Valued Workflow Satisfiability Problem

Jason Crampton
Information Security Group
Royal Holloway University of
London
Egham, Surrey, UK
jason.crampton@rhul.ac.uk

Gregory Gutin
Department of Computer
Science
Royal Holloway University of
London
Egham, Surrey, UK
gutin@cs.rhul.ac.uk

Daniel Karapetyan
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
daniel.karapetyan@gmail.com

ABSTRACT

A workflow is a collection of steps that must be executed in some specific order to achieve an objective. A computerised workflow management system may enforce authorisation policies and constraints, thereby restricting which users can perform particular steps in a workflow. The existence of policies and constraints may mean that a workflow is unsatisfiable, in the sense that it is impossible to find an authorised user for each step in the workflow and satisfy all constraints. In this paper, we consider the problem of finding the "least bad" assignment of users to workflow steps by assigning a weight to each policy and constraint violation. To this end, we introduce a framework for associating costs with the violation of workflow policies and constraints and define the *valued workflow satisfiability problem* (VALUED WSP), whose solution is an assignment of steps to users of minimum cost. We establish the computational complexity of VALUED WSP with user-independent constraints and show that it is fixed-parameter tractable. We then describe an algorithm for solving VALUED WSP with user-independent constraints and evaluate its performance, comparing it to that of an off-the-shelf mixed integer programming package.

Categories and Subject Descriptors

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

General Terms

Algorithms, Security, Theory

Keywords

workflow satisfiability, parameterized complexity, valued workflow satisfiability problem

1. INTRODUCTION

It is increasingly common for organisations to computerise their business and management processes. The coordination of the tasks or steps that comprise a computerised business process is managed by a workflow management system (or business process management system). A workflow is defined by the steps in a business process and the order in which those steps should be performed. A workflow is executed multiple times, each execution being called a *workflow instance.* Typically, the execution of each step in a workflow instance will be triggered by a human user, or a software agent acting under the control of a human user. As in all multi-user systems, some form of access control, typically specified in the form of policies and constraints, should be enforced on the execution of workflow steps, thereby restricting the execution of each step to some authorised subset of the user population.

Policies typically specify the workflow steps for which users are authorised, what Basin *et al.* call *history-independent* authorisations [2]. Constraints restrict which groups of users can perform sets of steps. It may be that a user, while authorised by the policy to perform a particular step *s*, is prevented (by one or more constraints) from executing *s* in a specific workflow instance because particular users have performed other steps in the workflow (hence the alternative name of *history-dependent* authorizations [2]). The concept of a Chinese wall, for example, limits the set of steps that any one user can perform [3], as does separation-of-duty, which is a central part of the role-based access control model [1]. We note that policies are, in some sense, discretionary, as they are defined by the workflow administrator in the context of a given set of users. However, constraints may be mandatory (and independent of the user population), in that they may encode statutory requirements governing privacy or separation-of-concerns or high-level organisational requirements.

It is well known that a workflow specification may be "unsatisfiable" in the sense that the combination of policies and constraints means that there is no way of allocating authorised users to workflow steps without violating at least one constraint. The workflow satisfiability problem is NP-hard [24] although relatively efficient algorithms have been developed on the assumption that the number of workflow steps is much smaller than the number of users that may perform steps in the workflow [7, 12, 18, 24]. Of course, the objectives of the business process associated with a workflow specification can never be achieved if the specification is unsatisfiable. Hence, it is interesting to consider an ex-

tended version of the workflow satisfiability problem that seeks the "best" allocation of users to steps in the event that the specification is unsatisfiable.

Accordingly, in this paper we study the *valued* workflow satisfiability problem (VALUED WSP). Informally, we associate constraint and authorisation violations with a cost, which may be regarded as an estimate of the risk associated with allowing those violations. We then compute an assignment of users to steps having minimal cost, this cost being zero when the workflow is satisfiable. In a sense, our work is related to recent work on risk-aware access control [4, 5, 13, 20], which seeks to compute the risk of allowing a user to perform an action, rather than simply computing an allow/deny decision, and ensure that cumulative risk remains within certain limits. However, unlike related work, we focus on computing user-step assignments of minimal cost, rather than access control decisions.

Our main contributions are: to define VALUED WSP and determine its complexity; to prove that VALUED WSP is fixed-parameter tractable for weighted user-independent constraints; to develop an algorithm to solve VALUED WSP with user-independent constraints; to provide a comprehensive experimental evaluation of our algorithm; and to demonstrate that the performance of our algorithm compares very favourably with an approach that uses the mixed integer programming solver CPLEX. Our experimental evaluation shows our algorithm enjoys a substantial advantage over CPLEX as the number of steps grows, with our algorithm being able to deal far better with problem instances containing more than 30 steps. Moreover, our algorithm is far better at solving instances that are unsatisfiable – precisely those instances for which VALUED WSP is relevant.

In the next section, we define VALUED WSP, having reviewed relevant concepts from the literature, including the workflow satisfiability problem and user-independent constraints. In Section 3, we prove that VALUED WSP is fixed-parameter tractable for user-independent constraints and describe an algorithm based on the concept of a *pattern*, which is, informally, a compact representation of a set of similar plans for a user-independent constraint. In Section 4, we present our experimental results. This section also includes a method of representing VALUED WSP as a mixed integer programming problem, which may be of use in subsequent research. We conclude the paper with a discussion of related work, a summary of contributions and some suggestions for future work.

2. BACKGROUND AND PROBLEM STATEMENT

We first briefly summarise relevant concepts from the literature, including *workflow authorisation schema*, *workflow constraints* and the *workflow satisfiability problem* (WSP). We then explain how a constrained workflow authorisation schema can be extended to assign costs to plans that do not satisfy the schema's policy and constraints. We conclude this section with a formal definition of VALUED WSP.

2.1 WSP

A directed acyclic graph $G = (V, E)$ is defined by a set of nodes V and a set of edges $E \subseteq V \times V$. The reflexive, transitive closure of a directed acyclic graph (DAG) defines a partial order, where $v \leqslant w$ if and only if there is a path

from v to w in G. We may write $v \geqslant w$ whenever $w \leqslant v$. We may also write $v < w$ whenever $v \leqslant w$ and $v \neq w$.

DEFINITION 1. *A workflow specification is defined by a directed, acyclic graph $G = (S, E)$, where S is a set of steps and $E \subseteq S \times S$. Given a workflow specification (S, E) and a set of users U, an* authorisation policy *for a workflow specification is a relation $A \subseteq S \times U$. A* workflow authorisation schema *is a tuple (G, U, A), where $G = (S, E)$ is a workflow specification and A is an authorisation policy.*

A workflow specification describes a sequence of steps and the order in which they must be performed when the workflow is executed, each such execution being called a *workflow instance*.[1] User u is authorised to perform step s only if $(s, u) \in A$.[2] We assume that for every step $s \in S$ there exists some user $u \in U$ such that $(s, u) \in A$ (otherwise the workflow is trivially unsatisfiable).

DEFINITION 2. *Let $((S, E), U, A)$ be a workflow authorisation schema. A* plan *is a function $\pi : T \to U$, where $T \subseteq S$. A plan π is* complete *if $T = S$.*

DEFINITION 3. *A* workflow constraint *has the form (T, Θ), where $T \subseteq S$ and Θ is a family of functions with domain T and range U. T is the* scope *of the constraint (T, Θ). A* constrained workflow authorization schema *is a pair $((S, E), U, A, C)$, where $((S, E), U, A)$ is a workflow authorization schema and C is a set of workflow constraints.*

Informally, a workflow constraint (T, Θ) limits the users that are allowed to perform a set of steps T in any given instance of the workflow. In particular, Θ identifies authorised (partial) assigments of users to workflow steps in T. More formally, let $\pi : S' \to U$, where $S' \subseteq S$, be a plan. Given $T \subseteq S'$, we write $\pi|_T$ to denote the function π restricted to domain T; that is $\pi|_T(s) = \pi(s)$ for all $s \in T$ (and is undefined otherwise). Then we say $\pi : S' \to U$ *satisfies* a workflow constraint (T, Θ) if $T \not\subseteq S'$ or $\pi|_T \in \Theta$.

DEFINITION 4. *Given a constrained workflow authorization schema $((S, E), U, A, C)$, we say a plan $\pi : S \to U$ is* valid *if it satisfies every constraint in C and, for all $t \in S$, $(t, \pi(t)) \in A$.*

In practice, we do not define constraints by enumerating all possible elements of Θ. Instead, we define different families of constraints that have "compact" descriptions. Thus, for example, we might define the family of simple separation-of-duty constraints, each of which is represented by a set

---

[1]In this paper, the ordering on the steps is not considered. Prior work has shown that the ordering is irrelevant to the question of satisfiability subject to certain assumptions about the constraints [12], assumptions that are satisfied by the constraints considered in this paper.

[2]In practice, the set of authorised step-user pairs, A, will not be defined explicitly. Instead, A will be inferred from other access control data structures. In particular, R²BAC – the role-and-relation-based access control model of Wang and Li [24] – introduces a set of roles R, a user-role relation $UR \subseteq U \times R$ and a role-step relation $SA \subseteq R \times S$ from which it is possible to derive the steps for which users are authorised. For all common access control policies (including R²BAC), it is straightforward to derive A. We prefer to use A in order to simplify the exposition.

$\{t_1, t_2\}$, the constraint being satisfied provided t_1 and t_2 are assigned to different users.

A constraint (T, Θ) is said to be *user-independent* if, for every $\theta \in \Theta$ and every permutation $\phi : U \to U$, $\phi \circ \theta \in \Theta$, where

$$\phi \circ \theta : T \to U \quad \text{and} \quad (\phi \circ \theta)(s) \stackrel{\text{def}}{=} \phi(\theta(s)).$$

Simple separation-of-duty constraints are user-independent, and it appears most constraints that are useful in practice are user-independent [7]. In particular, cardinality constraints and binding-of-duty constraints are user-independent. In this paper, we restrict our experimental evaluation to two particular types of user-independent constraints (in addition to separation-of-duty constraints):

- an *at-least-r counting* constraint has the form (T, r), where $r \leqslant |T|$, and is satisfied provided at least r users are assigned to the steps in T;

- an *at-most-r counting* constraint has the form (T, r), where $r \leqslant |T|$, and is satisfied provided at most r users are assigned to the steps in T.

It is important to stress that our approach works for any user-independent constraints. We chose to use counting constraints because such constraints have been widely considered in the literature (often known as cardinality constraints). Moreover, counting constraints can be encoded using mixed integer programming, so we can use off-the-shelf solvers to solve WSP and to compare with the performance of our bespoke algorithms.

We now introduce the workflow satisfiability problem, as defined by Wang and Li [24].

WORKFLOW SATISFIABILITY PROBLEM (WSP)
  Input: A constrained workflow authorisation schema
      $((S, E), U, A, C)$
  Output: A valid $\pi : S \to U$ or an answer that there
      exists no valid plan

2.2 Fixed-Parameter Tractability of WSP

A naïve approach to solving WSP would consider every possible assignment of users to steps in the workflow. There are n^k such assignments if there are n users and k steps, so an algorithm of this form would have complexity $O(mn^k)$, where m is the number of constraints. Moreover, Wang and Li showed that WSP is NP-hard, by reducing GRAPH k-COLORABILITY to WSP with separation-of-duty constraints [24, Lemma 3].

The importance of finding an efficient algorithm for solving WSP led Wang and Li to look at the problem from the perspective of *parameterised complexity*. Suppose we have an algorithm that solves an NP-hard problem in time $O(f(k)n^d)$, where n denotes the size of the input to the problem, k is some (small) parameter of the problem, f is some function in k only, and d is some constant (independent of k and n). Then we say the algorithm is a *fixed-parameter tractable* (FPT) algorithm. If a problem can be solved using an FPT algorithm then we say that it is an *FPT problem* and that it belongs to the class FPT [14, 21].

Wang and Li observed that fixed-parameter algorithmics is an appropriate way to study the problem, because the number k of steps is usually small and often much smaller

than the number n of users.[3] Wang and Li [24] proved that, in general, WSP is W[1]-hard and thus is highly unlikely to admit an FPT algorithm. However, WSP is FPT if we consider only separation-of-duty and binding-of-duty constraints [24]. Henceforth, we consider special families of constraints, but allow arbitrary authorisations. Crampton *et al.* [12] obtained significantly faster FPT algorithms that were applicable to "regular" constraints, thereby including the cases shown to be FPT by Wang and Li. Subsequent research has demonstrated the existence of FPT algorithms for WSP in the presence of other constraint types [9, 10]. Cohen *et al.* [7] introduced the class of user-independent constraints and showed that WSP remains FPT if only user-independent constraints are included. Note that every regular constraint is user-independent and all the constraints defined in the ANSI RBAC standard [1] are user-independent. Results of Cohen *et al.* [7] have led to algorithms which significantly outperform the widely used SAT-solver SAT4J on difficult instances of WSP with user-independent constraints [6, 18].

2.3 Valued WSP

There has been considerable interest in recent years in making the specification and enforcement of access control policies more flexible. Naturally, it is essential to continue to enforce effective access control policies. Equally, it is recognised that there may well be situations where a simple "allow" or "deny" decision for an access request may not be appropriate. It may be, for example, that the risks of refusing an unauthorised request are less significant than the benefits of allowing it. One obvious example occurs in healthcare systems, where the denial of an access request in an emergency situation could lead to loss of life. Hence, there has been increasing interest in context-aware policies, such as "break-the-glass", which allow different responses to requests in different situations. Risk-aware access control is another promising line of research that seeks to quantify the risk of allowing a request, where a decision of "0" might represent an unequivocal "deny" and "1" an unequivocal "allow", with decisions of intermediate values representing different levels of risk.

Similar considerations arise very naturally when we consider workflows. In particular, we may specify authorization policies and constraints that mean a workflow specification is unsatisfiable. Clearly, this is undesirable from a business perspective, since the business objective associated with the workflow can not be achieved. There are two possible ways of dealing with an unsatisfiable workflow specification:(i) modify the authorization policy and/or constraints; (ii) find the "least bad" complete plan. Prior work by Basin, Burri and Karjoth considered the former approach [2]. They restricted their attention to modification of the authorisation policy, what they called *administrable authorizations*. They assigned costs to modifying different aspects of a policy and then computed a strategy to modify the policy of minimal cost.

We adopt a different approach and consider minimising the cost of "breaking" the policies and/or constraints. (We will compare our approach to Basin *et al.* in the related work section.) Informally, given a workflow specification, for each

---

[3]The SMV loan origination workflow studied by Schaad *et al.*, for example, has 13 steps and identifies five roles [22]. It is generally assumed that the number of users is significantly greater than the number of roles.

plan π, we define the total cost or weight associated with the plan $w(\pi)$. The problem, then, is to find the complete plan with minimum total cost.

More formally, let $((S, E), U, A, C)$ be a constrained workflow authorization schema. Let Π denote the set of all possible plans from S to U. Then, for each $c \in C$, we define a weight function $w_c : \Pi \to \mathbb{Z}$, where

$$w_c(\pi) \begin{cases} = 0 & \text{if } \pi \text{ satisfies } c, \\ > 0 & \text{otherwise.} \end{cases}$$

The pair (c, w_c) is a *weighted constraint*.

The intuition is that $w_c(\pi)$ represents the extent to which π violates c. Consider, for example, an at-most-r counting constraint (T, r). Then $w_c(\pi)$ 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 $|\pi(T)| \leqslant r$; 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 $r < |\pi(T)| \leq |\pi'(T)|$. Similarly, for an at-least-r constraint c with scope T, we would have $w_c(\pi) = 0$ if $|\pi(T)| \geqslant r$; 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 $|\pi(T)| \geq |\pi'(T)| > 0$.

Then we define

$$w_C(\pi) = \sum_{c \in C} w_c(\pi),$$

which we call the *constraint weight* of π. Note that $w_C(\pi) = 0$ if and only if π satisfies all constraints in C. Note also that $w_C(\pi)$ need not be defined to be the linear sum: $w_C(\pi)$ may be defined to be an arbitrary function of the tuple $(w_c(\pi) : c \in C)$ and Theorem 1 below would still hold. However, we will not use this generalisation in this paper, but simply remark that it is possible, if needed.

We next introduce a function $w_A : \Pi \to \mathbb{Z}$, which assigns a cost for each plan 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 cost and the cost of a plan that violates the policy increases as the number of steps that are assigned to unauthorised users increases. More formally,

$$w_A(\pi) \begin{cases} = 0 & \text{if } (\pi(t), t) \in A \text{ for all } t, \\ > 0 & \text{otherwise} \end{cases}$$

is the *authorisation weight* of π.

The definition of w_A can be arbitrarily fine-grained. We could, for example, associate a weight $\omega(t, u)$ with every pair (t, u), where a zero weight indicates that u is authorised for t, and define

$$w_A(\pi) = \sum_{t \in T} \omega(t, \pi(t)).$$

One particularly simple instantiation of this idea is to define a single weight $\omega > 0$ to be associated with every policy violation. In this case, $w_A(\pi) = a\omega$, where a is the number of steps that are assigned to unauthorised users. Alternatively, we might distinguish between different types of users, so that, for example, assigning steps to external contractors is associated with a higher weight ω_e than the weight ω_i associated with assigning steps to (internal) unauthorised staff members.

We may now define the valued workflow satisfiability problem, which will be the subject of the remainder of the paper.

VALUED WSP
  Input: A constrained workflow authorisation schema
    $((S, E), U, A, C)$ with weights for constraints
    and authorisations, as above.
  Output: A plan $\pi : S \to U$ that minimises
    $w(\pi) = w_C(\pi) + w_A(\pi)$.

Before proceeding further, however, we introduce a weight function that is more fine-grained than those considered above, and the one that we shall use in the remainder of this paper. Specifically, for each user u and each subset T of S, we define a weight $\omega(T, u) \in \mathbb{Z}$, where

$$\omega(T, u) \begin{cases} = 0 & \text{if } (u, t) \in A \text{ for all } t \in T, \\ > 0 & \text{otherwise.} \end{cases}$$

We call $\omega : 2^S \times U$ the *(weighted) set-authorisation* function. Vacuously, we have $\omega(\emptyset, u) = 0$ for all $u \in U$. We write $\pi^{-1}(u)$ to denote the set of steps to which u is assigned by plan π. Then we define

$$w_A(\pi) = \sum_{u \in U} \omega(\pi^{-1}(u), u).$$

Clearly, this form of authorisation weight satisfies the required criteria.

We base w_A on weights of the form $\omega(T, u)$ because, in addition to allowing us to specify weights for every pair (t, u) if required, it allows us to express more complex ("non-linear") costs on plans. For example:

1. We can introduce a large penalty $\omega(T, u)$, effectively saying we prefer not to involve u in steps in T. (We use weights like this in our experimental work, described in Section 4.)

2. We can define a limit ℓ on the number of steps that can be executed by u, by setting a large penalty $\omega(T, u)$ for all T of cardinality greater than ℓ.

3. We can attempt to minimise the number of involved users by giving a small penalty for assigning a user to at least one step. This is similar to item 1 above, albeit with a different goal.

4. The weights associated with the same user executing different steps may not increase linearly. Once a user has performed one particular unauthorized step, the additional cost of executing a *related* unauthorized step may be reduced, while the additional cost of executing an *unrelated* unauthorized step may be the same as the original cost. Our formulation enables us to model this kind of situation.

5. We can implement separation-of-duty on a per-user basis, which is not possible with user-independent constraints. In particular, it may be acceptable for u_1 to perform steps s_1 and s_2, but not u_2, in which case $\omega(\{s_1, s_2\}, u_1)$ would be small, while $\omega(\{s_1, s_2\}, u_2)$ would be large.

The next claim is an important observation following directly from the definitions above.

PROPOSITION 1. *The optimal weight of an instance of* VALUED WSP *equals zero if and only if the corresponding* WSP *instance is satisfiable.*

3. SOLVING VALUED WSP WITH USER-INDEPENDENT CONSTRAINTS

In Section 3.1 we introduce the notion of weighted user-independent constraints and prove that VALUED WSP with only user-independent constraints is fixed-parameter tractable (FPT). In Section 3.2, we describe an FPT algorithm to solve VALUED WSP with user-independent constraints.

3.1 Weighted User-Independent Constraints and Patterns

A weighted constraint c is called *user-independent* if, for every permutation θ of U, $w_C(\pi) = w_C(\theta \circ \pi)$. Thus, a weighted user-independent constraint does not distinguish between users. Any (weighted) counting constraint for which the weight of plan π is defined in terms of the cardinality of the image of π is user-independent.

Given a plan $\pi : S' \to U$, where $S' \subseteq S$, the *pattern* $P(\pi)$ of π is the partition $\{\pi^{-1}(u) : u \in U, \pi^{-1}(u) \neq \emptyset\}$ of S' into non-empty sets. We say that two plans π and π' are *equivalent* if they have the same pattern. If all constraints are user-independent and π and π' are equivalent, then $w_C(\pi) = w_C(\pi')$. A pattern is said to be *complete* if $S' = S$.

Generalising the corresponding result for WSP with user-independent constraints [7, Theorem 2], we can prove the following theorem, which uses weighted set-authorisation. The proof uses ideas from [7] and [18]. We assume(i) that the weight of each assignment can be computed in time polynomial in the number of steps, users and constraints (denoted k, n and m, respectively); (ii) we can determine whether a plan satisfies a constraint in time polynomial in the number of steps and users.

THEOREM 1. VALUED WSP *in which all constraints are user-independent can be solved in time* $2^{k \log k}(k+n+m)^{O(1)}$. *Thus,* VALUED WSP *with user-independent constraints is FPT.*

PROOF. For a positive integer x, let $[x] = \{1, \ldots, x\}$. Recall that for equivalent complete plans π and π', we have $w_C(\pi) = w_C(\pi')$. However, $w_A(\pi) = \sum_{u \in U} w_A(\pi^{-1}(u), u)$ is, in general, different from $w_A(\pi')$ and so we must compute the minimum value of $w_A(\pi)$ among all equivalent complete plans π. To do so efficiently, for a complete plan π, we first construct a weighted complete bipartite graph G_π with partite sets $[p]$ and U, where $p = |P(\pi)|$ as follows. Let $P(\pi) = \{T_1, \ldots, T_p\}$. The weight of an edge $\{q, u\}$ is $\sum_{s \in T_q} \omega(s, u)$.

Now observe that $G_{\pi'} = G_\pi$ for every pair π, π' of equivalent complete plans and that $w_A(\pi)$ equals the weight of the corresponding matching of G_π covering all vertices of $[p]$. Hence, it suffices to find such a matching of G_π of minimum weight, which can be done by the Hungarian method [19] in time $O(n^3)$.

Observe that the number of partitions of the set $[k]$ into non-empty subsets, called the Bell number B_k, is smaller than $k!$ and there are algorithms of running time $O(B_k) = O(2^{k \log k})$ to generate all partitions of $[k]$ [15]. Thus, we can generate all patterns in time $O(2^{k \log k})$. For each of them we compute the corresponding complete plan of minimum weight, and, among all such plans, we choose the one of smallest weight. The total running time is $O(2^{k \log k}(k+n+m)^{O(1)})$. □

Cohen *et al.* proved [7] that WSP with user-independent constraints cannot be solved in time $2^{o(k \log k)}(k+n+m)^{O(1)}$, unless the widely believed Exponential Time Hypothesis[4] fails. This and Proposition 1 imply that the FPT algorithm of Theorem 1 is optimal, in a sense.

3.2 Pattern Branch and Bound

We present a branch-and-bound algorithm (Algorithm 1) for the VALUED WSP. This algorithm is inspired by the Pattern Backtracking algorithm for the WSP [18]. However, the original algorithm solves a decision problem, whereas the VALUED WSP is an optimisation problem. Thus our algorithm for VALUED WSP requires a completely different algorithmic framework. We call our algorithm for the VALUED WSP Pattern Branch and Bound (PBB). Given a pattern P, we will write $T(P) = \bigcup_{p \in P} p$ to denote the set $T \subseteq S$ on which P is the partition.

Algorithm 1: Entry point of the PBB algorithm.

input : VALUED WSP instance W
output: The optimal complete plan for W
1 $\lambda \leftarrow \lambda(W)$, where $\lambda(W)$ is the global lower bound;
2 $\pi \leftarrow H(W)$, where $H(W)$ is a VALUED WSP heuristic;
3 **return** $explore(\emptyset, \pi, \lambda)$;

Algorithm 2: Recursive procedure $explore(P, \pi^*, \lambda)$ of the PBB algorithm.

input : Node P of the search tree; best plan π^* found so far; the global lower bound λ
output: The best plan found after exploring this branch of search
1 **if** P *is a complete pattern* **then**
2   Let π' be an optimal plan with pattern P;
3    $\pi'' \leftarrow \arg\min_{\pi \in \{\pi', \pi^*\}} w(\pi)$;
4 **else**
5    $\pi'' \leftarrow \pi^*$;
6   Select $s \in S \setminus T(P)$ maximising $I(P, s)$;
7   **forall the** *extensions P' of P with step s* **do**
8   **if** $L(P') < w(\pi'')$ *and* $w(\pi'') > \lambda$ **then**
9   Let $\pi' \leftarrow explore(P', \pi'', \lambda)$;
10        $\pi'' \leftarrow \arg\min_{\pi \in \{\pi', \pi''\}} w(\pi)$;
11 **return** π'';

The general idea of the algorithm follows the proof of Theorem 1. The algorithm explores the space of patterns with a depth-first search and for each complete pattern P seeks an optimal plan π such that $P(\pi) = P$ (recall that such a plan can be found efficiently). Each node of the search tree is a pattern, with the root being an empty pattern and leaves being complete patterns. In each non-leaf node P, the algorithm selects a step $s \in S$ such that $s \notin T(P)$ (line 6 of Algorithm 2), and generates one child node for each possibility to extend P with s (line 10). By extensions of P with step s we mean patterns P' obtained from P by adding s to one of the subsets $p \in P$ or adding a new subset $\{s\}$ to the partition; hence, there are $|P| + 1$ extensions of P.

---

[4]The Exponential Time Hypothesis claims there is no algorithm of running time $O^*(2^{o(n)})$ for 3SAT on n variables [17].

Like any branch-and-bound algorithm, PBB utilises a lower bound $L(P)$ for pruning branches. The lower bound $L(P)$ in a node P is computed as follows:

$$L(P) = \sum_{c \in C} L_c(P) + \sum_{p \in P} \min_{u \in U} \omega(p, u),$$

where $L_c(P)$ is the lower bound of $w_c(\pi)$, where π is an extension of a plan with pattern P. The implementation of $L_c(P)$ depends on the constraint type. For example, for a counting constraint c with the scope T and weight function $w_c(\pi) = \omega_c(|\pi(T)|)$, the lower bound can be computed as $L_c(P) = l(q, a)$, where $q = |\{p \in P : p \cap T \neq \emptyset\}|$, $a = |T \cap T(P)|$ and $l(q, a)$ is the following recursive function:

$$l(q, a) = \begin{cases} \omega_c(q) & \text{if } a = |T|, \\ \min\{l(q, a+1), l(q+1, a+1)\} & \text{otherwise.} \end{cases}$$

Other speed-ups implemented in the PBB algorithm include a global lower bound $\lambda(W)$ (line 1 of Algorithm 1) and the heuristic $H(W)$ to obtain a good upper bound from the very beginning of the search (line 2 of Algorithm 1). In a simple implementation, the global lower bound $\lambda(W)$ could be a constant function $\lambda(W) = 0$; that would terminate the algorithm as soon as a complete plan satisfying all the constraints and authorisations is found. The heuristic algorithm can be as simple as a trivial plan assigning some user to all the steps S.

In our implementation, however, we translate the VALUED WSP instance W into WSP instances and solve them to obtain better global lower bound and upper bound. Let $\mathrm{WSP}(W, x)$ be a WSP instance obtained from W by eliminating all the constraints and authorisations with penalties below x and converting the rest of the constraints and authorisations into hard constraints. By solving $\mathrm{WSP}(W, x)$ we establish either the global lower bound or the upper bound. If $\mathrm{WSP}(W, x)$ is unsatisfiable, we conclude that there exists no complete plan π such that $w(\pi) < x$ and, hence, $\lambda(W) = x$. Otherwise, the plan valid in $\mathrm{WSP}(W, x)$ can be used for an upper bound in W.

We start from solving the $\mathrm{WSP}(W, 1)$ and, if it turns out to be unsatisfiable, we solve the $\mathrm{WSP}(W, M)$, where M is a large enough number, which usually gives us a good upper bound as it rules out plans that break highly-penalised constraints. We solve the WSPs using an improved version of the WSP algorithm of [18].

The order in which patterns are extended (a step at a time) makes no difference to the worst-case time complexity of the algorithm but is crucial to its performance in practice [18]. It is defined by the 'importance' function $I(P, s)$, the intention being to focus on the most important steps as early as possible to quickly prune fruitless branches of the search. The importance of a step mostly depends on the constraints that include the step in their respective scopes. For example, if a step is involved in several separation-of-duty constraints, adding it to the pattern may significantly reduce the search space and possibly result in increased penalties for some at-most constraints. Another example is if most of the steps of some constraint's scope are assigned, and adding the remaining steps to the pattern may have a severe effect on the penalty for that constraint. The 'importance' metric is context-dependent, i.e. the order of steps needs to be determined dynamically in each branch of the search tree.

The 'importance' function $I(P, s)$ is a heuristic expression which we parametrised and optimised by an automated parameter tuning method. Our function $I(P, s)$ takes into account the number and types of the constraints in which the step is involved. In addition, it accounts for the constraints with incomplete scopes. Finally, we check intersections of 'conflicting' constraints such as at-most and not-equals or at-most and at-least.

As shown in the proof of Theorem 1, finding the optimal assignment of users given a fixed pattern can be done in $O(n^3)$ time (if computing $w_C(\pi)$ takes $O(n^3)$ time and computing $\omega(T, u)$ takes $O(n^3/k)$ time). Each non-leaf node of the search tree has at least two child nodes and, hence, the size of the search tree is within $O(B_k)$. Then the worst case time complexity of the PBB algorithm is $O(B_k n^3)$.

With the exception of the 'step importance' function $I(P, s)$, which is easy to adjust for any type of instances, and the lower bound $L(P)$, our algorithm is a generic solver for the user-independent VALUED WSP. For example, it does not exploit the specifics of the counting constraints which could be used to preprocess problem instances [7]. This shows that our approach is generic, easy to implement and its performance can be further improved by implementing instance-specific speed-ups.

4. EXPERIMENTAL RESULTS

The pseudo-Boolean SAT solver SAT4J has been used to solve the WSP [24]. Recent work has demonstrated that a bespoke pattern-based algorithm can outperform SAT4J in solving WSP [6, 18]. Integer linear programming has been used by Basin *et al.* to solve the *allocation existence problem* [2], which is related to VALUED WSP. In this section, we describe the experimental work on VALUED WSP that we have undertaken. In particular, we will compare the performance of our PBB algorithm to that of the state-of-art commercial MIP solver in our computational experiments on VALUED WSP. We first describe the problem instances we used and how we represented VALUED WSP as a mixed integer programming (MIP) problem. We then present the results of our experiments.

4.1 Benchmark Instances

We use a pseudo-random instance generator to produce benchmark instances. Our generator is an extension of an existing instance generator for WSP [18]. The parameters of the generator are the number k of steps in the instance, the not-equals constraints density d in the range 0–100%, the multiplier α for the number of constraints and the seed value for initialisation of the pseudo-random number generator. The generator produces an instance with k steps and $10k+10$ users: $10k$ employees U' and 10 external consultants U''. The penalty for assigning steps $T \subseteq S$ to an employee $u \in U'$ is given by

$$\omega(T, u) = |T \cap B| \cdot 10 + |T \setminus (A \cup B)| \cdot 10^6,$$

where $A \subset S$ and $B \subset S$ are selected uniformly at random from S, with $A \cap B = \emptyset$ and $|A|$ being selected uniformly at random from $[1, \lceil (k-4)/2 \rceil]$, and $|B| = 2$. The penalty for assigning steps $T \subseteq S$ to an external consultant $u \in U''$ is

given by

$$\omega(T,u) = \begin{cases} 0 & \text{if } T = \emptyset, \\ 20 & \text{if } T \neq \emptyset \text{ and } T \subseteq A, \\ 10^6 \cdot |T \setminus A| & \text{if } T \subseteq S \setminus A, \\ 10^6 \cdot |T \setminus A| + 20 & \text{otherwise,} \end{cases}$$

where $A \subset S$ is selected uniformly at random having selected $|A|$ uniformly at random from $[1, \lceil k/4 \rceil]$.

Further, $\lfloor (dk(k-1)+1)/2 \rfloor$ distinct not-equals constraints are produced uniformly at random, each with penalty 10^6 for assigning one user to both steps. Finally, αk at-most-3 constraints and αk at-least-3 constraints are generated uniformly at random. The scopes of all the at-most-3 and at-least-3 constraints are set to 5 steps. The at-least-3 penalties are defined as $\omega_c(1) = 10^6$, $\omega_c(2) = 1$, $\omega_c(3) = \omega_c(4) = \omega_c(5) = 0$. The at-most-3 penalties are defined as $\omega_c(1) = \omega_c(2) = \omega_c(3) = 0$, $\omega_c(4) = 5$ and $\omega_c(5) = 10$.

The source code of our instance generator can be found at [11].

4.2 Mixed Integer Programming Formulation

In order to use an MIP solver, we propose an efficient MIP formulation of the Valued WSP. Note that the MIP formulation is specific to the particular constraints present in the instances, unlike the PBB algorithm. In this section we describe an MIP formulation for the instances described in Section 4.1.

Let $C = C_{\leq} \cup C_{\geq}$, where $C_{\leq}$ is the set of at-most-r constraints and $C_{\geq}$ is the set of at-least-r constraints. (Note that not-equals constraints can be modelled as at-least-2 constraints with the scope of two steps.) For each constraint $c \in C$ we are given its scope $T_c \subseteq S$, the minimum (maximum, respectively) number r_c of users that can be assigned to $c \in C$ such that the at-most (at-least, respectively) constraint c is satisfied and the penalty $\omega_c(q)$ for assigning q distinct users to T_c (note that $\omega_c(r_c) = 0$).

For each employee $u \in U'$ and each step $s \in S \setminus A(u)$ we are given an additive weight $\omega_{su} > 0$ of assigning u to s, which models the penalties for steps in both $B(u)$ and $S \setminus (A(u) \cup B(u))$. For each consultant $u \in U''$ we are given a set of steps $A(u) \subseteq S$, any of which can be assigned to u for a penalty $\omega_u > 0$, and a weight $\Omega_u > 0$ for assigning a step $s \in S \setminus A(u)$ to u.

The complete plan in our formulation is defined by binary decision variables x_{su}, $s \in S$, $u \in U$. Variable x_{su} takes value 1 if step s is assigned to user u and 0 otherwise. The Valued WSP is then encoded in (1)–(14):

$$\begin{aligned} \text{minimise } & \sum_{c \in C_{\leq}} \sum_{q=r_c+1}^{|T_c|} (\omega_c(q) - \omega_c(q-1)) p_{cq} \\ & + \sum_{c \in C_{\geq}} \sum_{q=1}^{r_c-1} (\omega_c(q) - \omega_c(q+1)) p_{cq} \\ & + \sum_{u \in U'} \sum_{s \in S \setminus A(u)} \omega_{su} x_{su} \\ & + \sum_{u \in U''} \omega_u z_u + \sum_{u \in U''} \Omega_u \sum_{s \in S \setminus A(u)} x_{su} \end{aligned} \qquad (1)$$

subject to

$$\sum_{u \in U} x_{su} = 1 \text{ for } s \in S, \qquad (2)$$

$$\sum_{u \in U} y_{cu} - \sum_{q=r_c+1}^{|T_c|} p_{cq} \leq r_c \text{ for each } c \in C_{\leq}, \qquad (3)$$

$$p_{cq} - p_{c,q+1} \geq 0 \text{ for } c \in C_{\leq} \text{ and } q = r_c+1, \ldots, |T_c|-1, \qquad (4)$$

$$\sum_{u \in U} y_{cu} + \sum_{q=1}^{r_c-1} p_{cq} \geq r_c \text{ for each } c \in C_{\geq}, \qquad (5)$$

$$p_{cq} - p_{c,q+1} \leq 0 \text{ for } c \in C_{\geq} \text{ and } q = 1, 2, \ldots, r_c-2, \qquad (6)$$

$$y_{cu} - x_{su} \geq 0 \text{ for each } c \in C_{\leq}, \ u \in U \text{ and } s \in T_c, \qquad (7)$$

$$y_{cu} - \sum_{s \in T_c} x_{su} \leq 0 \text{ for each } c \in C_{\geq} \text{ and } u \in U, \qquad (8)$$

$$z_u - x_{su} \geq 0 \text{ for each } u \in U'' \text{ and } s \in A(u), \qquad (9)$$

$$x_{su} \in \{0,1\} \text{ for } s \in S \text{ and } u \in U, \qquad (10)$$

$$0 \leq y_{cu} \leq 1 \text{ for } c \in C \text{ and } u \in U, \qquad (11)$$

$$p_{cq} \in \{0,1\} \text{ for } c \in C_{\leq} \text{ and } q = r_c+1, \ldots, |T_c|, \qquad (12)$$

$$p_{cq} \in \{0,1\} \text{ for } c \in C_{\geq} \text{ and } q = 1, 2, \ldots, r_c-1, \qquad (13)$$

$$0 \leq z_u \leq 1 \text{ for } u \in U''. \qquad (14)$$

In addition to binary variables x_{su}, we introduce some other variables. Binary variables y_{cu}, $c \in C$, $u \in U$ determine if user u is assigned to some steps in the scope T_c of constraint c. Since y_{cu} for $c \in C_{\leq}$ is minimised and it is limited from below by binary expressions (7), its integrality constraint can be waived. Since y_{cu} for $c \in C_{\geq}$ is maximised and it is limited from above by binary expressions (8), its integrality constraint can also be waived. Similar logic applies to z_u, which indicates if the consultant $u \in U''$ is assigned any steps in $B(u)$. Finally, we introduce the binary variables p_{cq} for each $c \in C$ and $q \in \mathbb{N}$ such that $w_c(q) > 0$. These variables are responsible for the constraint penalties and (with appropriate limitations imposed on the instances, as our instance generator does) the integrality of p_{cq} and constraints (4) and (6) can be waived.

The objective function (1) is the weight of the plan defined by x_{su}, and thus our aim is to minimise it.

4.3 Experimental Results

We conducted a series of computational experiments to test the performance of the Valued WSP solution methods. Our test machine is powered by two Intel Xeon CPU's E5-2630 v2 (2.6 GHz) and has 32 GB RAM installed. The PBB algorithm is implemented in C#, and the MIP formulation is solved with CPLEX 12.6. The source code of our implementation of the Pattern Branch and Bound algorithm can be found at [11]. In all our experiments, each solver run is allocated exactly one physical CPU core. Each result is reported as an average over 100 runs for 100 instances obtained by changing the random generator seed value.

Main computational results are reported in Table 1. The columns k, d and α indicate the parameters of the instances. For each combination of parameters, 100 instances were generated. The column "Sat." reports the percentage of the instances that are satisfiable. The column $w(\pi)$ shows the average weight of the optimal complete plans. The other columns compare the MIP-based solver to the PBB algo-

k	d	α	Sat.	$w(\pi)$	Solved		Time, sec		$w_C(\pi)$		$w_A(\pi)$		Best $w(\pi)$	
					PBB	MIP	PBB	MIP	PBB	MIP	PBB	MIP	PBB	MIP
20	10%	0.50	100%	0.0	100%	100%	0.0	5.9	0.0	0.0	0.0	0.0	—	—
20	20%	0.50	90%	0.4	100%	100%	0.0	19.8	0.4	0.4	0.0	0.0	—	—
20	30%	0.50	37%	4.0	100%	100%	0.0	65.0	3.8	4.0	0.2	0.0	—	—
20	10%	1.00	18%	4.4	100%	100%	0.1	556.0	4.3	4.2	0.1	0.2	—	—
20	20%	1.00	0%	14.2	100%	100%	0.1	532.9	13.5	13.8	0.7	0.4	—	—
20	30%	1.00	0%	24.3	100%	100%	0.1	469.9	23.4	23.5	0.9	0.8	—	—
25	10%	0.50	100%	0.0	100%	100%	0.0	32.0	0.0	0.0	0.0	0.0	—	—
25	20%	0.50	93%	0.4	100%	100%	0.0	102.2	0.4	0.4	0.0	0.0	—	—
25	30%	0.50	27%	5.0	100%	100%	0.0	319.3	5.0	5.0	0.0	0.0	—	—
25	10%	1.00	40%	2.3	100%	39%	0.3	?	2.3	?	0.0	?	—	4.1
25	20%	1.00	0%	14.3	100%	66%	0.8	?	13.7	?	0.6	?	—	14.9
25	30%	1.00	0%	29.9	100%	95%	2.0	?	28.9	?	1.0	?	—	29.9
30	10%	0.50	100%	0.0	100%	100%	0.0	72.2	0.0	0.0	0.0	0.0	—	—
30	20%	0.50	88%	0.6	100%	99%	0.0	?	0.6	?	0.0	?	—	0.6
30	30%	0.50	24%	5.7	100%	99%	0.1	?	5.7	?	0.0	?	—	5.7
30	10%	1.00	6%	5.8	100%	3%	5.1	?	5.7	?	0.1	?	—	14.9
30	20%	1.00	0%	22.8	100%	7%	36.3	?	22.4	?	0.4	?	—	31.5
30	30%	1.00	0%	43.5	100%	31%	173.7	?	43.2	?	0.3	?	—	52.8
35	10%	0.50	100%	0.0	100%	100%	0.0	195.9	0.0	0.0	0.0	0.0	—	—
35	20%	0.50	91%	0.4	100%	89%	0.2	?	0.4	?	0.0	?	—	0.6
35	30%	0.50	13%	7.7	100%	68%	18.5	?	7.7	?	0.0	?	—	8.0
35	10%	1.00	3%	6.2	100%	0%	64.7	?	6.2	?	0.0	?	—	33.5
35	20%	1.00	?	?	92%	0%	?	?	?	?	?	?	29.6	130071.1
35	30%	1.00	?	?	48%	0%	?	?	?	?	?	?	60.1	2990104.6

Table 1: Comparison of the PBB and MIP solvers, each being given one hour per instance A question mark shows that at least one of the runs failed for the MIP solver.

rithm. Each of them is given one hour for each instance. The "Solved" columns show the percentage of instances successfully solved within the one hour limit by each of the solvers. The "Time, sec" columns show the average time taken by each of the approaches. If at least one of the runs failed for a solver, a question mark is shown in the corresponding cell of the table. The $w_C(\pi)$ and $w_A(\pi)$ columns show the components of the weight corresponding to the constraints and the authorisations penalties, respectively. For those parameters where at least one of the runs failed, we use the "Best $w(\pi)$" columns to to report the average weight of the best plan obtained by each of the solvers.
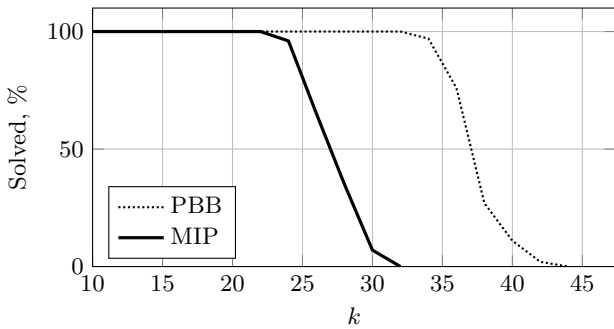
For each k, Table 1 includes a range of instances starting from lightly constrained instances, which are mostly satisfiable, to highly constrained instances, none of which is satisfiable. Naturally, the most interesting instances from the perspective of VALUED WSP are those that are unsatisfiable (since it is necessary to find an optimal plan of nonzero weight for such instances). We are most interested in the unsatisfiable instances with moderate weights of the optimal complete plans. A small weight $w(\pi)$ indicates that only a few minor exceptions are needed to implement the complete plan π. With such a plan, it is easy to identify the bottleneck of the problem and refine it or accept the exceptions to the constraints as the exceptions are likely to be mild. The $w_C(\pi)$ and $w_A(\pi)$ columns show that in most of the cases the authorisations were not broken. In fact, there were only a few highly-constrained instances in which the the optimal complete plans assigned some steps to consul-

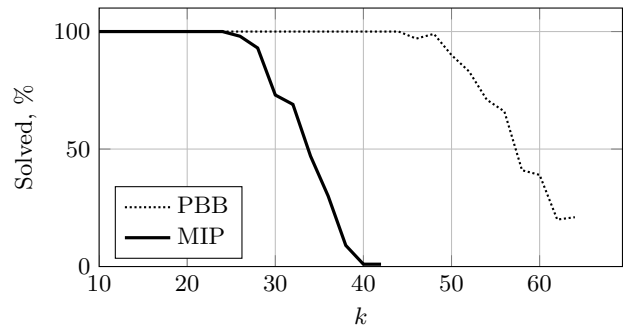tants, as the penalty for doing that is relatively high in our instances.

The complexity of the instances depends to a great extent on the number of steps k and the parameters of the instances. While small lightly-constrained instances can be easily tackled by either of the solvers, other instances clearly require an efficient algorithm. The MIP solver succeeds with all the instances of size $k = 20$ but fails to solve many of the larger instances within an hour. The PBB algorithm demonstrates a much better performance, solving all the instances of size up to $k = 30$ and the majority of the instances of size $k = 35$. It is worth noting that the running time of the MIP solver can reach 10 minutes for $k = 20$ while the PBB solver solves all such instances within a fraction of a second.

Exact algorithms for solving hard optimisation problems will, necessarily, take a long time to compute results for certain instances. However, such an algorithm may find an optimum or near-optimum result long before the whole solution space has been searched and can thus be used to compute a reasonable solution for instances that do not run to completion. The Best $w(\pi)$ column in Table 1 clearly shows that MIP is far less suitable than PBB for this purpose.

To establish the practical limit on the problem size that each of the solvers can tackle within a reasonable time, we conducted another experiment to determine the number of instances that the two solvers could solve given at most one hour for each instance. Figure 1 shows the results of the experiment. Each result is averaged over 100 experiments for each instance.

(a) $d = 20\%$ and $\alpha = 1.00$

(b) $d = 10\%$ and $\alpha = 0.75$

Figure 1: Comparison of the methods in terms of the ability to solve an instance within one hour

Figure 1a shows the performance of the methods on highly constrained instances. Being given one hour, PBB solves 100% of the instances of size up to $k = 32$. In contrast, MIP can only reliably manage instances for $k \leq 22$, and for $k = 32$ it fails to solve any instances at all. Figure 1b reports the results of the same experiment but for less constrained instances. The results are broadly similar, with PBB solving all the instances of up to $k = 44$, whereas MIP fails for some instances when $k > 24$. This experiment shows that the PBB algorithm significantly extends the range of solvable instances of VALUED WSP, something that will be important for large real-world workflow specifications. Considering that the running time of each of the methods grows exponentially with the size of the problem, large instances of VALUED WSP would require enormous computational power to be solved with MIP, while the PBB algorithm tackles them within minutes on a regular machine.

5. RELATED WORK

Recent work on workflow satisfiability has borrowed techniques from the literature on constraint satisfaction [7]. Indeed, WSP may be regarded as a constraint satisfaction problem, albeit with some unusual features which makes the study of WSP of interest in its own right. Recent work in the constraint satisfaction community has made a distinction between "hard" and "soft" constraints: the former must be satisfied, while the latter may be broken provided the "cost" of breaking the constraint is taken into account.

The *valued constraint satisfaction problem*, or VCSP for short, was introduced by Schiex, Fargier and Verfaillie [23] as a unifying framework for studying constraint programming with soft constraints. The study of a special case of VCSP, called *finite-valued* VCSP, was initiated by Cohen *et al.* [8]. In this case, useful for many applications, all weights are in $\mathbb{Z}$ (i.e., finite) and the objective function is the sum of appropriate weights. Valued CSP has influenced our framework for defining costs and VALUED WSP.

Recent work on WSP introduced the notion of a pattern for user-independent constraints, and bespoke algorithms, optimised to solve WSP using patterns, have been developed [7, 18]. The branch-and-bound algorithm in Section 3.2 is influenced by the work of Karapetyan, Gagarin and Gutin [18].

The most closely related work in the literature on access control in workflows is that of Basin, Burri and Karjoth [2], which considers the cost of modifying the authorisation policy when the workflow is unsatisfiable. They encode the problem of minimizing this cost as a integer linear programming problem and use off-the-shelf software to solve the resulting problem. We tackle the problem of an unsatisfiable workflow specification in a different way. We assume the constraints and authorisation policy are fixed and instead associate costs with breaking the constraints and/or policies. However, each violation will incur a cost and the goal of VALUED WSP is to minimise that cost. Thus our approach provides greater flexibility than that of Basin *et al.*: we can break constraints as well as override the existing authorisation policy. Obviously, there may be constraints (arising from statutory requirements, say) that cannot be broken. Violation of such a constraint is simply assigned the maximum cost. And of course, we can always refuse to implement a plan proposed by the algorithm.

Our work is also related to the growing body of research on risk-based and risk-aware access control [4, 5, 13, 20]. In such approaches, the decision returned by policy decision point for a given access request is not necessarily a simple "allow" or "deny". The decision may be a number in the range $[0, 1]$ indicating the risk associated with allowing the request, which allows the policy enforcement point to allow or deny the request on the basis of cumulative risk (either on a per-user or system basis). The decision may also include an obligation that must be fulfilled by the policy enforcement point or requester to ensure that the risk is recorded and/or mitigated appropriately.

There is little work in the security literature on risk-aware workflows. One exception is the MRARD framework of Han, Ni and Chen [16]. However, the emphasis of their work (and of similar work in the business processing literature) is on the modelling and computation of risk, rather than determining an optimal assignment of users to steps in a workflow given the risk metrics.

6. CONCLUDING REMARKS

We have established a framework that enables us to reason about unsatisfiable workflow specifications by associating costs with policy and constraint violations. This, in turn, enables us to formulate the VALUED WSP, whose solution provides an assignment of users to steps that minimises the total cost of violations. We have developed a bespoke algorithm for solving VALUED WSP and shown that its performance is far better than a generic solver, both in terms

of the time taken to solve Valued WSP and the range of instances that can be solved in a reasonable amount of time.

There are several interesting possibilities for future work. One obvious possibility is to move to a completely risk-based approach for the assignment of users to steps in workflows. Specifically, we retain the constraints but replace the authorisation policy with a risk matrix, associating each user-step pair with a risk. The goal would be to ensure that the risk associated with a workflow instance remains below some specified threshold.

A second possibility arises from the idea of associating each pair (T, u) with a cost, which provides the basis for an alternative "non-linear" approach to access control. Suppose that we consider a set of permissions P, as in conventional role-based access control, and we associate a cost $\omega(Q, u)$ with each pair, where u is a user and Q is a subset of P. Given an RBAC policy, expressed as a user-role relation $UA \subseteq U \times R$ and permission-role relation $PA \subseteq R \times P$, we write $P(u)$ to denote the set of permissions for which u is authorised: that is, $P(u) = \{p \in P : \exists r \in R, (u, r) \in UA, (r, p) \in PA\}$. Then we define the weight of the policy to be

$$w_A(UA, PA) = \sum_{u \in U} \omega(P(u), u).$$

This then raises some interesting questions that may have practical value. We might, for example, consider the following problem: given inputs U, P, $\{\omega(Q, u) : u \in U, Q \subseteq P\}$ and integer k, compute a set of roles of R of cardinality k and relations $UA \subseteq U \times R$ and $PA \subseteq R \times P$ such that at least one user is authorised for every permission and $w_A(UA, PA)$ is minimised. Alternatively, we may insist that a user session does not exceed a "budget", where the cost of a session in which user u invokes permissions Q is defined to be $\omega(Q, u)$.

7. ACKNOWLEDGMENTS

8. REFERENCES

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

[2] D. A. Basin, S. J. Burri, and G. Karjoth. Optimal workflow-aware authorizations. In V. Atluri, J. Vaidya, A. Kern, and M. Kantarcioglu, editors, *17th ACM Symposium on Access Control Models and Technologies, SACMAT'12, Newark, NJ, USA – June 20–22, 2012*, pages 93–102. ACM, 2012.

[3] D. F. C. Brewer and M. J. Nash. The Chinese Wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society, 1989.

[4] L. Chen and J. Crampton. Risk-aware role-based access control. In C. Meadows and M. C. F. Gago, editors, *Security and Trust Management - 7th International Workshop, STM 2011, Copenhagen, Denmark, June 27–28, 2011, Revised Selected Papers*, volume 7170 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2011.

[5] P. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20–23 May 2007, Oakland, California, USA*, pages 222–230. IEEE Computer Society, 2007.

[6] D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones. Engineering algorithms for workflow satisfiability problem with user-independent constraints. In J. Chen, J. Hopcroft, and J. Wang, editors, *Frontiers in Algorithmics, FAW 2014*, volume 8497 of *Lecture Notes in Computer Science*, pages 48–59. Springer, 2014.

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

[8] D. A. Cohen, M. C. Cooper, P. G. Jeavons, and A. A. Krokhin. The complexity of soft constraint satisfaction. *Artif. Intell.*, 170(11):983–1016, Aug. 2006.

[9] J. Crampton, R. Crowston, G. Gutin, M. Jones, and M. S. Ramanujan. Fixed-parameter tractability of workflow satisfiability in the presence of seniority constraints. In M. R. Fellows, X. Tan, and B. Zhu, editors, *FAW-AAIM*, volume 7924 of *Lecture Notes in Computer Science*, pages 198–209. Springer, 2013.

[10] J. Crampton and G. Gutin. Constraint expressions and workflow satisfiability. In M. Conti, J. Vaidya, and A. Schaad, editors, *SACMAT*, pages 73–84. ACM, 2013.

[11] J. Crampton, G. Gutin, and D. Karapetyan. Source codes of the pattern branch and bound algorithm for the valued workflow satisfiability problem. DOI:10.6084/m9.figshare.1367470 retrieved 5 April 2015.

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

[13] N. Dimmock, A. Belokosztolszki, D. M. Eyers, J. Bacon, and K. Moody. Using trust and risk in role-based access control policies. In T. Jaeger and E. Ferrari, editors, *SACMAT 2004, 9th ACM Symposium on Access Control Models and Technologies, Yorktown Heights, New York, USA, June 2–4, 2004, Proceedings*, pages 156–162. ACM, 2004.

[14] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Verlag, 1999.

[15] M. Er. A fast algorithm for generation set partitions. *The Computer Journal*, 31:283–284, 1988.

[16] W. Han, Q. Ni, and H. Chen. Apply measurable risk to strengthen security of a role-based delegation supporting workflow system. In *POLICY 2009, IEEE International Symposium on Policies for Distributed Systems and Networks, London, UK, 20–22 July 2009*, pages 45–52. IEEE Computer Society, 2009.

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

[18] D. Karapetyan, A. Gagarin, and G. Gutin. Pattern backtracking algorithm for the workflow satisfiability problem. In *Frontiers in Algorithmics, FAW 2015*, volume to appear of *Lecture Notes in Computer Science*. Springer, 2015.

[19] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[20] Q. Ni, E. Bertino, and J. Lobo. Risk-based access control systems built on fuzzy inferences. In D. Feng, D. A. Basin, and P. Liu, editors, *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, Beijing, China, April 13–16, 2010*, pages 250–260. ACM, 2010.

[21] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[22] A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In D. F. Ferraiolo and I. Ray, editors, *SACMAT 2006, 11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe, California, USA, June 7–9, 2006, Proceedings*, pages 139–149. ACM, 2006.

[23] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence – Volume 1*, IJCAI'95, pages 631–637, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

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