Constraint Branching in Workflow Satisfiability Problem

Gregory Gutin
Royal Holloway, University of London
Egham, United Kingdom
g.gutin@rhul.ac.uk

Daniel Karapetyan
University of Nottingham
Nottingham, United Kingdom
daniel.karapetyan@nottingham.ac.uk

ABSTRACT

There has been a considerable amount of interest in recent years in the problem of workflow satisfiability which seeks an allocation of authorised users to every step of the workflow, subject to workflow specification constraints. Unfortunately, the workflow satisfiability problem (WSP) where arbitrary constraints are allowed is computationally intractable. Wang and Li (2010) were the first to study WSP in the framework of parameterized complexity (with the parameter being the number of steps). Wang and Li proved that the WSP for arbitrary constraints is intractable even in the framework of parameterized complexity, i.e., it is highly unlikely to be fixed-parameter tractable (FPT). Extending the work of Wang and Li (2013) and Crampton et al. (2013), Cohen et al. (2014) introduced the family of user-independent (UI) constraints, which are constraints whose satisfiability does not depend on the identities of the users. Cohen et al. proved that WSP with UI constraints is FPT. Karapetyan et al. (2019) employed these ideas in practically efficient solution methods for WSP with UI constraints, including methods based on SAT and CSP general purpose solvers.

While the family of UI constraints includes the most common constraints used in practice, some real-world cases are outside of the family. In this paper, we generalise the concept of authorizations by making them context-dependent and show how to absorb some non-UI constraints into context-dependent authorizations. This allows us to extend algorithms and their implementations developed for WSP with UI constraints to arbitrary constraints. We carry out computational experiments with a general purpose SAT solver, SAT4J, to test practicality of solving WSP with UI and non-UI constraints using our approach.

KEYWORDS

workflow satisfiability; constraints; authorizations

ACM Reference Format:
Gregory Gutin and Daniel Karapetyan. 2020. Constraint Branching in Workflow Satisfiability Problem. In *The 25th ACM Symposium on Access Control Models and Technologies (SACMAT '20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3322431.3325418

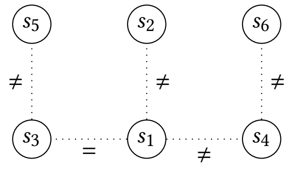
1 INTRODUCTION

Many businesses and other organizations use computerized systems to manage their business processes. A common example of such a system is a workflow management system, which is responsible for the coordination and execution of steps in a business process. Such a system is normally multi-user and thus it should include some form of access control which is facilitated by various restrictions on users to perform steps. It can be highly non-trivial to decide whether all the steps can be assigned to available users such that all restrictions are satisfied. Such a decision problem is called the Workflow Satisfiability Problem (WSP).

Example. Let us consider the following simple, illustrative example of an instance of the WSP. Figure 1 depicts a purchase order processing introduced in [9]. As shown in Figure 1(a), in the first two steps of the workflow, the purchase order is created and approved (and then dispatched to the supplier). 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.

s_1	create purchase order
s_2	approve purchase order
s_3	sign GRN
s_4	create payment
s_5	countersign GRN
s_6	approve payment

(a) Tasks



(b) Constraints

Figure 1: A simple constrained workflow for purchase order processing

Figure 1(b) shows constraints to prevent possible fraudulent use of the purchase order processing system. In our example, these constraints restrict the users that can perform pairs of steps in the workflow: the same user cannot sign and countersign the GRN, for example. There may also be a requirement that some steps are performed by the same user. In our example, the user that raises a purchase order is also required to sign for receipt of the goods. All in all, Part (b) shows five constraints

$$(s_1, s_2, \neq), (s_1, s_4, \neq), (s_3, s_5, \neq), (s_4, s_6, \neq), (s_1, s_3, =),$$

where $(s_i, s_j, \neq)$ is a binary separation-of-duty (SoD) constraint meaning that steps s_i and s_j have to be assigned to different users and $(s_i, s_j, =)$ is a binding-of-duty (BoD) constraint meaning that steps s_i and s_j have to be assigned to the same user. In particular, SoD and BoD constraints can be found in [1].

Let $S = \{s_i : i = 1, 2, \ldots, 6\}$ be the set of steps. To complete a WSP specification in this example, we introduce a set $U = \{u_i : i = 1, 2, \ldots, 8\}$ of users and describe authorization lists:

$$\begin{aligned} &A(s_1) = \{u_1, u_2\}, \quad A(s_2) = \{u_2, u_3\}, \quad A(s_3) = \{u_1, u_3\}, \\ &A(s_4) = \{u_3, u_4\}, \quad A(s_5) = \{u_3, u_4, u_5, u_8\}, \quad A(s_6) = \{u_5, u_6, u_7\}. \end{aligned}$$

The authorization list of s_i lists all users which can perform s_i.

It is not hard to verify that the following assignment $\pi : S \to U$ satisfies all the constraints and authorizations:

$$\begin{aligned} &\pi(s_1) = \pi(s_3) = u_1, \quad \pi(s_2) = u_2, \quad \pi(s_4) = u_4, \\ &\pi(s_5) = u_3, \quad \pi(s_6) = u_5. \end{aligned} \tag{1}$$

In general, the Workflow Satisfiability Problem (WSP)[1] contains a set S of steps and a set U of users and some constraints and authorizations restricting performance of steps by users (the difference between an authorization and a constraint is that while the former involves just one step, the latter at least two steps). The aim is to decide whether there is assignment of users to all steps such that all authorizations and steps are satisfied. Such WSP instances are called satisfiable; the other WSP instances are unsatisfiable.

Known WSP constraints. Research on workflow satisfiability began with the seminal work of Bertino, Ferrari and Atluri [3] and Crampton [9]. Some authors studying and using the WSP restrict themselves to binary SoD and BoD constraints only [4, 8, 16]. The interest in only binary SoD and BoD constraints can be explained by the fact that such constraints are the most widely used by many customers (in particular, those who use the appropriate SAP software) [17]. However, many authors consider more general WSP constraints, see e.g. [6, 12, 17, 24, 25] in order to facilitate more robust WSP systems.

In particular, Wang and Li [25] introduced relatively simple generalizations of binary SoD and BoD constraints and showed that while the WSP is already NP-hard when just binary SoD constraints are considered, even the WSP with their generalized SoD and BoD constraints admits efficient, fixed-parameter tractable (FPT) algorithms[2] when parameterized by $k = |S|$. Crampton et al. [12] introduced a more general family of WSP constraints, so-called regular constraints, for which the WSP still admits an FPT algorithm. While theoretically the algorithm of [12] is fast, it is unlikely to be so in practice as it uses the method of inclusion-exclusion which makes worst and best exponential running times pretty close to each other. Cohen et al. [6] introduced the concept of diversity in WSP constraints and showed that the WSP with constraints of relatively small diversity can be solved by an FPT algorithm. This algorithm is based on a practically useful concept of backtracking. Cohen et al. considered in particular user-independent constraints, which are of relatively small diversity. A constraint is called *user-independent (UI)* if its satisfiability does not depend on the identities of the users. UI constraints form a useful constraint family as it includes not only all constraints studied in [12] and [25], but also all constraints listed by the American National Standards Institute in [1].

Cohen et al. [7] implemented their algorithm for UI constraints and showed that it is practical interest. Karapetyan et al. [24] introduced and studied a different kind of backtracking algorithm for the WSP with UI constraints, which turned out to be significantly faster than the algorithm of [7]. Moreover, Karapetyan et al. showed that similar concepts can be used in formulations for general-purpose solvers such as the Pseudo-Boolean solver SAT4J [2], and the resulting methods are sufficient for many moderate-size WSP instances. Crampton et al. [10] introduced a generalization of UI constraints, the family of class-independent (CI) constraints, and proved that the WSP with CI constraints only, admits an FPT algorithm. However, to use CI algorithms, the set U of users has to have a hierarchical structure (e.g. if U are users in some hierarchical orgamization), which is rather restrictive. Dos Santos and Ranise [17] developed a software tool for the software company SAP which can be used to work, in particular, with the run-time version of the WSP which has UI and CI constraints. Apart from Pseudo-Boolean and Constraint Satisfaction Problem solvers used in [7, 24], many researchers studied WSP satisfiability using solvers based on Satisfiability Modulo Theories and Optimization Modulo Theories, see e.g. [5, 17]. We refer the reader to [18, 20] for surveys on workflow satisfiability approaches.

This work. In this paper, we introduce the notion of branching factor for every WSP constraint. This parameter is motivated by a basic algorithm of Karapetyan et al. [24], see Algorithm 1 in Section 2. To the best of our knowledge, the only other parameter for every WSP constraint is diversity which was introduced by Cohen et al. [6]. Diversity was also motivated by an algorithm (and used to estimate its running time), but the algorithm in [6] is less efficient from the theoretical point of view: its running time for UI constraints is $O(3^k B_k N^{O(1)})$ rather than $O(B_k N^{O(1)})$ of Algorithm 1, where $k = |S|$, B_k is the kth Bell number, and N is the size of the problem. More importantly, computational experiments in [24] clearly demonstrated that the implementation of the algorithm in [6] for the WSP with UI constraints only, is several orders of magnitude slower than that of the backtracking implementation of Algorithm 1 in [24]. Examples in Sections 4 and 5 demonstrate that for many interesting constraints it is not hard to provide a good upper bound constraint branching factor. This is not the case for diversity as it is a more complicated parameter [6].

It was shown in [24] that WSP with UI constraints can be efficiently solved using general-purpose solvers. Software solutions based on general-purpose solvers tend to be cheaper to develop and maintain than bespoke algorithms and thus strongly preferred by practitioners. Thus, in this paper, we test our approach using computational experiments with a general-purpose Pseudo-Boolean solver, SAT4J [2]. Benchmark instances are described in Section 5. They are chosen is such a way as to make them computationally hard to solve. We discuss our solution method in detail in Section 6. The results of the experiments on our benchmark instances are given in Section 7. They demonstrate that our approach provides a practical method of solving WSP with UI and non-UI constraints of relatively small branching factors. We conclude the paper in

[1]For a formal definition, see the beginning of Section 3.
[2]For a brief introduction to FPT algorithms, see Section 2.

Section 8, where in particular we discuss some directions of further research.

2 PARAMETERIZED ALGORITHMS AND COMPLEXITY

Unfortunately, the vast majority of non-trivial decision problems are intractable, i.e., NP-hard. One approach for solving an NP-hard decision problem is to use heuristics, but this means that the output is not always correct. Another approach is the use of parameterized algorithms. In this case, for the decision problem under consideration we assign a parameter (or, a collection of parameters which is usually aggregated to only one parameter, the sum of the original parameters). Examples of such parameters are the tree-width, tree-depth or other structural parameters of the input graph. This way the decision problem Π under consideration has two quantities: the size N of the problem and its parameter k. The usual choice of the parameter is such that k is much smaller than N on the instances of Π of interest. A problem together with its parameter is called a *parameterized problem.*

The reason for introducing the parameter is to try design an algorithm of running time $O(f(k)N^c)$, where f is a computable function of k only and c is an absolute constant. Such an algorithm is called *fixed-parameter tractable (FPT).* FPT algorithms exist for several parameterized NP-hard problems, in which cases $f(k)$ "absorbs" the classical computational complexity of the problem. A parameterized problem admitting an FPT algorithm is called an *FPT problem* (or, the problem is in the class FPT). Unfortunately, there are many parameterized problems for which there are good reasons to believe that they are not FPT. In parameterized complexity, such problems are called W[1]-hard and it is widely assumed that FPT≠W[1] [15].

To stress the fact that in an FPT algorithm running time $O(f(k)N^c)$ the function f is usually exponential (it does not have to be if the problem is polynomial-time solvable) and thus $f(k)$ is often the dominating factor in the running time, $O(f(k)N^c)$ is often simplified to $O^*(f(k))$. For more information about parameterized algorithms and complexity, we refer the reader to the monograph [15].

3 WSP, PLANS, PATTERNS AND PATTERN BASIC ALGORITHM

The WSP in all its generality is a problem with complicated constraints which may require a certain partial order in which the steps are performed, exclusion of certain steps from being performed by a plan under certain conditions, etc., see e.g. [5, 13, 14, 17]. However, it is possible to reduce such a general WSP to a number of more basic WSPs in which there are no constraints imposing a partial order of the steps (i.e. the order in which the steps are performed is immaterial), no steps are excluded from being performed, etc., see e.g. [13, 14]. Because of the reductions and since new WSP notions introduced in this paper are already of interest for the basic WSP, we will restrict our attention to the basic WSP (an instance of such a WSP is depicted in Figure 1). We will define it as follows.

Let S be a set of steps, U is a set of users, C is a set of non-unary constraints whose scopes are subsets of S (called just *constraints* in what follows[3]) and A is an *authorization function* (or, just *authorization*) $A: S \to 2^U$. Thus, every WSP instance is given by a quadruple (S, U, C, A). A *plan* π is a function $\pi : S \to U$. The aim is to decide whether there is plan which is *authorized* i.e. $\pi(s) \in A(s)$ for each $s \in S$, and *eligible* i.e. satisfies all constraints. A plan is called *valid* if it is authorized and eligible. If a WSP instance (S, U, C, A) has a valid plan, it is called *satisfiable* and otherwise *unsatisfiable.* It is easy to see that the WSP with only binary SoD constraints is NP-hard as it is equivalent to the well-known NP-hard Graph List Coloring problem.

Let us denote $|S|$ by k and $|U|$ by n. Wang and Li [25] observed that in real-world WSP instances k is often much smaller than n. This led them to introduce parameterization of the WSP by k. We will also study this parameterizations like several other papers, cf. [6, 7, 19, 24]. Wang and Li showed that the WSP is W[1]-hard.

Remark 1. *In fact, Wang and Li [25] proved the following result. Let ρ be a binary relation on U and let $S = \{s_1, \dots, s_k\}$. Then the WSP is W[1]-hard if $A(s) = U$ for every $s \in S$ and C consists of $\binom{k}{2}$ constraints $c_{i,j}$ with $1 \le j < i \le k$ such that a plan π satisfies $c_{i,j}$ if and only if $(\pi(s_i), \pi(s_j)) \in \rho$. The W[1]-hardness follows by a simple reduction from the* (Graph) Independent Set *problem*[4]*, which is W[1]-hard.*

Let us now define a WSP constraint formally. A constraint c with *scope* $T \subseteq S$ is a pair (T, Θ_c), where Θ_c is a set of functions $\theta : T \to U$ such that c is *satisfied by a plan* π if for some $\theta \in \Theta_c$, $\theta(t) = \pi(t)$ for each $t \in T$. For example, the scope of a (general) SoD constraint c is a subset T of S and $\theta \in \Theta_c$ if and only if θ is injective.[5] Satisfiability of essentially every real-world constraint c by a plan π can be decided without using the formal definition of a constraint (e.g., deciding whether an SoD constraint is satisfied by a plan π can be done by a simple inspection of π) and in polynomial time in k and n. In this paper, we assume that satisfiability of every constraint by a plan can be decided in polynomial time in k and n.

Recall that a constraint is called *user-independent (UI)* if its satisfiability does not depend on the identities of the users. For example, SoD and BoD constraints are UI. The following generalizations of SoD and BoD constraints are also UI. The scope of an *at-least-p-out-of-q constraint* is a subset T of S of size q and at least p users can be assigned to the steps of T. Analogously, the scope of *an at-most-p-out-of-q constraint* is a subset T of S of size q and at most p users can be assigned to the steps of T. Many WSP constraints are UI and in fact, all constraints listed by American National Standards Institute in [1] are UI. Formally, a constraint $c = (T, \Theta_c)$ is UI if for every $\theta \in \Theta_c$, and every permutation $f: U \to U$ of users there is a $\theta' \in \Theta_c$ such that $\theta'(t) = f(\theta(t))$ for each $t \in T$.

To naively decide whether a WSP instance (S, U, C, A) has a valid plan, one can generate plans one by one and check whether one of them is valid, stopping when a valid one is found. However, such an algorithm is not efficient as the total number of plans is n^k. Since the WSP is W[1]-hard, it is highly unlikely that much more efficient, i.e., FPT algorithms exist. However, there are FPT algorithms if all constraints are UI (and with no restrictions on authorizations). To

[3]We will formally define a constraint later in this section.
[4]Set ρ to be the non-adjacency relation in the input graph.
[5]For an example of a non-binary WSP SoD constraint, see Constraint 1 in [25, Example 1].

describe such an algorithm, we will use the notion of a pattern introduced by Cohen et al. [6] but we will follow the definition of patterns from Crampton et al. [11].

A *pattern* is a partition $p = \{S_1, \ldots, S_p\}$ of S into non-empty subsets $S_1, \ldots, S_p$ ($S_1 \cup \cdots \cup S_p = S$ and $S_i \cap S_j = \emptyset$ for every $i \neq j$) called *blocks*. We will denote the set of all patterns of S by $P(S)$. For example, if $S = \{s_1, s_2, s_3\}$ then $P(S) = \{p_1, p_2, p_3, p_4, p_5\}$, where $p_1 = \{\{s_1\}, \{s_2\}, \{s_3\}\}$ (every block of p_1 is a singleton), $p_2 = \{\{s_1\}, \{s_2, s_3\}\}$, $p_3 = \{\{s_2\}, \{s_1, s_3\}\}$, $p_4 = \{\{s_3\}, \{s_1, s_2\}\}$ and $p_5 = \{\{s_1, s_2, s_3\}\}$ (p_5 has only one block).

The *pattern p of a plan π* is a partition of S into blocks of steps such that all the steps in a block are assigned the same user but steps in different blocks are assigned different users. For example, the plan (1) has pattern $p = \{\{s_1, s_3\}, \{s_2\}, \{s_4\}, \{s_5\}, \{s_6\}\}$. The number of blocks in p will be denoted by $|p|$. By the definition of a UI constraint, a pair π', π'' of plans with the same pattern either both satisfy a UI constraint c or both violate it [6]. Thus, it suffices to know the pattern of a plan π (without knowing π itself) to decide whether a UI constraint c is satisfied by π or not and we can say whether a *pattern p* of S *satisfies* a UI constraint c (meaning that either all plans with pattern p satisfy c or none do). Thus, we have the following observation:

Remark 2. *To decide whether a WSP instance (S, U, C, A) with only UI constraints has an eligible plan, it suffices to go over all patterns of S and check whether there is a pattern that satisfies all constraints in C (we call such a pattern* eligible*).*

Recall that not every eligible plan is valid as authorizations A have to be satisfied, too. Following Karapetyan et al. [24], to decide whether there exists a valid plan with an eligible pattern p, we construct a bipartite graph G_p with partite sets p (where every vertex is a block of p) and U such that $bu \in E(G_p)$ (b is a block in p and $u \in U$) if and only if $u \in A(s)$ for every $s \in b$. Observe that a plan with pattern p is authorized if and only if G_p contains a matching M' saturating p, i.e. for every vertex (block) $b \in p$ there is an edge in M' incident to b [24]. Since $k \leq n$, this means that a plan with pattern p is authorized if and only if the size of a maximum matching M of G_p equals $|p|$, the number of blocks in p. If $|M| = |p|$, an authorized plan π *corresponding* to M can be constructed as follows: $\pi(s) = u$ if and only if $s \in b$ and $bu \in M$ [24]. For an example of graph G_p, see Figure 2.

This leads to a simple WSP algorithm using patterns whose pseudo-code is given in Algorithm 1, where UNSAT indicates that the input instance is unsatisfiable. A more sophisticated and much more efficient in computational experiments pattern-based backtracking algorithm was studied by Karapetyan et al. [24].

Note that $|P(S)|$, i.e., the number of partitions of a set of size k, equals the kth Bell number[6] $B_k \leq k! = O(2^{k \log k})$. Now it is not hard to see that the running time of Algorithm 1 is $O^*(2^{k \log k})$. More advanced algorithms in [6, 24] for the WSP with only UI constraints are also of running time $O^*(2^{k \log k})$. In fact, the time $O^*(2^{k \log k})$ is highly likely to be optimal: Cohen et al. [6] proved that unless the Exponential Time Hypothesis (ETH) fails, there is no algorithm of running time $O^*(2^{o(k \log k)})$ for the WSP with UI constraints; Gutin and Wahlström [19] showed that unless the Strong ETH fails, there is no algorithm of running time $O^*(c^{k \log k})$ for any constant $c < 2$ for the WSP with UI constraints only.[7]

[6]All logarithms in this paper are of base 2.

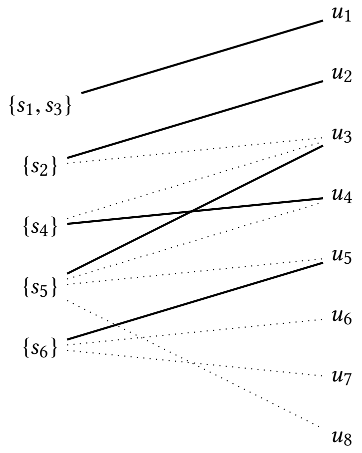


Figure 2: Graph G_p constructed for the example given in Section 1. The authorisations in this example are as follows: $A(s_1) = \{u_1, u_2\}, A(s_2) = \{u_2, u_3\}, A(s_3) = \{u_1, u_3\}, A(s_4) = \{u_3, u_4\}, A(s_5) = \{u_3, u_4, u_5, u_8\}$ **and** $A(s_6) = \{u_5, u_6, u_7\}$**. The pattern is** $\{\{s_1, s_3\}, \{s_2\}, \{s_4\}, \{s_5\}, \{s_6\}\}$**. The matching shown in bold lines corresponds to the valid plan (1):** $\pi(s_1) = \pi(s_3) = u_1$, $\pi(s_2) = u_2$, $\pi(s_4) = u_4$, $\pi(s_5) = u_3$, $\pi(s_6) = u_5$.

Algorithm 1: Pattern Basic Algorithm

```
   input  : WSP instance $W = (S, U, C, A)$
   output : Valid plan $\pi$ or UNSAT
1  for $p \in P(S)$ do
2      from $A$ compute $G_p$;
3      compute a maximum matching $M$ in $G_p$;
4      if $p$ is eligible and $|M| = |p|$ then
5          return plan $\pi$ corresponding to $M$;
6      else
7          return UNSAT
```

4 WSP WITH CONTEXT-DEPENDENT AUTHORIZATIONS AND CONSTRAINT BRANCHING FACTOR

While authorizations A in the WSP defined above are fixed, we may expect more expressibility from the WSP if A is not fixed. A careful consideration of the authorization part of Algorithm 1 shows that the algorithm may remain FPT if we introduce the following *context-dependent authorizations (CDAs)* $\mathcal{A} = \{\mathcal{A}_p : p \in P(S)\}$. Here $\mathcal{A}_p$ is a "disjunctive" set of authorization functions $\{A_1^{(p)}, \ldots, A_{d_p}^{(p)}\}$ ($d_p = |\mathcal{A}_p|$ and each $A_i^{(p)}$ maps S to 2^U). We will call $\mathcal{A}_p$ a *p-authorization family* and $\mathcal{A}$ an *authorization family.* A pattern p is

[7]The ETH claims that 3-SAT with n variables cannot be solved in time $O(2^{o(n)})$ [21]. The Strong ETH claims that SAT with n variables cannot be solved in time $O(c^n)$ for any $c < 2$ [22]. Note that both ETH and Strong ETH are conjectures.

Algorithm 2: CDA Basic Algorithm

```
input  : WSP instance $W = (S, U, C, \mathcal{A})$, where
         $\mathcal{A}_p = \left\{ A_1^{(p)}, \dots, A_{d_p}^{(p)} \right\}$ for each $p \in P(S)$.
output : Valid plan $\pi$ or UNSAT
1 for $p \in P(S)$ and $i \in [d_p]$ do
2     from $A_i^{(p)}$ compute $G_{p,i}$;
3     compute a maximum matching $M$ in $G_{p,i}$;
4     if $p$ is eligible and $|M| = |p|$ then
5         return plan $\pi$ corresponding to $M$;
6     else
7         return UNSAT
```

authorized for a WSP instance $(S, U, C, \mathcal{A})$ if it is authorized for at least one of the authorization functions in $\mathcal{A}_p$.

As an example, consider the instance of WSP given in Figure 1. Recall that it has $S = \{s_1, \dots, s_6\}$, $U = \{u_1, \dots, u_8\}$, and five constraints

$$(s_1, s_2, \neq), (s_1, s_4, \neq), (s_3, s_5, \neq), (s_4, s_6, \neq), (s_1, s_3, =).$$

Let us modify the WSP instance by letting $\mathcal{A} = \{\mathcal{A}_p : p \in P(S)\}$, where $\mathcal{A}_p = \{A_i^{(p)} : i = 1, 2\}$ for each $p \in P(S)$ as follows:

$$A_1^{(p)}(s_2) = \{u_2\}$$
$$A_1^{(p)}(s_6) = \{u_5, u_6\}$$
$$A_1^{(p)}(s_i) = U \text{ for } i = 1, 3, 4, 5$$

$$A_2^{(p)}(s_2) = \{u_3\}$$
$$A_2^{(p)}(s_6) = \{u_5, u_6, u_7\}$$
$$A_2^{(p)}(s_i) = U \text{ for } i = 1, 3, 4, 5$$

These CDAs describe the following logic. If user u_2 is assigned to approve purchase order (step s_2) then only users u_5 and u_6 are allowed to approve payment (step s_6); however if user u_3 is assigned to approve purchase order, user u_7 is also allowed to approve payment. It is not hard to see that plan (1) satisfies $A_1^{(p)}$ but does not satisfy $A_2^{(p)}$. Since (1) satisfies at least one authorization family, it is a valid plan for our modified instance.

Consider Algorithm 2, which is a straightforward modification of Algorithm 1, where $G_{p,i}$ is the bipartite graph corresponding to the authorization function $A_i^{(p)}$. It is not hard to see that Algorithm 2 still has the running time $O^*(2^{k \log k})$ if $\mathcal{A}_p$ is of polynomial size in both n and k for each pattern p. If we allow the size of each $\mathcal{A}_p$ to be bounded by $O(g(k)n^{r_p})$, where g is a computable function of k only and r_p is an absolute constant, then the modified basic algorithm will still be FPT (of running time $O^*(g(k)2^{k \log k})$). To distinguish the WSP introduced earlier in the paper and its generalization with CDAs, we will denote the latter by WSP-CDA.

While the WSP-CDA seems to be of interest in its own right, we will show below that its use will allow us to absorb *all* non-UI constraints into CDAs. Below we will define the branching factor of a constraint. Our absorption is efficient for constraints of small branching factor, especially those of branching factor 1. In fact, constraints of branching factor 1 are direct generalizations of UI constraints, see Proposition 4.2 below.

Let

$$\mathcal{A}' = \left\{ \mathcal{A}'_p : p \in P(S) \right\} \text{ and } \mathcal{A}'' = \left\{ \mathcal{A}''_p : p \in P(S) \right\}$$

be a pair of authorization families, where each

$$\mathcal{A}'_p = \left\{ A'_{p,1}, \dots, A'_{p,d'_p} \right\} \text{ and } \mathcal{A}''_p = \left\{ A''_{p,1}, \dots, A''_{p,d''_p} \right\}$$

is a p-authorization family. Then the *intersection* $\mathcal{A} := \mathcal{A}' \cap \mathcal{A}''$ is an authorization family such that

$$\mathcal{A}_p = \left\{ A'_{p,i} \cap A''_{p,j} : i \in [d'_p], j \in [d''_p] \right\},$$

where each $A'_{p,i} \cap A''_{p,j}$ is an athorization function such that

$$A'_{p,i} \cap A''_{p,j}(s) := A'_{p,i}(s) \cap A''_{p,j}(s).$$

A pair $(S, U, C', \mathcal{A}')$ and $(S, U, C'', \mathcal{A}'')$ of WSP-CDA instances are called *plan-equivalent* if for every plan $\pi : S \to U$, π is valid for one of them if and only if it is valid for the other.

A constraint c is called *m-branching* if there is an authorization family $\mathcal{A}^{(c)} = \left\{ \mathcal{A}_p^{(c)} : p \in P(S) \right\}$, where each

$$\mathcal{A}_p^{(c)} = \left\{ A_{p,1}^{(c)}, \dots, A_{p,m_p}^{(c)} \right\}$$

is a p-authorization family with $m_p \leq m$, such that every WSP-CDA instance $(S, U, C, \mathcal{A})$ with $c \in C$ is plan-equivalent to the WSP-CDA instance $(S, U, C \setminus \{c\}, \mathcal{A} \cap \mathcal{A}^{(c)})$. We call $\mathcal{A}^{(c)}$ an *authorization family* of c. Let $m(c, \mathcal{A}^{(c)}) := \max_{p \in P(S)} m_p$. The *branching factor* $m(c)$ of c is the minimum of $m(c, \mathcal{A}^{(c)})$ over all authorization families $\mathcal{A}^{(c)}$ of c. We have the following:

Proposition 4.1. *Every constraint c is m-branching for some m.*

Proof. Consider $\mathcal{A}_p^{(c)} = \left\{ A_{p,\pi}^{(c)} : \pi \in \Pi_p \right\}$, where Π_p is the set of all plans with pattern p and $A_{p,\pi}^{(c)}(s) = \pi(s)$ for every $s \in S$. We can set $m = \max_{p \in P(S)} |\mathcal{A}_p^{(c)}|$. □

The above proposition shows that every WSP constraint, even if it is a non-UI constraint, can be absorbed into CDAs. For some WSP constraints, though, the branching factor will be impractically large.

Let us consider examples of a constraint of branching factor 1 and a constraint of branching factor $m \geq 2$. Further examples of CDA constraints are given in Section 5.2.

Example 1. (Type 1 CDA constraint) *In the following constraint c, let $T \subseteq S$ be such that if at most h users are assigned to perform steps of T, then these users have to be of certain high qualification and/or security clearance, i.e., they have to belong to a set $X \subset U$.*

We can define a p-authorization family for c as follows: $\mathcal{A}_p^{(c)} = \left\{ A_p^{(c)} \right\}$ such that for every $s \in S$, $A_p^{(c)}(s) = X$ if p has at most h blocks covering T and $A_p^{(c)}(s) = \emptyset$, otherwise. Note that $m(c) = 1$, but c is not UI (see Proposition 4.2). Indeed, already for a set X of size 1, c has dependence on users.

Example 2. (Type 2 CDA constraint) *This constraint c is a generalization of Constraint 2 in [25, Example 1], where $|T| = 2$. There are d departments whose users are allowed to perform steps in $T \subseteq S$. However, c requires that users of only one department are allowed to perform all steps in T. Let U_i be the users of Department i, $1 \le i \le d$ such that $U_i \cap U_j = \emptyset$ for every $1 \le i < j \le d$.*

We can define a p-authorization family for c as follows:

$$\mathcal{A}_p^{(c)} = \left\{A_{p,1}^{(c)}, \ldots, A_{p,d}^{(c)}\right\},$$

where for $j \in [d]$, $A_{p,j}^{(c)}(t) = U_j$ for all $t \in T$ and $A_{p,j}^{(c)}(s) = U$ for all $s \in S \setminus T$. Thus, $m(c) \le d$.

The next result shows that constraints of branching factor 1 generalize UI constraints. This follows from Proposition 4.2 and the fact that type 1 CDA constraints are non-UI constraints of branching factor 1.

Proposition 4.2. *Every UI constraint is of branching factor 1.*

Proof. Let c be a UI constraint and let P_c be the set of patterns in $P(S)$ which satisfy c. By Remark 2, we can define a p-authorization family for c as follows: $\mathcal{A}_p^{(c)} = \left\{A_p^{(c)}\right\}$, where for every $s \in S$, we have $A_p^{(c)}(s) = U$ if $p \in P_c$ and $A_p^{(c)}(s) = \emptyset$, otherwise. □

For the purpose of building an algorithm for WSP-CDA, we can interpret m-branching constraints as follows. We will say that a constraint c is *absorbed* into $\mathcal{A}$, i.e., c is removed from C at the cost of replacing $\mathcal{A}$ by $\mathcal{A} \cap \mathcal{A}^{(c)}$. The size of $\mathcal{A} \cap \mathcal{A}^{(c)}$ will depend on the branching factor of c, the best case being $m(c) = 1$. Since $(S, U, C \setminus \{c\}, \mathcal{A} \cap \mathcal{A}^{(c)})$ is a WSP-CDA instance, if $(S, U, C, \mathcal{A})$ has more than one constraint, we can similarly absorb all of them into $\mathcal{A}$ one by one. Algorithm 2 shows that there is no need to absorb UI constraints and so our absorption approach may be efficient if both the number of non-UI constraints and their branching factors are relatively small. We will formalise this remark as follows.

Theorem 4.3. *Let $W = (S, U, C, \mathcal{A})$ be a WSP-CDA instance in which all but ℓ constraints are UI and the ℓ constraints are of branching factor at most m. Then W can be solved in time $O^*(m^\ell 2^{k \log k})$.*

Proof. Let each p-authorization family $\mathcal{A}_p$ of W contain at most d authorizations functions for every $p \in P(S)$. Let us first absorb the ℓ non-UI constraints one by one. The resulting WSP-CDA instance W' will have each p-authorization family with at most dm^ℓ authorization functions. Thus, the running time to absorb all non-UI constraints will be $O^*(\ell dm^\ell) = O^*(m^\ell)$. Now we can apply Algorithm 2 to W'. The running time of Algorithm 2 will be $O^*(m^\ell 2^{k \log k})$. Hence, the total running time will be $O^*(m^\ell 2^{k \log k})$. □

Corollary 4.4. *Let $W = (S, U, C, \mathcal{A})$ be a WSP-CDA instance in which all but ℓ constraints are UI and the ℓ constraints are of branching factor at most m. We have the following:*

1. *If ℓ and m are computable functions of k only, then W can be solved in FPT time.*
2. *If either $m = 1$ or ℓ and m are constants, then W can be solved in time $O^*(2^{k \log k})$.*
3. *Consider constraints $c_{i,j}$, $1 \le j < i \le k$ defined in Remark 1. Unless FPT $= W[1]$, there is no computable function h of k only such that $m(c_{i,j}) \le h(k)$ for any $1 \le j < i \le k$.*

Proof. (1) and (2) immediately follow from Theorem 4.3. To prove (3), suppose that there is computable function h of k only such that $m(c_{i,j}) \le h(k)$ for some $1 \le j < i \le k$. By symmetry of constraints $c_{i,j}$, we have $m(c_{i,j}) \le h(k)$ for all $1 \le j < i \le k$. Consider the WSP with $S = \{s_1, \ldots, s_k\}$, $C = \left\{c_{i,j} : 1 \le j < i \le k\right\}$ and $A(s_i) = U$ for every $i = 1, \ldots, k$. Note that $\ell = |C| = \binom{k}{2}$. By (1), the WSP is FPT. However, by Remark 1 the WSP is W[1]-hard. If there is no contradiction, FPT must be equal to W[1]. □

5 FURTHER EXAMPLES AND BENCHMARK INSTANCES

For our computational experiments, we developed a WSP-CDA pseudo-random instance generator. It is based on the WSP-UI generator described in [23, 24]. The original generator produces users with random uniformly distributed authorizations and user-independent constraints of three types: not-equals, requiring that two steps are assigned different users; at-least-3-of-5, requiring that at least three distinct users are assigned the five-step scope; and at-most-3-of-5, requiring that at most three distinct users are assigned the five-step scope. The constraints are produced uniformly at random. For more details refer to [24].

Our WSP-CDA generator extends the WSP-UI generator by adding CDA constraints of several types, specifically the two constraints described above (1 and 2), and the three constraints described in Section 5.2. The parameters of each benchmark instance are discussed in Section 5.3.

5.1 Parameters for Type 1 and 2 CDA Constraints

When generating Type 1 CDA constraints, we randomly choose scope $T \subset S$ of size 5, the number h is set to 2 and we randomly choose 10 users for the set $X \subset U$.

When generating Type 2 CDA constraints, we randomly choose scope $T \subset S$ of size 3, the number of teams is fixed to 3 and each team consists of 10 randomly chosen users (we guarantee that each user belongs to at most one team).

5.2 Further Examples of CDA Constraints

In this section, we consider three more constraints and describe their branching families, which we believe to be optimal, i.e., corresponding to the branching factor. However, we have no optimality proofs.

Example 3. (Type 3 CDA constraint) *Let $T \subseteq S$ and let X be a non-empty set of users. The constraint c requires that at least one step in T is performed by a user from X, e.g., to ensure that at least one step in a subflow is overseen by a senior member of staff.*

A p-authorization family of c can be written as follows, where $T = \{s_1, \ldots, s_t\}$. For each $p \in P(S)$, let

$$\mathcal{A}_p^{(c)} = \left\{A_{p,1}^{(c)}, \ldots, A_{p,t}^{(c)}\right\},$$

where for $j \in \{1, \ldots, t\}$, $A^{(c)}_{p,j}(s_j) = X$ *and*

$$A^{(c)}_{p,j}(s_i) = U \text{ for } i \in \{1, \ldots, k\} \setminus \{j\}\,.$$

Thus, $m(c) \le t$.

When generating Type 3 CDA constraints, we randomly choose the scope T of size 5 and randomly choose the set of users X of size 10.

Example 4. (Type 4 CDA constraint) *This constraint c is defined as follows. Let* s_1 *and* s_2 *be two distinct steps, and* $X \subsetneq U$ *be a set of users. Constraint c requires that only the users X are allowed to be assigned both steps within a workflow. This can be useful to ensure that only most trusted users can perform both steps.*

A p-authorization family of c can be written as follows. For each $p \in P(S)$ *such that* $(s_1, s_2, =)$ *holds, let* $\mathcal{A}^{(c)}_p = \left\{A^{(c)}_p\right\}$, *where*

$$A^{(c)}_p(s_1) = X,$$
$$A^{(c)}_p(s_2) = X,$$
$$A^{(c)}_p(s_i) = U \text{ for } 3 \le i \le k.$$

For each $p \in P(S)$ *such that* $(s_1, s_2, \ne)$ *holds, let* $\mathcal{A}^{(c)}_p = \left\{A^{(c)}_p\right\}$, *where*

$$A^{(c)}_p(s_i) = U \text{ for } 1 \le i \le k.$$

Thus, $m(c) = 1$. Note that c is not a UI constraint.

When generating Type 4 CDA constraints, we randomly select distinct steps s_1 and s_2 and the set X of size 5.

Example 5. (Type 5 CDA constraint) *In this constraint c, if an employee from a defence department is assigned* s_1*, then* s_2 *(a signature step) must be assigned to someone from the security department. Let* U_1 *be the set of users in the defence department, and let* U_2 *be the set of users in the security department. We assume that* $U_1 \cap U_2 = \emptyset$.

Let $p \in P(S)$. *A p-authorization family of c can be written as follows. Let* $\mathcal{A}^{(c)}_p = \left\{A^{(c)}_{p,1}, A^{(c)}_{p,2}\right\}$, *where*

$$A^{(c)}_{p,1}(s_1) = U_1,$$
$$A^{(c)}_{p,1}(s_2) = U_2,$$
$$A^{(c)}_{p,1}(s_j) = U \text{ for } 3 \le j \le k,$$
$$A^{(c)}_{p,2}(s_1) = U \setminus U_1,$$
$$A^{(c)}_{p,2}(s_j) = U \text{ for } 2 \le j \le k.$$

Thus, $m(c) \le 2$.

When generating Type 5 CDA constraints, we randomly choose distinct steps s_1 and s_2 and sets of users U_1 and U_2 such that $U_1 \cap U_2 = \emptyset$ and $|U_1| = 100$ and $|U_2| = 10$.

5.3 Instance Generator Parameters

Following [24], we choose to include only phase-transition instances into our benchmark. Phase-transition instances are instances at the boundary between under-subscribed and over-subscribed regions. These are typically the hardest to solve instances, and are most suitable for empirical evaluation of the average running time.

To illustrate phase transition when the number of constraints is varied, we plotted the running time and the probability of an instance being satisfiable in Figure 3. We chose to use only the Type 5 CDA constraints, and we vary the number of these constraints in a wide range. All other parameters are fixed; $k = 45$. For each combination of parameters, 100 random instances are generated and then the median running time is reported.

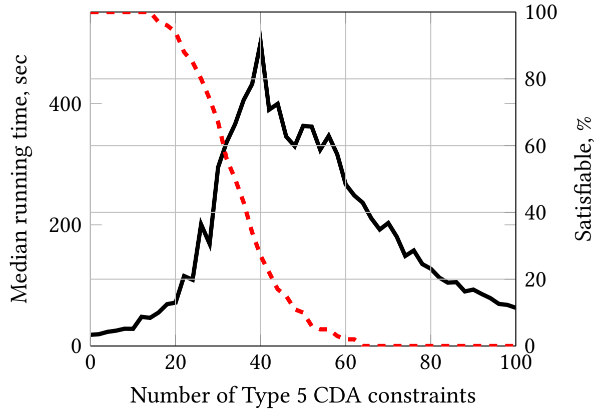


Figure 3: Solution time (solid black line) and satisfiability probability (red dashed line) as they change with the number of Type 5 CDA constraints.

In this experiment we observe that if the number of Type 5 CDA constraints is low then the instances are very likely to be satisfiable, with multiple solutions, and finding one is relatively easy. On the other end we see over-subscribed instances, with the satisfiability probability being very low, and proving the unsatisfiability is also relatively easy due to intensive propagation. The hardest instances are in between, when the satisfiability probability is close to 50%.

This shows the importance of selecting phase-transition instances; without appropriate analysis, we would likely choose under- or over-subscribed instances that are easy to solve and our conclusions regarding the performance of the solution methods would be premature.

To generate the benchmark instances, we used the following parameters for our instance generator. The number of users was set to $n = 10k$, the number of not-equals, at-least and at-most constraints was set to k each. The number of CDA constraints was chosen individually for each instance, to ensure that the probability of such an instance being satisfiable is close to 50%, i.e. that the instance is in the phase transition region. The specific values we used are reported in Table 1.

Each combination of parameters is assigned a name based on the parameters, and for each of such combinations we generated 100 random instances, by using different random number generator seed values.

6 SOLUTION METHODS

To fully exploit the FPT properties of WSP-CDA, our first choice would be a bespoke algorithm. However, it was shown in [24] that

		CDA constraint type				
Instance	k	T. 1	T. 2	T. 3	T. 4	T. 5
wsp-cda-30-1	30	89	0	0	0	0
wsp-cda-30-2	30	0	1	0	0	0
wsp-cda-30-3	30	0	0	3	0	0
wsp-cda-30-4	30	0	0	0	23	0
wsp-cda-30-5	30	0	0	0	0	14
wsp-cda-35-1	35	130	0	0	0	0
wsp-cda-35-2	35	0	1	0	0	0
wsp-cda-35-3	35	0	0	4	0	0
wsp-cda-35-4	35	0	0	0	31	0
wsp-cda-35-5	35	0	0	0	0	22
wsp-cda-40-1	40	125	0	0	0	0
wsp-cda-40-2	40	0	1	0	0	0
wsp-cda-40-3	40	0	0	4	0	0
wsp-cda-40-4	40	0	0	0	31	0
wsp-cda-40-5	40	0	0	0	0	25
wsp-cda-45-1	45	166	0	0	0	0
wsp-cda-45-2	45	0	1	0	0	0
wsp-cda-45-3	45	0	0	4	0	0
wsp-cda-45-4	45	0	0	0	43	0
wsp-cda-45-5	45	0	0	0	0	34

Table 1: This table gives the number of CDA constraints of different types used in the benchmark instances.

WSP with UI constraints can be efficiently solved using general-purpose solvers. Software solutions based on general-purpose solvers tend to be cheaper to develop and maintain and thus strongly preferred by practitioners. In this paper, we focus on solving WSP-CDA with Pseudo-Boolean (PB) solver to verify that this approach works well for WSP-CDA too. We use the PBPB formulation from [24] as a base while extending it with encodings of the CDA constraints. We produce two PB encodings of each CDA constraint:

- **'CDA'** is based on the CDA concept, that is it exploits the M variables (see formulation (2)–(14) for details) whenever the authorizations depend on the pattern and introduces auxiliary variables to choose between the authorization functions in the family;
- **'Naive'** is based on straightforward encodings of each constraint.

6.1 PB Formulation for WSP with UI constraints

To make the paper self-contained, we give here the PBPB formulation from [24].

$$M_{s_1,s_2} = M_{s_2,s_1} \qquad \forall s_1 \neq s_2 \in S, \tag{2}$$

$$M_{s,s} = 1 \qquad \forall s \in S, \tag{3}$$

$$M_{s_1,s_2} \geq M_{s_1,s_3} + M_{s_2,s_3} - 1 \qquad \forall s_1 \neq s_2 \neq s_3 \in S, \tag{4}$$

$$M_{s_1,s_2} \leq M_{s_2,s_3} - M_{s_1,s_3} + 1 \qquad \forall s_1 \neq s_2 \neq s_3 \in S, \tag{5}$$

$$\sum_{u \in U} x_{s,u} = 1 \qquad \forall s \in S, \tag{6}$$

$$x_{s_1,u} - x_{s_2,u} \leq 1 - M_{s_1,s_2} \qquad \forall s_1 \neq s_2 \in S \text{ and } \forall u \in U, \tag{7}$$

$$x_{s_1,u} + x_{s_2,u} \leq 1 + M_{s_1,s_2} \qquad \forall s_1 \neq s_2 \in S \text{ and } \forall u \in U, \tag{8}$$

$$x_{s,u} = 0 \qquad \forall s \in S \text{ and } \forall u \in U \setminus A^{-1}(s), \tag{9}$$

$$M_{s_1,s_2} = 0 \qquad \forall \text{ not-equals constr. } \{s_1, s_2\}, \tag{10}$$

$$\sum_{s_1 < s_2 \in T} M_{s_1,s_2} \geq 2 \qquad \forall \text{ at-most constr. with scope } T, \tag{11}$$

$$\sum_{s_1 < s_2 \in T} M_{s_1,s_2} \leq 3 \qquad \forall \text{ at-least constr. with scope } T, \tag{12}$$

$$M_{s_1,s_2} \in \{0,1\} \qquad \forall s_1, s_2 \in S, \tag{13}$$

$$x_{s,u} \in \{0,1\} \qquad \forall s \in S \text{ and } \forall u \in U. \tag{14}$$

The $x_{s,u}$ variables encode the assignment of users to steps; if $x_{s,u} = 1$ then user u is assigned step s. The M_{s_1,s_2} are auxiliary variables; $M_{s_1,s_2} = 1$ if and only if steps s_1 and s_2 are assigned the same user. While not compulsory, the M variables allow compact and efficient encoding of UI constraints as they capture the concept of patterns.

It was shown in [24] that any UI constraint can be formulated using the M-variables only. Clearly, to encode the CDA constraints, one may need the x-variables. Below we give Naive and CDA-based formulations of each of the five CDA constraints introduced above.

6.2 Type 1 CDA Constraint Formulation

Recall that Type 1 CDA constraint requires that, if the number of users assigned steps $T \subset S$ is at most h, then these users have to be from set $X \subsetneq U$.

Our 'CDA' formulation of such a constraint depends on h and $|T|$; below we give a formulation for $h = 2$ and scope of size 5:

$$7z \leq 10 - \sum_{s_1 < s_2 \in T} M_{s_1,s_2} \tag{15}$$

$$x_{s,u} \leq z \qquad \forall u \in U \setminus X, \ \forall s \in T \tag{16}$$

$$z \in \{0,1\} \tag{17}$$

Here z is forced to 0 if the number of users assigned to the scope T is at most 2, see (15). Indeed, if the number of users assigned to T is one or two, the number of pairs of steps s_1, s_2 assigned equal users has to be at least 4, see [24]. If z is zero, the users not from X are prohibited from performing any step in T, see (16).

Our 'Naive' formulation of such a constraint is based on counting the number of users assigned to the scope:

$$y_u \le \sum_{s \in T} x_{s,u} \qquad \forall u \in U, \tag{18}$$

$$|T|(z-1) + h + 1 \le \sum_{u' \in U} y_{u'} \tag{19}$$

$$x_{s,u} \le z \qquad \forall u \in U \setminus X,\ \forall s \in T \tag{20}$$

$$z \in \{0,1\} \tag{21}$$

$$y_u \in \{0,1\} \qquad \forall u \in U \tag{22}$$

Variable y_u is forced to 0 if none of the steps in T is assigned user u, see (18). Then z is forced to 0 if the number of users assigned to T is at most h, see (19). Finally, users other than X are prohibited from being assigned to any $s \in T$ if z is 0, see (20).

6.3 Type 2 CDA Constraint Formulation

Recall that the Type 2 CDA constraint requires that all the users assigned to steps $T \subseteq S$ should be from exactly one department. In this constraint, we are given a set of steps $T \subseteq S$ and d departments $U_1, U_2, \dots, U_d \subset U$ such that $U_i \cap U_j = \emptyset$ for all $i \neq j$.

Our 'CDA' formulation of such a constraint is as follows:

$$x_{s,u} = 0 \qquad \forall s \in T \text{ and } \forall u \notin \bigcup_{i=1}^{d} U_i \tag{23}$$

$$x_{s,u} \le d_i \qquad \forall u \in U_i \text{ and } \forall i \in \{1,2,\dots,d\} \text{ and } \forall s \in T \tag{24}$$

$$\sum_{i=1}^{d} d_i = 1 \tag{25}$$

$$d_i \in \{0,1\} \qquad \forall i \in \{1,2,\dots,d\} \tag{26}$$

Here variable d_i takes 1 if and only if users from department i are assigned to steps T. This is enforced by (24). Constraint (25) ensures that exactly one department is selected. Constraints (23) are needed to make sure that none of the steps are assigned users from outside the listed departments.

Our 'Naive' formulation of such a constraint is as follows:

$$x_{s',u} \le 1 - \sum_{u' \in U_i^-} x_{s,u'} \qquad \forall s' \in T \setminus \{s\},\ \forall u \in U_i,\ \forall i \tag{27}$$

$$x_{s',u} = 0 \qquad \forall s' \in T \text{ and } \forall u \notin \bigcup_{i=1}^{d} U_i \tag{28}$$

where $s \in T$ is a step in T selected arbitrarily and

$$U_i^- = \left(\bigcup_{j=1}^{d} U_j \right) \setminus U_i\,,$$

i.e. the set of users from all the departments except department i. The point of (27) is that if a member of team i is assigned to step s then no users from any other teams are allowed to perform steps in T. Constraints (28) are to ensure that no users from outside the listed departments are assigned any of the steps in T.

6.4 Type 3 CDA Constraint Formulation

Recall that Type 3 CDA constraint requires that at least one of steps T is performed by a user from $X \subsetneq U$. We are given $T = \{s_1, s_2, \dots, s_t\} \subseteq S$ and a non-empty set of users $X \subsetneq U$.

Our 'CDA' formulation of such a constraint introduces variables $c_i \in \{0,1\}$ for $i = 1, 2, \dots, t$ that will control which of the two authorization families is used.

$$x_{s_i,u} \le 1 - c_i \qquad \forall u \in U \setminus X, \tag{29}$$

$$\sum_{i=1}^{t} c_i = 1 \tag{30}$$

$$c_i \in \{0,1\} \qquad i = 1, 2, \dots, t. \tag{31}$$

Our 'Naive' formulation of this constraint is as follows:

$$\sum_{s \in T} \sum_{u \in X} x_{s,u} \ge 1. \tag{32}$$

6.5 Type 4 CDA Constraint Formulation

Recall that Type 4 CDA constraint requires that only the users $X \subsetneq U$ are actually allowed to be assigned both steps $s_1, s_2 \in S$ at the same time.

Our 'CDA' formulation of such a constraint uses the M variables to distinguish between different patterns:

$$x_{s_1,u} \le 1 - M_{s_1,s_2} \qquad \forall u \in U \setminus X, \tag{33}$$

$$x_{s_2,u} \le 1 - M_{s_1,s_2} \qquad \forall u \in U \setminus X. \tag{34}$$

When steps s_1 and s_2 are assigned the same user, $M_{s_1,s_2} = 1$ and this forces $x_{s_1,u} = x_{s_1,u} = 0$ for every $u \notin X$.

The 'Naive' formulation works with the x-variables prohibiting any user not from X to perform both s_1 and s_2 at the same time:

$$x_{s_1,u} + x_{s_2,u} \le 1 \qquad \forall u \in U \setminus X. \tag{35}$$

6.6 Type 5 CDA Constraint Formulation

Recall that Type 5 CDA constraint requires that, if a user from department 1 is assigned step s_1, then s_2 has to be assigned to someone from department 2. Let $U_1 \subsetneq U$ be the set of users in the department 1, and $U_2 \subsetneq U$ be the set of users in department 2. We assume that $U_1 \cap U_2 = \emptyset$.

Our 'CDA' formulation is as follows:

$$x_{s_1,u} \le i \qquad \forall u \in U_1, \tag{36}$$

$$\sum_{u \in U_2} x_{s_2,u} \ge i, \tag{37}$$

$$i \in \{0,1\}\,. \tag{38}$$

It makes use of an auxiliary variable i. Constraint (37) forces i to 0 unless s_2 is assigned to someone from department 2. If $i = 0$ then no one from department 1 can be assigned to s_1.

Our 'Naive' formulation encodes the constraint in one straightforward inequality:

$$\sum_{u \in U_1} x_{s_1,u} \le \sum_{u \in U_2} x_{s_2,u}\,. \tag{39}$$

7 EXPERIMENTAL RESULTS

We conducted a series of computational experiments to establish the practical difficulty of the WSP-CDA and the effectiveness of the two formulations discussed in the previous section.

We used the SAT4J solver [2] with our Pseudo-Boolean formulations. Our experimental machine is based on two Intel Xeon E5-2690 (2.6 GHz) CPUs, 128 GB RAM. Each solver was restricted to using one CPU core, and we never ran more than one solver per physical core.

The results are reported in Table 2. Each running time is averaged over 100 runs, one run per instance (recall that we generated 100 instances for each combination of instance parameters). The lower running time in a row is underlined. Note that we provide only the lower bound for the Naive formulation solution time for the 'wsp-cda-45-1' instances as we could not complete the experiment.

Instances	Naive formulation, sec	CDA formulation, sec
wsp-cda-30-1	48.8	<u>10.8</u>
wsp-cda-30-2	<u>4.4</u>	4.4
wsp-cda-30-3	<u>8.3</u>	9.3
wsp-cda-30-4	6.8	<u>5.5</u>
wsp-cda-30-5	<u>7.7</u>	7.9
wsp-cda-35-1	514.2	<u>52.6</u>
wsp-cda-35-2	6.8	<u>6.6</u>
wsp-cda-35-3	<u>29.8</u>	30.9
wsp-cda-35-4	16.0	<u>11.6</u>
wsp-cda-35-5	<u>25.3</u>	26.2
wsp-cda-40-1	1711.5	<u>189.9</u>
wsp-cda-40-2	12.7	<u>12.6</u>
wsp-cda-40-3	94.0	<u>91.6</u>
wsp-cda-40-4	47.3	<u>30.9</u>
wsp-cda-40-5	94.1	<u>84.0</u>
wsp-cda-45-1	$>$ 10000.0	<u>2370.7</u>
wsp-cda-45-2	42.4	<u>39.4</u>
wsp-cda-45-3	746.2	<u>739.0</u>
wsp-cda-45-4	206.8	<u>144.3</u>
wsp-cda-45-5	699.7	<u>622.7</u>

Table 2: Results of computational experiments with the benchmark instances.

We conclude that the WSP-CDA benchmark instances can be efficiently solved even for the large instances with $k = 45$ and $n = 450$. We further observe that the 'CDA' formulations are, overall, more efficient. Specifically, the 'CDA' formulation is significantly more efficient for Type 1 CDA constraints and considerably more efficient for Type 4 CDA constraints. The 'CDA' formulation also outperforms the 'Naive' formulation on the large instances with Type 5 CDA constraints. In all other cases, the two formulations perform very similarly.

Using the results reported in Table 2, we also analyse how the solution times vary with the size of the problem instance. Figure 4 shows how the average running time of the 'CDA' formulation changes with k, for each constraint type.

Observe that the curves are slightly non-linear. Given the logarithmic scale of the vertical axis, this indicates that the running time grows a little quicker than exponentially. This is consistent with the other results reported in the literature showing that the average running times of FPT algorithms for WSP scale approximately as $k^{\text{const}\cdot k}$.

To compare the practical difficulty of the CDA constraints, we also show the solution times for instances with UI constraints only.

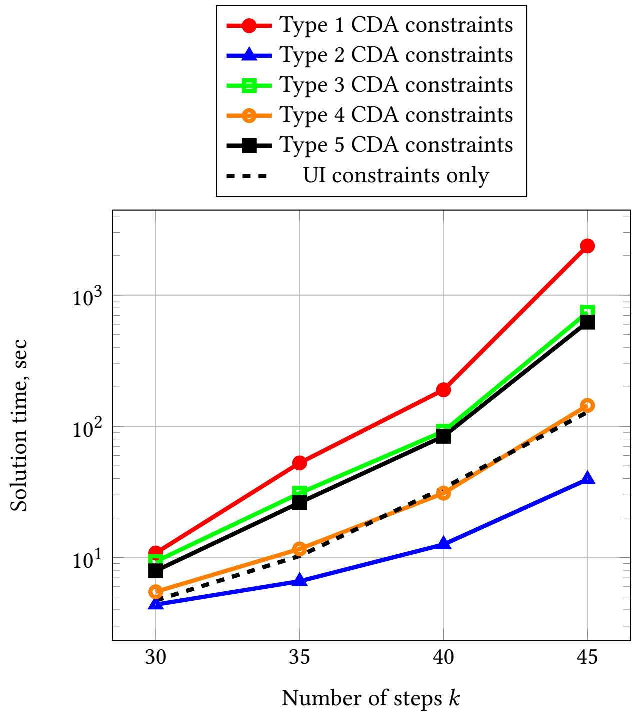


Figure 4: Average solution times as they vary with the size of the problem k. Each curve corresponds to benchmark instances with a certain type of the CDA constraints.

To enable fair comparison, we use instances with the number of not-equals constraints adjusted to achieve around 50% satisfiability probability (i.e., phase transition), exactly as in [24]. It follows from our experiments that CDA constraints do not significantly affect the solution times; some instances are harder than the instances with UI constraints only while others are easier. This is well-aligned with the theoretical findings given in Section 4; the WSP remains tractable with this extended family of constraints.

8 CONCLUSIONS

While the previous research on the WSP was mainly focused on the family of UI constraints or some of its representatives, in this paper, we consider non-UI constraints. We generalise the concept of authorizations by making them context-dependent and show how to absorb non-UI constraints into context-dependent authorizations. This allows us to extend algorithms developed for WSP with UI constraints to arbitrary constraints. We carry out computational experiments with a general purpose Pseudo-Boolean solver, SAT4J, to test practicality of solving WSP with UI and non-UI constraints using our approach.

It would be interesting to characterize non-UI constraints whose branching factor is bounded from above by a constant, in particular, constraints of branching factor 1. It would also be interesting to carry out computational experiments with other types of general purpose solvers, in particular, with CSP solvers, as well as with other types of non-UI constraints.

Acknowledgement. Gregory Gutin's research was supported by the Leverhulme Trust award RPG-2018-161.

REFERENCES

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

[2] Berre, D. L., and Parrain, A. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation 7*, 2-3 (2010), 59–6.

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

[4] Bertolissi, C., dos Santos, D. R., and Ranise, S. Automated synthesis of run-time monitors to enforce authorization policies in business processes. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, 2015* (2015), F. Bao, S. Miller, J. Zhou, and G. Ahn, Eds., ACM, pp. 297–308.

[5] Bertolissi, C., dos Santos, D. R., and Ranise, S. Solving multi-objective workflow satisfiability problems with optimization modulo theories techniques. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, Indianapolis, IN, USA, June 13-15, 2018* (2018), E. Bertino, D. Lin, and J. Lobo, Eds., ACM, pp. 117–128.

[6] Cohen, D., Crampton, J., Gagarin, A., Gutin, G., and Jones, M. Iterative plan construction for the workflow satisfiability problem. *J. Artif. Intell. Res. 51* (2014), 555–577.

[7] Cohen, D. A., Crampton, J., Gagarin, A., Gutin, G., and Jones, M. Algorithms for the workflow satisfiability problem engineered for counting constraints. *J. Comb. Optim. 32*, 1 (2016), 3–24.

[8] Compagna, L., dos Santos, D. R., Ponta, S. E., and Ranise, S. Cerberus: Automated synthesis of enforcement mechanisms for security-sensitive business processes. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings* (2016), M. Chechik and J. Raskin, Eds., vol. 9636 of *Lecture Notes in Computer Science*, Springer, pp. 567–572.

[9] Crampton, J. A reference monitor for workflow systems with constrained task execution. In *10th ACM Symposium on Access Control Models and Technologies, SACMAT 2005, Stockholm, Sweden, June 1-3, 2005, Proceedings* (2005), E. Ferrari and G. Ahn, Eds., ACM, pp. 38–47.

[10] Crampton, J., Gagarin, A., Gutin, G., Jones, M., and Wahlström, M. On the workflow satisfiability problem with class-independent constraints for hierarchical organizations. *ACM Trans. Priv. Secur. 19*, 3 (2016), 8:1–8:29.

[11] Crampton, J., Gutin, G., and Karapetyan, D. Valued workflow satisfiability problem. In *Proceedings of the 20th SACMAT* (2015), E. R. Weippl, F. Kerschbaum, and A. J. Lee, Eds., ACM, pp. 3–13.

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

[13] Crampton, J., Gutin, G. Z., Koutecký, M., and Watrigant, R. Parameterized resiliency problems. *Theor. Comput. Sci. 795* (2019), 478–491.

[14] Crampton, J., Gutin, G. Z., and Majumdar, D. Bounded and approximate strong satisfiability in workflows. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies, SACMAT 2019, Toronto, ON, Canada, June 03-06, 2019* (2019), F. Kerschbaum, A. Mashatan, J. Niu, and A. J. Lee, Eds., ACM, pp. 179–184.

[15] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms.* Springer, 2015.

[16] dos Santos, D. R., Ponta, S. E., and Ranise, S. Modular synthesis of enforcement mechanisms for the workflow satisfiability problem: Scalability and reusability. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies, SACMAT 2016, Shanghai, China, June 5-8, 2016* (2016), X. S. Wang, L. Bauer, and F. Kerschbaum, Eds., ACM, pp. 89–99.

[17] dos Santos, D. R., and Ranise, S. On run-time enforcement of authorization constraints in security-sensitive workflows. In *Software Engineering and Formal Methods - 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, Proceedings* (2017), A. Cimatti and M. Sirjani, Eds., vol. 10469 of *Lecture Notes in Computer Science*, Springer, pp. 203–218.

[18] dos Santos, D. R., and Ranise, S. A survey on workflow satisfiability, resiliency, and related problems. *CoRR abs/1706.07205* (2017).

[19] Gutin, G., and Wahlström, M. Tight lower bounds for the workflow satisfiability problem based on the strong exponential time hypothesis. *Inf. Process. Lett. 116*, 3 (2016), 223–226.

[20] Holderer, J., Accorsi, R., and Müller, G. When four-eyes become too much: a survey on the interplay of authorization constraints and workflow resilience. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015* (2015), R. L. Wainwright, J. M. Corchado, A. Bechini, and J. Hong, Eds., ACM, pp. 1245–1248.

[21] Impagliazzo, R., and Paturi, R. On the complexity of k-SAT. *J. Comput. Syst. Sci. 62*, 2 (2001), 367–375.

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

[23] Karapetyan, D., Gagarin, A. V., and Gutin, G. Z. Pattern backtracking algorithm for the workflow satisfiability problem with user-independent constraints. In *Frontiers in Algorithmics - 9th International Workshop, FAW 2015, Guilin, China, July 3-5, 2015, Proceedings* (2015), J. Wang and C. Yap, Eds., vol. 9130 of *Lecture Notes in Computer Science*, Springer, pp. 138–149.

[24] Karapetyan, D., Parkes, A. J., Gutin, G. Z., and Gagarin, A. Pattern-based approach to the workflow satisfiability problem with user-independent constraints. *J. Artif. Intel. Res. 66* (2019).

[25] Wang, Q., and Li, N. Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur. 13*, 4 (2010), 40.