International Journal of Graph Computing
Vol. 1, No. 1 (2020) 25-38
© KS Press, Institute for Semantic Computing Foundation
DOI: 10.35708/GC1868-126722

Algorithms for Workflow Satisfiability Problem with User-independent Constraints*

Gregory Gutin[1] and Daniel Karapetyan[2]

[1] Roayl Holloway University of London, Egham, Surrey, UK
g.gutin@rhul.ac.uk
[2] University of Essex, Colchester, UK
daniel.karapetyan@gmail.com

Received (10/01/2019)
Revised (01/15/2020)
Accepted (02/28/2020)

Abstract. The Workflow Satisfiability Problem (WSP) is a problem of interest in access control of information security. In its simplest form, the problem coincides with the Constraint Satisfiability Problem, where the number of variables is usually much smaller than the number of values. Wang and Li (ACM Trans. Inf. Syst. Secur. 2010) were the first to study the WSP as a problem parameterized by the number of variables. Their paper initiated very fruitful research surveyed by Cohen, Crampton, Gutin and Wahlström (2017). In this paper, we overview more recent WSP algorithmic developments and discuss computational experiments performed on two new testbeds of WSP instances. These WSP instances are closer to real-world ones than those by Karapetyan et al. (JAIR 2019). One of the two testbeds is generated using a novel iterative approach for obtaining computationally hard WSP instances.

Keywords: Workflow Satisfiability Problem; Unser-independent Constraints; Fixed-parameter Tractability.

1 Introduction

The Workflow Satisfiability Problem (WSP) is a problem of interest in access control of information security. Its instance is defined by a partially ordered list of steps required to be performed by users subject to certain authorizations and constraints. Our aim is to decide whether all steps can be assigned to users

* This paper is an extended version of a conference paper [12]. The main addition is Section 5 where we discuss computational experiments performed with two testbeds which contain instances that are closer to real-world ones than those used in [15].

such that all authorizations and constraints are satisfied, i.e., the instance is *satisfiable* or not. This problem generalizes the well-known Constraint Satisfiability Problem (CSP): CSP variables are non-ordered WSP steps, CSP values are WSP users, CSP unary constraints are WSP authorizations, and CSP non-unary constraints are WSP constraints. Thus, the WSP is NP-hard. Wang and Li [19] observed that usually the number k of steps in WSP is relatively small and much smaller than the number of users. This led Wang and Li to study the WSP parameterized by k. Their first aim was to determine whether the WSP parameterized by k is *fixed-parameter tractable (FPT)*, i.e., can be solved by an algorithm of running time $O(f(k)N^c)$, where f is a computable function in k only, N is the size of the problem and c is a constant. An algorithm of running time $O(f(k)N^c)$, often denoted by $O^*(f(k))$ to hide not only constant factors but also polynomial ones, is called an *FPT algorithm.* (For excellent, recent introductions to parameterized algorithms and complexity, see [8] and [9].)

Wang and Li [19] proved that the WSP is W[1]-hard implying that it is highly unlikely to be FPT (it is widely believed that the parameterized complexity classes FPT and W[1] do not coincide). Then they considered the WSP restricted to simple constraints (with no restrictions on authorizations) and proved fixed-parameter tractability in that case. Cohen, Crampton, Gagarin, Gutin and Jones [2] introduced a wide set of constraints called *user-independent (UI) constraints* (such constraints do not depend on identity of users) and proved that the WSP with only UI constraints (with no restrictions on authorizations) is FPT. All constraints considered by Wang and Li [19] and many other WSP researchers are UI; in fact, all constraints listed by American National Standards Institute in [1] are UI. Cohen et al. [3] introduced and studied an algorithm for the WSP with only UI constraints, which is FPT but of exponential space in k. Karapetyan, Parkes, Gutin and Gagarin [15] introduced another FPT algorithm for the WSP with only UI constraints, which is of polynomial space and much more efficient in practical computing than that in [3]. A key ingredient of the algorithm of Karapetyan et al. is the use of maximum matchings in specially constructed bipartite graphs to decide on authorizations.

Note that while the algorithms Cohen et al. and Karapetyan et al. are of running time[3] $O^*(2^{k\log k})$, in practical computing the latter is much more efficient than the former. In fact, the time $O^*(2^{k\log k})$ is likely to be optimal: Cohen et al. [2] 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 [11] 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.[4]

The rest of the paper is organized as follows. In the next section we give a formal definition of the WSP, UI constraints and other related notions. Section 3

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

[4] The ETH claims that 3-SAT with n variables cannot be solved in time $O(2^{o(n)})$ [13]. The Strong ETH claims that SAT with n variables cannot be solved in time $O(c^n)$ for any $c < 2$ [14].

is devoted to describing a basic version of the algorithm of Karapetyan et al. and an overview of computational experiments with the algorithm and its 'competitors', the algorithm of Cohen et al. and Pseudo-Boolean and CSP solvers. In Section 5, we discuss new computational experiments. They are performed with the WSP instances produced by two novel instance generators. Results on the WSP with UI constraints, some of which (as well as authorizations) can be 'soft' (i.e. allowed to be falsified) are briefly overviewed in Section 6.

2 Workflow Satisfiability Problem

In its basic form, the WSP is defined as follows: We are given a set S of steps, a set U of users, a set $A(s) \subseteq U$ of unary constraints that determines which users can perform step s, and a set C of non-unary constraints over S. Our aim is to decide whether there is a function (called a *plan*) π from S to U such that all authorizations are satisfied (π is *authorized*) and all constraints in C are satisfied (π *is eligible*). If π is both authorized and eligible, then π is called *valid*.

A *constraint* is a pair $c = (L, \Theta)$, where $L \subseteq S$ and Θ is a set of functions from L to U: L is the *scope* of the constraint; Θ specifies those assignments of elements of U to elements of L that *satisfy* the constraint c. In practice, the elements of Θ are not explicitly defined; usually the members of Θ are defined implicitly for different constraint types. Following WSP literature, we assume in this paper that every WSP constraint under consideration can be checked in time polynomial in the numbers of users and steps. Also, following WSP literature and the WSP definition above, only non-unary constraints, i.e., constraints with scope of size at least two are called constraints; the unary constraints are called authorizations.

A constraint (L, Θ) is *user-independent (UI)* if whenever $\theta \in \Theta$ and $\psi : U \to U$ is a permutation then $\psi \circ \theta$, the composition of the two functions, also belongs to Θ. In other words, UI constraints do not distinguish between users. Throughout this paper, we will use the following two types of UI constraints: an *at-most-p-out-of-q* constraint $(T, \leq p)$ (the steps of T can be assigned to at most p users, $|T| = q$), and an *at-least-p-out-of-q* constraint $(T, \geq p)$ (the steps of T can be assigned to at least p users, $|T| = q$) In particular, the well-known constraint AllDiff(T) is $(T, \geq |T|)$.

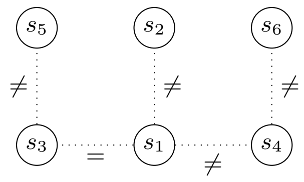
Note that while the order of steps is irrelevant in the WSP in the basic form with only UI constraints, in more advanced models of WSP it is no longer the case (see e.g. [7]).

Figure 1 shows a simple, illustrative example for purchase order processing introduced in [4]. As shown in Part (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.

Part (b) shows constraints to prevent possible fraudulent use of the purchase order processing system. In our example, these constraints restrict the users

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

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

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. Clearly, all constraints of the example are UI.

To complete a WSP specification in this example, we introduce a set of users $U = \{u_i : \ i = 1, 2, \dots, 8\}$ and describe authorization lists: $A(s_1) = \{u_1, u_2\}$, $A(s_2) = \{u_2, u_3\}$, $A(s_3) = \{u_1, u_3\}$, $A(s_4) = \{u_3, u_4\}$, $A(s_5) = \{u_3, u_4, u_5, u_8\}$ and $A(s_6) = \{u_5, u_6, u_7\}$. It is not hard to verify that the following plan $\pi : \ S \to U$ is valid: $\pi(s_1) = \pi(s_3) = u_1$, $\pi(s_2) = u_2$, $\pi(s_4) = u_4$, $\pi(s_5) = u_3$, $\pi(s_6) = u_5$.

3 Pattern Algorithms

The algorithms below were introduced in [15].

3.1 Basic Pattern Algorithm

Both the algorithm of Cohen et al. [3] and Karapetyan et al. [15] are based on the concept of a pattern introduced in [2]. A *pattern* is a partition $\mathcal{P} = \{S_1, \dots, S_p\}$ of S into subsets $S_1, \dots, S_p$ ($S_1 \cup \dots \cup S_p = S$ and $S_i \cap S_j = \emptyset$ for every $i \neq j$) called *blocks* such that every step in each S_i is assigned to the same user, but steps in different S_i's are assigned to different users. For a plan $\pi : \ S \to U$ and $T \subseteq S$, let $\pi(T) = \{\pi(t) : t \in T\}$. A plan π *corresponds to* $\mathcal{P}$ if $\pi(S_i) = \{u_{j_i}\}$ for each $i = 1, \dots, p$ such that $i_j \neq i_q$ for every $1 \leq j \neq q \leq p$. A pattern is *valid* if there is a corresponding plan which is valid.

The basic pattern algorithm consists of generating all different patterns one by one (two patterns are the same if their blocks coincide) and for each generating pattern checking whether all constraints and authorizations are satisfied. Note that the number of different patterns equals to the k'th Bell number $\mathcal{B}_k \leq k!$.

To decide whether all UI constraints are satisfied, we can assign every block a different arbitrary user and check each UI constraint in polynomial time (as we assumed earlier). To decide whether all authorizations are satisfied we construct a bipartite graph $B_{\mathcal{P}}$ whose left part vertices are blocks S_i of the pattern and the right part vertices are the users. There is an edge in $B_{\mathcal{P}}$ between S_i and u_j if all steps of S_i are authorized to be performed by u_j. Observe that all authorizations are satisfied if and only if $B_{\mathcal{P}}$ has a matching saturating its left part. The basic algorithm stops whenever either a valid pattern is found (then the instance is satisfiable) or all patterns have been generated and none are valid (then the instance is unsatisfiable).

Since $\mathcal{B}_k \leq k! = O(2^{k \log k})$, it is not hard to see that the running time of the basic pattern algorithm is $O^*(2^{k \log k})$. Since we generate patterns one by one, the algorithm requires only polynomial space.

3.2 Pattern Backtracking Algorithm

The theoretical running time of the basic pattern algorithm and, as we will see below, the pattern backtracking algorithm are $O^*(2^{k \log k})$ and thus are very likely to be optimal, as we stated in Section 1. However, in practical computing backtracking allows us to usually avoid generating all different patterns for unsatisfiable instances and many patterns for satisfiable instances, and as a result usually solve the instances faster and for larger values of k.

To use backtracking, we generalize the notion of a pattern as follows. A *partial pattern* is a partition of some subset T of S. We can decide whether a partial pattern $\mathcal{Q} = \{T_1, \dots, T_q\}$ can potentially be extended to a valid pattern by deciding whether all constraints with scopes being subsets of T are satisfied and there is a matching saturating the left part of a bipartite graph $B_\mathcal{Q}$ whose left side vertices are blocks T_i of $\mathcal{Q}$ and the right side vertices are the users; the edges of $B_\mathcal{Q}$ are defined in the same way as those of $B_\mathcal{P}$ in the previous subsection.

If a partial pattern $\mathcal{Q}$ can potentially be extended to a valid pattern, the algorithm adds a new step s to the partial pattern by either creating a new block $\{s\}$ or adding s to one of the blocks of $\mathcal{Q}$. For an illustration, see Fig. 2.

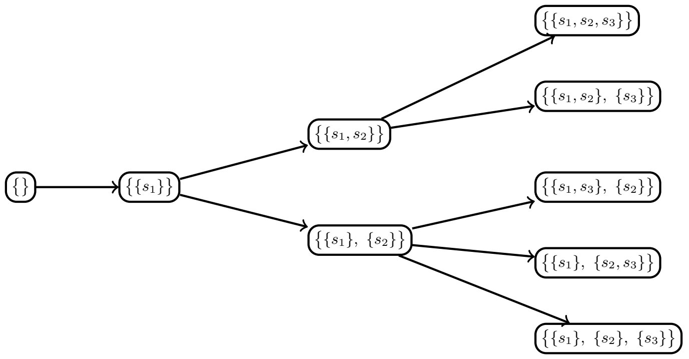


Fig. 2: Illustration of the backtracking mechanism.

It is important to use an efficient procedure which chooses a new step to add to the partial pattern. Such a procedure depends on the types of constraints used in the WSP instances under consideration. For such a procedure and further particulars of the algorithm, see Karapetyan et al. [15].

To bound the running time of the pattern backtracking algorithm, observe that the number of partial patterns on the set $\{s_1, \dots, s_k\}$ is at most twice the number of patterns on $\{s_1, \dots, s_k\}$. Indeed, every partial pattern $\mathcal{Q}$, which is not a pattern, can be completed to a pattern by adding a new block containing all steps in $\{s_1, \dots, s_k\}$ but not in $\mathcal{Q}$. Thus, the running time of the pattern backtracking algorithm is $O^*(2^{k \log k})$.

4 WSP Formulations for SAT Solvers

To computationally evaluate the pattern backtracking algorithm, Karapetyan et al. [15] compared it with Pseudo-Boolean and CSP solvers. Such solvers require appropriate formulations and, in the rest of this section, we very briefly discuss some formulations for SAT solvers.

To solve the WSP using a Pseudo-Boolean solver SAT4J [17], Wang and Li [19] and Cohen et al. [3] considered a pseudo-Boolean formulation of WSP which is based on binary variables $x_{s,u}$ such that $x_{s,u} = 1$ if and only if user u is assigned to step s. We will call such a formulation an *'ordinary' pseudo-Boolean formulation of WSP.* For only not-equals constraints $(s_i \neq s_j)$ (i.e. s_i and s_j must be assigned different users), we have the following WSP Pseudo-Boolean formulation:

$$\begin{aligned} \textstyle\sum_{u \in U} x_{s,u} &= 1 \quad \forall s \in S, \\ x_{s,u} &= 0 \qquad \forall s \in S \text{ and } \forall u \in U \setminus A(s), \\ x_{s_i,u} + x_{s_j,u} &\leq 1 \ \forall \text{ not-equals } (s_i \neq s_j) \text{ and } \forall u \in U. \end{aligned}$$

Karapetyan et al. [15] introduced a more computationally efficient pseudo-Boolean formulation. We may call it an *'FPT' formulation* as alongside variables $x_{s,u}$'s, it uses 'FPT variables' $M_{s,s'}$ such that $M_{s,s'} = 1$ if steps s, s' are assigned the same user and $M_{s,s'} = 0$, otherwise. The M-variables are of interest due to the following:

Theorem [15] *On solving an instance of the WSP, the decision variables M are sufficient to encode any UI constraint.*

4.1 Computational Experiments in [15]

Karapetyan et al. [15] carried out computational experiments to compare the pattern backtracking algorithm with the algorithm of Cohen et al. [3] and other solvers. The experiments were performed on a computer with Intel Xeon CPU E5-2630 v2 (2.6 GHz) and 32 GB RAM. Hyper-threading was enabled, but Karapetyan et al. never ran more than one experiment per physical CPU core concurrently, and concurrency was not exploited in any of the tested solution methods.

The WSP instances used in [15] were generated as follows. In the variable-k study, the number of steps was $18 \leq k \leq 58$ and the number of users was $n = 10k$ (also the $n = 100k$ case as well as the fixed-k variable n case were studied and the results were similar). The authorizations were obtained as follows: for each $u \in U$, first choose $b_u = |A^{-1}(u)|$ randomly and uniformly from $\{1, \dots, \lfloor k/2 \rfloor\}$, then $A^{-1}(u)$ from all subsets of S of size b_u. The constraints used were k at-least-3-out-of-5 constraints, k at-most-3-out-of-5 constraints and e not-equals constraints, where e is chosen such that the probability of (S, U, C, A) being satisfiable is approximately 0.5. (They were showed to be the hardest instances in [15] due to WSP phase transition, see also Section 5.1 for similar results.)

The following solvers were compared:

PBT Pattern backtracking algorithm of Karapetyan et al. [15] (coded in C#);
PUI The algorithm of Cohen et al. [3] (coded in C++);
UDBP-R SAT4J using 'ordinary' pseudo-Boolean formulation of WSP in the resolution proof system mode (cutting plane mode was much worse);
PBPB-R SAT4J using 'FPT' pseudo-Boolean formulation of WSP in the resolution proof system mode;
PBPB-C SAT4J using 'FPT' pseudo-Boolean formulation of WSP in the cutting plane mode;
CP-SAT CP-SAT from Google OR-tools [10] using a CSP formulation of WSP.

For each pair (k, n), 100 experiments were run. The clear winner was PBT with runner-ups CP-SAT and the best of PBPB-R and PBPB-C. In particular, for $k = 50$, PBT took median time 100 sec. for unsatisfiable instances. CP-SAT was 1-2 orders of magnitude slower than PBT and used much more memory. What Karapetyan et al. found particularly remarkable was that CP-SAT used an 'ordinary' CSP formulation of WSP, not a special 'FPT' one, but nevertheless displayed an FPT-like behaviour. Karapetyan et al. hypothesized that the likely explanation was that CP-SAT first reduced the CSP formulation to a SAT formulation, which was 'FPT'-like, and then solved the corresponding SAT problem.

5 New Computational Experiments

An important part of the computational experiments in [15] was a phase transition study. Karapetyan et al. demonstrated that the instance generator described above leads to WSP phase transition (PT), a phenomenon when the properties of the instances sharply change when a parameter crosses a threshold value. In [16], the authors selected the number e of not-equals constraints for that parameter. This parameter has immediate impact on the harder part of the problem, i.e. the part related to the UI constraints. At the same time, the authors demonstrated that the authorisations in the benchmark instances are relatively loose and it is typical that an instance has only a few eligible patterns (i.e. patters satisfying all the constraints) but millions of valid plans (i.e. plans that satisfy all the constraints and authorisations).

The *authorisation density* of a WSP instance (S, U, A) is $|A|/(|S||U|)$, where $|A|$ is the total number of authorisations over all the users. In Section 5.1, instead of varying the parameter e, we vary the authorisation density. The reason for doing so is that it is closer to real-world situations: authorisations can change more frequently than constraints due to changes related to the set of users (the set U of users may change itself and/or the authorisations for some users may change, too, e.g. due to new security clearances or qualifications for users). Also note that all the real-world WSP instances considered in [15] have fixed constraints and varied authorisations. (Karapetyan et al. considered all real-world

WSP instances publicly available from the literature. Unfortunately, they all are computationally easy and thus of no interest in this paper.)

In Section 5.2 we give a completely new approach that leads to another testbed with even harder instances. The results of computational experiments with the two new testbeds are reported in Sections 5.3 and 5.4, respectively.

5.1 Phase Transition by Adjustment of Authorisation Density

We first give an instance generator, which we call Authorisation Density Adjustment Generator (ADAG), that produces PT instances by adjustment of authorisation density.

Figure 3 shows how authorisation density affects the number of satisfiable instances and also the running time of the PBT solver. Each reported running time is the median of the running times in 100 experiments conducted with 100 instances generated with the same parameters but different random number generator seed values. One can observe that adjustments of this parameter lead to behaviour typical for phase transition. The under- and over-subscribed instances are much easier to solve than the instances near the threshold, and the change from unsatatisfiable (unsat) to satisfiable (sat) instances is very rapid.

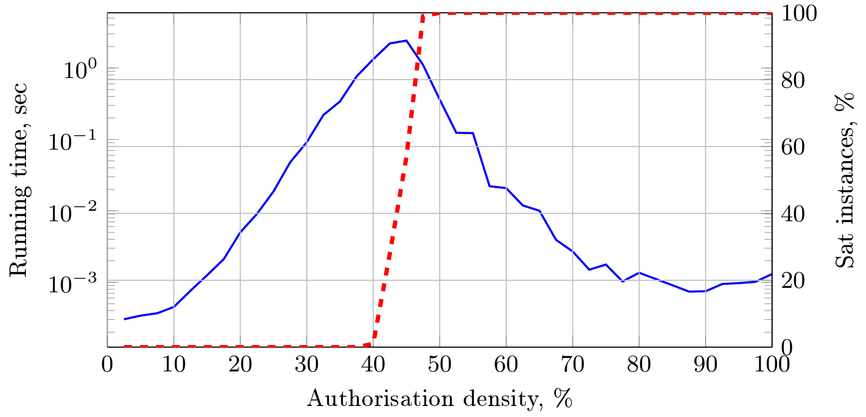


Fig. 3: Phase transition observed in variation of the authorisation density. The red dashed line shows the number of sat instances and the solid blue line shows the running time of PBT. The number of steps is $k = 40$ and the number of users is $n = 400$.

For a given k and n, ADAG fixes the number of not-equals constraints to k, at-most constraints to k and at-least constraints to k. Then it finds the authorisation density that gives an even split between sat and unsat instances. Then the obtained authorisation density is used to produce instances for this combination of k and n.

5.2 Phase Transition with Maximised Authorisation Density

The ADAG instance generator produces hard instances as evident from Figure 3. In this section we give a new instance generator, which we call Maximised Authorisation Density Generator (MADG), capable of generating even harder instances. Specifically, it implements a heuristic that maximises the number of authorisations while the instances remain in the PT region. By maximising the number of authorisations, we reduce the amount of pruning and propagation possible within the solvers and hence make the instances harder. In order to remain in the PT region, the generator guarantees that if the generated instance is sat, then removing a single authorisation can make it unsat, and if the generated instance is unsat, then adding a single authorisation can make it sat.

To summarise, MADG is both more accurate in placing the instances in the PT region and also maximises the number of authorisations which, we expect, will make the instances more challenging.

MADG works as follows.

1. Produce a PT instance using ADAG instance generator described in Section 5.1.
2. If the instance is sat, randomly select a user authorised to at least two steps, randomly select one of the steps to which this user is authorised and remove this authorisation. Repeat this step as long as the instance remains sat.
3. (Note that at this point the instance is guaranteed to be unsat.) Randomly select a user authorised to less than all steps, randomly select a step to which this user is not authorised and add the corresponding authorisation. Repeat this step as long as the instance remains unsat.
4. (Note that at this point the instance is guaranteed to be sat.) Randomly select a user authorised to at least two steps, randomly select one of the steps to which this user is authorised and remove this authorisation. If the modified instance is still sat, backtrack (i.e. restore the removed authorisation) and repeat this step. If the modified instance is unsat, go to Step 3 or terminate if the prescribed number of iterations of the loop formed by Steps 3 and 4 have been performed.

Observe that Step 4 is guaranteed to find a single authorisation whose removal will turn the instance unsat; indeed, removing the last authorisation added in Step 3 will do exactly that. There is no guarantee that Steps 2 and 3 will terminate, however, considering that the initial instance is in the PT region, it is highly unlikely that they will never terminate.

5.3 Computational Experiments with ADAG Instances

The experiments were conducted on a Dell XPS 15 laptop with Intel Core i7-8750H 2.2 GHz CPU and 32 GB RAM.

We expect that, compared to the standard instances, ADAG instances will have more challenging embedded matching problems. This will give advantage to the PBT algorithm and PBPB (CP) – the solvers that are guaranteed to efficiently solve the matching problems. On the other hand, tighter authorisations will lead to optimal branching that involves user assignments or at least takes user authorisations into account; only the off-the-shelf solvers are capable of doing that whereas PBT has a rigid branching heuristic.

Our computational results are reported in Figure 4. Each reported running time is the median of the running times in 100 experiments with 100 instances generated with the same parameters except for the random number generator seed value.

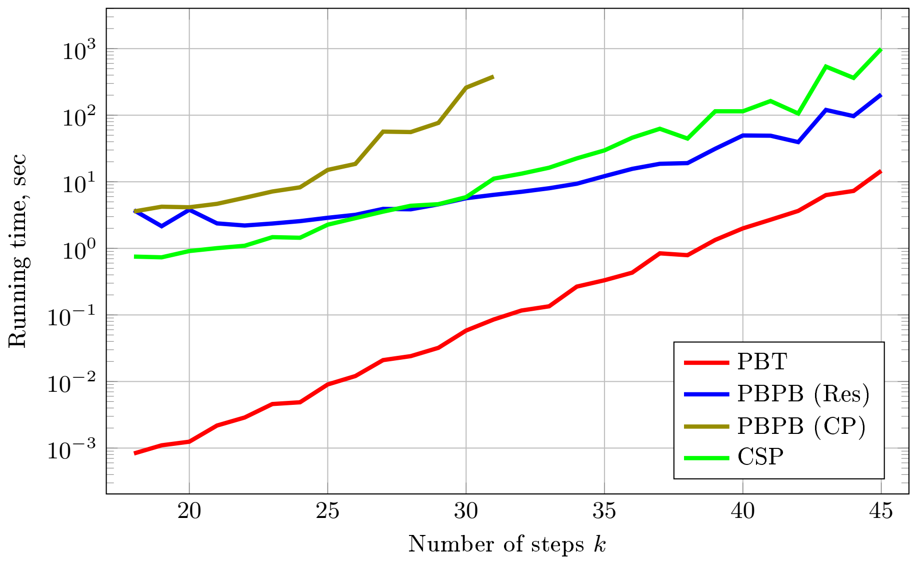


Fig. 4: Evaluation of the solvers on the ADAG instances.

The results are very close to those reported in [15]. (Note that we used a different machine for the experiments in this paper, but we expect that the single-threaded performance of the two machines is similar.) It could be that the two effects discussed in the beginning of this section cancel each other out. It could also be that the authorisations are not tight enough to make the user assignments challenging. This hypothesis led us to designing and experimenting with the second instance generator, MADG.

5.4 Computational Experiments with MADG Instances

Since generation of the MADG instances is expensive, we restricted the number of iterations of MADG in order to produce a figure similar to Figure 4. Specifically, we used the instances obtained after 10 alternations between sat and unsat instances within MADG. Compared to the ADAG instances, this approach considerably increased the authorisation density but still the running times of the solvers were not much unaffected.

For our next experiment, we allowed more iterations within MADG. We then plotted how the authorisation density affects the running time of the different solvers. The results are reported in Figure 5. We used only the unsat instances to plot this graph, as the running times for the sat instances have higher variance. It is clear that all the solvers are to some degree sensitive to the authorisation density. The dependency is approximately linear; linear functions quite closely approximate each of the datasets. Note that the vertical axis in this plot has logarithmic scale and as a result linear functions are curved.

We then computed how sensitive each solver is to the authorisation density. The results are summarised in Table 1.

	Time at 24%, sec	Time at 36%, sec	Ratio
PBT	0.01	0.05	3.6
PBPB (Res)	3.11	5.87	1.9
PBPB (CP)	39.14	308.59	7.9
CSP	5.21	23.13	4.4

Table 1: Summary of how sensitive each of the solvers is to the change of the authorisation density.

We used the fitted linear functions to estimate the running time of each solver at authorisation densities 24% and 36% (columns 2 and 3). The last column of the table gives the ratio between those values for each solver. It turns out that PBPB (Res) is by far the least sensitive to the authorisation density. In this sense, it significantly outperforms its competitor CSP. Also, PBPB (CP) is the most sensitive one. Explaining these results however will require further research. We also note that PBT is moderately sensitive to the authorisation density. We assume that this is because it employs heuristics for restricting the set of users considered for the matching problems, however these heuristics will not be as efficient if the number of authorisations is increased. Nevertheless, PBT still outperforms all the other methods by several orders of magnitude and hence is not in direct competition with PBPB.

We also conducted experiments with MADG instances obtained by changing n at a fixed k. The preliminary results were similar to those reported in [15] for the fixed-k slice, however due to the computational cost of producing MADG instances we could not complete the experiment and thus we do not report the details here.

6 Other Research on WSP with UI Constraints

Sometimes a WSP instance cannot be satisfied, but it would be acceptable to falsify some 'soft' constraints especially if they are not falsified 'too much', e.g., for an at-most-3-out-of-5 constraint only four users rather than three are assigned. Crampton, Gutin and Karapetyan [5] formalized this as follows. For each constraint c and plan π, if π satisfies c then $w_c(\pi) = 0$, otherwise $w_c(\pi) > 0$ with

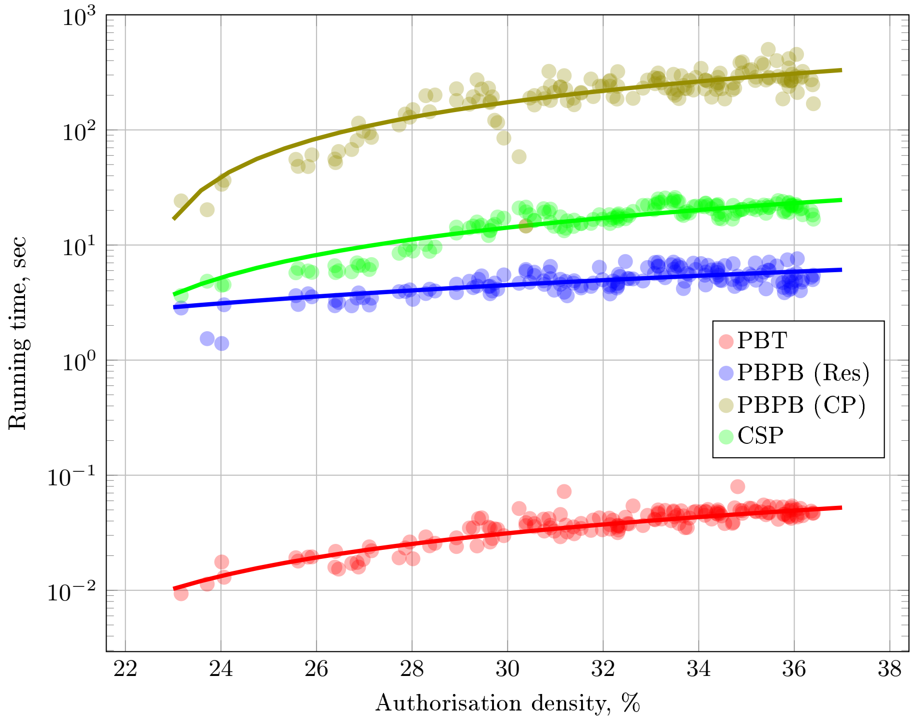


Fig. 5: Evaluation of how the authorisation density affects the running times. The instances with higher authorisation density were obtained by running more iterations of MADG. The number of steps is $k = 25$ and the number of users is $n = 250$. Only the unsat instances are selected for this experiment. The solid lines are linear functions fitted into each dataset.

the actual value of $w_c(\pi)$ depending on how much c is not satisfied by π. Let $w_C(\pi) = \sum_{c \in C} w_c(\pi)$, the weight of all constraints in C for plan π.

Consider, for example, an at-most-r constraint $(T, \leq r)$. Then it is natural to assume that $w_c(\pi)$ depends only on the number of users assigned to the steps in T (and the penalty should increase as the number of users increases). Let $\pi(T)$ denote the set of users assigned to steps in T. Then $w_c(\pi) = 0$ if $|\pi(T)| \leq r$; for plans π and π', we have $w_c(\pi) = w_c(\pi')$ if $|\pi(T)| = |\pi(T')|$; and $0 < w_c(\pi) \leq w_c(\pi')$ if $r < |\pi(T)| \leq |\pi'(T)|$.

A weighted constraint c is called *UI* if, for every permutation θ of U, $w_C(\pi) = w_C(\theta \circ \pi)$. Thus, a weighted UI constraint does not distinguish between users. Clearly, an at-most-r constraint is UI.

We can also introduce $w_A(\pi)$, which assigns a weight for each plan π with respect to the authorisation policy. The intuition is that a plan in which every user is authorised for the steps to which she is assigned has zero weight and the weight of a plan that violates the policy increases as the number of steps that are assigned to unauthorised users increases. Now, we can introduce the total weight of a plan π: $w(\pi) = w_C(\pi) + w_A(\pi)$ with the aim to minimize it. This problem is called the Valued WSP, analogously to the Valued CSP introduced by Schiex, Fargier and Verfaillie [18].

Crampton et al. [5] designed an $O^*(2^{k \log k})$-time algorithm, denoted PBnB, to find the minimum of $w(\pi)$ when all constrains are UI. Computational experiments in [5] showed that PBnB is superior to MIP solver CPLEX 12.6: while PBnB solved every instance for $k \leq 30$ and a large majority for $k = 35$ within one hour, CPLEX 12.6 solved all instances only for $k = 20$, for $k = 35$ less than 43%.

Crampton, Gutin, Karapetyan and Watrigant [6] considered a more general problem: the Bi-objective WSP (BOWSP), where the two objectives are $w_C(\pi)$ and $w_A(\pi)$, respectively. They proved that computing a Pareto front for BOWSP is FPT and can be done in time $O^*(2^{k \log k})$, if we restrict our attention to UI constraints. Computational experiments with the FPT algorithm and one based on MIP solver CPLEX12.6 demonstrated clear superiority of the FPT algorithm.

Crampton et al. [6] also studied the important question of workflow resiliency and proved new results establishing that known resiliency decision problems are FPT when restricted to UI constraints. They also proposed a new way of modeling the availability of users and demonstrated that many questions related to resiliency in the context of the new model may be reduced to instances of BOWSP. The new model has probabilistic elements and allows one to estimate the probability of a valid plan to be completed when, with some probabilities, users may be unavailable (e.g. due to their absence from work) to perform some steps.

Acknowledgment

Gutin would like to thank the Leverhulme Trust for supporting his research on the WSP and related problems.

References

1. American National Standards Institute: ANSI INCITS 359-2004 for Role Based Access Control (2004)
2. Cohen, D., Crampton, J., Gagarin, A., Gutin, G., Jones, M.: Iterative plan construction for the workflow satisfiability problem problem. J. Artif. Intell. Res. (JAIR) **51**, 555–577 (2014)
3. Cohen, D.A., Crampton, J., Gagarin, A., Gutin, G.Z., Jones, M.: Algorithms for the workflow satisfiability problem engineered for counting constraints. J. Comb. Optim. **32**(1), 3–24 (2016)
4. Crampton, J.: A reference monitor for workflow systems with constrained task execution. In: Ferrari, E., Ahn, G.J. (eds.) SACMAT. pp. 38–47. ACM (2005)
5. Crampton, J., Gutin, G., Karapetyan, D.: Valued workflow satisfiability problem. In: Proceedings of the 20th ACM Symposium on Access Control Models and Technologies (SACMAT 2015). pp. 3–13 (2015)
6. Crampton, J., Gutin, G., Karapetyan, D., Watrigant, R.: The bi-objective workflow satisfiability problem and workflow resiliency. Journal of Computer Security **25**(1), 83–115 (2017)

7. Crampton, J., Gutin, G.Z., Watrigant, R.: On the satisfiability of workflows with release points. In: Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2017, Indianapolis, IN, USA, June 21-23, 2017. pp. 207–217 (2017)
8. Cygan, M., Fomin, F., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)
9. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer Verlag (2013)
10. Google OR-tools (2019), `https://developers.google.com/optimization/`
11. Gutin, G., Wahlström, M.: Tight lower bounds for the workflow satisfiability problem based on the strong exponential time hypothesis. Inf. Process. Lett. **116**(3), 223–226 (2016)
12. Gutin, G.Z.: The workflow satisfiability problem with user-independent constraints. In: Proceedings of Graph Computing for Industries 2019 (GC4I 2019). pp. 1–4 (2019)
13. Impagliazzo, R., Paturi, R.: On the Complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367 – 375 (2001)
14. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
15. Karapetyan, D., Parkes, A.J., Gutin, G.Z., Gagarin, A.: Pattern-based approach to the workflow satisfiability problem with user-independent constraints. J. AI Research **66**, 85–122 (2019)
16. Karapetyan, D., Parkes, A.J., Gutin, G.Z., Gagarin, A.: Pattern-based approach to the workflow satisfiability problem with user-independent constraints (full version). arXiv 1604.05636 (2019)
17. Le Berre, D., Parrain, A.: The SAT4J library, release 2.2. J. Satisf. Bool. Model. Comput. **7**, 59–64 (2010)
18. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence – Volume 1. pp. 631–637. IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995)
19. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM Trans. Inf. Syst. Secur. **13**(4), 40 (2010)