On the Workflow Satisfiability Problem with Class-Independent Constraints for Hierarchical Organizations[1]

JASON CRAMPTON, ANDREI GAGARIN, GREGORY GUTIN, MARK JONES AND MAGNUS WAHLSTRÖM, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK

A workflow specification defines a set of steps, a set of users, and an access control policy. The policy determines which steps a user is authorized to perform and imposes *constraints* on which sets of users can perform which sets of steps. The *workflow satisfiability problem* (WSP) is the problem of determining whether there exists an assignment of users to workflow steps that satisfies the policy. Given the computational hardness of WSP and its importance in the context of workflow management systems, it is important to develop algorithms that are efficient as possible to solve WSP.

In this paper, we study the fixed-parameter tractability of WSP in the presence of class-independent constraints, which enable us to (i) model security requirements based on the groups to which users belong and (ii) generalize the notion of a user-independent constraint. Class-independent constraints are defined in terms of equivalence relations over the set of users. We consider sets of nested equivalence relations because this enables us to model security requirements in hierarchical organizations. We prove that WSP is fixed-parameter tractable (FPT) for class-independent constraints defined over nested equivalence relations and develop an FPT algorithm to solve WSP instances incorporating such constraints. We perform experiments to evaluate the performance of our algorithm and compare it with that of SAT4J, an off-the-shelf pseudo-Boolean SAT solver. The results of these experiments demonstrate that our algorithm significantly outperforms SAT4J for many instances of WSP.

Categories and Subject Descriptors: D4.6 [**Operating Systems**]: Security and Protection—*Access controls*; F2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; H2.0 [**Database Management**]: General—*Security, integrity and protection*

Additional Key Words and Phrases: workflow satisfiability problem; fixed-parameter tractability; user-independent constraints; class-independent constraints

ACM Reference Format:
Jason Crampton, Andrei Gagarin, Gregory Gutin, Mark Jones, and Magnus Wahlström. 2015. On the Workflow Satisfiability Problem with Class-Independent Constraints for Hierarchical Organizations *ACM Trans. Info. Syst. Sec.* V, N, Article A (January YYYY), 29 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

It is increasingly common for organizations to computerize their business and management processes. The co-ordination of the tasks or steps that comprise a computerized business process is managed by a workflow management system (or business process management system). A simple, illustrative example for purchase order processing [Crampton et al. 2013] is shown in Figure 1 (on page 3). In the first step of the workflow, the purchase order is created and approved (and then dispatched to the sup-

[1]A preliminary version of this paper appeared at IPEC 2015 [Crampton et al. 2015]. For a description of the differences between the two versions, see the last paragraph of Section 1.

plier). The supplier will submit an invoice for the goods ordered, which is processed by the create payment step. When the supplier delivers the goods, a goods received note (GRN) must be signed and countersigned. Only then may the payment be approved and sent to the supplier. Note that a workflow specification need not be linear: the processing of the GRN and of the invoice can occur in parallel, for example.

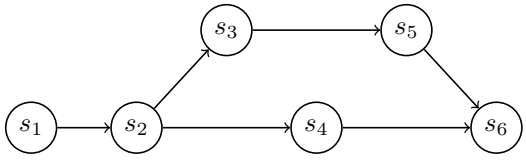
Typically, the execution of a step in a workflow instance will be triggered by a human user, or a software agent acting under the control of a human user, and each step may only be executed by an *authorized* user. Thus a workflow specification will include an *authorization policy* defining which users are authorized to perform which steps. Such a policy may be specified directly as a relation associating each user with those steps for which the user is authorized. Alternatively, one may specify a set of roles, associating each role with the steps for which that role is authorized, and associating each user with the roles for which the user is authorized. We could, for example, define a `financeClerk` role which is authorized for the create payment step.

In addition, many workflows require controls on the users that perform certain sets of steps [American National Standards Institute 2004; Basin et al. 2014; Bertino et al. 1999; Brewer and Nash 1989; Crampton 2005; Wang and Li 2010]. In the context of our purchase order workflow, these controls seek to prevent fraudulent use of the purchase order processing system. The controls are specified as constraints on users that can perform pairs of steps in the workflow. Examples include the same user must not sign and countersign the GRN; and the same user must create the purchase order and sign the associated GRN. These and other similar constraints are illustrated in Figure 1(b). Such constraints are said to be *user-independent* because the satisfaction of these constraints is not dependent on specific user identities. The first constraint, for example, is satisfied provided *any* two users are involved in signing and countersigning the GRN.

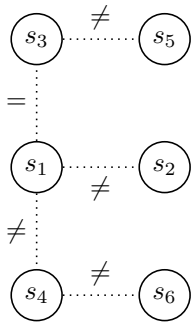
However, there are constraints, of practical value, that are not user-independent. Similarly, there are business rules that cannot be expressed using user-independent constraints. It is common, for example, for an organization to be partitioned into departments (or similar) and each department to be partitioned into sections (or similar). Then we might wish to impose the following rules on purchase order processing.

— The create purchase order and sign GRN steps in a workflow instance must be performed by users in the same section. (This replaces, and is a weakening of, the requirement that these steps be performed by the same user.)
— The create and approve purchase steps in a workflow instance must be performed by users in the same department. (This augments the constraint that the users performing these steps must be different.)
— The sign and countersign GRN steps in a workflow instance must be performed by users in the same department but those users must not belong to the same section. (This is a strengthening of the requirement that these steps be performed by different users.)
— The create purchase order and create payment steps must be performed by users in different departments. (This replaces, and is a strengthening of, the requirement that these steps be performed by different users.)
— The approve purchase order and approve payment steps must be performed by users in different departments. (This is an additional constraint.)

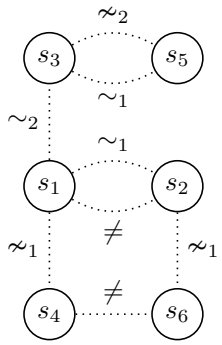
There is little work in the literature on constraints of this form, although prior work has recognized that such constraints are likely to be important in practice [Crampton 2005; Wang and Li 2010], and it has been shown that even simple constraints of this form present additional difficulties when incorporated into WSP [Crampton et al. 2013].



(a) Ordering on steps



(b) User-independent constraints



(c) Class-independent constraints

Steps		Constraint types	
s_1	create purchase order	$\neq$	different users must perform steps
s_2	approve purchase order	$=$	same user must perform steps
s_3	sign GRN	$\nsim_1$	users in different departments must perform steps
s_4	create payment	$\sim_1$	users in the same department must perform steps
s_5	countersign GRN	$\nsim_2$	users in different sections must perform steps
s_6	approve payment	$\sim_2$	users in the same section must perform steps

(d) Legend

Fig. 1. A simple constrained workflow for purchase order processing

It is also important to note that there is no obvious way to enforce the requirements listed above by specifying an authorization policy. In particular, role-based access control – a natural choice for specifying an authorization policy for a workflow (by authorizing roles to perform steps and users to activate roles) – does not distinguish between users, much less sections or departments. Suppose, for example, that every user at the clerk level in a hierarchical organization (having many different departments) is authorized to create purchase orders (step s_1) and sign GRNs (step s_3). It would be natural to define a role r_1, which is authorized for steps s_1 and s_3, and authorize clerks for this role. Similarly, we might authorize every user at administrator level for role r_2, which is authorized to countersign GRNs (step s_5).

We note, in passing, that we could assign users to "organizational" (rather than "functional") roles on the basis of membership in different departments and sections. Thus, we can distinguish between two users u_1 and u_2, each assigned to role r_1, and belonging to departments d_1 and d_2, respectively. However, to ensure that two steps s_1 and s_4 are executed by users assigned to r_1 but belonging to different departments by specifying constraints on the organizational roles involved in the execution of s_1 and s_4, we would need to specify multiple constraints, one for each pair of department roles. In other words, while it may be possible to articulate some of the constraints in Figure 1 in terms of roles, it is more natural and compact to express these constraints in terms of equivalence relations on the set of users, and checking the satisfaction of

those constraints will be more efficient. More generally, roles (in the context of role-based access control) and constraints defined in terms of organizational structure are best suited for enforcing rather different security requirements.

We model hierarchical organization structures as nested equivalence relations (each of which induces a partition of the set of users). We then generalize the definition of a user-independent constraint to a class-independent constraint. Returning to our purchase order processing example, let us write $u \sim_1 v$ if and only if users u and v belong to the same department, and $u \sim_2 v$ if and only if u and v belong to the same section. Then we may represent the constraints listed above in terms of $\sim_1$ and $\sim_2$, as shown schematically in Figure 1(c).

The existence of an authorization policy and constraints on the execution of workflow steps may mean it is impossible to find a *valid plan* – an assignment of authorized users to workflow steps such that all constraints are satisfied. The WORKFLOW SATISFIABILITY PROBLEM (WSP) takes a workflow specification as input and outputs a valid plan if one exists. WSP is known to be NP-hard, even when the set of constraints only includes very simple constraints [Wang and Li 2010]. Clearly, it is important to be able to determine whether a workflow specification is satisfiable at design time. Equally, when users select steps to execute in a workflow instance, it is essential that the access control mechanism can determine whether (a) the user is authorized to execute the step, and (b) allowing the user to execute the step does not render the instance unsatisfiable. Thus, the access control mechanism must incorporate an algorithm to solve WSP, and that algorithm needs to be as efficient as possible.

Wang and Li [2010] observed that, in practice, the number k of steps in a workflow will be small, relative to the size of the input to WSP; specifically, the number of users is likely to be an order of magnitude greater than the number of steps. This observation led them to set k as the parameter and to study the problem using tools from parameterized complexity. In doing so, they proved that the problem is *fixed-parameter tractable* (FPT) for simple classes of constraints. However, Wang and Li also showed that in general WSP is fixed-parameter *intractable* (subject to a widely accepted complexity theory hypothesis). Recent research has made significant progress in identifying those classes of constraints for which WSP remains FPT. In particular, Cohen et al. [2014] proved that WSP is FPT if all constraints are user-independent. We provide a brief overview of fixed-parameter tractability and known results for WSP in Section 2.4.

In this paper, we extend the notion of a user-independent constraint to that of a *class-independent* constraint with respect to an equivalence relation on the set of users (such as belonging to the same department). In particular, every user-independent constraint is an instance of a class-independent constraint with respect to the identity relation. An equivalence relation $\sim$ is said to be a *refinement* of an equivalence relation $\sim'$ if any two elements that are equivalent under $\sim$ are also equivalent under $\sim'$. A sequence $\{\sim_1, \ldots, \sim_r\}$ of equivalence relations is said to be *nested* if $\sim_{q+1}$ is a refinement of $\sim_q$ for each $q \in \{1, \ldots, r-1\}$. Our second contribution is to demonstrate that patterns for user-independent constraints [Cohen et al. 2014] can be generalized to patterns for class-independent constraints with respect to a set of nested equivalence relations – that is, every constraint is class-independent with respect to an equivalence relation $\sim_q$ from a nested sequence $\{\sim_1, \ldots, \sim_r\}$. The resulting algorithm, using these new patterns, remains FPT (irrespective of the authorization policy), although its running time is somewhat slower than that of the algorithm for WSP with user-independent constraints only (and r must be included as an additional parameter). In short, our first two contributions identify a large class of constraints for which WSP is shown to be FPT, and subsume prior work on user-independent constraints [Crampton et al. 2013; Cohen et al. 2014; Karapetyan et al. 2015; Wang and Li 2010]. Our final

contribution is an implementation of our algorithm in order to investigate whether the theoretical advantages implied by its fixed-parameter tractability can be realized in practice. We compare the performance of our FPT algorithm with that of SAT4J, an off-the-shelf pseudo-Boolean SAT solver, for solving WSP. The results of our experiments suggest that our FPT algorithm enjoys some significant advantages over SAT4J for hard instances of WSP.

In the next section, we define WSP and user-independent constraints in more formal terms, discuss related work in more detail, and introduce the notion of class-independent constraints. In Section 3 we state our main theoretical result, which is that WSP is fixed-parameter tractable when all constraints are class-independent with respect to nested equivalence relations on r levels. The foundation of our main algorithm is the concept of nested equivalence relation patterns. We establish that every nested equivalence relation plan has a pattern, and that to check whether a plan satisfies all constraints it is enough to consider the nested equivalence relation pattern corresponding to that plan. Thus our main algorithm will focus on nested equivalence relation patterns, as the number of such patterns is much smaller than the number of possible plans. In Section 4, we describe an implementation of our algorithm and report the results of our experiments. We summarize our results in Section 5.

There are four main differences between this version of the paper, and the earlier conference version [Crampton et al. 2015]:

(1) we provide a full description of our approach for an arbitrary number r of nested equivalence relations, unlike the conference version which only considers the case $r = 2$;
(2) we significantly improve the upper bound on the complexity of our algorithm with respect to r (our new bound indicates that the dependency of the running time on r is much weaker than previously assumed);
(3) we improve the efficiency of our algorithm implementation; and
(4) we consider a wider range of WSP instances in our computational experiments, specifically considering WSP instances with fewer constraints and larger numbers of steps.

2. PRELIMINARIES AND RELATED WORK

Work on authorization constraints as a means of implementing security requirements in workflow systems was first studied in the seminal paper by Bertino et al. [1999]. A considerable amount of research now exists on the topic, extending the work of Bertino et al. in a variety of ways, including more complex constraints [Crampton et al. 2013; Cohen et al. 2014]; more complex control flow in the workflow specification [Basin et al. 2014; Yang et al. 2014]; and the complexity of algorithms required to evaluate authorization requests [Crampton et al. 2013; Cohen et al. 2014; Karapetyan et al. 2015; Wang and Li 2010].

In this paper, we extend results on the fixed-parameter tractability of WSP to include workflow specifications containing class-independent constraints. Accordingly, we adopt an existing and well-established model for a constrained workflow authorization schema [Crampton et al. 2013].

A *workflow specification* is a partially ordered set of steps $(S, \leqslant)$. The partial order determines the *control flow*: that is, the order in which workflow steps must be executed. In particular, if $s < s'$ then s must be performed before s' in any instance of the workflow; and if $s \not< s'$ and $s \not> s'$ then s and s' may be performed in either order. Our definition of workflow specification does not permit repetition of steps (loops) or repetition of sub-workflows (cycles). Nor do we consider workflows where the control flow includes exclusive branching: that is, of two parallel branches only one branch is

executed in any given instance of the workflow. Informally, we omit control flow patterns that mean the set of steps in a given workflow instance is not fully determined in advance. However, these omissions do not have significant theoretical implications. Recent work has shown that workflow specifications with control flows that lead to indeterminism can be handled by "unwinding" the workflow specification to produce multiple (deterministic) workflow specifications [Crampton and Gutin 2013; Yang et al. 2014].

Given a workflow specification $(S, \leqslant)$ and a set of users U, an *authorization policy* for $(S, \leqslant)$ is a relation $A \subseteq S \times U$, which we may represent as a set of authorization lists $\mathcal{A} = \{A(u) : u \in U\}$, where $A(u) = \{s \in S : (s, u) \in A\}$. User u is authorized to perform step s only if $(s, u) \in A$.[2] We assume that for every step $s \in S$ there exists some user $u \in U$ such that $(s, u) \in A$. A *workflow authorization schema* is a tuple $(S, U, \leqslant, A)$, where $(S, \leqslant)$ is a workflow specification and A is an authorization policy for $(S, \leqslant)$.

We are interested in assigning users to steps subject to a given authorization policy and constraints. We may be interested in simply confirming that such an assignment exists to ensure that workflow instances can be completed. However, in a more dynamic setting, we may also be interested in determining whether allowing a user's request to execute a particular step in a workflow instance would make it impossible to complete the workflow instance (while complying with the authorization policy and constraints).

In other words, among the set $\Pi(S, U)$ of all functions from S to U, there are some that represent "legitimate" assignments of steps to users and some that do not. The legitimacy or otherwise of a particular assignment is determined by the authorization policy and the constraints that complete the workflow specification. Formally, a *constraint* $c \in C$ may be viewed as a pair (T, Θ), where $T \subseteq S$ is the *scope* of c and Θ is a set of functions from T to U, specifying the assignments of steps in T to users in U that satisfy the constraint. In practice, we do not enumerate all the elements of Θ. Instead, we define its members implicitly using some constraint-specific syntax. For example, we write (s, s', ρ), where $s, s' \in S$ and ρ is a binary relation defined on U, to denote a constraint that has scope $\{s, s'\}$ and is satisfied by any assignment $\pi : S \to U$ such that $(\pi(s), \pi(s')) \in \rho$. In particular, $(s, s', \neq)$ requires s and s' to be performed by different users (and so represents a separation-of-duty constraint). Similarly, $(s, s', =)$ requires that s and s' be performed by the same user (a binding-of-duty constraint).

2.1. The Workflow Satisfiability Problem

A *plan* is a function in $\Pi(S, U)$. Given a *workflow* $W = (S, U, \mathcal{A}, C)$, a plan π is *authorized* if for all $s \in S$, $s \in A(\pi(s))$, i.e. the user assigned to s is authorized for s. A plan π is *eligible* if for all $(T, \Theta) \in C$, $\pi|_T \in \Theta$, i.e. every constraint is satisfied.[3] A plan π is *valid* if it is both authorized and eligible. In the *workflow satisfiability problem* (WSP), we are given a workflow (specification) W, and our aim is to decide whether W has a valid plan. If W has a valid plan, W is *satisfiable*; otherwise, W is *unsatisfiable*.[4]

---

[2] In practice, the set of authorized step-user pairs, A, will not be defined explicitly. Instead, A will be inferred from other access control data structures. In particular, R^{2}BAC – the role-and-relation-based access control model of Wang and Li [2010] – introduces a set of roles R, a user-role relation $UR \subseteq U \times R$ and a role-step relation $SA \subseteq R \times S$ from which it is possible to derive the steps for which users are authorized. For all common access control policies (including R^{2}BAC), it is straightforward to derive A. We prefer to use A in order to simplify the exposition.

[3] Given a plan $\pi : S \to U$ and a subset T of S, $\pi|_T$ denotes the function with domain T such that $\pi|_T(s) = \pi(s)$ for all $s \in T$.

[4] WSP has many similarities to the Constraint Satisfaction Problem (CSP). In fact, WSP, is exactly the *conservative* CSP (i.e., CSP with arbitrary unary constraints, corresponding to step authorizations in the WSP terminology). However, unlike a typical instance of CSP, where the number of variables is significantly

Notice that our definition of a plan is independent of the ordering on the steps. We could, in principle, define a plan to be a pair (π, ℓ), where ℓ is a list of steps, indicating the order in which steps should be executed (for that plan). For such a definition to be meaningful, this would imply that for some $\pi : S \to U$, there exist lists of steps ℓ and ℓ' such that (π, ℓ) is a valid plan and (π, ℓ') is not. We have never seen any constraints or workflow specifications, either in practice or in the research literature, where this situation arises. Henceforth, therefore, we will ignore the ordering on workflow steps, as it is irrelevant to the construction of a valid plan.

We assume that in all instances of WSP we consider, all constraints can be checked in time polynomial in n. Thus it takes polynomial time to check whether any plan is eligible. The correctness of our algorithm is unaffected by this assumption, but using constraints not checkable in polynomial time would naturally affect the running time. All constraints of practical interest known to us satisfy this assumption.

Table I. A simple authorization policy

	$dept_1$					$dept_2$			
	$section_{11}$		$section_{12}$			$section_{21}$		$section_{22}$	
	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
s_1	1		1	1	1	1			
s_2		1						1	1
s_3	1		1						1
s_4							1	1	
s_5				1	1			1	
s_6								1	1

Example 2.1. Table I shows a simple authorization policy. Consider this policy in conjunction with the workflow specification and the user-independent constraints shown in Figures 1(a) and 1(b), respectively. Then the following plan is valid:

$$\pi(s_1) = \pi(s_3) = u_1;\ \pi(s_2) = \pi(s_4) = \pi(s_5) = u_8;\ \pi(s_6) = u_9. \tag{1}$$

Table I also illustrates the partitioning of the user set into departments and sections. Note that we cannot construct a valid plan using only users from $dept_1$, since no user is authorized for s_6. Equally, we cannot construct a valid plan using only users from $dept_2$, since no user is authorized for both s_1 and s_3.

A *partial plan* π is a function from a subset T of S to U. In particular, a plan is a partial plan. To avoid confusion, we may refer to a plan $\pi : S \to U$ as a *complete plan*. We extend the definitions of *eligible*, *authorized* and *valid* to partial plans in the natural way: the only difference is that we only consider authorizations for steps in T and constraints with scope being a subset of T.

An algorithm to solve WSP is important because it can establish whether a workflow specification is satisfiable. Clearly, a specification that is not satisfiable is of little practical use.[5] Moreover, we can use the algorithm as a sub-routine in a run-time reference monitor, to determine whether allowing a user to execute a workflow step would render a workflow instance unsatisfiable [Crampton and Gutin 2013, Section 2.2]. Hence, it is important to develop algorithms to solve WSP that are as efficient as possible.

larger than the number of values, a typical instance of WSP has many more values (i.e., users) than variables (i.e., steps).

[5]Note we can test whether a specification is satisfiable in the absence of an authorization policy. Any plan requires at most k users, so if no authorization policy is available, we may simply consider a set of k "abstract" users, each of which is authorized for all steps. If the resulting specification is unsatisfiable, the specification will be unsatisfiable for any other authorization policy.

2.2. Constraints using Equivalence Relations

Crampton et al. [2013] introduced constraints defined in terms of an equivalence relation $\sim$ on U: a plan π satisfies constraint $(s, s', \sim)$ if $\pi(s) \sim \pi(s')$ (and satisfies constraint $(s, s', \nsim)$ if $\pi(s) \nsim \pi(s')$). Hence, we could, for example, specify the pair of constraints $(s, s', \neq)$ and $(s, s', \sim)$, which, collectively, require that s and s' are performed by different users that belong to the same equivalence class with respect to $\sim$. As we noted in the introduction, such constraints are very natural in the context of organizations that partition the set of users into departments, sections, groups or teams.

Crampton et al. [2013, Theorem 5.4] proved that WSP remains FPT when some simple extensions of constraints $(s, s', \sim)$ and $(s, s', \nsim)$ are included. Our extension of constraints $(s, s', \sim)$ and $(s, s', \nsim)$ is much more general: it is similar to generalizing simple constraints $(s, s', =)$ and $(s, s', \neq)$ to the wide class of user-independent constraints [Cohen et al. 2014]. This leads, in particular, to a significant generalization of Theorem 5.4 in [Crampton et al. 2013].

Let $c = (T, \Theta)$ be a constraint and let $\sim$ be an equivalence relation on U. Let $U^{\sim}$ denote the set of equivalence classes induced by $\sim$ and let $u^{\sim} \in U^{\sim}$ denote the equivalence class containing u. Then, for any function $\pi : S \to U$, we may define the function $\pi^{\sim} : S \to U^{\sim}$, where $\pi^{\sim}(s) = (\pi(s))^{\sim}$. If, for example, $\pi(s_1) = u_1$, then (using the partitions defined in Table I), we have

$$\pi^{\sim_1}(s_1) = dept_1 \quad \text{and} \quad \pi^{\sim_2}(s_1) = section_{11}.$$

In particular, $\sim$ induces a set of functions $\Theta^{\sim} = \{\theta^{\sim} : \theta \in \Theta\}$.

Informally, given an equivalence relation $\sim$ on U, we say a constraint c is class-independent if, for each plan π that satisfies c, for each step s, and for each permutation π of equivalence classes in $U^{\sim}$, replacing $\pi(s)$ by any user in the class $\phi(\pi(s)^{\sim})$ results in a valid plan. More formally, $c = (T, \Theta)$ is *class-independent* (CI) *for* $\sim$ (or simply $\sim$*-independent*) if for any function θ, $\theta^{\sim} \in \Theta^{\sim}$ implies $\theta \in \Theta$, and for any permutation $\phi : U^{\sim} \to U^{\sim}$, $\theta^{\sim} \in \Theta^{\sim}$ implies $\phi \circ \theta^{\sim} \in \Theta^{\sim}$. In particular, constraints of the form $(s, s', \sim)$ and $(s, s', \nsim)$ are $\sim$-independent.

We say a constraint is *user-independent* (UI) if it is CI for the identity relation. (In particular, every UI constraint is CI.) In other words, if a plan $\pi : s \mapsto \pi(s)$ satisfies a UI constraint c and we replace any user in $\{\pi(s) : s \in S\}$ by an arbitrary user such that the replacement users are all distinct, then the new plan satisfies c. Many constraints of practical interest, such as separation-of-duty, binding-of-duty and cardinality constraints are user-independent.

Example 2.2. Table I illustrates two partitions of the set of users, one into two departments, the other into four sections. Now consider these partitions and the class-independent constraints defined in Figure 1(c). We can satisfy all the constraints in Figure 1(c) by:

— allocating users from $dept_1$ to steps s_1, s_2, s_3 and s_5, ensuring that the users allocated to steps s_1 and s_2 are different and the users allocated to steps s_3 and s_5 belong to different sections;
— allocating two users from $dept_2$ to steps s_4 and s_6, ensuring that the users allocated to these steps are different.

Notice that we could equally well allocate steps s_1, s_2, s_3 and s_5 to users from $dept_2$ and steps s_4 and s_6 to users from $dept_1$. Equally, it doesn't matter which specific sections the users allocated to steps s_3 and s_5 belong to, only that the sections are distinct. In other words, these are class-independent constraints.

In terms of constructing a plan, notice that the plan defined in (1) is no longer valid because it violates the constraints $(s_1, s_2, \sim_2)$ and $(s_3, s_5, \sim_1)$. The workflow is, never-

theless, satisfiable. The following plan, for example, is valid:

$$\pi_1(s_1) = \pi_1(s_3) = u_1;\ \pi_1(s_2) = u_2;\ \pi_1(s_5) = u_4;\ \pi_1(s_4) = u_8;\ \pi_1(s_6) = u_9. \quad (2)$$

2.3. Nested equivalence relations and WSP

Many organizations have a hierarchical structure and may wish to define operational constraints based on that structure. We now explain, based on material previously published by Crampton et al. [2013], how such structures may be modeled using equivalence relations.

We will write $[m]$ to denote the set $\{1, \ldots, m\}$ for any integer $m \geqslant 1$. For a set X, we say $(B_1, \ldots, B_\ell)$ is a *partition of* X if $B_1 \cup \cdots \cup B_\ell = X$, $B_i \neq \emptyset$ all $i \in [\ell]$, and $B_i \cap B_j = \emptyset$ for all $i \neq j \in [\ell]$. We refer to the sets B_i as *blocks*. Let $\mathcal{B} = (B_1, \ldots, B_\ell)$ and $\mathcal{B}' = (B'_1, \ldots, B'_{\ell'})$ be partitions of X. We say that $\mathcal{B}'$ is a *refinement* of $\mathcal{B}$ if every block of $\mathcal{B}'$ is contained in a block of $\mathcal{B}$. The set of equivalence classes of an equivalence relation on X defines a partition of X. Hence, we may speak unambiguously of one equivalence class being a refinement of another. We say a sequence $(\mathcal{P}_1, \ldots, \mathcal{P}_r)$ of partitions of a set X is a *nested partition* (*in r levels*) if $\mathcal{P}_i$ is a refinement of $\mathcal{P}_j$ for all $i > j$. Similarly, we say a series of equivalence relations $\sim_1, \ldots, \sim_r$ is *nested* if the partition of X defined by $\sim_i$ is a refinement of the partition of X defined by $\sim_j$ for all $i > j$, i.e. the partitions induced by the equivalence relations form a nested partition. Two special cases are worth mentioning: the trivial relation, where $u \sim v$ if and only if u and v belong to the set of users; and the equality relation, where $u \sim v$ if and only if $u = v$. The former equivalence relation induces a single equivalence class containing every user, so any other partition of U is a refinement of this partition. The latter induces an equivalence class for each user, and the resulting partition refines every other partition.

Modern organizations typically have a hierarchical structure, the organization being successively divided into increasingly fine-grained organizational units. An organization might, for example, have divisions, which are split into departments, which are split into sections, which are split into teams. We may represent such an organization by defining a number of nested equivalence relations: $u \sim_1 v$ iff u and v belong to the same organization, $u \sim_2 v$ iff u and v belong to the same division, etc. A simple example of this kind of organization into departments and sections (and individual users) is shown in Table I.

As we illustrated in Figure 1(c), many security requirements can be expressed in terms of constraints of the form "the users that perform these steps must belong to different equivalence classes" or "the users that perform these steps must belong to the same equivalence class"; all such constraints are class-independent. (Moreover, such constraints cannot be expressed in terms of user-independent constraints.) It is possible, for example, to specify the requirement that two steps are performed by users belonging to the same department but to different sections within that department. In short, the use of nested equivalence relations to model organizations and constraints defined over the resulting equivalence classes extend considerably the expressivity of workflow specifications.

ASSUMPTION 1. *Unless stated otherwise, we assume throughout the paper that all constraints are $\sim_q$-independent for some $q \in [r]$. Furthermore, we assume we are given a series of nested relations $\{\sim_1, \ldots, \sim_r\}$ as part of the input.*

ASSUMPTION 2. *We may assume without loss of generality that $\sim_r$ is the equality relation; that is, all $\sim_r$-independent constraints are user-independent.*[6]

[6]If this assumption did not hold then we may always add the equality relation as $\sim_{r+1}$ (which trivially refines $\sim_r$), with no user-independent constraints. This has the effect on our algorithm's running time of increasing r by 1.

Recall the definitions of $U^{\sim}, u^{\sim}, \pi^{\sim}$ with respect to an equivalence relation $\sim$. When considering the sequence $(\sim_1, \ldots, \sim_r)$ of equivalence relations, we will abuse notation slightly in the interests of clarity and write $U^{\sim q}$ in place of $U^{\sim_q}$. Similarly, we will write $u^{\sim q}$ for $u^{\sim_q}$ and $\pi^{\sim q}$ for $\pi^{\sim_q}$.

We conclude this section with a claim whose simple proof is omitted.

PROPOSITION 2.3. Given two equivalence relations $\sim$ and $\approx$ such that $\sim$ is a refinement of $\approx$, and any plan $\pi : S \to U$, $\pi^{\sim}(s) = \pi^{\sim}(s')$ implies $\pi^{\approx}(s) = \pi^{\approx}(s')$.

2.4. FPT Results for WSP

Wang and Li [2010] showed that WSP is NP-hard by a reduction from GRAPH COLORING. However, they also noted that the number of steps k is generally much smaller than the size of the input to WSP. In particular, k is typically an order of magnitude smaller than the number of users n. Accordingly, they studied the fixed-parameter tractability of WSP using k as the parameter. We now provide a brief introduction to fixed-parameter tractability and a survey of the FPT results that are known for WSP.

An algorithm that solves an NP-hard problem in time $O(f(k)N^d)$, where N denotes the size of the input to the problem, k is some (small) parameter of the problem, f is some function in k only, and d is some constant (independent of k and N), is said to be *fixed-parameter tractable* (FPT). If a problem can be solved using an FPT algorithm then we say that it is an *FPT problem* and that it belongs to the class FPT. Throughout the paper, we will use the O^* notation to suppress polynomial factors: that is, a function $g(n, r, k) = O^*(f(n, r, k))$ if $g(n, r, k) = O(f(n, r, k)(n + r + k)^d)$ for some constant d. Throughout this paper, all log values are given in base 2. The total size of the input to WSP is determined by n, k, c and r, denoting the number of users, steps, constraints and equivalence relations, respectively.

There exists an infinite collection of parameterized complexity classes, $\mathrm{W}[1], \mathrm{W}[2], \ldots$, with FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots$. Informally, a parameterized problem belongs to the complexity class W[i] if there exists an FPT algorithm that transforms every instance of the problem into an instance of WEIGHTED CIRCUIT SATISFIABILITY for a circuit of depth $i + 1$, of which i levels have unbounded fan-in. FPT is the class W[0]; the problem INDEPENDENT SET, for example, is in W[1]. It is widely-believed and generally assumed that FPT $\neq$ W[1] (see, for example, Flum and Grohe [2006]).

Wang and Li [2010, Lemma 8] showed, using an elementary argument, that WSP is FPT when all constraints have the form $(s, s', \neq)$ and can be solved in time $O^*(k^{k+1})$. They also proved that WSP is W[1]-hard in general, using a reduction from INDEPENDENT SET [Wang and Li 2010, Theorem 10]. Crampton et al. [2013] introduced the notion of a "regular" constraint[7], subsuming those constraints studied by Wang and Li for which WSP was known to be FPT. They developed an FPT algorithm to solve WSP when all constraints are regular whose running time, $O^*(2^k)$, significantly improved on that obtained by Wang and Li. They also studied WSP in the presence of constraints defined over nested equivalence relations defined on the set of users and proved that WSP remains FPT.

Recent work by Cohen et al. [2014] has introduced the notion of a *pattern*, an abstraction of a plan, which leads to a significant reduction in the search space and improved running times for FPT algorithms for WSP. As a result, Cohen et al. [2014] were able to show that WSP is FPT when all constraints are UI. Using a modified pattern approach, Karapetyan et al. [2015] obtained a short proof that WSP with only UI constraints is FPT and designed a very efficient algorithm for WSP with UI con-

[7]Regular constraints are a special family of UI constraints.

straints. In this paper, we show that WSP remains FPT when all constraints are CI and we generalize patterns to develop the fastest known FPT algorithms for WSP in the presence of constraints defined over nested equivalence relations.

3. AN FPT ALGORITHM FOR WSP WITH CI CONSTRAINTS

In this section, we will prove our main theoretical result:

THEOREM 3.1. *WSP with nested class-independent constraints in r levels and with k steps is FPT with a running time of $O^*(2^{k \log(rk)})$.*

In particular, note that when r is small enough relative to k (e.g. when r is a constant), the dominant term in the exponent is just $k \log k$. Recently, it was shown that (under a standard complexity assumption) even the basic case of WSP with only UI constraints admits no algorithm with a running time of $O^*(2^{(1-\varepsilon)k \log k})$ for any $\varepsilon > 0$ [Gutin and Wahlström 2015]. Hence, for small r, the bound in Theorem 3.1 is close to optimal.

In Section 3.1 we introduce the concept of patterns, which form the foundation of our main algorithm. We establish that every plan has a pattern, and that to check whether a plan is eligible it is enough to consider its pattern. The pattern of a plan depends on the type of constraints under consideration. In our case, for each equivalence relation $\sim_q$, $q \in [r]$, we define a *$\sim_q$-pattern* for any plan, and then define a *nested equivalence relation* (NER-) *pattern* as a sequence of $\sim_q$-patterns.

As the number of patterns is much smaller than the number of possible plans, we will solve WSP by considering patterns instead of plans. We will show in Section 3.1 that the eligibility of a plan can be determined from its NER-pattern. In Section 3.2, we describe an efficient method of enumerating a minimal set of NER-patterns that need to be considered. Given an eligible NER-pattern, it is not necessarily the case that a WSP instance has a solution, as there may be no plan with the corresponding NER-pattern that is also authorized. Thus we also have to check whether a given NER-pattern is the NER-pattern of an authorized plan. We say that such a pattern is *realizable*, and we show in Section 3.3 how to check whether a NER-pattern is realizable. These results are combined to give our main algorithm in Section 3.4.

3.1. Plans and Patterns

If a constraint is CI then, by definition, the mapping of steps to equivalence classes by a plan may be assumed to be independent of the specific equivalence classes chosen. All that matters is that certain steps are mapped to the same equivalence class, while other are mapped to different ones. In the case of the constraints shown in Figure 1(c), for example, any eligible plan must map the create purchase order and approve purchase order steps to (users in) the same section, while the sign GRN and countersign GRN steps must be allocated to different sections. (Notice that the department-level constraint that the create purchase order and create payment steps be executed by users in different departments necessarily requires that these steps are executed by users in different sections, because of the nesting of equivalence relations.) The concept of a pattern, which provides a "template" for an eligible plan has proved to be very important in the study of WSP with UI constraints. In this section, we explain how patterns can be generalized to WSP with constraints defined over nested equivalence relations.

Let $W = (S, U, \mathcal{A}, C = C_1 \cup \cdots \cup C_r)$ be a workflow, where C_q is a set of $\sim_q$-independent constraints for each $q \in [r]$. Consider a vector $p = (x_1, \ldots, x_k)$, where $x_i \in [k]$ for all $i \in [k]$. Such a vector may be used to define an equivalence relation $\approx_p$ on S, where $s_i \approx_p s_j$ if and only if $x_i = x_j$. Conversely, any plan $\pi : S \to U$ may be used to define an equivalence relation $\approx_{\pi,q}$ on S, where $s_i \approx_{\pi,q} s_j$ if and only if $\pi(s_i) \sim_q \pi(s_j)$. We

say p is a $\sim_q$*-pattern* for π if the set of equivalence classes induced by $\approx_p$ equals the set induced by $\approx_{\pi,q}$. We say p is *eligible for* C_q if any plan π with p as its $\sim_q$-pattern is eligible for C_q. An (eligible) pattern for $\sim_2$ in our running example is $(1,1,1,2,3,2)$, indicating that steps s_1, s_2 and s_3 should be allocated to the same section, s_5 should be allocated to a different section, and s_4 and s_6 should be allocated to a third distinct section; the plan π_1, defined in (2), has this $\sim_2$-pattern.

PROPOSITION 3.2. Let p_q be a $\sim_q$-pattern for a plan π, for some $q \in [r]$. Then p_q is eligible for C_q if and only if π is eligible for C_q.

PROOF. Suppose that π is eligible for C_q. We show that for any plan π_0 that has p_q as its $\sim_q$-pattern, π_0 is eligible for C_q, and so p_q is eligible for C_q.

Let $p_q = (p_{(q,1)}, \ldots, p_{(q,k)})$. Observe that for any s_i, s_j, we have

$$\begin{aligned}\pi(s_i) \sim_q \pi(s_j) &\Leftrightarrow p_{(q,i)} = p_{(q,j)}\\ &\Leftrightarrow \pi_0(s_i) \sim_q \pi_0(s_j)\end{aligned}$$

Then define a permutation $\phi : U^{\sim q} \to U^{\sim q}$ as follows: $\phi(u^{\sim q}) = \pi_0^{\sim q}(s_i)$ if there exists $s_i \in S$ such that $\pi^{\sim q}(s_i) = u^{\sim q}$, and $\phi(u^{\sim q}) = u^{\sim q}$ otherwise. As $\pi_0^{\sim q}(s_i) = \pi_0^{\sim q}(s_j)$ for any s_i, s_j such that $\pi^{\sim q}(s_i) = \pi^{\sim q}(s_j) = u^{\sim q}$, ϕ is well-defined. Furthermore $\pi_0^{\sim q} = \phi \circ \pi^{\sim q}$.

Then it follows from the definition of a class-independent constraint that for any $c = (T, \Theta) \in C_q$,

$$\begin{aligned}\pi|_T \in \Theta &\Leftrightarrow \pi^{\sim q}|_T \in \Theta^{\sim q}\\ &\Leftrightarrow \phi \circ (\pi^{\sim q}|_T) \in \Theta^{\sim q}\\ &\Leftrightarrow \pi_0^{\sim q}|_T \in \Theta^{\sim q}\\ &\Leftrightarrow \pi_0|_T \in \Theta.\end{aligned}$$

Since π satisfies every constraint in C_q, π_0 satisfies every constraint in C_q and so π_0 is eligible for C_q, as required.

For the converse, it follows by definition that if p_q is eligible for C_q then π is eligible for C_q. □

Now let $p = (p_1, \ldots, p_r)$ be a series of patterns (where for each $q \in [r]$, p_q is to be interpreted as a $\sim_q$-pattern). Then we call p a *nested equivalence relation* (NER-) *pattern*. We say that p is a *NER-pattern for* π if p_q is a $\sim_q$-pattern for π, for each $q \in [r]$. We say that p is *eligible for* $C = C_1 \cup \cdots \cup C_r$ if p_q is eligible for C_q for each $q \in [r]$.

Example 3.3. Consider the plan from Example 2.2:

$$\pi_1(s_1) = \pi_1(s_3) = u_1;\ \pi_1(s_2) = u_2;\ \pi_1(s_5) = u_4;\ \pi_1(s_4) = u_8;\ \pi_1(s_6) = u_9.$$

Then the following patterns are associated with this plan:

— for $\sim_1$ we have the pattern $(1,1,1,2,1,2)$ (since the steps s_1, s_2, s_3 and s_5 are mapped to users in $dept_1$, while the remaining two steps are mapped to users in $dept_2$);
— for $\sim_2$ we have the pattern $(1,1,1,2,3,2)$ (here s_5 is mapped to a user in a different section from the users performing s_1, s_2 and s_3);
— and for $=$ (that is, $\sim_3$ in this example) we have the pattern $(1,2,1,3,4,5)$.

Thus the NER-pattern associated with this plan is

$$((1,1,1,2,1,2),(1,1,1,2,3,2),(1,2,1,3,4,5)).$$

The next two results follow immediately from Proposition 3.2 and the definition of $\sim_q$-patterns.

LEMMA 3.4. *Let $p = (p_1, \ldots, p_r)$ be a NER-pattern for a plan π. Then p is eligible for $C = C_1 \cup \cdots \cup C_r$ if and only if π is eligible for C_q for each $q \in [r]$.*

PROPOSITION 3.5. There is a NER-pattern p for every plan π.

We now introduce further terminology related to NER-patterns. We say a NER-pattern $p = (p_1, \ldots, p_r)$ is *realizable* if there exists a plan π such that π is authorized and p is a NER-pattern for π. We say p is *consistent* if for any $p_a = (p_{(a,1)} \ldots, p_{(a,k)})$ and $p_b = (p_{(b,1)}, \ldots, p_{(b,k)})$ with $a > b$, and any $i, j \in [k]$, we have $p_{(a,i)} = p_{(a,j)}$ implies $p_{(b,i)} = p_{(b,j)}$. Recall that if p is the NER-pattern for a plan π, then $p_{(a,i)} = p_{(a,j)}$ iff $\pi(s_i) \sim_a \pi(s_j)$, and similarly $p_{(b,i)} = p_{(b,j)}$ iff $\pi(s_i) \sim_b \pi(s_j)$. Thus Proposition 2.3 implies that if p is the NER-pattern for some plan, then p is consistent. Take the plan from Examples 2.2 and 3.3, which yields the following NER-pattern:

$$((1,1,1,2,1,2),(1,1,1,2,3,2),(1,2,1,3,4,5)).$$

By inspection, we can see that if two elements in one tuple are equal, then the same two elements in any tuple to its left are also equal. In other words, this NER-pattern is consistent (as one would expect, given that the plan from which it is derived is valid). Notice that this NER-pattern (and indeed any consistent NER-pattern) induces nested partitions on the set of steps. In this example, the $\sim_1$-pattern yields the partition $\{\{s_1, s_2, s_3, s_5\}, \{s_4, s_6\}\}$; the $\sim_2$-pattern yields $\{\{s_1, s_2, s_3\}, \{s_5\}, \{s_4, s_6\}\}$; and the $\sim_3$-pattern yields $\{\{s_1, s_3\}, \{s_2\}, \{s_4\}, \{s_5\}, \{s_6\}\}$.

For a $\sim_q$-pattern $p_q = (p_{(q,1)}, \ldots, p_{(q,k)})$ and a subset $T \subseteq S$, we define the *partial pattern* $p_q|_T = (z_1, \ldots, z_k)$, where $z_i = p_{(q,i)}$ if $s_i \in T$, and $z_i = 0$ otherwise. Given a partial plan $\pi : T \to U$, we say that $p|_T$ is a *NER-pattern for* π if $p|_T$ with all coordinates with 0 values removed is a NER-pattern for π. We therefore have that if p is a NER-pattern for a plan π, then $p|_T$ is a NER-pattern for π restricted to T.

Given Lemma 3.4 and Proposition 3.5, in order to solve a WSP instance with nested equivalence constraints, it is enough to decide whether there exists a NER-pattern p such that (i) p is realizable by some plan π, and (ii) p is eligible (and hence π is eligible) for $C = C_1 \cup \cdots \cup C_r$. Our algorithm will therefore enumerate all possible NER-patterns (except for certain NER-patterns that we show can be disregarded), and for each one check whether the two conditions hold. As every realizable pattern is consistent, it is enough to enumerate all consistent patterns.

We give the details of the enumeration in Section 3.2, and explain how to determine whether p is realizable in Section 3.3. In the remainder of this section, we show it is possible to check whether a NER-pattern $p = (p_1, \ldots, p_r)$ is eligible in time polynomial in the input size N. Indeed, in polynomial time, we can construct a plan π_q with $\sim_q$-pattern p_q for each $q \in [r]$. For each label $p_{(q,i)} \in p_q$, we may choose a distinct equivalence class $u^{\sim_q}$ in $U^{\sim_q}$ and, for each step s_j with $p_{(q,j)} = p_{(q,i)}$, we assign s_j to a user from $u^{\sim_q}$. By Lemma 3.4 and Proposition 3.2, p is eligible if and only if π_q is eligible for C_q for each $q \in [r]$. By assumption, eligibility of each π_q can be checked in polynomial time, which is enough to check the eligibility of p.[8] Note, however, that π_q may be different for different q, so this simple check for eligibility does not give us a check for realizability of p.

3.2. Enumerating Patterns

As a NER-pattern consists of a $\sim_q$-pattern for each $q \in [r]$, and each $\sim_q$-pattern is a vector of k integers from $[k]$, it is clear that the number of (not necessarily consis-

[8]Clearly, it is not hard to check eligibility of p without explicitly constructing each π_q, as we do in the algorithm implementation, described in Section 4.

tent) patterns is k^{rk}. Thus, we can naively enumerate all consistent patterns in time $O^*(k^{rk}) = O^*(2^{kr \log k})$.

Recall the definition of a nested partition. Observe that a $\sim_q$-pattern p_q defines a partition on the set of steps, where two steps are in the same block if and only if they are assigned the same label by p_q. Furthermore, if two $\sim_q$-patterns p_q and p'_q define the same partition, then p_q is eligible for C_q if and only if p'_q is eligible for C_q. Observe also that for a consistent NER-pattern $p = (p_1, \dots p_r)$, the tuple of partitions defined by $p_1, \dots, p_r$ respectively is a nested partition in r levels. It follows that in order to determine if there exists a realizable and eligible NER-pattern, it is enough to consider one pattern for each nested partition in r levels. The next theorem provides a bound on the number of such patterns.

THEOREM 3.6. *Let S be a set with $|S| = k$ and let r be an integer. The number of nested partitions of S in r levels is at most $r^{k-1}k!$.*

PROOF. Note that by Assumption 2 the blocks of the final partition $\mathcal{P}_r$ are singleton sets.

First observe that each nested partition can be represented as a leaf-labeled rooted tree embedded into the plane, with $r + 1$ levels and k leaves. Indeed, we can create such a tree as follows. Create a root node; then for every block in $\mathcal{P}_1$, create one node representing the block as a child of the root; then recursively, for every block B in $\mathcal{P}_i$, $1 < i \leq r + 1$, we create a node at level $i + 1$ of the tree representing the block, and make it a child of the node representing the block $B' \in \mathcal{P}_{i-1}$ for which $B \subseteq B'$. Hence it suffices to bound the number of such trees.

We give a characteristic of the tree as a permutation of $[k]$ followed by $k - 1$ integers between 1 and r. Clearly, there are exactly $r^{k-1}k!$ such characteristics; we will be able to reproduce the tree from the characteristic, which will finish the proof.

Leaves of the tree listed from left to right form a permutation of $[k]$. For a fixed permutation $\pi = x_1 \dots x_k$, to recreate the full shape of the tree, it suffices to have $k - 1$ numbers between 1 and r as follows. For every pair of elements x_i, x_{i+1}, we write down the distance between the corresponding leaves and their lowest common ancestor. Hence we have described a characteristic of the tree, from a set of size $k!r^{k-1}$, from which we can recreate the tree and hence also the nested partition. This completes the proof. □

It remains to show how we can enumerate NER-patterns in such a way that we cover exactly one pattern for each nested partition on r levels.

THEOREM 3.7. *We can enumerate a set of NER-patterns in such a way that there is a NER-pattern for every nested partition on r levels in time $O^*(2^{k \log(rk)})$.*

PROOF. We begin by defining an empty partial NER-pattern $p = (\mathbf{0}, \dots, \mathbf{0})$, where $\mathbf{0}$ is a pattern in which all k entries are 0. We then recursively replace 0 entries in p to obtain a complete NER-pattern (in which no entry is 0). This replacement is performed using a recursive, backtracking algorithm that outputs every complete NER-pattern, as implemented by the pseudocode in Algorithm 3 (on page 20).

Let $p' = (p'_1, \dots, p'_r)$, where $p'_i = (p'_{(i,1)}, \dots, p'_{(i,k)})$ for $i \in [k]$, be a partial NER-pattern. If p' is complete, then we add p' to the output of the algorithm. Otherwise, there exists a largest $q \leq r$ such that p'_q is not complete. Choose $i \in [k]$ such that $p'_{(q,i)} = 0$. Then define $k' = \max\{p'_{(q,j)} : j \in [k]\} + 1$ and, for every $a \in [k']$:

(1) for every $j \in [k]$, if $p'_{(q+1,j)} = p'_{(q+1,i)}$ then set $p'_{(q,j)} = a$;[9]
(2) make a recursive call with the resulting NER-pattern.

To prove that this algorithm enumerates all required patterns, it is enough to show that for each possible plan π the algorithm generates a NER-pattern p which is a NER-pattern for π. So consider a plan π. At every recursion point, corresponding to the specification of a value $p'_{(q,i)}$, there is exactly one value of a consistent with π (either $k' = 1$ in which case there is no choice; or p'_q places s_i in the same equivalence class as some previously specified step s_j; or s_i must be placed in a new equivalence class and we let $a = k'$). It is also clear that this recursive path is not aborted. Hence the process results in a complete NER-pattern p for π.

To bound the running time of our algorithm, we argue very similarly to show that the number of leaves in the algorithm's pattern generation tree $\mathcal{T}$ is bounded by the number of distinct nested partitions. Clearly, for an upper bound on the running time we may assume that no recursive search branch is aborted (i.e., every possible pattern is eligible). Then we find, as above, that for every nested partition $\mathcal{P}$, we can trace exactly one path from the root of the calling tree to a leaf, where at every point there is exactly one value $p'_{(q,i)} = a$ that is consistent with $\mathcal{P}$. We also find that every leaf of $\mathcal{T}$, corresponding to a complete NER-pattern p, corresponds to only exactly one nested partition. Since every non-leaf node of $\mathcal{T}$ has at least two children, the total number of nodes in $\mathcal{T}$ is less than twice the number of leaves. Hence Theorem 3.6 bounds the number of nodes by $\mathcal{T}$ by

$$2r^{k-1}k! = O^*(2^{k\log(rk)}).$$

The total running time is bounded by a polynomial factor times this number, whence the result follows. □

3.3. Checking Realizability

Given a consistent NER-pattern $p = (p_1, \ldots, p_r)$, we must determine whether p can be realized, given the authorization lists defined on users. Recall that each equivalence relation $\sim_q$ defines a set of equivalence classes on U, and each $\sim_q$-pattern $p_q = (p_{(q,1)}, \ldots, p_{(q,k)})$ defines a set of equivalence classes on S. We must determine whether there exists a plan $\pi : S \to U$ that: (i) has $\sim_q$-pattern p_q for each $q \in [r]$; and (ii) assigns an authorized user to each step.

Informally, checking realizability is based on r authorization relations, one for each equivalence relation $\sim_q$, $q \in [r]$. These authorization relations may be visualized as binary matrices in which the rows are indexed by equivalence classes in S and the columns are indexed by equivalence classes in U. We say the users in equivalence class $V \subseteq U$ are *collectively authorized* for the steps in $T \subseteq S$ if and only if for each $s \in T$ there exists a user $u \in V$ such that user u is authorized for step s (that is, $s \in A(u)$). Then the (T, V)-th entry in the matrix is set to 1 if and only if the users in the equivalence class V are collectively authorized for the steps in T.

We illustrate, informally, what an algorithm for checking realizability must do, using our running example with NER-pattern

$$((1,1,1,2,1,2),(1,1,1,2,3,2),(1,2,1,3,4,5)).$$

[9]Thus, $p'_{(q,i)} = a$ and p'_q remains consistent with p'_{q+1}. If $q = r$, then we simply define $p'_{(q,i)} = a$.

This gives rise to the following authorization relations (from which we omit set delimiters and zero entries in the interests of clarity):

$\sim_3$	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
s_1s_3	1		1						
s_2		1						1	1
s_5				1	1			1	
s_4							1	1	
s_6								1	1

$\sim_2$	u_1u_2	$u_3u_4u_5$	u_6u_7	u_8u_9
$s_1s_2s_3$	1			
s_5		1		1
s_4s_6				1

$\sim_1$	$u_1u_2u_3u_4u_5$	$u_6u_7u_8u_9$
$s_1s_2s_3s_5$	1	1
s_4s_6		1

From each of these authorization relations, we must construct a mapping such that each set indexing a row is mapped to a *distinct* set indexing a column (by definition of a pattern). And the three mappings, must be "coherent" in the following sense: if $T \subseteq S$ is an equivalence class induced by pattern p_q and is mapped to an equivalence class $V \subseteq U$ induced by $\sim_q$, then there must exist an injective mapping from the equivalence classes into which T is partitioned by p_{q+1} to the equivalence classes of $\sim_{q+1}$ that partition V. Clearly, this is possible for the three relations above:

$$\pi_1(s_1s_2s_3s_5) = u_1u_2u_3u_4u_5,\ \pi_1(s_4s_6) = u_6u_7u_8u_9$$
$$\pi_2(s_1s_2s_3) = u_1u_2,\ \pi_2(s_5) = u_3u_4u_5,\ \pi_2(s_4s_6) = u_8u_9$$
$$\pi_3(s_1s_3) = u_1,\ \pi_3(s_2) = u_2,\ \pi_3(s_5) = u_4,\ \pi_3(s_4) = u_8,\ \pi_3(s_6) = u_9$$

We may construct the final plan π from these mappings in the obvious way.

To illustrate that not every eligible NER-pattern is realizable, consider the NER-pattern

$$((1,1,1,2,1,2),(1,2,1,3,2,3),(1,2,1,3,4,5)).$$

(This NER-pattern only differs from our running example in p_2, which now requires that steps s_2 and s_5 be performed by users in the same section.) In this case, we derive the following authorization relation for pattern p_2 and $\sim_2$:

$\sim_2$	u_1u_2	$u_3u_4u_5$	u_6u_7	u_8u_9
s_1s_3	1	1		
s_2s_5				1
s_4s_6				1

Notice that it is now impossible to find an injective mapping (since we cannot map s_2s_5 and s_4s_6 to distinct equivalence classes in U).

Additionally, a pattern may not be realizable because it is impossible to find a coherent set of mappings. Consider, the following simple example with NER-pattern

$((1,1,2,2),(1,2,3,4))$:

$\sim_2$	u_1	u_2	u_3	u_4
s_1	1	1		1
s_2	1		1	1
s_3		1	1	
s_4		1	1	

$\sim_1$	u_1u_2	u_3u_4
s_1s_2	1	1
s_3s_4	1	1

Clearly we can find an injective mapping for $\sim_1$, with either of the following mappings being acceptable:

$$\pi_1(s_1s_2) = u_1u_2,\ \pi_1(s_3s_4) = u_3u_4; \quad \pi_1'(s_1s_2) = u_3u_4,\ \pi_1'(s_3s_4) = u_1u_2.$$

However, there is no injective mapping π_2 for $\sim_2$ such that π_1 and π_2 are coherent or such that π_1' and π_2 are coherent: whether we choose π_1 or π_1', there is no way we can map s_3 and s_4 to distinct authorized users.[10]

Our algorithm for checking realizability is recursive. Essentially, it computes authorization relations of the kind illustrated in the above tables. However, we insert a 1 in the (T,V)-th entry in the table for $\sim_q$ if and only if

1. the users in V are collectively authorized for the steps in T, *and*
2. there exists an injective mapping from the equivalence classes in T induced by p_{q+1} to the equivalence classes in V induced by $\sim_{q+1}$.

In the example above, then, the authorization relation computed by our algorithm for $\sim_1$ would be

$\sim_1$	u_1u_2	u_3u_4
s_1s_2	1	1
s_3s_4		

since there is no injective mapping from the equivalence classes comprising s_3s_4 in p_2 (that is s_3 and s_4) to the equivalence classes comprising u_1u_2 in $\sim_2$ (u_1 and u_2) or to the equivalence classes comprising u_3u_4 (u_3 and u_4). Hence there can be no injective mapping for $\sim_1$.

Essentially, our algorithm recursively computes coherent authorization relations for $\sim_r$, $\sim_{r-1}$, …, $\sim_1$, using information from $\sim_{q+1}$ to compute $\sim_q$. If we are able to find an injective mapping for the authorization relation for $\sim_1$ then the pattern is realizable. In our algorithm, each of these coherent authorization relations is represented as a bipartite graph: the partite sets comprise the indexing sets for the rows and columns; and an edge connects two vertices if and only if there is a 1 in the corresponding entry in the coherent authorization relation. A bipartite graph G with vertex set $X \cup Y$ has a *matching covering* X if there exist a set of edges $M \subseteq E(G)$ such that no pair of edges has a vertex in common and for every $x \in X$ there exists an edge in M containing x. Hence, a matching defines an injective mapping from X to Y. Then a pattern is realizable if, for each of these bipartite graphs, there exists a matching covering X.

More formally, fix a consistent NER-pattern $p = (p_1 = ((p_{(1,1)},\ldots,p_{(1,k)}),\ldots,p_r = (p_{(r,1)},\ldots,p_{(r,k)}))$. For notational convenience, let $\sim_0$ be the trivial equivalence relation having a single equivalence class equal to U, and let $p_0 = (1,\ldots,1)$ be the pattern assigning every step to the same label.

[10]Note, however, that the NER-pattern $((1,2,1,2),(1,2,3,4))$, for example, is realizable.

Definition 3.8. Given a consistent NER-pattern p, an integer $q \in \{0, 1, \ldots, r-1\}$, an equivalence class T induced by p_q, and an equivalence class V induced by $\sim_q$, we define the *q-level bipartite graph* $G(T, V, q)$ as follows: Let $\mathcal{T}$ denote the equivalence classes induced by p_{q+1} and contained in T, and let the vertices of $G(T, V, q)$ be $\mathcal{T} \cup V^{\sim_{q+1}}$. For each equivalence class $T' \in \mathcal{T}$, and for each equivalence class $V' \in V^{\sim_{q+1}}$, let $G(T, V, q)$ have an edge between T' and V' if and only if there exists an authorized partial plan $\pi' : T' \to V'$ such that $p_{q'}|_{T'}$ is a $\sim_{q'}$-pattern for π', for each $q' \geq q + 1$.

LEMMA 3.9. *Given a consistent NER-pattern p, an integer $q \in \{0, 1, \ldots, r-1\}$, an equivalence class T induced by p_q, and an equivalence class V induced by $\sim_q$, the following are equivalent:*

(i) There exists an authorized partial plan $\pi : T \to V$ such that $p_{q'}|_T$ is a $\sim_{q'}$-pattern for π, for each $q' \geq q$;
(ii) $G(T, V, q)$ has a matching covering $\mathcal{T}$.

PROOF. (i) $\Rightarrow$ (ii):
Suppose such a plan π exists. Then consider an equivalence class T' induced by p_{q+1} and contained in T. It follows from the definition of π that $p_{q'}|_{T'}$ is a $\sim_{q'}$-pattern for $\pi|_{T'}$ for each $q' \geq q + 1$.

As $p_{q+1}|_T$ is a $\sim_{q+1}$-pattern for π, and T' is an equivalence class induced by p_{q+1}, it must be the case that T' is an equivalence class induced by $\approx_{\pi,q+1}$, which implies that $\pi|_{T'}(T')$ is contained in an equivalence class induced by $\sim_{q+1}$. Let $V(T')$ denote this equivalence class. Then $\pi|_{T'}$ is an authorized partial plan $\pi|_{T'} : T' \to V(T')$ such that $p_{q'}|_{T'}$ is a $\sim_{q'}$-pattern for $\pi|_{T'}$, for each $q' \geq q+1$. Thus there is an edge between T' and $V(T')$ in $G(T, V, q)$.

Furthermore, for two different equivalence classes T', T'' induced by p_{q+1} and contained in T, we have that T' and T'' are different equivalence classes induced by $\approx_{\pi,q+1}$, and so $\pi(T')$ and $\pi(T'')$ are contained in different equivalence classes induced by p_{q+1}. Thus, $V(T') \neq V(T'')$. By taking the edge between T' and $V(T')$ for each equivalence class T', we get a matching covering $\mathcal{T}$.

(ii) $\Rightarrow$ (i):
Suppose we have a matching covering $\mathcal{T}$. Then for each equivalence class T' induced by p_{q+1} and contained in T, there exists an equivalence class $V(T')$ induced by $\sim_{q+1}$ and contained in V, such that there is an edge between T' and $V(T')$ in $G(T, V, q)$, and furthermore $V(T') \neq V(T'')$ for any $T' \neq T''$.

Then by definition of $G(V, T, q)$, for each T' there exists an authorized partial plan $\pi' : T' \to V(T')$ such that $p_{q'}|_{T'}$ is a $\sim_{q'}$-pattern for π', for each $q' \geq q+1$. Let $\pi : T \to V$ be the union of these partial plans. Then π is clearly an authorized partial plan. It remains to show that $p_{q'}|_T$ is a $\sim_{q'}$-pattern for π, for each $q' \geq q$.

Consider $s_i, s_j \in T$. We will show that for each $q' \geq q$, s_i, s_j are in the same class induced by $p_{q'}$ if and only if they are in the same class induced by $\approx_{\pi,q'}$ i.e. $\pi(s_i) \sim_{q'} \pi(s_j)$. It follows that $p_{q'}|_T$ is a $\sim_{q'}$-pattern for π, as required.

If $q' = q$, then by construction $s_i, s_j \in T$ and $\pi(s_i), \pi(s_j) \in V$. Thus s_i, s_j are in the same class induced by p_q and $\pi(s_i), \pi(s_j)$ are in the same class induced by $\sim_q$, and we are done. Otherwise, we have $q' \geq q + 1$. In this case, suppose that s_i, s_j are in the same class T' induced by p_{q+1}. Then there exists an authorized partial plan $\pi' : T' \to V(T')$ such that $p_{q'}|_{T'}$ is a $\sim_{q'}$-pattern for π', and in particular $\pi'(s_i) \sim_{q'} \pi'(s_j)$ if and only if s_i, s_j are in the same class induced by $p_{q'}$. By construction, $\pi(s_i) = \pi'(s_i)$ and $\pi(s_j) = \pi'(s_j)$, and so we are done. Finally, suppose that s_i, s_j are different classes induced by p_{q+1}. Then as p is consistent and $q' \geq q+1$, they are also in different classes induced by $p_{q'}$. Furthermore, by construction $\pi(s_i)$ and $\pi(s_j)$ are in different classes induced by $\sim_{q+1}$, and therefore in different classes induced by $p_{q'}$, and we are done. □

By definition, S and U are the only equivalence classes induced by p_0 and $\sim_0$, respectively. Thus, Lemma 3.9 implies that there exists an authorized plan $\pi : S \to U$ such that p is a NER-pattern for π if and only if the 0-level bipartite graph $G(S, U, 0)$ has a matching covering all equivalence classes of S induced by p_1. For $q < r$, Lemma 3.9 and Definition 3.8 imply that $G(T, V, q)$ has an edge between T' and V' if and only if $G(T', V', q + 1)$ has a matching covering T'.

Algorithm 1: isRealizable(W, p, T, V, i)

```
1  if i = r then
2      if T ⊆ A(V) ;        /* V is singleton user authorized for steps in T */
3      then
4          return SAT;
5      else
6          return UNSAT;
7  else
8      for each equivalence class T' in T induced by p_{i+1} do
9          for each equivalence class V' in V induced by ∼_{i+1} do
10             if isRealizable(T', V', i + 1) = SAT then
11                 Add edge between T' and V';
12     if there exists a covering matching in the q-level bipartite graph G(T, V, q) then
13         return SAT;
14     else
15         return UNSAT;
```

Together, these observations lead to a recursive algorithm (isRealizable, described in Algorithm 1) to decide whether a NER-pattern p is realizable. Given a level q, an equivalence class T induced by p_q, and an equivalence class V induced by $\sim_q$, our algorithm will return SAT if there exists an authorized partial plan $\pi : T \to V$ such that $p_{q'}|_T$ is a $\sim_{q'}$-pattern for π, for each $q' \geq q$. Otherwise, the algorithm will return UNSAT. For $q \leq r-1$, the algorithm generates a bipartite graph and checks it for a matching, where each edge of the graph is determined by a recursive call to the algorithm for $q + 1$.

To analyze the running time of the isRealizable algorithm, observe that a single call to the algorithm requires polynomial time:

— checking whether a bipartite graph has a matching covering one partite set can be done in polynomial time; and
— for $q = r$, the algorithm performs a simple check to see whether the single user v (since, for $q = r$, $V = \{v\}$ for some $v \in U$) is authorized for all steps in T, which also takes polynomial time.

Moreover, for each tuple $(T', V', q + 1)$, isRealizable$(T', V', q + 1)$ is called at most once (during an iteration of isRealizable(T, V, q), where T is the equivalence class induced by p_q that contains T', and V is the equivalence class induced by $\sim_q$ that contains V'). Thus the running time is bounded by a polynomial multiplied by the number of tuples of the form (T, V, q), where T is an equivalence class induced by p_q and V is an equivalence class induced by $\sim_q$. As $|S| = k$ and $|U| = n$, the number of such tuples is at most rkn, and so we have the following result:

Algorithm 2: main(W)

input : WSP instance $W = (S, U, \mathcal{A}, \{\sim_1, \ldots, \sim_r\}, C)$
output: UNSAT or SAT

```
p_0 = (1, ..., 1);
for each q ∈ [r] do
    p_q = (0, ..., 0);
p = (p_0, p_1, ..., p_r);
return patternBacktrack(W, p, r);
```

THEOREM 3.10. *Given a NER-pattern p, we can determine whether p is realizable in polynomial time.*

3.4. Proof of Theorem 3.1 and FPT Algorithm

In order to solve a WSP instance with nested class-independent constraints in r levels, by Lemma 3.4 and Proposition 3.5, it is enough to decide whether there exists a NER-pattern p such that (i) p is realizable, and (ii) p is eligible for $C = C_1 \cup \cdots \cup C_r$. By the discussion in Section 3.2, in order to check if there is a realizable and eligible NER-pattern, it is enough to consider one pattern for each nested partition in r levels. By Theorem 3.7, such a set of NER-patterns can be enumerated in time $O^*(2^{k \log(rk)})$. For each pattern in this set, we can decide whether it is realizable in polynomial time by Theorem 3.10, and we can decide whether it is eligible in polynomial time as discussed at the end of Section 3.1. Thus, overall, we can solve the WSP instance in time $O^*(2^{k \log(rk)})$, as required.

Algorithms 1 – 3 provide pseudo-code for our FPT algorithm. Note that in the implementation of our algorithm, as an additional heuristic we check whether partial patterns are eligible (i.e. the pattern constructed so far does not imply that a constraint will be violated), reducing the search space by immediately discarding ineligible patterns.

Algorithm 3: patternBacktrack(W, p, q)

```
if q = 0 then
    return isRealizable(W, p, S, U, 0);
else
    if p_q is complete then
        return patternBacktrack(W, p, q − 1);
    else
        Choose i such that p_(q,i) = 0;
        for each a ∈ {1, ..., max{p_(q,j) : j ∈ [k]} + 1} do
            if q = r then
                p_(q,i) = a;
            else
                for each j such that p_(q+1,j) = p_(q+1,i) do
                    p_(q,j) = a;
            if p_q is eligible and patternBacktrack(W, p, q) = SAT then
                return SAT;
        return UNSAT;
```

4. ALGORITHM IMPLEMENTATION AND COMPUTATIONAL EXPERIMENTS

Thus far, we have investigated the worst-case complexity of an algorithm to solve WSP in the presence of CI constraints. However, there can be a huge difference between the performance of an algorithm in principle and the performance of its actual implementation as a computer code [Bartz-Beielstein et al. 2010; Myrvold and Kocay 2011]. In this section, we discuss an implementation of the algorithm described in the previous section, its performance on synthetic instances of WSP, and compare its performance with that of the pseudo-Boolean SAT solver SAT4J. We refer to the implementation of our algorithm as PB4CI – pattern-backtracking for class-independent (constraints).

The goals of our experimental work are to:

— evaluate the performance of PB4CI on a range of instances of WSP with CI constraints; and
— compare the performance of PB4CI with a state-of-the-art off-the-shelf SAT solver on the same instances.

We chose the solver SAT4J as this has been used in previous work on WSP [Cohen et al. 2015; Karapetyan et al. 2015; Wang and Li 2010]. We hope to confirm our intuition that PB4CI would perform well on instances in which the constraints meant there were relatively few eligible patterns, while SAT4J would perform well on instances for which there were many possible solutions.

In Section 4.1, we describe the generation of test instances, in particular focusing on our choice of parameters. In Section 4.2, we present the results of our experiments and compare the performance of PB4CI and SAT4J. In the appendix we provide some further information about the implementation of PB4CI, including our branching heuristics, and details of the encoding of WSP with CI constraints as a pseudo-Boolean satisfiability problem. The FPT algorithm's executable code and experimental data set are publicly available [Gagarin et al. 2015].

4.1. Generation of Test Instances

To be able to provide a meaningful analysis of experimental results, we decided, as in prior experimental work on WSP [Cohen et al. 2015; Karapetyan et al. 2015], to have a small number of constraint types. In particular, we focus on the special case $r = 2$: that is, we use a single equivalence relation $\sim$ on U (in addition to the identity relation). We use constraints of the following types: (i) not-equals constraints $(s, s', \neq)$; (ii) equivalence constraints $(s, s', \sim)$; (iii) non-equivalence constraints $(s, s', \nsim)$; and (iv) at-most-3 constraints with a scope $Q \subseteq S$ of size 5.[11] The values we used for the parameters k and n were chosen by inspection of case studies provided by standards bodies and the academic literature. In particular, the number of steps in the workflows in the BPM Academic Initiative repository[12] and the workflow patterns database[13] ranges in value between 10 and 50. Empirical access control studies suggest that the number of roles in an organization, typically a similar order of magnitude to the number of steps in workflow specifications, is an order of magnitude smaller than the number of users [Schaad et al. 2001]. For every user $u \in U$ we generated $A(u)$ by choosing the size α of $A(u)$ from the set $\{1, 2, \ldots, \lceil k/2 \rceil\}$ randomly and uniformly and then picking α distinct steps from S also randomly and uniformly.

[11] A constraint of this form is satisfied by a plan π if π assigns at most three users to the steps in Q. Constraints of this form have been used in previous experimental work [Cohen et al. 2015; Karapetyan et al. 2015].

[12] http://www.signavio.com/bpm-academic-initiative/

[13] http://www.workflowpatterns.com/

We generated a range of WSP instances with approximately equal numbers of satisfiable and unsatisfiable instances. We did this by varying the number of constraints of each type. The two parameters we varied substantially in order to generate our test instances were the number of constraints of the form $(s, s', \nsim)$ and the number of at-most-3 constraints. The satisfaction of constraint $(s, s', \nsim)$ implies the satisfaction of the not-equals constraint $(s, s', \neq)$. Therefore, we do not vary the number of not-equals constraints a great deal (in contrast to existing work in the literature [Cohen et al. 2015]). Conversely, satisfaction of constraint $(s, s', =)$ implies the satisfaction of constraint $(s, s', \sim)$ for any equivalence relation $\sim$. Thus the inclusion of constraints of the form $(s, s', \sim)$ usually does not make an instance more difficult to satisfy. Therefore, although our algorithm solves instances that include constraints of the form $(s, s', \sim)$, we use few (if any) constraints of this form in our experiments.

In total, we generated $525 = 5 \times 3 \times 7 \times 5$ test instances, which ranged from relatively "lightly constrained" instances, which are likely to be satisfiable and easy to solve, to "highly constrained" instances, which are likely to be unsatisfiable and also (relatively) easy to solve. In between these extremes, there are instances that may or may not be satisfiable and are likely to be difficult to solve. Of the 525 test instances we generated, 250 are satisfiable (47.62%). The distribution of satisfiable and unsatisfiable instances with respect to the number of steps is shown in Figure 2. The values of all parameters used in the generation of our test instances are summarized in Table II.

Table II. Parameters and their values used in the experiments

Parameter	**Values**
Number of steps, k	$20, 25, 30, 35, 40$
Number of users, n	$10k$
Number of equivalence classes of users, c	$2k$
Number of not-equals constraints $(s, s', \neq)$	$k \pm 5i,\ i = 0, 1$
Number of at-most-3 constraints	$k \pm 5i,\ i = 0, 1, 2, 3$
Number of equivalence constraints $(s, s', \sim)$	$\frac{k}{5} - 4$
Number of non-equivalence constraints $(s, s', \nsim)$	$k \pm 5i,\ i = 0, 1, 2$

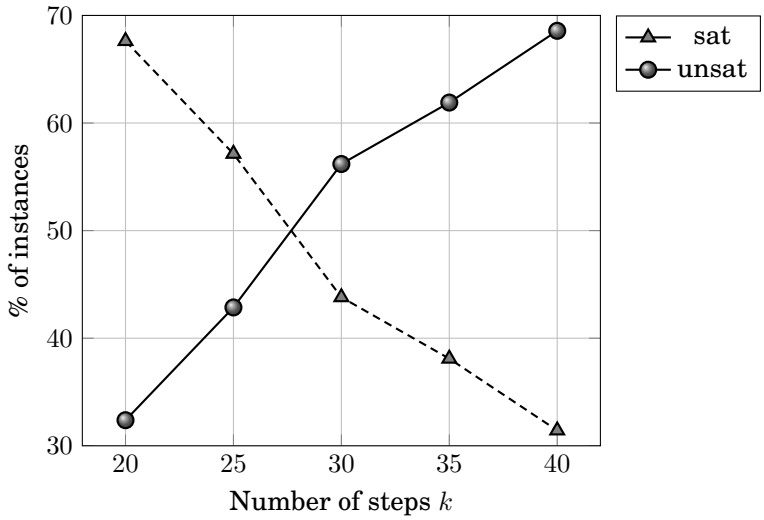


Fig. 2. Distribution of satisfiable and unsatisfiable instances by the number of steps k.

All the constraints, authorizations, and equivalence classes of users are generated for each instance separately and uniformly at random. The random generation of authorizations, not-equals, and at-most-3 constraints uses existing techniques [Cohen et al. 2015]. The generation of equivalence and non-equivalence constraints has to be controlled more carefully to ensure that an instance is not trivially unsatisfiable and does not have too many vacuous constraints. For example, we need to discard a constraint of the form $(s, s', \nsim)$ if we have already generated a constraint of the form $(s, s', \sim)$.

The equivalence classes of users are generated by linearly ordering the user set and then splitting the list into intervals of consecutive users. The number of users in each class varies from 3 to 7, chosen uniformly at random and adjusted, if necessary, so that the total number of users in all the classes is equal to n.

4.2. Experimental Results

All our experiments were performed on a MacBook Pro computer having a 2.6 GHz Intel Core i5 processor, 8 GB 1600 MHz DDR3 RAM, and running Mac OS X 10.9.5. PB4CI and SAT4J were each given one hour to solve each instance of the problem. One of three outcomes was possible: the instance was solved and found to be satisfiable; the instance was solved and found to be unsatisfiable; the instance was not solved (either because the one-hour time limit was exceeded or because of insufficient memory resources).

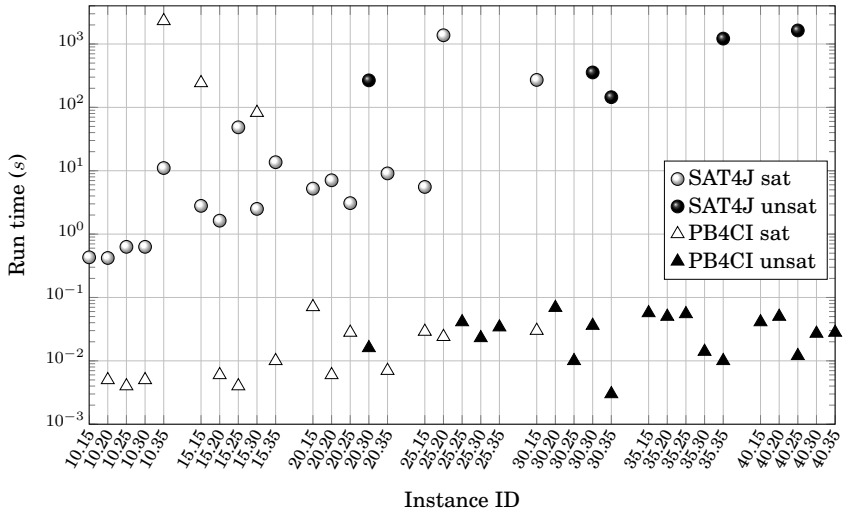


Fig. 3. Sample of test results for $k = 25$ steps. An Instance ID $a.b$, denotes an instance with a at-most-3 constraints and b non-equivalence constraints. If no node is shown for a particular instance then the instance remained unsolved.

Figure 3 reports detailed representative results obtained for 35 instances, of which 17 are satisfiable. Each instance has 25 steps, 30 not-equals constraints and 1 equivalence constraint.

— Each instance is labeled with an identifier of the form $a.b$, where a and b denote the number of at-most-3 constraints and non-equivalence constraints, respectively.
— Satisfiable instances are represented by unfilled nodes; unsatisfiable instances by filled nodes.

— Times to solve the instances using PB4CI are represented by triangles; times using SAT4J are represented by circles.
— If no node is shown for a particular instance then the instance remained unsolved. For example, PB4CI failed to solve (satisfiable) instance 10.15, while SAT4J failed to solve (unsatisfiable) instance 25.25.

More generally, PB4CI successfully solves around 95% of the test instances (502 instances out of 525), while SAT4J solves fewer than 50% (253 instances). The results are summarized in Figures 4–6, which show the success rates for PB4CI and SAT4J on all instances, unsatisfiable instances and satisfiable instances, respectively. These figures, together with Figure 3, clearly illustrate the following points:

— SAT4J is much more effective on satisfiable instances than on unsatisfiable ones;
— PB4CI is extremely effective in solving highly constrained instances and unsatisfiable instances, particularly in comparison to SAT4J;
— SAT4J is more effective, on average, than PB4CI on satisfiable instances with small numbers of constraints, although for many such instances PB4CI is faster than SAT4J, and any advantage SAT4J enjoys over PB4CI on satisfiable instances disappears for instances with more than 30 steps.

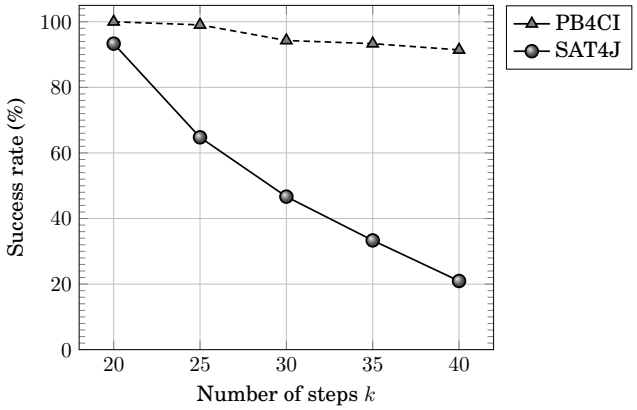


Fig. 4. Performance of SAT4J vs PB4CI on all instances

It is evident from Figures 3–6 that PB4CI is least effective for lightly constrained instances, whereas SAT4J is least effective on highly constrained instances. This reflects the different strategies used by PB4CI and SAT4J. PB4CI explores a search space that is, in the worst case, exponential in k. If the instance is lightly constrained that search space will necessarily be large, because most patterns will be eligible. In contrast, SAT4J is a more conventional solver, presumably with many heuristics and optimizations geared towards solving pseudo-Boolean satisfiability problems. If the instance is highly constrained, the encoding of the instance will contain more variables and SAT4J will have to explore many more possible assignments if it is to find a solution (and will have to explore every possible solution if the instance is unsatisfiable). In contrast, the number of possible patterns – the size of the search space for PB4CI – will be rather small (irrespective of whether the instance is satisfiable or not). The contrast in performance is particularly noticeable on unsatisfiable instances: for $k = 40$, for example, PB4CI solves all unsatisfiable instances, while SAT4J is only able to solve one instance.

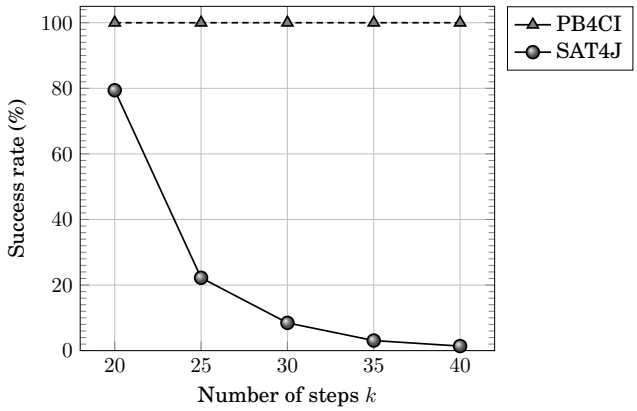


Fig. 5. Performance of SAT4J vs PB4CI on unsatisfiable instances

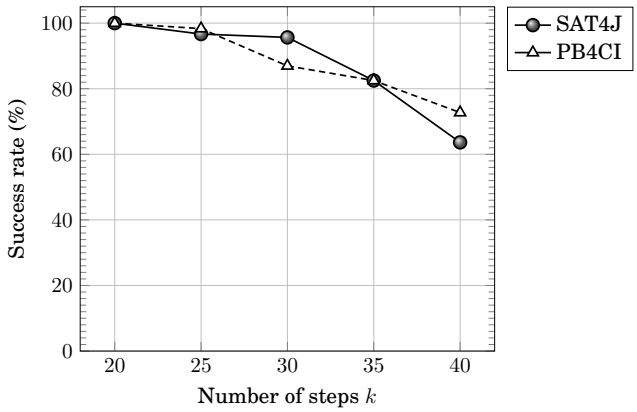


Fig. 6. Performance of SAT4J vs PB4CI on satisfiable instances

Table III shows summary statistics for all the experiments, including average and median CPU times. Where an instance is not solved, we take the time at which execution was halted – either 60 minutes or the point at which the computer's memory was exhausted. Thus, reported average and median times represent estimated lower bounds on these statistics: if more time or memory had been available to run these instances to completion, execution times would have been higher.

Despite the evidence that PB4CI is generally more effective and quicker at solving WSP instances, we recognize that SAT4J is quicker, sometimes orders of magnitude quicker, for some satisfiable WSP instances (see instances 10.35, 15.15 and 15.30 in Figure 3). This is due to a relatively small number of constraints in these instances, which requires PB4CI to generate a large number of eligible patterns. Since pattern realizability checking is mostly independent from eligibility testing and quite time consuming, PB4CI may spend a lot of time trying to find a realizable pattern in the large set of all eligible patterns. Conversely, there are many satisfiable assignments for these instances, so SAT4J's heuristics are likely find solutions for such instances rather quickly.

Table III. Summary statistics for $k \in \{20, 25, 30, 35, 40\}$

		SAT4J				PB4CI			
k	Result	Solved	Unsolved	Mean Time	Median Time	Solved	Unsolved	Mean Time	Median Time
20	SAT	71	0	11.55	1.28	71	0	0.005	0.003
	UNSAT	27	7	1,312.95	1,022.01	34	0	0.008	0.006
	Total	98	7	432.96	7.02	105	0	0.006	0.004
25	SAT	58	2	151.18	5.00	59	1	111.20	0.028
	UNSAT	10	35	1,938.31	2,228.01	45	0	0.057	0.041
	Total	68	37	917.09	86.06	104	1	63.57	0.036
30	SAT	44	2	229.09	13.56	40	6	368.41	0.38
	UNSAT	5	54	2,061.26	2,145.42	59	0	0.83	0.26
	Total	49	56	1,258.59	1,667.80	99	6	161.86	0.28
35	SAT	33	7	644.28	151.22	33	7	443.54	4.08
	UNSAT	2	63	2,023.41	2,010.80	65	0	2.82	1.95
	Total	35	70	1,498.03	1,808.43	98	7	170.72	2.48
40	SAT	21	12	1,014.75	627.31	24	9	403.74	111.10
	UNSAT	1	71	1,834.73	1,855.97	72	0	31.38	12.02
	Total	22	83	1,577.02	1,794.52	96	9	148.41	13.98

Finally, we note that at least one of the solvers was able to (correctly) solve every instance in Figure 3 in no more than 11 seconds (and usually much quicker).[14] In practice, therefore, it would be prudent to implement both solvers in a workflow management system and run both solvers on a WSP instance, terminating whichever solver hasn't finished when the other one returns a result.

5. CONCLUSION

In this paper, we introduce the concept of a class-independent constraint. In their own right, CI constraints represent a significant generalization of user-independent constraints, which are, themselves, highly expressive (and sufficient to model all constraints defined in the ANSI RBAC standard). However, the use of nested equivalence relations and CI constraints defined over those relations, provides an even richer framework for workflow constraints and substantially extends the range of real-world business requirements that can be modeled.

Nevertheless, an expressive framework is of limited use if reasoning about its effects is computationally infeasible. We have established that the introduction of CI constraints and nested equivalence relations does not affect the fixed-parameter tractability of WSP. In particular, we have designed an FPT algorithm for WSP with class-independent constraints, providing a more careful analysis of the worst-case complexity than that given in the preliminary version of this paper [Crampton et al. 2015].

Again, a theoretical algorithm to solve a problem is of little use if its realization in practice is very slow. Accordingly, we developed an implementation of our algorithm and compared its performance with that of SAT4J, an off-the-shelf state-of-the-art pseudo-Boolean SAT solver, on a wide range of test instances with realistic parameter values. Our results demonstrate that our FPT algorithm is useful in practice for WSP with class-independent constraints, notably for instances that are highly constrained. In this case, the number of patterns, which characterizes the size of the search space for our algorithm, is small. For such instances, conventional solvers like SAT4J have to explore a large number of assignments of users to steps. The performance of such

[14]The times taken for instances with $k < 25$ are correspondingly faster, as can be seen in Table III.

solvers is particularly poor for unsatisfiable instances because of the combinatorial blow-up in the size of the search space.

There remain opportunities for further work in this area. In particular, there are problems that are closely related to WSP that have been studied in the literature, typically only in the context of simple separation-of-duty and binding-of-duty constraints [Basin et al. 2014; Yang et al. 2014]. Of particular interest here is the work of Basin et al. [2014] and their use of release points to limit the scope of constraints in workflows with loops. It would be interesting to study workflow satisfiability (from the perspective of fixed-parameter tractability) when the workflow specification includes release points and user- and class-independent constraints.

In this paper we have focused on organizations with a hierarchical structure. While this is true of many organizations, modern management structures are often more complex so we would like to investigate whether it is possible to extend our work to accommodate richer organizational structures. Moreover, our work assumes that the organizational units partition the user population. Again, this may be true of many organizations, but not all. We would like to investigate team-based constraints, where each team comprises users from different parts of an organization but the union of those teams is neither disjoint nor the entire user population.

In addition, one can imagine that an organization might incorporate multiple hierarchical structures, defined in terms of partitions of the user set. Our current approach works for a single hierarchy, based on nested partitions. It would be interesting to see whether our approach can accommodate multiple hierarchies using different nested partitions, thus enriching workflow specifications further. Of course, the number of hierarchies and the maximum value of their respective depths become parameters to the problem but, provided these parameters are all small relative to the size of the user set, we may still be able to obtain useful theoretical results and reasonably efficient algorithms.

Acknowledgement. This research was partially supported by an EPSRC grant EP/K005162/1. Gutin's research was partially supported by Royal Society Wolfson Research Merit Award. We thank the anonymous referees for their invaluable feedback.

REFERENCES

American National Standards Institute 2004. *ANSI INCITS 359-2004 for Role Based Access Control*. American National Standards Institute.

Thomas Bartz-Beielstein, Marco Chiarandini, Lus Paquete, and Mike Preuss (Eds.). 2010. *Experimental Methods for the Analysis of Optimization Algorithms*. Springer.

David A. Basin, Samuel J. Burri, and Günter Karjoth. 2014. Obstruction-free authorization enforcement: Aligning security and business objectives. *J. Comput. Security* 22, 5 (2014), 661–698. DOI:http://dx.doi.org/10.3233/JCS-140500

Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. 1999. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM Trans. Inf. Syst. Secur.* 2, 1 (1999), 65–104.

D. F. C. Brewer and Michael J. Nash. 1989. The Chinese Wall Security Policy. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 206–214.

David Cohen, Jason Crampton, Andrei Gagarin, Gregory Gutin, and Mark Jones. 2014. Iterative Plan Construction for the Workflow Satisfiability Problem. *J. Artif. Intel. Res.* 51 (2014), 555–577.

David Cohen, Jason Crampton, Andrei Gagarin, Gregory Gutin, and Mark Jones. 2015. Algorithms for the workflow satisfiability problem engineered for counting constraints. *J. Comb. Optim.* (4 2015), 1–22. DOI:10.1007/s10878-015-9877-7.

Jason Crampton. 2005. A reference monitor for workflow systems with constrained task execution. In *SACMAT*, Elena Ferrari and Gail-Joon Ahn (Eds.). ACM, 38–47.

Jason Crampton, Andrei Gagarin, Gregory Gutin, and Mark Jones. 2015. On the Workflow Satisfiability Problem with Class-independent Constraints. In *10th International Symposium on Parameterized and*

Exact Computation (IPEC 2015) (Leibniz International Proceedings in Informatics (LIPIcs)), Thore Husfeldt and Iyad Kanj (Eds.), Vol. 43. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 66–77. DOI:http://dx.doi.org/10.4230/LIPIcs.IPEC.2015.66

Jason Crampton and Gregory Gutin. 2013. Constraint expressions and workflow satisfiability. In *SACMAT*, Mauro Conti, Jaideep Vaidya, and Andreas Schaad (Eds.). ACM, 73–84.

Jason Crampton, Gregory Gutin, and Anders Yeo. 2013. On the Parameterized Complexity and Kernelization of the Workflow Satisfiability Problem. *ACM Trans. Inf. Syst. Secur.* 16, 1 (2013), 4.

Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory*. Springer Verlag.

Andrei Gagarin, Jason Crampton, Gregory Gutin, Mark Jones, and Magnus Wahlström. 2015. The pattern-backtracking FPT algorithm and experimental data set for the WSP with class-independent constraints. (2015). Figshare. http://dx.doi.org/10.6084/m9.figshare.1603424. Retrieved Nov 16, 2015.

Gregory Gutin and Magnus Wahlström. 2015. Tight Lower Bounds for the Workflow Satisfiability Problem Based on the Strong Exponential Time Hypothesis. (2015). Inf. Processing Letters, to appear. Preprint available at http://arxiv.org/abs/1508.06829.

Daniel Karapetyan, Andrei Gagarin, and Gregory Gutin. 2015. Pattern Backtracking Algorithm for the Workflow Satisfiability Problem. In *Frontiers in Algorithmics 2015*. Lect. Notes Comput. Sci., Vol. 9130. Springer, 138–149.

William Kocay and Donald L. Kreher. 2004. *Graphs, Algorithms, and Optimization*. Chapman & Hall/CRC Press.

Wendy Myrvold and William Kocay. 2011. Errors in graph embedding algorithms. *J. Comput. Syst. Sci.* 77, 2 (2011), 430–438.

Andreas Schaad, Jonathan D. Moffett, and Jeremy Jacob. 2001. The role-based access control system of a European bank: a case study and discussion. In *SACMAT*. 3–9. DOI:http://dx.doi.org/10.1145/373256.373257

Qihua Wang and Ninghui Li. 2010. Satisfiability and Resiliency in Workflow Authorization Systems. *ACM Trans. Inf. Syst. Secur.* 13, 4 (2010), 40.

Ping Yang, Xing Xie, Indrakshi Ray, and Shiyong Lu. 2014. Satisfiability Analysis of Workflows with Control-Flow Patterns and Authorization Constraints. *IEEE T. Services Computing* 7, 2 (2014), 237–251. DOI:http://dx.doi.org/10.1109/TSC.2013.31

A. APPENDIX

A.1. Implementation Details

Our implementation of the theoretical FPT algorithm in C++ uses a number of heuristics in order to improve performance, which we now describe in more detail.

The FPT algorithm and pattern generation used by Cohen et al. [2015] assumed a fixed ordering $s_1, \ldots, s_k$ of steps in S. One of the key advantages of the pattern-backtracking framework used in our FPT algorithm is that we can consider the steps as arbitrarily permuted at any stage of the recursion. This flexibility is used in the choice of steps for branching during the recursion, and allows us to reduce the search space of NER-patterns dramatically. During the construction of $p_2 = p_=$ and $p_1 = p_\sim$, our algorithm uses a heuristic subroutine to decide which zero-valued label $p_{(2,i)}$ and $p_{(1,i)}$, $i \in [k]$, respectively, should be considered next. Our heuristic simply chooses a zero-valued label corresponding to a step of maximum weight. The weights for steps with zero-valued labels in p_2 and p_1 are computed separately, depending on the type of constraints in the WSP instance ($\sim$-independent or UI) and properties of the problem instance. Also, in the current version of the algorithm, the weights of steps for branching are computed locally, in each iteration, which is different from the approaches used by Crampton et al. [2015] in which weights are pre-computed and used for every iteration.

For the two types of constraints used in our computational experiments, the weights are computed as follows. The computation of weight for $p_{(2,i)}$ is based on the type, number, and current state of UI constraints containing the corresponding step s_i, $i \in [k]$, with larger weight given if other steps in the constraint have already been assigned to some non-zero values in the corresponding p_2-pattern. Since a constraint $(s, s', \nsim)$

implies $(s, s', \neq)$, such constraints accounted for in these weight computations as well. The computation of weight for $p_{(1,i)}$ is based on the type, number, and current state of $\sim$-independent constraints containing the corresponding step s_i, $i \in [k]$. Much larger weight is given for inclusion in the scope of the equivalence constraints and larger weight given when the other step in the constraint is already assigned to some non-zero value in the corresponding partial p_1-pattern. The intuition behind these weights and the corresponding choice of steps in iteration is that using a step participating in more, "more influential," and "more satisfied" constraints is more beneficial for pruning the search space branches, and this usually reduces the pattern search space more effectively. The precise values of parameters for weight functions have been tuned empirically. In a certain sense, this approach is a development of the approach used in Karapetyan et al. [2015], and often gives better results than the approach with global weights used by Crampton et al. [2015].

Checking realizability of complete NER-patterns is a subtle procedure involving the use of efficient data structures and construction of corresponding bipartite graphs on two levels. The construction of bipartite graphs itself is one of the most time consuming-parts of the code. The algorithm searching for required matchings in the bipartite graph uses a modified version of the Hungarian algorithm and data structures [Kocay and Kreher 2004], in combination with some simple speed-ups and Proposition 1 used by Karapetyan et al. [2015]. In case of satisfiable instances of the problem, when the existence of required matchings in the bipartite graphs is confirmed, the algorithm uses a modification of the construction of bipartite graphs and finding matchings subroutines to find an actual solution assignment.

A.2. Reduction of WSP to a Pseudo-Boolean Satisfiability Problem

In order to compare the performance of our algorithm with SAT4J, we transformed our test instances of WSP into pseudo-Boolean satisfiability problem instances, which we then solved using the off-the-shelf SAT solver SAT4J. We now briefly describe this transformation, which involves the introduction of a number of binary $(0, 1)$-variables and pseudo-Boolean constraints. Similar transformations have been used elsewhere [Cohen et al. 2015; Karapetyan et al. 2015; Wang and Li 2010].

— For each step $s \in S$ and each user in $A(s)$, we define a binary variable $x_{u,s}$. Assignments to these variables will be used to represent a plan π, where $x_{u,s} = 1$ if and only if $\pi(s) = u$.
— For each equivalence class $X \subseteq U$ (with respect to $\sim$) and each step $s \in S$, we define a binary variable $z_{X,s}$, where $z_{X,s} = 1$ if and only if $\pi(s) \in X$.
— For each at-most constraint c with scope Q and each user $u \in U$, we define a binary variable $y_{u,c}$, where $y_{u,c} = 1$ if an only if $x_{u,s} = 1$ for some $s \in Q$.
— For each step $s \in S$, we require $\sum_{u \in A(s)} x_{u,s} = 1$ (to ensure that every step is assigned to exactly one user).
— For every not-equals constraint $(s, s', \neq)$ and each user $u \in A(s) \cap A(s')$, we require $x_{u,s} + x_{u,s'} \leqslant 1$ (to ensure that no user performs both s and s').
— For every at-most constraint c with scope Q, every step $s \in Q$ and every user $u \in A(s)$, we require (i) $x_{u,s} \leqslant y_{u,c}$; and (ii) $\sum_{u \in U} y_{u,c} \leqslant 3$ (at most three users can be assigned to steps in Q).
— For every step $s \in S$ and every equivalence class $X \subseteq U$, we require $\sum_{u \in X \cap A(s)} x_{u,s} = z_{X,s}$.
— For each constraint $(s, s', \sim)$ and equivalence class $X \subseteq U$, we require $z_{X,s} = z_{X,s'}$.
— For each constraint $(s, s', \nsim)$ and equivalence class $X \subseteq U$, we require $z_{X,s} + z_{X,s'} \leqslant 1$.