$ will satisfy the constraints $(s_2, s_3, \neq), (s_1, s_4, \neq)$. Thus $\pi_1 \cup \pi'$ is eligible if and only if π' is eligible. Similarly, $\pi_2 \cup \pi'$ is eligible if and only if π' is eligible. Thus $\pi_1 \cup \pi'$ is eligible if and only if $\pi_2 \cup \pi'$ is eligible, and so π_1 and π_2 are extension-equivalent.

Suppose that we had a polynomial time algorithm to check whether two eligible plans are extension-equivalent. Then in Algorithm 1, we could keep track of just one plan from each equivalence class: when constructing Π_i, we will only add π_2 to Π_i if there is no π_1 extension-equivalent to π_2 already in Π_i; when we construct Π_{i+1}, we may use π_1 as a "proxy" for π_2. If the number of extension-equivalent classes is small compared to the number of plans, then the worst-case running time of the algorithm may be substantially lower than that of Algorithm 1.

Unfortunately, it is not necessarily easy to decide if two eligible plans are extension-equivalent, so this approach is not practical. However, we can always refine[4] extension-equivalence to an equivalence relation for which equivalence *is* easy to determine. For example, the identity equivalence relation where each plan is only equivalent to itself is such a refinement.

This refined equivalence relation may have more equivalence classes than extension-equivalence, but substantially fewer than the identity relation, so we may obtain a better running time than the naive algorithm.

Definition 2. *Given an instance (S, U, A, C) of the* WSP*, let Π be the set of all eligible plans and let $\approx$ be an equivalence relation refining extension-equivalence on Π. We say $\approx$ is a* plan-indistinguishability relation *(with respect to C) if, for all eligible π_1, π_2 such that $\pi_1 \approx \pi_2$, and for any plan π' disjoint from π_1 and π_2 such that $\pi_1 \cup \pi'$ is eligible, we have that $\pi_1 \cup \pi' \approx \pi_2 \cup \pi'$.*

Example 3. *Let $\approx$ be the identity relation on plans. That is, $\pi_1 \approx \pi_2$ if and only if $\text{USER}(\pi_1) = \text{USER}(\pi_2)$, $\text{TASK}(\pi_1) = \text{TASK}(\pi_2)$, and $\pi_1(s) = \pi_2(s)$ for all $s \in \text{USER}(\pi_1)$. Then $\approx$ is a*

4. An equivalence relation $\approx_2$ is a *refinement* of an equivalence relation $\approx_1$ if every equivalence class of $\approx_2$ is a subset of some equivalence class of $\approx_1$.

plan-indistinguishability relation. This shows that not every plan-indistinguishability relation is the extension-equivalence relation. Indeed, the plans given in Example 2 are extension-equivalent but not identical.

Recall that we refined extension-equivalence since it may be hard to determine whether two eligible plans are extension-equivalent. It is therefore natural to assume the following:

Assumption 1. *Given a plan-indistinguishability relation $\approx$, it takes polynomial time to check whether two eligible plans are equivalent under $\approx$.*

The correctness of our algorithms does not depend on this assumption. However, a poor choice of the plan-indistinguishability relation could affect the running times.

We now describe appropriate plan-indistinguishability relations for the constraints that we will be using. In each case determining if two eligible plans are equivalent under $\approx$ will take polynomial time.

3.1 Plan-Indistinguishability Relation for User-Independent Constraints

Lemma 1. *Suppose all constraints are user-independent, and let $\approx_{ui}$ be a relation such that $\pi_1 \approx_{ui} \pi_2$ if and only if*

1. $\text{USER}(\pi_1) = \text{USER}(\pi_2)$ *and* $\text{TASK}(\pi_1) = \text{TASK}(\pi_2)$*;*
2. *For all $s, t \in \text{TASK}(\pi_1)$, $\pi_1(s) = \pi_1(t)$ if and only if $\pi_2(s) = \pi_2(t)$.*

Then $\approx_{ui}$ is a plan-indistinguishability relation on the set of eligible plans.

Proof. By definition of user-independent constraints, if π is an eligible plan and $\psi : U \to U$ is a permutation, then $\psi \circ \pi$ is also eligible. Suppose that $\pi_1 \approx_{ui} \pi_2$, and let $T = \text{TASK}(\pi_1)$ and $X = \text{USER}(\pi_1)$. Let $\psi' : \pi_1(T) \to \pi_2(T)$ be a function such that $\psi'(\pi_1(t)) = \pi_2(t)$ for any task t. Let $\psi'' : X \setminus \pi_1(T) \to X \setminus \pi_2(T)$ be an arbitrary bijection (note that $|\pi_1(T)| = |\pi_2(T)|$ by Condition 2 of $\approx_{ui}$). Let $\psi = \psi' \cup \psi''$. Then ψ is a permutation such that $\pi_2 = \psi \circ \pi_1$. Thus π_1 is eligible if and only if π_2 is eligible.

Now consider two eligible plans π_1, π_2 such that $\pi_1 \approx_{ui} \pi_2$, and a plan π' disjoint from π_1 and π_2. First we show that $\pi_1 \cup \pi' \approx_{ui} \pi_2 \cup \pi'$. It is clear that $\text{USER}(\pi_1 \cup \pi') = \text{USER}(\pi_2 \cup \pi')$ and $\text{TASK}(\pi_1 \cup \pi') = \text{TASK}(\pi_2 \cup \pi')$. Now for any $s, t \in \text{USER}(\pi_1 \cup \pi')$, if $(\pi_1 \cup \pi')(s) = (\pi_1 \cup \pi')(t)$, then either s, t are both in $\text{TASK}(\pi')$, in which case $(\pi_2 \cup \pi')(s) = (\pi_2 \cup \pi')(t)$ trivially, or s, t are both in $\text{TASK}(\pi_1)$, in which case $\pi_2(s) = \pi_2(t)$ since $\pi_1 \approx_{ui} \pi_2$, and hence $(\pi_2 \cup \pi')(s) = (\pi_2 \cup \pi')(t)$. Thus if $(\pi_1 \cup \pi')(s) = (\pi_1 \cup \pi')(t)$ then $(\pi_2 \cup \pi')(s) = (\pi_2 \cup \pi')(t)$ and, by a similar argument, the converse holds. Thus $\pi_1 \cup \pi' \approx_{ui} \pi_2 \cup \pi'$. Furthermore, it follows by the argument in the first paragraph that $\pi_1 \cup \pi'$ is eligible if and only if $\pi_2 \cup \pi'$ is eligible. Thus, the condition of Definition 2 and the second condition of Definition 1 hold.

The first condition of $\approx_{ui}$ trivially satisfies the first condition of Definition 1. Thus, $\approx_{ui}$ satisfies all the conditions of a plan-indistinguishability relation. □

Example 4. *Consider an instance of the* WSP *with users $u_1, \ldots u_6$ and tasks $s_1, \ldots, s_6$ in which all constraints are user-independent. Let $\approx_{ui}$ be the plan-indistinguishability relation given by Lemma 1. Let c_1 be the constraint with scope $\{s_2, s_3, s_4, s_5\}$ such that c_1 is satisfied if and only if an even number of users are assigned to tasks in $\{s_2, s_3, s_4, s_5\}$. Let c_2 be the constraint with scope*

$\{s_1, s_3, s_4, s_6\}$ such that c_2 is satisfied if and only if either s_1 and s_3 are assigned to different users, or s_4 and s_6 are assigned to different users. Suppose that c_1 and c_2 are the only constraints whose scope contains tasks from both $\{s_1, s_2, s_3\}$ and $\{s_4, s_5, s_6\}$.

Now consider the plans $\pi_1, \pi_2 : \{s_1, s_2, s_3\} \to \{u_1, u_2, u_3, u_4\}$ such that $\pi_1(s_1) = u_1, \pi_1(s_2) = u_2, \pi_1(s_3) = u_1$, and $\pi_2(s_1) = u_3, \pi_2(s_2) = u_4, \pi_2(s_3) = u_3$, and suppose that π_1, π_2 are both eligible. Then π_1 and π_2 are equivalent under $\approx_{ui}$.

Observe that for any plan π' disjoint from π_1 and π_2, $\pi_1 \cup \pi'$ is eligible if and only if $\pi_2 \cup \pi'$ is eligible. As π_1 and π_2 both assign two users to $\{s_2, s_3\}$, π' must assign two users to $\{s_4, s_5\}$ in order to satisfy c_1. As π_1 and π_2 both assign s_1 and s_3 to the same user, π' must assign s_4 and s_5 to different users in order to satisfy c_2. As long as these conditions are satisfied, and π' satisfies all constraints with scope in $\{s_4, s_5, s_6\}$, then $\pi_1 \cup \pi'$ and $\pi_2 \cup \pi'$ will both be eligible.

3.2 Plan-Indistinguishability Relation for Equivalence Relation Constraints

Recall that given a binary relation ρ on U, a constraint of the form (s_i, s_j, ρ) is satisfied by a plan π if $\pi(s_i)\ \rho\ \pi(s_j)$. Recall that such constraints are not user-independent in general.

Lemma 2. *Suppose $\sim$ is an equivalence relation on U. Let $V_1, \dots, V_l$ be the equivalence classes of $\sim$ over U. Suppose all constraints are of the form $(s_i, s_j, \sim)$ or $(s_i, s_j, \not\sim)$. Let $\approx_e$ be a relation such that $\pi_1 \approx_e \pi_2$ if and only if*

1. $\textsc{User}(\pi_1) = \textsc{User}(\pi_2)$ *and* $\textsc{Task}(\pi_1) = \textsc{Task}(\pi_2)$*;*
2. *For all equivalence classes V_j such that $V_j \cap \textsc{User}(\pi_1) \neq \emptyset$ and $V_j \setminus \textsc{User}(\pi_1) \neq \emptyset$, we have that for all $s \in \textsc{Task}(\pi_1)$, $\pi_1(s) \in V_j$ if and only if $\pi_2(s) \in V_j$.*

Then $\approx_e$ is a plan-indistinguishability relation.

Proof. It is clear that $\approx_e$ satisfies the first condition of Definition 1. Now suppose π_1, π_2 are eligible plans such that $\pi_1 \approx_e \pi_2$, and let π' be a plan disjoint from π_1 and π_2. We first show that $\pi_1 \cup \pi'$ is eligible if and only if $\pi_2 \cup \pi'$ is eligible.

Suppose that $\pi_1 \cup \pi'$ is eligible. Consider two tasks $t, t' \in \textsc{Task}(\pi_2 \cup \pi')$. If $\{t, t'\} \subseteq \textsc{Task}(\pi')$, then $\pi_2 \cup \pi'$ will not falsify any constraint on t and t' since it is equal to $\pi_1 \cup \pi'$ when restricted to $\{t, t'\}$ and $\pi_1 \cup \pi'$ is eligible. If $\{t, t'\} \subseteq \textsc{Task}(\pi_2)$, then $\pi_2 \cup \pi'$ will not break any constraints since π_2 is eligible.

So we may assume that $t \in \textsc{Task}(\pi_2), t' \in \textsc{Task}(\pi')$. By definition, $(\pi_2 \cup \pi')(t) \sim (\pi_2 \cup \pi')(t')$ if and only if there exists $j \in [l]$ such that $\pi_2(t), \pi'(t') \in V_j$. Then $V_j \cap \textsc{User}(\pi_2) \neq \emptyset$ and $V_j \setminus \textsc{User}(\pi_2) \neq \emptyset$. Therefore, by definition of $\approx_e$, $\pi_1(s) \in V_j$ if and only if $\pi_2(s) \in V_j$, for all $s \in \textsc{Task}(\pi_1)$. In particular, $\pi_1(t) \in V_j$, and so $(\pi_1 \cup \pi')(t) \sim (\pi_1 \cup \pi')(t')$. By a similar argument, if $(\pi_1 \cup \pi')(t) \sim (\pi_1 \cup \pi')(t')$ then $(\pi_2 \cup \pi')(t) \sim (\pi_2 \cup \pi')(t')$. Therefore, every constraint is satisfied by $(\pi_1 \cup \pi')$ if and only if it is satisfied by $(\pi_2 \cup \pi')$. Therefore if $\pi_1 \cup \pi'$ is eligible then so is $\pi_2 \cup \pi'$, and by a similar argument the converse holds.

It remains to show that $\pi_1 \cup \pi' \approx_e \pi_2 \cup \pi'$. It is clear that the user and task sets are the same. As they have the same user set, the sets $\{V_j : V_j \cap \textsc{User}(\pi_1 \cup \pi') \neq \emptyset, V_j \setminus \textsc{User}(\pi_1 \cup \pi') \neq \emptyset\}$ and $\{V_j : V_j \cap \textsc{User}(\pi_2 \cup \pi') \neq \emptyset, V_j \setminus \textsc{User}(\pi_2 \cup \pi') \neq \emptyset\}$ are the same. Furthermore, for each V_j in this set and any $s \in \textsc{Task}(\pi_1 \cup \pi')$, if $(\pi_1 \cup \pi')(s) \in V_j$ then $(\pi_2 \cup \pi')(s) \in V_j$, as either $s \in \textsc{Task}(\pi_1)$, in which case $V_j \cap \textsc{User}(\pi_1) \neq \emptyset, V_j \setminus \textsc{User}(\pi_1) \neq \emptyset$ and so $\pi_2(s) \in V_j$, or

$s \in \text{TASK}(\pi')$, in which case $(\pi_2 \cup \pi')(s) = \pi'(s) = (\pi_1 \cup \pi')(s)$. By a similar argument, if $(\pi_2 \cup \pi')(s) \in V_j$, then $(\pi_1 \cup \pi')(s) \in V_j$. Thus $\pi_1 \cup \pi' \approx_e \pi_2 \cup \pi'$. $\square$

Example 5. *Let $\sim$ be an equivalence relation on users with equivalence classes $\{u_1\}, \{u_2\}, \{u_3, u_4, u_5\}, \{u_6, u_7, u_8\}$. Consider an instance of the* WSP *with users $u_1, \ldots, u_8$ and tasks $s_1, \ldots, s_6$ in which all constraints are of the form $(s_i, s_j, \sim)$ or $(s_i, s_j, \not\sim)$. Let $\approx_e$ be the plan-indistinguishability relation given by Lemma 2. Suppose that the only constraints whose scope contains tasks from both $\{s_1, s_2, s_3\}$ and $\{s_4, s_5, s_6\}$ are the constraints $(s_1, s_5, \not\sim)$, $(s_2, s_5, \sim)$ and $(s_2, s_6, \not\sim)$.*

Now consider the plans $\pi_1, \pi_2 : \{s_1, s_2, s_3\} \to \{u_1, u_2, u_3, u_4\}$ such that $\pi_1(s_1) = u_1, \pi_1(s_2) = u_3, \pi_1(s_3) = u_3$, and $\pi_2(s_1) = u_2, \pi_2(s_2) = u_3, \pi_2(s_3) = u_4$, and suppose that π_1, π_2 are both eligible. Then π_1 and π_2 are equivalent under $\approx_e$.

Observe that for any plan π' disjoint from π_1 and π_2, $\pi_1 \cup \pi'$ is eligible if and only if $\pi_2 \cup \pi'$ is eligible. The only $\sim$-equivalence class with members in $\{u_1, u_2, u_3, u_4\}$ and members not in $\{u_1, u_2, u_3, u_4\}$ is the class $\{u_3, u_4, u_5\}$. π_1 and π_2 both assign members of $\{u_3, u_4\}$ to exactly the set $\{s_2, s_3\}$. Thus for any plan π' disjoint from π_1 and π_2, $\pi_1 \cup \pi'$ and $\pi_2 \cup \pi'$ will both satisfy the constraint $(s_1, s_5, \not\sim)$ whatever π' assigns to s_5. They will both satisfy $(s_2, s_5, \sim)$ only if π' assigns s_5 to u_5, and they will both satisfy $(s_2, s_6, \not\sim)$ only if π' does not assign s_6 to u_5. As long as these conditions are satisfied, and π' satisfies all constraints with scope in $\{s_4, s_5, s_6\}$, then $\pi_1 \cup \pi'$ and $\pi_2 \cup \pi'$ will both be eligible.

4. A Generic Algorithm for the WSP

In what follows, for each $X \subseteq U, T \subseteq S$, we let $\Pi[X, T]$ denote the set of all eligible plans π with $\text{USER}(\pi) = X$ an $\text{TASK}(\pi) = T$. In this section we will introduce an algorithm that works in a similar way to Algorithm 1, except that instead of storing all valid plans over a particular set of users or tasks, we will construct $\Pi[X, T]$-representative sets for each task set T and certain user sets X. By definition, the equivalence classes of any plan-indistinguishability relation necessarily partition $\Pi[X, T]$. Hence any such equivalence class has a representation of the form $(X, T, \cdot)$, where $\cdot$ is dependent on the constraint language. In the remainder of this section we describe the algorithm and give examples of these representations.

4.1 Encodings and Patterns

In our generic algorithm, we will construct plans iteratively, using at most one plan from each equivalence class under a plan-indistinguishability relation. The running time of the algorithm will depend on the number of equivalence classes under this relation, over certain sets of plans. To ensure that sets of equivalence classes can be ordered and therefore searched and sorted efficiently, we introduce the notion of encodings and patterns. Loosely speaking, an *encoding* is a function that maps all the plans in a $\approx$-equivalence class to the same element (the *pattern* of those plans). These encodings ensure logarithmic-time access and insertion operations into a representative set of plans, rather than the linear time that a naive method would allow.

Note that the use of encodings and patterns is not necessary for any of our fixed-parameter tractability results; the same problems could be solved without the use of patterns and encodings in fixed-parameter time, but the function in k would grow more quickly.

Definition 3. *Given an instance (S, U, A, C) of the* WSP *and a plan-indistinguishability relation $\approx$, let Π be the set of all plans. Let* PAT *be some set and consider a function* ENC : $\Pi \to$ PAT*. For any $X \subseteq U, T \subseteq S$, let* $\text{PAT}[X,T] = \text{ENC}(\Pi[X,T])$*. Then we say* ENC *is a* $\approx$-encoding *(or an* encoding for $\approx$*) if, for any $X \subseteq U, T \subseteq S$ and any $\pi_1, \pi_2 \in \Pi[X,T]$, we have that*

1. $\text{ENC}(\pi_1) = \text{ENC}(\pi_2)$ *if and only if* $\pi_1 \approx \pi_2$*;*
2. $\text{ENC}(\pi_1)$ *can be calculated in time polynomial in* n, k, m*;*
3. *There exists a linear ordering $\preceq$ on* $\text{PAT}[X,T]$ *such that, for* $p, p' \in \text{PAT}[X,T]$*, we can decide whether $p \preceq p'$ in time polynomial in n, k, m.*

The elements of PAT *are called* $\approx$-patterns. *If* $\text{ENC}(\pi) = p$ *then we say p is the* $\approx$-pattern of π.

The second and third conditions of Definition 3 ensure that we may use encodings to organise our plans in a reasonable time. When $\approx$ is clear from the context, we will refer to a $\approx$-encoding as an *encoding* and $\approx$-patterns as *patterns*.

We note some complexity consequences of Definition 3 in the following:

Proposition 2. *For an encoding of a plan-indistinguishability relation $\approx$ and a set of patterns* PAT^**, by assigning patterns in* PAT^* *to the nodes of a balanced binary tree, we can perform the following two operations in time* $O^*(\log(|\text{PAT}^*|))$*: (i) check whether* $p \in \text{PAT}^*$*, and (ii) insert a pattern* $p \notin \text{PAT}^*$ *into* PAT^* .

Proof. Recall that comparisons are polynomial in n, k, m. Now our result follows from the well-known properties of balanced binary trees (e.g., see (Cormen, Stein, Rivest, & Leiserson, 2001)). □

We now show that the plan-indistinguishability relations given in the previous section have encodings. We first need to define a lexicographic order.

Definition 4. *Given a totally ordered set $(A, \leq)$, the (total)* lexicographic order $\leq$ *on d-tuples from A^d is defined as follows. We say that $(x_1, \ldots, x_d) \leq (y_1, \ldots, y_d)$ if either $x_j = y_j$ for all $j \in [d]$ or there is an i with $x_i < y_i$ such that $x_j = y_j$ for all $j < i$.*

Taking $A = \mathbb{N}$ and $d = k$ we obtain the natural lexicographic order on $\mathbb{N}_0^k$.

We can also lexicographically order the sets of disjoint subsets of an ordered set $T = \{t_1, \ldots, t_k\}$, where $t_1 < \cdots < t_k$.

Definition 5. *We associate a k-tuple $(x_1, \ldots, x_k) \in \mathbb{N}_0^k$ with each set $\mathcal{S}$ of disjoint subsets $\{S_1, \ldots, S_r\}$ of $\{t_1, \ldots, t_k\}$ as follows. We have $x_i = 0$ if $t_i \notin \cup_{m=1}^r S_m$. For $t_i \in \cup_{m=1}^r S_m$,*

- *if there are $j < i$ and m such that $\{t_i, t_j\} \subseteq S_m$ then $x_i = x_j$,*
- *otherwise $x_i = \max\{x_1, \ldots, x_{i-1}\} + 1$, where $\max \emptyset = 0$.*

We will write $\text{VEC}(\mathcal{S}) = (x_1, \ldots, x_k)$*. Note that* $\text{VEC}(\mathcal{S})$ *can be computed in time $O(k^2)$.*

Thus, tasks in the same subset are assigned the same value; this assignment of integers to tasks can be performed iteratively. For example, for $T = \{1, \ldots, 8\}$ and the sets $\mathcal{A} = \{\{2,4\}, \{3\}, \{5,7\}\}$ and $\mathcal{B} = \{\{2,3,4\}, \{5,7\}\}$, we have $\text{VEC}(\mathcal{A}) = (0,1,2,1,3,0,3,0)$ and $\text{VEC}(\mathcal{B}) = (0,1,1,1,2,0,2,0)$. So $\mathcal{A}$ is lexicographically bigger than $\mathcal{B}$.

Corollary 1. *Let $\approx_{ui}$ be the plan-indistinguishability relation given for a set of user-independent constraints in Lemma 1. Then there exists an encoding for $\approx_{ui}$.*

Proof. Let $s_1, \ldots, s_k$ be an ordering of S and π a plan. Let $\mathcal{S}^\pi = \{\pi^{-1}(u) : u \in \textsc{User}(\pi)\}$ and let $\textsc{vec}(\pi) = \textsc{vec}(\mathcal{S}^\pi)$. For a plan π, let $\textsc{enc}(\pi)$ be the tuple $(\textsc{User}(\pi), \textsc{Task}(\pi), \textsc{vec}(\pi))$.

It is clear that $\textsc{enc}(\pi_1) = \textsc{enc}(\pi_2)$ if and only if $\pi_1 \approx_{ui} \pi_2$, as $\pi_r(s_i) = \pi_r(s_j)$ if and only if $y_i = y_j$ in $\textsc{vec}(\pi_r) = (y_1, \ldots, y_k)$, for $r \in \{1, 2\}$. Furthermore it is clear that $\textsc{enc}(\pi)$ can be determined in polynomial time for any π.

It remains to define a linear ordering on $\textsc{pat}[X, T]$ for a given $X \subseteq U, T \subseteq S$. For two patterns $p = (X, T, (x_1, \ldots, x_k)), p' = (X, T, (y_1, \ldots, y_k)) \in \textsc{pat}[X, T]$, we define $p \preceq p'$ if $(x_1, \ldots, x_k) \leq (y_1, \ldots, y_k)$. □

Example 6. *Let* ENC *be the encoding given in the proof of Corollary 1. Let π_1, π_2 be the plans given in Example 4. Then $\textsc{enc}(\pi_1) = \textsc{enc}(\pi_2) = \{\{u_1, u_2, u_3, u_4\}, \{s_1, s_2, s_3\}, (1, 2, 1, 0, 0, 0)\}$.*

Corollary 2. *Let $\approx_e$ be the plan-indistinguishability relation given for a set of constraints on equivalence relations in Lemma 2. Then there exists an encoding for $\approx_e$.*

Proof. Suppose $\sim$ is an equivalence relation on users, and let $V_1, \ldots, V_p$ be the equivalence classes of $\sim$ over U. Suppose all constraints are of the form $(s_i, s_j, \sim)$ or $(s_i, s_j, \not\sim)$.

For a plan π, define $\textsc{enc}(\pi)$ to be $(\textsc{User}(\pi), \textsc{Task}(\pi), \mathcal{T}^\pi)$, where

$$\mathcal{T}^\pi = \left\{\pi^{-1}(V_j \cap \textsc{User}(\pi)) : V_j \cap \textsc{User}(\pi) \neq \emptyset, V_j \setminus \textsc{User}(\pi) \neq \emptyset, 1 \leq j \leq p\right\}.$$

It is clear that $\textsc{enc}(\pi_1) = \textsc{enc}(\pi_2)$ if and only if $\pi_1 \approx_e \pi_2$, as $\pi_i(s) \in V_j$ if and only if $s \in \pi_i^{-1}(V_j)$, for $i \in \{1, 2\}$. Furthermore it is clear that $\textsc{enc}(\pi)$ can be determined in polynomial time for any π.

It remains to define a linear ordering on $\textsc{pat}[X, T]$ for a given $X \subseteq U, T \subseteq S$. Let $\pi : T \to X$ be a plan. As $\mathcal{T}^\pi$ is a set of disjoint subsets of $\textsc{Task}(\pi)$, and T has a natural order, we can order patterns in $\textsc{pat}[X, T]$ according to the lexicographic order of $\mathcal{T}^\pi$.

□

Example 7. *Let* ENC *be the encoding given in the proof of Corollary 2. Let π_1, π_2 be the plans given in Example 5. Then $\textsc{enc}(\pi_1) = \textsc{enc}(\pi_2) = \{\{u_1, u_2, u_3, u_4\}, \{s_1, s_2, s_3\}, \{\{s_2, s_3\}\}\}$.*

4.2 The Generic Algorithm

We use the notion of *diversity* introduced in the next definition to analyse the running time of our generic algorithm.

Definition 6. *Let (S, U, A, C) be an instance of the* WSP, *with $n = |U|, k = |S|$ and $m = |C|$, and suppose $\approx$ is a plan-indistinguishability relation with respect to C. Given an ordering $u_1, \ldots, u_n$ of U, let $U_i = \{u_1, \ldots, u_i\}$ for each $i \in [n]$. Let w_i be the number of equivalence classes of $\approx$ over the set $\Pi[U_i, T]$ of eligible plans. Then we define the* diversity of $\approx$ with respect to $u_1, \ldots, u_n$ *to be $w = \max_{i \in [n]} w_i$.*

Since our generic algorithm only stores one plan from each equivalence class under $\approx$, we need the notion of a representative set.

Definition 7. *Given an instance (S, U, A, C) of the* WSP*, let Π' be a set of eligible plans and let $\approx$ be a plan-indistinguishability relation. A set Π'' is said to be a Π'-*representative set with respect to *$\approx$ if the following properties hold:*

1. $\Pi'' \subseteq \Pi'$*; every plan in Π'' is valid;*
2. *for every valid $\pi' \in \Pi'$, there exists a $\pi'' \in \Pi''$ such that $\pi' \approx \pi''$.*

When $\approx$ is clear from context, we will say Π'' is a Π'-*representative set* or a *representative set for* Π'. Our generic algorithm is based on finding plan-indistinguishability relations for which there exist small representative sets.

Theorem 1. *Let (S, U, A, C) be an instance of the* WSP*, with $n = |U|, k = |S|$ and $m = |C|$. Let $u_1, \dots, u_n$ be an ordering of U, and let $U_i = \{u_1, \dots, u_i\}$ for each $i \in [n]$, and $U_0 = \emptyset$. Suppose $\approx$ has diversity w with respect to $u_1, \dots, u_n$. Furthermore suppose that there exists a $\approx$-encoding* ENC. *Then (S, U, A, C) can be solved in time $O^*(3^k w \log w)$.*

Proof. The proof proceeds by demonstrating the correctness and then bounding the running time of Algorithm 2, which solves the WSP. To begin the proof, we give an overview of Algorithm 2.

- For each $i \in [n]$ in turn and each $T \subseteq S$, we will construct a representative set for $\Pi[U_i, T]$, denoted by $\Pi[U_i, T]^*$.
- As well as constructing the set $\Pi[U_i, T]^*$, we also maintain a companion set $\text{PAT}[U_i, T]^* = \text{ENC}(\Pi[U_i, T]^*)$. This provides an efficient way of representing the equivalence classes of $\Pi[U_i, T]^*$. In particular, it allows us to check whether a given valid plan π should be added to $\Pi[U_i, T]^*$, faster than by searching $\Pi[U_i, T]^*$ linearly.
- After $\Pi[U_n, S]^*$ has been constructed, it remains to check whether $\Pi[U_n, S]^*$ is non-empty, as if there exists any valid complete plan π, there exists a valid complete plan $\pi' \in \Pi[U_n, S]^*$ with $\pi \approx \pi'$.

Algorithm 2 gives the details on how to construct $\Pi[U_i, T]^*$ for each i and T.

The proof of correctness of Algorithm 2 proceeds by induction. Observe first that for the case of $\Pi[U_0, T]^*$, if $T \neq \emptyset$ then there is no possible plan in $\Pi[U_0, T]$, and so we set $\Pi[U_0, T]^* = \emptyset$. If $T = \emptyset$ then the only possible plan is the empty plan $\emptyset \rightarrow \emptyset$). This plan is added to $\Pi[U_0, \emptyset]^*$, as it is trivially valid. Thus $\Pi[U_0, T]^*$ is a $\Pi[U_0, T]$-representative set for each T.

So now assume that for all $T \subseteq S$ the set $\Pi[U_i, T]^*$ has been constructed and is a $\Pi[U_i, T]$-representative set. Now consider the construction of $\Pi[U_{i+1}, T]^*$ for some $T \subseteq S$. It is clear that for any π added to $\Pi[U_{i+1}, T]^*$, $\pi \in \Pi[U_{i+1}, T]$, and π is eligible. Furthermore π is authorized, as it is the union of the authorized plans $\pi' \in \Pi[U_i, T']$ and $(T'' \rightarrow u_{i+1})$. Thus every plan in $\Pi[U_{i+1}, T]^*$ is a valid plan in $\Pi[U_{i+1}, T]$. On the other hand, suppose π is a valid plan in $\Pi[U_{i+1}, T]$. Then let $T'' = \pi^{-1}(\{u_{i+1}\})$ and $T' = T \setminus T''$, and let $\pi' = \pi|_{U_i}$, so that $\pi = \pi' \cup (T'' \rightarrow u_{i+1})$. By assumption, there exists $\pi'^* \in \Pi[U_i, T]^*$ such that $\pi'^* \approx \pi'$. Consider the plan $\pi^* = \pi'^* \cup (T'' \rightarrow u_{i+1})$. It is clear that π^* will be considered during the algorithm. Furthermore, as $\pi'^* \approx \pi'$ and $\pi = \pi' \cup (T'' \rightarrow u_{i+1})$, we have that $\pi^* \approx \pi$. Therefore π^* is eligible (as π is eligible) and also authorized (as it is the union of two authorized plans). Therefore

Algorithm 2: Generic algorithm for the WSP

```
input : An instance (S, U, A, C) of the WSP, an ordering u_1, ..., u_n of U, a
        plan-indistinguishability relation ≈
Set Π[U_0, ∅]* = {(∅ → ∅)};
foreach ∅ ≠ T ⊆ S do
    Set Π[U_0, T]* = ∅;
end
Set i = 0;
while i < n do
    foreach T ⊆ S do
        Set Π[U_{i+1}, T]* = ∅;
        Set PAT[U_{i+1}, T]* = ∅;
        foreach T' ⊆ T do
            Set T'' = T \ T';
            if u_{i+1} ∈ A(s) for all s ∈ T'' then
                foreach π' ∈ Π[U_i, T']* do
                    Set π = π' ∪ (T'' → u_{i+1});
                    if π is eligible then
                        Set p = ENC(π);
                        if p ∉ PAT[U_{i+1}, T]* then
                            Insert p into PAT[U_{i+1}, T]*;
                            Set Π[U_{i+1}, T]* = Π[U_{i+1}, T]* ∪ {π};
                        end
                    end
                end
            end
        end
    end
    Set i = i + 1;
end
if Π[U_n, S]* ≠ ∅ then
    return π ∈ Π[U_n, S]*;
else
    return NULL;
end
```

π^* is valid and will be added to $\Pi[U_{i+1},T]^*$ unless $\Pi[U_{i+1},T]^*$ already contains another plan $\approx$-equivalent to π. Thus, $\Pi[U_{i+1},T]^*$ contains a plan $\approx$-equivalent to π, from which it follows that $\Pi[U_{i+1},T]^*$ is a $\Pi[U_{i+1},T]$-representative set, as required.

It remains to analyse the running time of the algorithm. By Proposition 2, testing whether a pattern p is in $\textsc{pat}[U_i,T]^*$ and inserting p into $\textsc{pat}[U_i,T]^*$ takes $O^*(\log(|\textsc{pat}[U_i,T]^*|))$ time. Since by Assumption 1 and our assumption on the time to check constraints and authorizations it takes polynomial time to check eligibility, authorization and $\approx$-equivalence of plans, the running time of the algorithm is $O^*(\sum_{i=0}^{n-1}\sum_{T\subseteq S}\sum_{T'\subseteq T}\sum_{\pi\in\Pi[U_i,T']^*}\log(|\Pi[U_{i+1},T]^*|))$. It is clear by construction that $\Pi[U_i,T']^*$ contains at most one plan for each $\approx$-equivalence class over $\Pi[U_i,T']$, and so by definition $|\Pi[U_i,T']^*| \leq w$ for all i, T'. It follows that the running time of the algorithm is $O^*(\sum_{i=0}^{n-1}\sum_{T\subseteq S}\sum_{T'\subseteq T} w\log w) = O^*(3^k w\log w)$. □

Remark 1. *Rather than checking whether $\Pi[U_n,S]^*$ is non-empty at the end of the algorithm, we could instead check whether $\Pi[U_i,S]^*$ is non-empty after the construction of $\Pi[U_i,S]^*$ for each i. That is, we can stop our search as soon as we have a valid plan with task set S. This is likely to lead to a saving in the running time of an implementation of the algorithm. As this paper is concerned with the worst-case running time, which would be unaffected by this change, we perform the check at the end of the algorithm in the interest of clarity.*

4.3 Application to User-Independent Constraints and its Optimality

In this subsection, we show that the WSP with user-independent constraints is FPT. Let B_k denote the kth Bell number, the number of partitions of a set with k elements.

Lemma 3. *Let $u_1,\ldots,u_n$ be any ordering of U, and let $\approx_{ui}$ be the plan-indistinguishability relation given in Lemma 1. Then $\approx_{ui}$ has diversity B_k with respect to $u_1,\ldots,u_n$.*

Proof. For any plan π, the set $\{\pi^{-1}(u) : u \in \textsc{User}(\pi)\}$ is a partition of the tasks in $\textsc{Task}(\pi)$. Furthermore, two plans that generate the same partition are equivalent under $\approx_{ui}$. Therefore the number of equivalence classes of $\approx_{ui}$ over $\Pi[U_i,T]$ is exactly the number of possible partitions of T, which is $B_{|T|}$. Thus, B_k is the required diversity. □

Theorem 2. *If all constraints are user-independent, then the* WSP *can be solved in time $O^*(2^{k\log k})$.*

Proof. Let $u_1,\ldots,u_n$ be any ordering of U, and let $\approx_{ui}$ be the plan-indistinguishability relation given in Lemma 1.

By Lemma 3, $\approx_{ui}$ has diversity B_k with respect to $u_1,\ldots,u_n$. Furthermore, by Corollary 1, there exists an encoding for $\approx_{ui}$. Therefore, we may apply Theorem 1 with $w = B_k$, to get an algorithm with running time $O^*(3^k B_k\log B_k) = O^*(2^{k\log k})$ as $B_k < (0.792k/\ln(k+1))^k$ for every k (Berend & Tassa, 2010). □

The running time $O^*(2^{k\log k})$ obtained is optimal in the sense that no algorithm of running time $O^*(2^{o(k\log k)})$ exists, unless the ETH fails. In the proof of the following theorem, we use a result from Lokshtanov, Marx, & Saurabh, 2011 (Theorem 2.2).

Theorem 3. *There is no algorithm for the* WSP *with user-independent constraints of running time $O^*(2^{o(k\log k)})$, unless the ETH fails.*

Proof. We give a reduction from the problem $k\times k$ INDEPENDENT SET: Given an integer parameter k and a graph G with vertex set $V = \{(i,j) : i,j\in[k]\}$, decide whether G has an independent set I such that $|I| = k$ and for each $r\in[k]$, there exists i such that $(r,i)\in I$.

Informally, $k\times k$ INDEPENDENT SET gives us a graph on a $k\times k$ grid of vertices, and asks whether there is an independent set with one vertex from each row. Lokshtanov et al. (2011) proved that there is no algorithm to solve $k\times k$ INDEPENDENT SET in time $2^{o(k\log k)}$, unless the ETH fails.

Consider an instance of $k \times k$ INDEPENDENT SET with graph G. We will first produce an equivalent instance of the WSP in which the constraints are not user-independent. We will then refine this instance to one with user-independent constraints.

Let $U = \{u_1, \ldots, u_k\}$ be a set of k users and $S = \{s_1, \ldots, s_k\}$ a set of k tasks. Let the authorization lists be $A(s_i) = U$ for all $i \in [k]$. For $i, j, h, l \in [k]$, let $c((i,j),(h,l))$ denote the constraint with scope $\{s_i, s_h\}$, and which is satisfied by any plan π unless $\pi(s_i) = u_j$ and $\pi(s_h) = u_l$. For every pair of vertices $(i,j),(h,l)$ which are adjacent in G, add the constraint $c((i,j),(h,l))$ to C.

We now show that (S,U,A,C) is a YES-instance of the WSP if and only if G has an independent set with one vertex from each row. Suppose (S,U,A,C) is a YES-instance of the WSP and let π be a valid complete plan. Then for each $i \in [k]$, let $f(i)$ be the unique j such that $\pi(s_i) = u_j$. Then $I = \{(i, f(i)) : i \in [k]\}$ is a set with one vertex from each row in G; furthermore, as π satisfies every constraint, no edge in G contains more than one element of I, and so I is an independent set.

Conversely, suppose G is a YES-instance of $k \times k$ INDEPENDENT SET. For each $i \in [k]$, let $f(i)$ be an integer such that $(i, f(i)) \in I$. Then observe that $\bigcup_{i=1}^{k}(\{s_i\} \to u_{f(i)})$ is a valid complete plan.

We now show how to reduce (S,U,A,C) to an instance of the WSP in which all constraints are user-independent. The main idea is to introduce some new tasks representing the users, and in the constraints, replace the mention of a particular user with the mention of the user that performs a particular task.

Create k new tasks $t_1, \ldots, t_k$ and let $S' = S \cup \{t_1, , \ldots, t_k\}$. Let the authorization lists be $A'(s) = U$ for each $s \in S$ and $A'(t_i) = \{u_i\}$ for each $i \in [k]$. For each constraint $c((i,j),(h,l))$ in C, let $d((i,j),(h,l))$ be the constraint with scope $\{s_i, s_h, t_j, t_l\}$, which is satisfied by any plan π unless $\pi(s_i) = \pi(t_j)$ and $\pi(s_h) = \pi(t_l)$. Let initially $C' = C$. Now replace, in C', every constraint $c((i,j),(h,l))$ with $d((i,j,),(h,l))$.

Since they are defined by equalities, and no users are mentioned, the constraints in C' are user-independent. We now show that (S',U,A',C') is equivalent to (S,U,A,C). First, suppose that π is a valid complete plan for (S,U,A,C). Then let $\pi' : S' \to U$ be the plan such that $\pi'(s_i) = \pi(s_i)$ for all $i \in [k]$, and $\pi'(t_j) = u_j$ for all $j \in [k]$. It is easy to check that if π satisfies every constraint of C then π' satisfies every constraint of C'. Since π' is an authorized and eligible plan, π' is a valid complete plan for (S',U,A',C').

Conversely, suppose that π' is a valid complete plan for (S',U,A',C'). Since $A'(t_i) = \{u_i\}$ for each $i \in [k]$, $\pi'(t_i) = u_i$ for every $i \in [k]$. For each $i \in [k]$, let $f(i)$ be the unique integer such that $\pi'(s_i) = u_{f(i)}$. Then define $\pi : S \to U$ by $\pi(s_i) = u_{f(i)}$, and observe that all constraints in C are satisfied by π. So, π is a valid complete plan for (S,U,A,C). □

4.4 Application to Equivalence Relation Constraints

It is known that restricting the WSP to have only equivalence relation constraints is enough to ensure that the problem is FPT (Crampton et al., 2013). However, we can derive this result by applying our algorithm directly having shown the appropriate properties of the language of equivalence relation constraints. This serves to demonstrate the wide applicability of our approach.

Lemma 4. *Let $\approx_e$ be the plan-indistinguishability relation given for a set of equivalence relation constraints in Lemma 2. Then there exists an ordering $u_1, \ldots, u_n$ of U such that $\approx_e$ has diversity 2^k with respect to U.*

Proof. Suppose $\sim$ is an equivalence relation on users, and let $V_1, \ldots, V_p$ be the equivalence classes of $\sim$ over U. Suppose all constraints are of the form $(s_i, s_j, \sim)$ or $(s_i, s_j, \not\sim)$.

Let $u_1, \ldots, u_n$ be an ordering of U such that all the elements of V_j appear before all the elements of $V_{j'}$, for any $j < j'$. Thus, for any i and any plan π with $\textsc{User}(\pi) = U_i = \{u_1, \ldots, u_i\}$, there is at most one integer j_i such that $V_{j_i} \cap \textsc{User}(\pi) \neq \emptyset, V_{j_i} \setminus \textsc{User}(\pi) \neq \emptyset$.

It follows that any two plans $\pi_1, \pi_2 \in \Pi[U_i, T]$ are $\approx_e$-equivalent, for any $i \in [n], T \subseteq S$, provided that $\pi_1(t) \in V_{j_i}$ if and only if $\pi_2(t) \in V_{j_i}$ for any $t \in T$. Therefore $\approx_e$ has at most 2^k equivalence classes over $\Pi[U_i, T]$, as required. □

Theorem 4. *Suppose $\sim$ is an equivalence relation on U. Suppose all constraints are of the form $(s_i, s_j, \sim)$ or $(s_i, s_j, \not\sim)$. Then the* WSP *can be solved in time $O^*(6^k)$.*

Proof. Let $u_1, \ldots, u_n$ be the ordering of U given by Lemma 4, and let $\approx_e$ be the plan-indistinguishability relation given in Lemma 2.

By Lemma 4, $\approx_e$ has diversity 2^k with respect to $u_1, \ldots, u_n$. Furthermore by Corollary 2, there exists an encoding for $\approx_e$. Therefore, we may apply Theorem 1 with $w = 2^k$, to get an algorithm with running time $O^*(3^k 2^k \log(2^k)) = O^*(6^k)$. □

5. Unions of Constraint Languages

In this section we show how our approach allows us easily to combine constraint languages shown to be FPT for the WSP. We do not need to build bespoke algorithms for the new constraint language obtained, only to show that the two languages are in some sense compatible.

This highlights the advantages of our approach over previous methods, which required the development of new algorithms when different constraint languages were combined in an instance of the WSP (e.g., see Crampton et al., 2013).

Theorem 5. *Let $(S, U, A, C_1 \cup C_2)$ be an instance of the* WSP*, and suppose $\approx_1$ is a plan-indistinguishability relation with respect to C_1 and $\approx_2$ is a plan-indistinguishability relation with respect to C_2. Given an ordering $u_1, \ldots, u_n$ of U, let W_1 be the diversity of $\approx_1$ with respect to $u_1, \ldots, u_n$ and W_2 the diversity of $\approx_2$ with respect to $u_1, \ldots, u_n$.*

Let $\approx$ be the equivalence relation such that $\pi \approx \pi'$ if and only if $\pi \approx_1 \pi'$ and $\pi \approx_2 \pi'$. Then $\approx$ is a plan-indistinguishability relation with respect to $C_1 \cup C_2$, and $\approx$ has diversity $W_1 W_2$ with respect to $u_1, \ldots, u_n$.

Proof. We first show that $\approx$ is a plan-indistinguishability relation with respect to $C_1 \cup C_2$. Let π and π' be eligible plans (with respect to $C_1 \cup C_2$). As $\pi \approx \pi'$ implies $\pi \approx_1 \pi'$ and $\approx_1$ satisfies the conditions of a plan-indistinguishability relation, it is clear that if $\pi \approx \pi'$ then $\textsc{User}(\pi) = \textsc{User}(\pi')$ and $\textsc{Task}(\pi) = \textsc{Task}(\pi')$. Now consider a plan π'' disjoint from π and π'. As $\approx_1$ is a plan-indistinguishability relation with respect to C_1 and $\pi \approx_1 \pi'$, we have that $\pi \cup \pi''$ is C_1-eligible if and only if $\pi' \cup \pi''$ is. Similarly $\pi \cup \pi''$ is C_2-eligible if and only if $\pi' \cup \pi''$ is. Observing that a plan is $C_1 \cup C_2$-eligible if and only if it is C_1-eligible and C_2-eligible, this implies that $\pi \cup \pi''$ is $C_1 \cup C_2$-eligible if and only if $\pi' \cup \pi''$ is. Thus we have that π and π' are extension equivalent.

As $\approx_1$ and $\approx_2$ are plan-indistinguishability relations, we have that $\pi \cup \pi'' \approx_1 \pi' \cup \pi''$ and $\pi \cup \pi'' \approx_2 \pi' \cup \pi''$, and therefore $\pi \cup \pi'' \approx \pi' \cup \pi''$. Thus, $\approx$ satisfies all the conditions of a plan-indistinguishability relation.

To bound the diversity of $\approx$ with respect to $u_1, \ldots, u_n$, consider any $T \subseteq S$ and $U_i = \{u_1, \ldots, u_i\}$. It is enough to note that any $\approx$-equivalent plans in $\Pi[U_i, T]$ must be in the same $\approx_1$ and $\approx_2$-equivalence classes. As there are at most W_1 choices for the $\approx_1$-equivalence class and at most W_2 choices for the $\approx_2$ equivalence class, $\approx$ has at most $W_1 W_2$ equivalence classes over $\Pi[U_i, T]$. □

Remark 2. *Given an encoding* ENC_1 *for* $\approx_1$ *and an encoding* ENC_2 *for* $\approx_2$*, we may construct an encoding for* $\approx$*. Given a plan* π*, let* $\text{ENC}(\pi)$ *be the ordered pair* $(\text{ENC}_1(\pi), \text{ENC}_2(\pi))$*. It is clear that* $\text{ENC}(\pi) = \text{ENC}(\pi')$ *if and only if* $\pi \approx \pi'$*.*

Given sets $T \subseteq S$ *and* $U_i = \{u_1, \ldots, u_i\}$*, fix linear orderings of* $\text{ENC}_1(\Pi[U_i, T])$ *and* $\text{ENC}_2(\Pi[U_i, T])$*. Then let* $\preceq$ *be the lexicographic ordering of* $\text{ENC}(\Pi[U_i, T]) = \text{ENC}_1(\Pi[U_i, T]) \times \text{ENC}_2(\Pi[U_i, T])$*.*

There is nothing to stop us applying Theorem 5 multiple times, in order to get a plan-indistinguishability relation with bounded diversity for a union of several constraint languages. Note that the diversity can be expected to grow exponentially with the number of languages in the union. Thus, it makes sense to only apply Theorem 5 to the union of a small number of languages. However, as long as there is a fixed number of languages, and each has a plan-indistinguishability relation with fixed-parameter diversity, the resulting union of languages will also have a plan-indistinguishability relation with fixed-parameter diversity.

We can now use this result directly to show that if all our constraints are either user independent or equivalence relation constraints, then the WSP is still FPT.

Theorem 6. *Suppose* $\sim$ *is an equivalence relation on* U*. Let* (S, U, A, C) *be an instance of the* WSP*, and suppose that all constraints are either of the form,* $(s_1, s_2, \sim)$*,* $(s_1, s_2, \not\sim)$ *or user-independent constraints. Then the* WSP *can be solved in time* $O^*(2^{k \log k + k})$*.*

Proof. Let $C_e \subseteq C$ be the set of constraints of the form $(s_1, s_2, \sim)$, $(s_1, s_2, \not\sim)$, and let C_{ui} be the remaining (user-independent) constraints.

Let $u_1, \ldots, u_n$ be the ordering of U given by Lemma 4. By Lemmas 2 and 4, there exists a plan-indistinguishability relation $\approx_e$ for C_e that has diversity 2^k with respect to $u_1, \ldots, u_n$. Furthermore by Corollary 2, $\approx_e$ has an encoding. By Lemmas 1 and 3, there exists a plan-indistinguishability relation $\approx_{ui}$ for C_{ui} that has diversity B_k with respect to $u_1, \ldots, u_n$. Furthermore by Corollary 1, $\approx_{ui}$ has an encoding.

Therefore by Theorem 5, we may find a plan-indistinguishability relation $\approx$ for C, such that $\approx$ has diversity $B_k \cdot 2^k$ with respect to $u_1, \ldots, u_n$ and $\approx$ has an encoding. Thus we may apply Theorem 1 with $w = B_k \cdot 2^k$, to get a running time of $O^*(3^k B_k 2^k \log(B_k 2^k)) = O^*(3^k 2^{k \log k(1-o(1))+k} \log(2^{k \log k(1-o(1))+k})) = O^*(2^{k \log k + k})$. □

6. Computational Experiments with WSP Algorithms

Apart from conducting theoretical research on the WSP, Wang and Li (2010) carried out an experimental study of the problem. Due to the difficulty of acquiring real-world workflow instances, Wang and Li used synthetic data in their experimental study. They encoded instances of the WSP into pseudo-Boolean SAT in order to use a pseudo-Boolean SAT solver SAT4J.

We have implemented our algorithm and compared its performance to SAT4J on another set of synthetic instances of the WSP (Cohen et al., 2014). These instances use $k = 16$, 20 and 24, $n = 10k$ and user-independent (cardinality) constraints of three different types: we vary the number of constraints and the proportions of the different constraint types; each user is authorized for between 1 and 8 tasks for $k = 16$, between 1 and 10 tasks for $k = 20$, and between 1 and 12 tasks for $k = 24$. The algorithm was implemented in C++ and has been enhanced by the inclusion of techniques employed in CSP solving, such as propagation. We also converted WSP instances into pseudo-Boolean problems for processing by SAT4J. All experiments were performed on a MacBook Pro computer with a 2.6 GHz Intel Core i5 processor and 8 GB 1600 MHz DDR3 RAM (running Mac OS X 10.9.2).

For lightly-constrained instances, SAT4J was often faster than our algorithm, largely because the number of patterns considered by our algorithm is large for such instances. However, for highly-constrained instances, SAT4J was unable to compute a decision for a number of instances (because it ran out of memory), in sharp contrast to our algorithm which solved all instances. Overall, on average, our algorithm was faster than SAT4J and, in particular, was two orders of magnitude faster for $k = 16$. Moreover, the time taken by our algorithm varies much less than that of SAT4J, even for unsatisfiable instances, because the time taken is proportional to the product of the number of patterns and the number of users. (In particular, in tested instances, it is much less dependent on the number of constraints, a parameter that can cause significant fluctuations in the time taken by SAT4J because it leads to a sharp increase in the number of variables in the pseudo-Boolean encoding.) Full details of our results have been published (Cohen et al., 2014).

7. Conclusion

In this paper we introduced an algorithm based on the notion of plan-indistinguishability, applicable to a wide range of WSP instances. We showed that our algorithm is powerful enough to be optimal, in a sense, for the wide class of user-independent constraints. The generic algorithm is also a fixed-parameter algorithm for equivalence relation constraints, which are not user-independent. We showed how to deal with unions of different types of constraints using our generic algorithm. In particular, we proved that the generic algorithm is a fixed-parameter algorithm for the union of user-independent and equivalence relation constraints.

Acknowledgments

Our research was supported by EPSRC grant EP/K005162/1. We are grateful to the referees for their useful comments and suggestions.

References

American National Standards Institute (2004). *Role Based Access Control (ANSI INCITS RBAC 359-2012).*

Basin, D. A., Burri, S. J., & Karjoth, G. (2014). Obstruction-free authorization enforcement: Aligning security and business objectives. *Journal of Computer Security*, *22*(5), 661–698.

Beldiceanu, N., Carlsson, M., & Rampon, J.-X. (2012). Global constraint catalog, 2nd edition (revision a). working copy 5195, Swedish Institute of Computer Science, Kista, Sweden.

Berend, D., & Tassa, T. (2010). Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, *30*(2), 185–205.

Bertino, E., Bonatti, P. A., & Ferrari, E. (2001). TRBAC: a temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, *4*(3), 191–233.

Bertino, E., Ferrari, E., & Atluri, V. (1999). The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, *2*(1), 65–104.

Bodlaender, H. L., Cygan, M., Kratsch, S., & Nederlof, J. (2013). Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part I*, ICALP'13, pp. 196–207, Berlin, Heidelberg. Springer-Verlag.

Bulatov, A., Jeavons, P., & Krokhin, A. (2005). Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, *34*, 720–742.

Cohen, D., Crampton, J., Gagarin, A., Gutin, G., & Jones, M. (2014). Engineering algorithms for workflow satisfiability problem with user-independent constraints. In Chen, J., Hopcroft, J. E., & Wang, J. (Eds.), *Frontiers in Algorithmics - 8th International Workshop, FAW 2014, Zhangjiajie, China, June 28-30, 2014. Proceedings*, Vol. 8497 of *Lecture Notes in Computer Science*, pp. 48–59. Springer.

Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. (2001). *Introduction to Algorithms* (2nd edition). McGraw-Hill Higher Education.

Crampton, J., Crowston, R., Gutin, G., Jones, M., & Ramanujan, M. (2013). Fixed-parameter tractability of workflow satisfiability in the presence of seniority constraints. In Fellows, M., Tan, X., & Zhu, B. (Eds.), *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, Vol. 7924 of *Lecture Notes in Computer Science*, pp. 198–209. Springer Berlin Heidelberg.

Crampton, J. (2005). A reference monitor for workflow systems with constrained task execution. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, SACMAT '05, pp. 38–47, New York, NY, USA. ACM.

Crampton, J., & Gutin, G. (2013). Constraint expressions and workflow satisfiability. In *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*, SACMAT '13, pp. 73–84, New York, NY, USA. ACM.

Crampton, J., Gutin, G., & Yeo, A. (2013). On the parameterized complexity and kernelization of the workflow satisfiability problem. *ACM Trans. Inf. Syst. Secur.*, *16*(1), 4:1–4:31.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA 94104-3205.

Downey, R. G., & Fellows, M. R. (2013). *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Fomin, F. V., Lokshtanov, D., & Saurabh, S. (2014). Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pp. 142–151. SIAM.

Gligor, V., Gavrila, S., & Ferraiolo, D. (1998). On the formal definition of separation-of-duty policies and their composition. In *1998 IEEE Symposium on Security and Privacy, 1998. Proceedings.*, pp. 172–183.

Impagliazzo, R., Paturi, R., & Zane, F. (2001). Which problems have strongly exponential complexity?. *J. Comput. Syst. Sci.*, *63*(4), 512–530.

Jayaraman, K., Ganesh, V., Tripunitara, M. V., Rinard, M. C., & Chapin, S. J. (2011). ARBAC policy for a large multi-national bank. *CoRR*, *abs/1110.2849*.

Joshi, J. B. D., Bertino, E., Latif, U., & Ghafoor, A. (2005). A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering,*, *17*(1), 4–23.

Lokshtanov, D., Marx, D., & Saurabh, S. (2011). Slightly superexponential parameterized problems. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pp. 760–776. SIAM.

Niedermeier, R. (2006). *Invitation to Fixed-Parameter Algorithms*. Oxford University Press.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *The Handbook of Constraint Programming*. Elsevier.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, *29*(2), 38–47.

Schaad, A., Moffett, J., & Jacob, J. (2001). The role-based access control system of a european bank: A case study and discussion. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, pp. 3–9, New York, NY, USA. ACM.

Simon, R., & Zurko, M. (1997). Separation of duty in role-based environments. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pp. 183–194.

Wang, Q., & Li, N. (2010). Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur.*, *13*(4), 40:1–40:35.

Wolter, C., & Schaad, A. (2007). Modeling of task-based authorization constraints in bpmn. In *Proceedings of the 5th International Conference on Business Process Management*, BPM'07, pp. 64–79, Berlin, Heidelberg. Springer-Verlag.