Bounded and Approximate Strong Satisfiability in Workflows

Jason Crampton
Royal Holloway, University of London
Egham, United Kingdom
jason.crampton@rhul.ac.uk

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

Diptapriyo Majumdar
Royal Holloway, University of London
Egham, United Kingdom
diptapriyo.majumdar@rhul.ac.uk

ABSTRACT

There has been a considerable amount of interest in recent years in the problem of workflow satisfiability, which asks whether the existence of constraints in a workflow specification makes it impossible to allocate authorized users to each step in the workflow. Recent developments have seen the workflow satisfiability problem (WSP) studied in the context of workflow specifications in which the set of steps may vary from one instance of the workflow to another. This, in turn, means that some constraints may only apply to certain workflow instances. Inevitably, WSP becomes more complex for such workflow specifications. Other approaches have considered the possibility of associating costs with the violation of "soft" constraints and authorizations. Workflow satisfiability in this context becomes a question of minimizing the cost of allocating users to steps in the workflow. In this paper, we introduce new problems, which we believe to be of practical relevance, that combine these approaches. In particular, we consider the question of whether, given a workflow specification with costs and a "budget", all possible workflow instances have an allocation of users to steps that does not exceed the budget. We design a fixed-parameter tractable algorithm to solve this problem parameterized by the total number of steps, release points and xor branchings.

KEYWORDS

workflow satisfiability; release points; workflow composition; xor-branching

ACM Reference Format:

Jason Crampton, Gregory Gutin, and Diptapriyo Majumdar. 2019. Bounded and Approximate Strong Satisfiability in Workflows. In *The 24th ACM Symposium on Access Control Models and Technologies (SACMAT '19), June 3–6, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3322431.3325418

1 INTRODUCTION

Many businesses use computerized systems to manage their business processes. A common example of such a system is a workflow management system, which is responsible for the co-ordination and execution of steps in a business process. The overall structure of the business process is fixed and may be defined as a workflow specification comprising a set of steps that must be performed in a particular sequence. However, the specific steps that are performed may vary from one instance of the workflow to another. For example, certain steps may only be relevant in a purchase order workflow if the value of a specific order exceeds a particular value.

The steps in a workflow instance are executed by users and these users will vary from instance to instance. We may wish to impose some form of access control on the execution of those steps. This control may take the form of an authorization policy and a set of authorization constraints: the former defines which users are authorized to perform which steps; and the latter limits the combinations of users that may perform certain sets of steps in the business process.

There may be constraints that only apply when certain sub-workflows are executed in a particular workflow instance. Basin, Burri and Karjoth introduced a mechanism for modeling such constraints using *release points* [3]. Informally, release points allow a constraint to be "switched off" when some specified points in a workflow instance are reached. In particular, when different release points are located in different mutually exclusive sub-processes, it is possible to encode *conditional* constraints.

An assignment of users to workflow steps is a *plan*. A plan is *valid* if all users are authorized and no constraint is violated. We say a workflow specification is *satisfiable* if there exists a *valid plan* for the specification. An efficient algorithm for deciding the so-called *workflow satisfiability question* (WSP) is important from the point of view of static analysis of workflow specifications and as an on-line access control decision problem [5, Section 2.2].

Crampton, Gutin and Karapetyan [6] introduced the *valued* workflow satisfiability problem (VWSP), where constraint and authorization violations are associated with costs, which may be regarded as an estimate of the risk associated with allowing those violations. A solution to VWSP is a plan having minimal cost.

In this paper, we introduce a new class of workflow satisfiability problems, based on extended workflow specifications and costs associated with policy and constraint violation. The notion of *strong* satisfiability is associated with workflow specifications in which the set of steps executed in a workflow instance may vary from instance to another. Typically, this variation arises because of conditional branching in the specification. Strong satisfiability requires that there exists a valid plan for every possible set of steps that could form a workflow instance. A weaker form of strong satisfiability asks whether there exists a "reasonable" plan for every possible set of steps, where "reasonable" means its cost does not exceed some threshold value that is part of the input to the problem. An alternative question would be that of "approximate satisfiability", which asks whether there exists a valid plan for at least some fraction (stated as part of the input to the problem) of all possible sets of steps. An organization might be prepared to make a workflow

SACMAT '19, June 3–6, 2019, Toronto, ON, Canada
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6753-0/19/06…$15.00
https://doi.org/10.1145/3322431.3325418

specification operational if, for example, it is known that there exists a valid plan for at least 99% of the possible workflow instances.

We define three decision problems based on the informal notions described above. It can be shown that these problems are fixed-parameter tractable (FPT), subject to reasonable assumptions about the workflow specification. Informally, this means that relatively efficient algorithms exist to solve these problems.

In Section 2, we introduce relevant notation and terminology. Then, in Section 3, we describe relevant workflow models and satisfiability problems. We formally define bounded and approximate WSP problems in Section 4. We show that one of the bounded problems is FPT in Section 5. The other problems can also be shown to be FPT [7]. The full version of this paper [7] also contains a running example and more background information.

2 NOTATION AND TERMINOLOGY

A directed graph (*digraph* for short) is a pair $G = (V, E)$, where V is the set of vertices, and $E \subseteq V \times V$ is the set of edges. A directed acyclic graph (DAG) is a digraph which does not contain any directed cycle, *i.e.* no sequence $(u_0, u_1 \dots, u_{k-1}, u_0)$ such that each pair of consecutive vertices belongs to E.

For $u \in V$, we define the *in-neighborhood* of u to be the set $N^-(u) = \{t \in V : (t, u) \in E\}$; the *in-degree* of u is the size of its in-neighborhood $|N^-(u)|$. Similarly, the *out-neighborhood* of u is the set $N^+(u) = \{w \in V : (u, w) \in E\}$ and the *out-degree* of u is $|N^+(u)|$. A vertex of in-degree 0 is called a *source*, while a vertex of out-degree 0 is called a *sink*. For $S \subseteq V$, we denote by $G[S]$ the *induced subgraph* $(S, E \cap (S \times S))$. By abuse of notation, we will sometimes write $G \setminus S$ as a shortcut for $G[V \setminus S]$. For additional information about directed graphs, we refer the reader to [2].

Sometimes, it is convenient to represent a DAG as a partial order on V. Indeed, we may write $u \leq v$ for $u, v \in V$ whenever $u = v$ or there exists a directed path from u to v. By extension, we may write $u < v$ if $u \leq v$ and $u \neq v$.

For any positive integer n, let $[n]$ denote $\{1, \dots, n\}$. An ordered sequence $\sigma = (v_1, \dots, v_q)$ of distinct vertices of V is called a *linear subextension* of G if and only if for every $i, j \in [q]$, $v_i \leq v_j$ implies $i \leq j$. If σ contains all vertices of V, then we say that σ is a *linear extension* of G.

Many decision problems take several parameters as input. Multivariate analysis of a hard problem's complexity may yield efficient algorithms when it can be assumed that certain parameters take small values in practice. We say that a decision problem is *fixed-parameter tractable* (FPT) if there exists an algorithm that decides if an instance is positive in $O(f(\kappa)p(n))$ time for some computable function f and some polynomial p, where n denotes the size of an instance, and κ is a (small) parameter of the instance. Accordingly, we will call such an algorithm an *FPT algorithm*.

In many cases, the decision problem under consideration has several parameters $\kappa_1, \dots, \kappa_p$. This is reduced to the case of just one parameter by considering the parameter $\kappa = \kappa_1 + \dots + \kappa_p$. In fixed-parameter tractable algorithmics, the time $O(f(\kappa)p(n))$ is often written as $O^*(f(\kappa))$. Thus, O^* hides not only constant factors like O, but also polynomials (exponential functions are usually more important when evaluating the running time of FPT algorithms than polynomial factors). For more details about parameterized complexity, we refer the reader to the monograph [9].

3 THE WORKFLOW SATISFIABILITY PROBLEM

A *workflow specification* is defined by a directed acyclic graph $G = (S, E)$, where S is the set of steps to be executed, and $E \subseteq S \times S$ defines a partial ordering on the set of steps in the workflow, in the sense that $(s_1, s_2) \in E$ means that step s_1 must be executed before s_2 in every instance of the workflow. Note that the order is not required to be total, so the exact sequence of steps may vary from instance to instance. In addition, we are also given a set of users U and an *authorization policy* $A \subseteq S \times U$, where $(s, u) \in A$ means that user u is authorized to execute step s. A workflow specification $G = (S, E)$ together with an authorization policy is called a *workflow schema*. Throughout the paper, we will assume that for every step $s \in S$, there exists some user $u \in U$ such that $(s, u) \in A$.

A workflow *constraint* (T, Θ) limits the users that are allowed to perform a set of steps T in any execution of the workflow. In particular, Θ identifies authorized (partial) assignments of users to steps in T, i.e. Θ is a set of functions from T to U. A (partial) plan is a function $\pi : S' \to U$, where $S' \subseteq S$. A plan $\pi : S \to U$ represents an allocation of steps to users. The workflow satisfiability problem (WSP) is concerned with the existence or otherwise of a plan that is authorized and satisfies all constraints.

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. Then we say $\pi : S' \to U$ *satisfies* a workflow constraint (T, Θ) if $T \nsubseteq S'$ or $\pi|_T \in \Theta$.

In practice, we do not define a constraint by giving the family of functions Θ extensionally, as the size of such set might be exponential in the number of users and steps. Instead, we will assume that constraints have "compact" descriptions, in the sense that it takes polynomial time to test whether a given plan satisfies a constraint. All constraints of relevance in practice satisfy this property. For instance, the two most well-known constraints are *binding-of-duty* (BoD) and *separation-of-duty* (SoD). The *scope* of these constraints is binary: a plan π satisfies a BoD constraint $(\{s_1, s_2\}, =)$ if and only if $\pi(s_1) = \pi(s_2)$; and π satisfies an SoD constraint $(\{s_1, s_2\}, \neq)$ if and only if $\pi(s_1) \neq \pi(s_2)$. Natural generalizations of these constraints are atmost and $\mathsf{atleast}$ constraints, in which the scope may be of arbitrary size, and the definition of such constraints includes an additional integer k. Given $T \subseteq S$, a plan satisfies $\mathsf{atmost}(T, k)$ (resp. $\mathsf{atleast}(T, k)$) if and only if $|\pi(T)| \leq k$ (resp. $|\pi(T)| \geq k$).

User-independent constraints generalize all these forms of constraints [4]. Informally, such a constraint limits the execution of steps in a workflow, but is indifferent to the particular users that execute the steps. More formally, a constraint (T, Θ) is user-independent if whenever $\theta \in \Theta$ and $\psi : U \to U$ is a permutation then $\psi \circ \theta \in \Theta$ (where $\circ$ denotes function composition). A separation of duty constraint, on two steps for example, simply requires that two *different* users execute the steps, not that, say, Alice and Bob (in particular) must execute them. Similarly, a binding of duty constraint on two steps only requires that the *same* user executes the steps. More generally, $\mathsf{atleast}$ and atmost constraints are user-independent.

It appears most constraints that are useful in practice are user-independent: all constraints defined in the ANSI-RBAC standard [1], for example, are user-independent.

A *constrained workflow authorization schema* is a tuple $(G = (S, E), U, A, C)$, where (G, U, A) is a workflow schema, and C is a set of constraints. We say that a plan $\pi : S \to U$ is *authorized* if $(s, \pi(s)) \in A$ for every $s \in S$, and we say that π is *valid* if it is authorized and if it satisfies all $c \in C$. Then the Workflow Satisfiability Problem is defined in the following way:

> Workflow Satisfiability Problem (WSP)
> *Input:* A constrained workflow authorization schema $W = (G = (S, E), U, A, C)$
> *Question:* Is there a valid plan $\pi : S \to U$?

Henceforth, "workflow schema" will mean "constrained workflow authorization schema".

3.1 Valued workflow satisfiability

We now review the problem of minimizing the cost of "breaking" the policies and/or constraints. Informally, given a workflow schema, for each plan π, we define the weight (total cost) $w(\pi)$ associated with the plan π. The problem, then, is to find a plan with minimum total cost. More formally, let $((S, E), U, A, C)$ be a workflow 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*. Then we define the *constraint weight* of π to be $w_C(\pi) = \sum_{c \in C} w_c(\pi)$. Note that $w_C(\pi) = 0$ if and only if π satisfies all constraints in C.

We next introduce a function $w_A : \Pi \to \mathbb{Z}$, which assigns a cost for each plan with respect to the authorization policy. The intuition is that a plan in which every user is authorized 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 unauthorized users increases. More formally, we define the *authorization weight* of π to be

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

Then the valued Valued Workflow Satisfiability Problem is defined in the following way [6].

> Valued WSP
> *Input:* A constrained workflow authorization schema $((S, E), U, A, C)$ with weights for constraints and authorizations, as above.
> *Output:* A plan $\pi : S \to U$ that minimizes $w(\pi) = w_C(\pi) + w_A(\pi)$.

Under the assumption that for every plan π its weight $w(\pi)$ can be computed in time polynomial in $|S| + |U| + |C|$ (this assumption is justified in many practical situations [6]), the following holds.

Theorem 3.1. *[6]* Valued WSP *can be solved in time* $O^*(2^{k \log k})$, *where* $k = |S|$.

3.2 Compositional workflows

This section summarizes the model introduced by Crampton, Gutin and Watrigant [8]. A *compositional workflow specification* is defined recursively using three operations: serial composition, parallel branching and xor branching. Like a "classical" workflow specification, it can be represented as a DAG $G = (V, E)$. However, in the case of a compositional workflow, not all vertices represent steps. In addition to the set of (classical) steps, V also contains R, the set of *release points* [3], and O, the set of *orchestration points*. We will sometimes directly define a compositional workflow specification as $G = (S \cup R \cup O, E)$.

The DAG of a compositional workflow always contains two special orchestration points: a source vertex α, called *input* and a sink vertex ω, called *output*. Moreover, an *atomic* compositional workflow specification (*i.e.* the base case for constructing such a workflow) consists of a single step or release point v, and can be represented by the DAG $G = (\{\alpha, v, \omega\}, \{(\alpha, v), (v, \omega)\})$. Given two compositional workflows $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with respective input and output vertices α_1, ω_1 and α_2, ω_2, respectively, we may construct new compositional workflows using serial composition, and parallel branching and xor branching, denoted by $G_1; G_2$, $G_1 \parallel G_2$ and $G_1 \otimes G_2$, respectively. We assume that $V_1 \cap V_2 = \emptyset$.

For *serial composition*, all the steps in G_1 must be completed before the steps in G_2. Hence, the DAG of $G_1; G_2$ is formed by taking the union of V_1 and V_2, the union of E_1 and E_2, and the addition of a single edge from ω_1 to α_2. Thus, α_1 (resp. ω_2) is the input (resp. output) vertex of $G_1; G_2$.

For *parallel composition*, the execution of the steps in G_1 and G_2 may be interleaved. Hence, the DAG of $G_1 \parallel G_2$ is formed by taking the union of V_1 and V_2, the union of E_1 and E_2, the addition of new input and output vertices $\alpha_{\parallel}$ and $\omega_{\parallel}$, and the addition of edges $(\alpha_{\parallel}, \alpha_1), (\alpha_{\parallel}, \alpha_2), (\omega_1, \omega_{\parallel})$ and $(\omega_2, \omega_{\parallel})$. This form of composition is sometimes known as an *AND-fork* [10] or a *parallel gateway* [12].

In both serial and parallel compositions, all steps in G_1 and G_2 are executed. In *xor composition*, either the steps in G_1 are executed or the steps in G_2, but not both. In other words, xor composition represents non-deterministic choice in a workflow specification. The DAG $G_1 \otimes G_2$ is formed by taking the union of V_1 and V_2, the union of E_1 and E_2, the addition of new input and output vertices $\alpha_{\otimes}$ and $\omega_{\otimes}$, and the addition of edges $(\alpha_{\otimes}, \alpha_1), (\alpha_{\otimes}, \alpha_2), (\omega_1, \omega_{\otimes})$ and $(\omega_2, \omega_{\otimes})$. Given $G_1 \otimes G_2$, we will say that every pair of vertices $(v, v') \in V_1 \times V_2$ are *exclusive*. We say that a compositional workflow is *xor-free* if it can be constructed with only serial and parallel operations.

For the sake of readability, we will sometimes simplify the representation of a compositional workflow by replacing an orchestration point having a single in-neighbor u and a single out-neighbor v by the edge (u, v) (for instance, a path $(\alpha_1, s_1, \omega_1, \alpha_2, s_2, \omega_2)$ will be replaced by $(\alpha_1, s_1, s_2, \omega_2)$). A compositional workflow specification $G = (V, E)$ together with an authorization policy $A \subseteq S \times U$ will be called a *compositional workflow schema*.

3.2.1 *Execution sequences.*

In a compositional workflow having an xor branching, there exists more than one set of steps that could comprise a workflow instance. And in a compositional workflow having only parallel branching, two different workflow instances will contain the same steps but they may occur in different orders.

We characterize the set of possible workflow instances in terms of *execution sequences.*

An execution sequence is an ordered sequence of steps and release points and may be empty. For execution sequences $\sigma = (a_1, \ldots, a_k)$ and $\sigma' = (b_1, \ldots, b_k)$, we define the following:

$$\sigma + \sigma' = \{(a_1, \ldots, a_k, b_1, \ldots, b_\ell)\}$$
$$\sigma * \sigma' = \{(a_1) + \sigma'' : \sigma'' \in (a_2, \ldots, a_k) * (b_1, \ldots, b_\ell)\} \cup \{(b_1) + \sigma'' : \sigma'' \in (a_1, \ldots, a_k) * (b_2, \ldots, b_\ell)\}$$
$$\sigma * () = () * \sigma = \sigma$$

In other words, $\sigma + \sigma'$ represents concatenation of σ and σ'; and $\sigma * \sigma'$ represents all possible interleavings of σ and σ' that preserve the ordering of elements in both σ and σ'. Given sets of execution sequences Σ and Σ', we write $\Sigma + \Sigma'$ and $\Sigma * \Sigma'$ to denote $\{\sigma + \sigma' : \sigma \in \Sigma, \sigma' \in \Sigma'\}$ and $\{\sigma * \sigma' : \sigma \in \Sigma, \sigma' \in \Sigma'\}$, respectively.

For a compositional workflow G, we write $\Sigma(G)$ to denote the set of execution sequences for G. Then for workflow specification G comprising a single step or release point v, $\Sigma(G) = \{(v)\}$; $\Sigma(G_1; G_2) = \Sigma(G_1) + \Sigma(G_2)$; $\Sigma(G_1 \parallel G_2) = \Sigma(G_1) * \Sigma(G_2)$; and $\Sigma(G_1 \otimes G_2) = \Sigma(G_1) \cup \Sigma(G_2)$.

For an execution sequence σ, let σ_S and σ_R be the restriction of σ to the set of steps and release points, respectively. Similarly, let $S(\sigma)$ and $R(\sigma)$ be respectively the set of steps and release points contained in σ.[1] Given an execution sequence $\sigma = (v_1, \ldots, v_n)$ of G and $i \in [n]$, we define $\text{left}_\sigma(v_i) = (v_1, \ldots, v_{i-1})$, $\text{right}_\sigma(v_i) = (v_{i+1}, \ldots, v_n)$. Also, if $1 \le i < j \le n$, then define $\text{btw}_\sigma(v_i, v_j) = (v_{i+1}, \ldots, v_{j-1})$. We will omit the σ subscript from left_σ, right_σ and btw_σ when it is obvious from context.

3.2.2 *Constraints with release points.*

The model for constraints with release points described below [8] is more general than that of Basin, Burri and Karjoth [3]. Let $W = (S \cup R \cup O, E, U, A)$ be a compositional workflow schema. A *constraint with release points* has the form $c = (T, \Theta, P)$, where $T \subseteq S$ is the scope of the constraint, $P \subseteq R$ represents the release points of the constraints, and Θ is a family of functions with domain T and range U. For $Q \subseteq S$, we denote by $\Theta|_Q = \{f|_Q : f \in \Theta\}$ the restriction of the family Θ to Q.

Let σ be an execution sequence of W, and $\sigma_P = (r_1, \ldots, r_q)$ be the ordering of release points of P in σ. For every $i \in \{1, \ldots, q-1\}$, define

$$T_0 = T \cap S(\text{left}(r_1));$$
$$T_i = T \cap S(\text{btw}(r_i, r_{i+1})), \text{ for } i \in [q-1];$$
$$T_q = T \cap S(\text{right}(r_q)).$$

In other words, for $i \in [q-1]$, T_i is the set of steps of T occurring between r_i and r_{i+1} in σ.

Given a constraint $c = (T, \Theta, P)$ and an execution sequence σ, we define the *restriction* of c to T_i to be the constraint $c_i = (T_i, \Theta|_{T_i})$. (That is, a constraint with scope limited to T_i and having no release points.) We say that a plan $\pi : S(\sigma) \to U$ *satisfies* c if and only if for all $i \in \{0, \ldots, q\}$, $\pi|_{T_i}$ satisfies c_i, *i.e.* if $\pi|_{T_i} \in \Theta|_{T_i}$. Informally, a plan satisfies c if and only if its restriction to each subscope T_i, $i \in \{0, \ldots, q\}$, can be extended to a valid tuple (*i.e.* a tuple which belongs to Θ). We say σ *satisfies* c if there exists a plan $\pi : S(\sigma) \to U$ that satisfies c.

A constrained compositional workflow schema (CCWS for short) is a tuple $(G = (S \cup R \cup O, E), U, A, C)$, where (G, U, A) is a compositional workflow schema, and C is a set of constraints with release points. We assume the scope of a constraint does not contain two exclusive steps. This is a reasonable assumption since two exclusive steps never occur in the same execution sequence. We say constraint $c = (T, \Theta, P)$ is user-independent (UI) if and only if for every $\theta \in \Theta$ and every permutation $\phi : U \to U$, we have $\phi \circ \theta \in \Theta$.

3.2.3 *WSP with release points.*

Given a CCWS $W = (S \cup R \cup O, E, U, A, C)$, we say that an execution sequence σ is *satisfied* if there exists an authorized plan $\pi : S(\sigma) \to U$ that satisfies all constraints in C. (Note that authorization does not depend on the ordering of steps or release points.) We say that W is *strongly satisfiable* if and only if every execution sequence of W is satisfiable. We then define the following decision problem:

> WSP WITH RELEASE POINTS
> *Input:* A constrained compositional workflow schema $W = (S \cup R \cup O, E, U, A, C)$
> *Question:* Is W strongly satisfiable ?

Clearly WSP WITH RELEASE POINTS is a generalization of WSP (indeed, a WSP WITH RELEASE POINTS with no xor branching and whose all constraints have no release point is equivalent to a WSP instance), and is thus NP-hard and $W[1]$-hard when parameterized by $k = |S|$ [11]. Despite the seeming difficulty of the problem (since all execution sequences have to be considered), WSP WITH RELEASE POINTS is FPT parameterized by the total number of steps, release points and xor-branchings [8].

4 APPROXIMATE AND BOUNDED WORKFLOW SATISFIABILITY

One may specify authorization policies and constraints that result in a workflow specification being unsatisfiable. This is undesirable from a business perspective, since the business objective associated with the workflow cannot be achieved. Valued WSP provides a way of determining the minimum cost, in terms of violating constraints and/or the authorization policy, of a plan for the given workflow specification.

In this paper, we extend Valued WSP to CCWSs. Given a CCWS, we define for every pair (σ, π), where $\pi : S(\sigma) \to U$, a cost $w(\sigma, \pi)$. In practice, this cost will be determined by the sum of the costs of all constraint and authorization policy violation(s) incurred by the plan, as described in Section 3.1. We assume $w(\sigma, \pi)$ can be computed in time polynomial in the size of the workflow specification.

Let $f : \Sigma \to (0, 1]$ be a frequency distribution ($\sum_{\sigma \in \Sigma} f(\sigma) = 1$), where $f(\sigma)$ denotes the relative frequency of σ occurring as the set of steps in a workflow instance. The simplest case is a uniform distribution $f(\sigma) = \frac{1}{|\Sigma|}$, for all $\sigma \in \Sigma$. In this case, every execution sequence is equally likely to occur as a workflow instance. Of course, for some workflow specifications, the uniform distribution will not be appropriate and some execution sequences will be much more likely to occur than others. In this paper, we assume that

[1]Hence, the difference between σ_S and $S(\sigma)$ (resp. σ_R and $R(\sigma)$) is that the former is an ordered sequence, while the latter is a set. In particular, it might be the case, for two ordered sequences σ, σ', that, say, $S(\sigma) = S(\sigma')$ while $\sigma_S \neq \sigma'_S$, in the case where σ and σ' are two different orderings of the same set of steps.

f is distributed uniformly. Then, given an execution sequence σ and a plan $\pi : S(\sigma) \to U$, we define $\overline{w}(\sigma,\pi) = f(\sigma) \cdot w(\sigma,\pi)$ to be the *relative cost* of the pair (σ,π). We may wish to impose an upper bound on the relative cost of every execution sequence, or bound the expected cost of the workflow, or insist that a particular proportion of workflow instances will have a bounded cost. More formally, we introduce the following definition.

Definition 4.1. Let $B \geqslant 0$ denote a *budget.* We say a workflow schema has *bounded cost* if for every $\sigma \in \Sigma$, there exists a plan $\pi : S(\sigma) \to U$ such that $\overline{w}(\sigma,\pi) \leqslant \frac{B}{|\Sigma|}$; and *bounded expected cost* if for every $\sigma_i \in \Sigma$, there exists a plan $\pi_i : S(\sigma_i) \to U$ such that $\sum_{\sigma_i \in \Sigma} \overline{w}(\sigma_i,\pi_i) \leqslant B$.

In the special case $B = 0$, a workflow with bounded cost or with bounded expected cost is satisfiable. And in the special case that f is uniform, bounded cost simply means that for every execution sequence σ, there exists a plan π such that $w(\sigma,\pi) \leqslant B$. The above definitions naturally give rise to two decision problems:

Bounded Cost WSP (BC-WSP). Given a workflow specification and a budget, does the workflow specification have bounded cost?

Bounded Expected Cost WSP (BEC-WSP), Given a workflow specification and a budget, does the workflow specification have bounded expected cost?

A specification with bounded cost means that the relative cost of every execution sequence can be bounded. In the special case that f is uniform, the cost of every execution sequence can be bounded by B. A specification with bounded expected cost allows some execution sequences to exceed the budget but the cumulative cost is bounded. Such a specification allows for some very rare execution sequences whose only plans are relatively expensive, provided all the more commonly occurring plans have plans that are relatively cheap. We may also define related search problems: For a workflow specification, what is the smallest budget B for which the workflow has (i) bounded cost (ii) bounded expected cost?

Definition 4.2. Let $\Sigma_B = \{\sigma \in \Sigma : \exists\, \pi : S(\sigma) \to U, w(\sigma,\pi) \leqslant B\}$ denote the set of execution sequences for which there exists a plan with cost no greater than B. We say a workflow specification has *probability p of completing within budget* if $\sum_{\sigma \in \Sigma_B} f(\sigma) \geqslant p$.

In the simple case that f is uniform, the above definition reduces to $|\Sigma_B| \,/\, |\Sigma| \geqslant p$. Then, we may define a third decision problem called Approximate BC-WSP: Given a workflow specification, a budget B and a probability p, does the workflow have probability p of completing within budget?

5 SOLVING BOUNDED COST WSP

We now describe an algorithm to solve Bounded Cost WSP. Notice that our goal is to determine whether, for every execution sequence, there exists a complete plan with bounded cost. One naive approach would be to enumerate all execution sequences, and solve Valued WSP for each of them. We show, however, that there is a more efficient way to solve the problem. We will define an equivalence relation similar to the one defined by Crampton et al. [8] for the execution sequences, and will see that all equivalent execution sequences have the same weight.

5.1 Execution arrangements and their enumeration

We say that execution sequences σ and σ' are *equivalent* [8] if (i) $\sigma_R = \sigma'_R$, (ii) $S(\sigma) = S(\sigma')$, and (iii) for all $s \in S, R(right_\sigma(s)) = R(right_{\sigma'}(s))$. Informally, two execution sequences are in the same equivalence class when their release points are the same and occur in the same sequences, they have the same set of steps, and for every step, the set of release points occurring to the right are the same. Essentially this means that the set of steps occurring between two release points are also the same. We call each equivalence class, an *execution arrangement.*

Based on the above characterization, we now define a *compact* representation of execution arrangement similar to that used by Crampton *et al* [8]. For an equivalent class containing an execution sequence σ, an execution arrangement is an ordered sequence $(S_1, r_1, S_2, r_2, \ldots, r_{q-1}, S_q\}$ which satisfies the following properties:

(1) $\{S_1, \ldots, S_q\}$ is a partition of $S(\sigma)$. Note that we may have $S_i = \emptyset$ for some $i \in [q]$;
(2) $(r_1, \ldots, r_{q-1})$ is a linear sub-extension of G containing all release points; and
(3) for all $(s_1, \ldots, s_q) \in S_1 \times \ldots \times S_q$, $(s_1, r_1, \ldots, s_{q-1}, r_{q-1}, s_q)$ is a linear sub-extension of G.

The alert reader will notice that we have abused the notation slightly in the last property if $S_i = \emptyset$ for some $i \in [q]$. If $S_i = \emptyset$ for some $i \in [q]$, we simply omit such steps s_i in the sequence $(s_1, r_1, \ldots, r_{q-1}, s_q)$. Now, we have the following lemma.

Lemma 5.1. *Let $W = ((S \cup R \cup O, E), U, A, C)$ be a CCWS. Let σ and σ' be two execution sequences and $\sigma \sim \sigma'$. Then, for a plan $\pi : S(\sigma) \to U$, we have that $w_\sigma(\pi) = w_{\sigma'}(\pi)$.*

Proof. Let $c = (T, \Theta, R)$ be a constraint. By definition of $\sim$, we have that $\sigma_R = \sigma'_R = (r_1, \ldots, r_q)$. Now, let $i \in [q-1]$, and denote by T_i the set $T \cap S(\text{btw}_\sigma(r_i, r_{i+1}))$, and by T' the set $T \cap S(\text{btw}_{\sigma'}(r_i, r_{i+1}))$. We know by definition of $\sim$ that $R(\text{right}_\sigma(s)) = R(\text{right}_{\sigma'}(s))$ for every $s \in \sigma'$. Hence, constraint c is violated by a plan π in σ if and only if the constraint c is violated by a plan π in σ'. Also, authorization does not depend on the ordering of steps or release points. So, if a particular authorization policy is violated by π in σ, then the same authorization policy is violated by π in σ' as well. Therefore we have that $w_\sigma(\pi) = w_{\sigma'}(\pi)$. □

Suppose that we find a plan π of minimum cost for each of the execution arrangements. Then, by Lemma 5.1, this will be sufficient to find a plan for each of the execution sequences. So, we may proceed as follows. First, we enumerate the set of execution arrangements. Then for each of the execution arrangements, we find a plan of minimum cost. If all such plans are of weight bounded by B, then Bounded Cost WSP has a solution; otherwise, it has no solution. Below we will describe details of this approach. It follows from [8] that the number of execution arrangements is $(|S| + |R|)!$, and their enumeration is non-trivial. The presence of xor branching means that the set of steps and release points differ depending on the executions. We can use the approach of [8] to enumerate all execution arrangements. We provide an outline of the enumeration, rather than describing it in detail; a full description is available in [8]. The overall approach is decomposed into two parts:

(1) elimination of xor branching [8, Section 3.2]; and
(2) enumeration of all execution arrangements in an xor branching [8, Algorithm 2].

Let G be the DAG of the initial workflow instance W. After eliminating xor-branchings, we obtain a collection of workflow instances $W[G_1], W[G_2], \ldots, W[G_t]$. Then W is satisfiable if and only if for every $i \in [t]$, $W[G_i]$ is satisfiable [8, Lemma 3.2]. The reason here is that an execution of $W[G_i]$ is also an execution sequence of W. Also, for every execution sequence σ of W, there exists $i \in [t]$ such that σ is an execution sequence of $W[G_i]$. Hence, consider an arbitrary $i \in [t]$, and consider an arbitrary execution sequence σ of $W[G_i]$. If a plan π has weight $w^\star$ for σ in $W[G_i]$, then π also has weight $w^\star$ for the execution sequence σ in W since the execution sequences are same. Hence, if a plan π for an execution sequence in an xor-free instance has bounded weight, then the same plan also has the weight with same bound in the original workflow instance. Thus, it is sufficient to compute the execution arrangements for each of the xor-free instances and find if there is a plan with bounded weight for each of the execution arrangements.

The existing algorithm for enumerating execution arrangements can be used, unchanged, as it applies to workflow instances containing release points and orchestration points, and is independent of whether costs are associated with policy and constraint violations. We may also make use of the following theorem [8, Theorem 3.5].

Theorem 5.2. *For an instance of CCWS with release points, there exists an algorithm that removes all xor-branchings, and enumerates the set of all execution arrangements in time* $O^*(2^{|\mathcal{B}|}|R|!(|R|+1)^{|S|})$.

Having enumerated the execution arrangements, we must determine whether there exists a plan π with bounded cost for each of the execution arrangements. Next we consider this problem.

5.2 Reduction to Valued WSP

Now for each execution arrangement, we have to determine whether there exists a plan π with bounded cost. We show that this can be reduced to solving finitely many instances of the classical Valued WSP (stated in Section 3.1), which contains no release points or orchestration points. Note that for every execution arrangement, we want to solve Bounded Cost WSP for one execution sequence. The idea is similar to the ideas used in [8]; we describe it here for completeness, and also in order to highlight the differences.

Suppose that $\Sigma = (S_1, r_1, S_2, r_2, \ldots, r_{q-1}, S_q)$ is an execution arrangement, and $c = (T, \Theta, P)$ is a constraint with release points $P = \{r_{p_1}, \ldots, r_{p_{|P|}}\}$. We assume without loss of generality that the ordering is a linear extension of release points $R(\Sigma)$. As before, for all $i \in [P-1]$, we define $T_i = T \cap S(\text{btw}(r_{p_i}, r_{p_{i+1}}), T_0 = T \cap S(\text{left}(r_{p_0})), T_{|P|} = T \cap S(\text{right}(r_{p_{|P|}})$, and the "classical" constraint $c_i = (T_i, \Theta|_{T_i})$. We know that c is satisfied by an execution sequence σ if and only if there exists a plan π such that π_{T_i} satisfies c_i for every $i \in \{0, 1, \ldots, q\}$. Now, suppose that for a plan π, the constraint c_i is violated by π_{T_i} for some $i \in \{0, 1, \ldots, q\}$. Then, the constraint c is also violated by π. So, for each $i \in \{1, \ldots, |P|\}$, we define the classical Valued WSP instance $W_i = (G_i = (S_i, E_i), U, A_i, C_i)$ that defines the partial order restricted to S_i, $A_i = A \cap (S_i \times U)$ and $C_i = \{c_i | c \in C\}$.

For our case, we reduce it to finitely many instances of Valued WSP. Suppose that for every $i \in [|P|]$, we find the weight $w_i^\star$ of a minimum weight plan for the Valued WSP W_i. If this $w_i^\star$ meets the bound for every $i \in [|P|]$, then we say that Bounded Cost WSP is a yes-instance. Otherwise we say that Bounded Cost WSP is a no-instance. Hence, we have the following lemma.

Lemma 5.3. Σ *is a yes-instance for* Bounded Cost WSP *if and only if for every* $i \in [|P|]$, *the cost of the plan output for* W_i *is at most* B.

5.3 Running time of the algorithm

The worst case running time of the algorithm is determined by the respective running times of the algorithms for solving the following subproblems: (i) enumerating all xor-free sub-instances; (ii) given an xor-free instance, enumeration of all execution arrangements; (iii) given an execution arrangement, reduction to Valued WSP and solving Valued WSP for each of them. By Theorem 5.2, steps (i) and (ii) take $O(2^{|\mathcal{B}|}|R|!(|R|+1)^{|S|})$ time. Hence, by Lemma 5.3 and Theorems 5.2 and 3.1, we have the following:

Theorem 5.4. *Let* $\kappa = |S| + |R| + |\mathcal{B}|$. Bounded Cost WSP *parameterized by* κ *is fixed-parameter tractable, and can be solved in* $O^*(2^{|\mathcal{B}|}|R|!(|R|+1)^{|S|}|S|^{|S|})$ *time.*

Acknowledgement. The research in this paper was supported by the Leverhulme Trust award RPG-2018-161.

REFERENCES

[1] American National Standards Institute. *ANSI INCITS 359-2004 for Role Based Access Control*, 2004.
[2] Bang-Jensen, J., and Gutin, G. *Digraphs - theory, algorithms and applications.* Springer, 2002.
[3] Basin, D. A., Burri, S. J., and Karjoth, G. Obstruction-free authorization enforcement: Aligning security and business objectives. *Journal of Computer Security 22*, 5 (2014), 661–698.
[4] Cohen, D., Crampton, J., Gagarin, A., Gutin, G., and Jones, M. Iterative plan construction for the workflow satisfiability problem. *J. Artif. Intell. Res. 51* (2014), 555–577.
[5] Crampton, J., and Gutin, G. Constraint expressions and workflow satisfiability. In *Proceedings of 18th SACMAT* (2013), M. Conti, J. Vaidya, and A. Schaad, Eds., ACM, pp. 73–84.
[6] Crampton, J., Gutin, G., and Karapetyan, D. Valued workflow satisfiability problem. In *Proceedings of the 20th SACMAT* (2015), E. R. Weippl, F. Kerschbaum, and A. J. Lee, Eds., ACM, pp. 3–13.
[7] Crampton, J., Gutin, G., and Majumdar, D. Bounded and approximate strong satisfiability in workflows. *CoRR abs/1904.07234* (2019).
[8] Crampton, J., Gutin, G. Z., and Watrigant, R. On the satisfiability of workflows with release points. In *SACMAT* (2017), ACM, pp. 207–217.
[9] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms.* Springer, 2015.
[10] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. Workflow patterns. *Distributed and Parallel Databases 14*, 1 (2003), 5–51.
[11] Wang, Q., and Li, N. Satisfiability and resiliency in workflow authorization systems. *ACM Trans. Inf. Syst. Secur. 13*, 4 (2010), 40.
[12] White, S., and Miers, D. *BPMN Modeling and Reference Guide: Understanding and Using BPMN.* Future Strategies Incorporated, 2008.