Fixed-Parameter Tractability of Workflow Satisfiability in the Presence of Seniority Constraints

J. Crampton[1], R. Crowston[1], G. Gutin[1], M. Jones[1], and M.S. Ramanujan[2]

[1] Royal Holloway University of London, United Kingdom
[2] The Institute of Mathematical Sciences, Chennai, India

Abstract. The workflow satisfiability problem is concerned with determining whether it is possible to find an allocation of authorized users to the steps in a workflow in such a way that all constraints are satisfied. The problem is NP-hard in general, but is known to be fixed-parameter tractable for certain classes of constraints. The known results on fixed-parameter tractability rely on the symmetry (in some sense) of the constraints. In this paper, we provide the first results that establish fixed-parameter tractability of the satisfiability problem when the constraints are asymmetric. In particular, we introduce the notion of seniority constraints, in which the execution of steps is determined, in part, by the relative seniority of the users that perform them. Our results require new techniques, which make use of tree decompositions of the graph of the binary relation defining the constraint. Finally, we establish a lower bound for the hardness of the workflow satisfiability problem.

1 Introduction

A business process is a collection of interrelated steps that are performed in some predetermined sequence in order to achieve some objective. It is increasingly common to automate business process and for business process management systems or workflow management systems to control the execution of the steps comprising the business process. A workflow specification is an abstract representation of a collection of business steps, together with dependencies on the order in which steps should be performed. A workflow specification may be instantiated and its execution controlled by a workflow management system.

In many situations, we wish to restrict the users that can perform certain steps. On the one hand, we may wish to specify which users are authorized to perform particular steps. The workflow management system will prevent a user from performing any step for which that user is not authorized. In addition, we may wish, either because of the particular requirements of the business application or because of statutory requirements, to prevent certain combinations of users from performing particular combinations of steps. In particular, there may be pairs of steps that must be executed in any given instance of the workflow by different users, the so-called "two-man rule" (or "four-eyes rule"). Similarly,

we may require that two or more steps in any given instance are performed by the same user. These constraints are sometimes known in the literature as separation-of-duty and binding-of-duty constraints, respectively.

The existence of constraints on the execution of a workflow raises the question of whether a workflow specification can be realized in practice. As a trivial example, a workflow with two steps and the requirement that a different user performs each of the two steps cannot be realized by a user population with a single user. Therefore, it is important to be able to determine whether a workflow is satisfiable: Does there exist an allocation of authorized users to workflow steps such that every step is performed by an authorized user and are all constraints on the execution of steps satisfied?

A brute-force approach to answering the question gives rise to an algorithm that has running time $O(cn^k)$, where c is the number of constraints[1], n is the number of users and k is the number of steps. Moreover, it is known that determining the satisfiability of a workflow specification is NP-hard in general [15]. However, it has also been shown that some interesting special cases of the problem are fixed-parameter tractable, meaning that there exists an algorithm to solve them with running time $O(f(k)n^d)$, where d is some constant (independent of k and n). The existence of such an algorithm suggests that relatively efficient methods can be developed to solve interesting cases of the workflow satisfiability problem.

Wang and Li established that satisfiability is fixed-parameter tractable when we restrict attention to separation- and binding-of-duty constraints [15]. Crampton *et al.* developed a novel analysis of the problem, which reduced the complexity considerably, but retained the focus on separation- and binding-of-duty constraints [6]. In this paper, we consider a new class of constraints, in which the users that perform two steps are different and one is senior to the other. Seniority constraints are asymmetric, in contrast to separation- and binding-of-duty constraints, and this means that existing techniques for determining workflow satisfiability cannot be applied to workflow specifications that contain such constraints.

In this paper, we introduce novel techniques for determining workflow satisfiability when the specification includes seniority constraints. These techniques are based on the tree decomposition of the graph of the seniority relation and the application of dynamic programming to a particular form of tree decomposition. This enables us to establish that the workflow satisfiability problem is fixed-parameter tractable when the partial order defined over the set of users has Hasse diagram (viewed as an acyclic digraph) of bounded treewidth[2]. As we will see, many user hierarchies that arise in practice have bounded treewidth. However, our result is highly unlikely to hold for an arbitrary partial order defined over the set of users. Moreover, we show that it is impossible to obtain

[1] Here and in the rest of the paper, all constraints are binary and a constraint can be checked in constant time.

[2] We define treewidth of a graph in Sec. 3.

an algorithm for the general case of WSP with running time significantly better than $O(cn^k)$, assuming the Exponential Time Hypothesis (ETH) [11] holds.

We conclude this section by providing some terminology and notation on directed and undirected graphs. In the next section, we introduce the workflow satisfiability problem and further justify the relevance of seniority constraints. In Sec. 3, we describe tree decompositions, define treewidth and show its relevance to practical seniority constraints, and establish some elementary, preparatory results. Section 4 establishes fixed-parameter tractability of the above-mentioned "treewidth bounded" case of the problem and the following section establishes a lower bound for the complexity of the general problem (assuming ETH holds). We conclude the paper with a summary of our contributions, a discussion of the significance of our results, and some suggestions for future work.

Terminology and Notation for Graphs Let G be a directed or undirected graph and let X be a set of vertices of G. The subgraph $G[X]$ of G *induced by* X is obtained from G by deleting all vertices not in X. Let D be a directed graph. The *underlying graph* $U(D)$ is the undirected graph obtained from D by removing orientations from all arcs of D. We say that D is *connected* if $U(D)$ is connected. We say that D is *transitive* if for every pair x, y of distinct vertices, if there is a directed path from x to y then D contains an arc from x to y. We say that a directed graph H is the *transitive closure* of D if there is an arc from x to y in H whenever there is a directed path from x to y in D. The *degree* of a vertex x of D is its degree in $U(D)$. Let H be a directed or undirected graph. For a natural number ℓ, we say that H is ℓ-degenerate if $H[X]$ has a vertex of degree at most ℓ for each set of vertices X of H. As an example, consider a forest. Note that it is 1-degenerate. Let D be a digraph, Y a set of vertices of D, and y, z vertices in $D - Y$. We say that Y *separates* y *from* z if $D - Y$ has no directed path from y to z.

2 Workflow Satisfiability

Suppose we are given a workflow specification comprising a set S of k steps. A workflow constraint has the form (ρ, s', s''), where $s', s'' \in S$ and ρ is a binary relation defined over a set U of n users. For each step $s \in S$, there is a list $L(s)$ of users authorized to perform s. A function π from S to U is called a *plan*. We say that a plan π *satisfies* constraint (ρ, s', s'') if $(\pi(s'), \pi(s'')) \in \rho$.

For a set, $\{\rho_1, \ldots, \rho_t\}$, of binary relations on U, an instance $\mathcal{I}$ of the workflow satisfiability problem $\mathrm{WSP}(\rho_1, \ldots, \rho_t)$ is given by a list $L(s)$ for each $s \in S$ and a set C of constraints of the form (ρ, s', s''), where $s', s'' \in S$ and $\rho \in \{\rho_1, \ldots, \rho_t\}$; we are to decide whether there is a *valid* plan, i.e., a plan π such that the following hold:

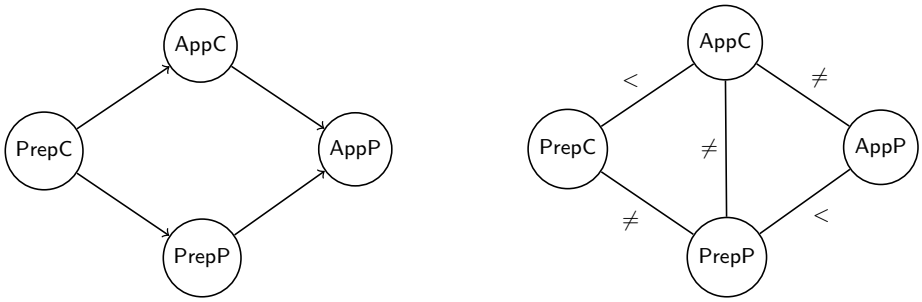
- for each $s \in S$, $\pi(s) \in L(s)$;
- π satisfies each constraint $(\rho_i, s', s'') \in C$.

If $\mathcal{I}$ has a valid plan, it is called a YES-instance. Otherwise, it is a NO-instance.

Let $<$ be a partial order on U. We will consider constraints of the form (ρ, s', s''), where ρ is one of $=$, $\neq$, $<$, and $s', s'' \in S$. A plan π satisfies:

- $(=, s', s'')$ if $\pi(s') = \pi(s'')$;
- $(\neq, s', s'')$ if $\pi(s') \neq \pi(s'')$;
- $(<, s', s'')$ if $\pi(s') < \pi(s'')$.

Consider a business process for handling expenses claims, which is illustrated in Fig. 1. Such a workflow might include four steps: the preparation of an expenses claim (PrepC), the approval of the claim (AppC), the preparation of the payment (PrepP), and the approval of the payment (AppP). We might assume that most, if not all, users in an organization are authorized to prepare an expenses claim. We require that the user who approves a claim is senior to the user who prepares a claim. Note that it would be either difficult or impractical to enforce this rule simply by restricting the users who are authorized to approve claims. (We could authorize only the most senior user to approve expenses claims, but this is unnecessarily limiting and places an onerous burden on a single individual.) Similarly, we require that the user who approves a payment be senior to the user who prepares the payment. In addition, we require that the user who prepares the expenses claim is different from the one who prepares the payment, and the user that approves the claim is different from the user who prepares the payment and from the one who approves the payment.



(a) Ordering on steps (b) Constraint graph

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

It is perhaps worth noting at this stage that the use of an access control model that incorporates some notion of seniority (role-based access control and information flow models being obvious candidates) does not necessarily enforce the desired constraints. We might assign the PrepC and AppC steps to two different roles r and r', say, with $r < r'$. However, this does not enforce the desired constraint: a user assigned to r' is indirectly assigned to r and is, therefore, authorized to perform both steps.

It is worth noting, however, that access control models do define (albeit indirectly) an ordering on the set of users. In particular, we may define $u < u'$

if the set of steps for which u is authorized is a strict subset of the set of steps for which u' is authorized. The relation $<$ is transitive. The relation $\leq$, where $u \leq u'$ if and only if $u < u'$ or $u = u'$ is transitive, reflexive and anti-symmetric; that is, $\leq$ defines a partial order on U.

We also note that many organizations have user hierarchies that define the reporting and management lines within those organizations. If such a hierarchy exists, we may evaluate our seniority constraints with respect to such a hierarchy (rather than an ordering defined by the authorization policy). In many cases, such a user hierarchy will be a rooted tree, although our results do not require this and more complex hierarchies do arise in practice. At Royal Holloway, University of London, for example, each of the three faculty Deans reports to and is managed by each of the three Vice Principals, as shown in Fig. 2. The complete bipartite subgraph within a user hierarchy that is a feature of this hierarchy also arises in the (graphs of the) relations of the preorders that are obtained from an authorization policy: each user in the set of users authorized for $S' \subseteq S$ is senior to each user in the set of users authorized for $S'' \subset S'$.

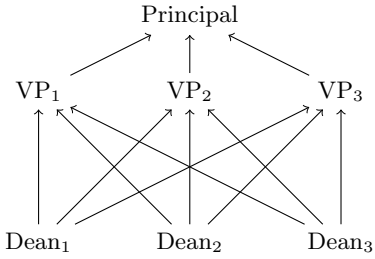


Fig. 2. Part of the user hierarchy at Royal Holloway

2.1 Constraint Graphs

Given a partial (irreflexive) order $<$ on U, let H be the transitive acyclic graph with vertex set U such that $u < v$ if and only if there is an arc from u to v in H. We say H is the *full graph* of $(U, <)$. Let D be an directed acyclic graph such that H is the transitive closure of D and the transitive closure of every subgraph $D - a$, where a is an arc of D, is not equal to H. Note that since H is acyclic, D is unique [1] (see also Sec. 2.3 of [3]). We say that D is the *reduced graph* (or *Hasse diagram*) of $(U, <)$.

A *mixed graph* consists of a set of vertices together with a set of undirected edges and a set of directed arcs. We may represent the set of constraints with a mixed graph as follows.

First, we eliminate constraints of the form $(=, s', s'')$. Specifically, we construct a graph P with vertices S in which $s', s'' \in S$ are adjacent if $\mathcal{I}$ has a constraint $(=, s', s'')$. Observe that the same user must necessarily be assigned to all

steps in a connected component Q of P. Thus, if there is a pair $s', s'' \in Q$ such that $\mathcal{I}$ has a constraint $(\neq, s', s'')$ or $(<, s', s'')$, then clearly $\mathcal{I}$ is a NO-instance; thus we may assume that there is no such pair for any connected component of H. For each connected component Q of P, replace all steps of Q in S by a "superstep" q. A user u is authorized to perform q if u is authorized to perform all steps of Q. That is, $L(q) = \bigcap_{s \in Q} L(s)$.

The above procedure eliminates all constraints of the type $(=, s', s'')$ for the reduced set S of steps. All constraints of the types $(\neq, s', s'')$ and $(<, s', s'')$ remain, but steps s' and s'' are replaced by the corresponding supersteps. For simplicity of notation, we will denote the new instance of the problem also by $\mathcal{I}$.

Now we construct a mixed graph with vertex set S. For each constraint of the type $(\neq, s', s'')$, add an *edge* between s' and s''. For each constraint of the type $(<, s', s'')$, add an *arc* from s' and s''. We will refer to the resulting graph as the *constraint graph* (of $\mathcal{I}$). We will say an edge or arc in a constraint graph is *satisfied* by a plan π if π satisfies the corresponding constraint.

It is worth noting that WSP($\neq$) is rather closely related to graph colorability, where the assignment of users to tasks in such a way that separation-of-duty constraints are satisfied provides a coloring of the constraint graph and vice versa[3]. Note that the selection of a color for step s in the constraint graph prevents the use of only one color for steps connected by an edge to s. WSP($<, \neq$) is an even more complex problem because it imposes a structure on the set of colors that are available, meaning that the selection of a color for s may preclude the use of many other colors for steps connected to s by an arc.

Consider, for example, an organization with three users – Alice, Bob and Carol, where Alice is senior to Bob and Carol and all three users are authorized for all tasks. Then, our expenses claim workflow is not satisfiable. However, the workflow specification is satisfiable if we replace the seniority constraints with separation-of-duty constraints.

2.2 Related Work

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[4].

Wang and Li initiated the study of the fixed-parameter tractability of workflow satisfiability [15]. They showed that the problem is W[1]-hard, in general, which implies that it is not FPT (unless the parameterized complexity hypothesis FPT $\neq$ W[1] fails, which is believed to be highly unlikely). However, they were able to show that WSP($=, \neq$) is FPT.

[3] In fact, WSP($\neq$) is equivalent to the more general problem List Coloring, as the list $L(s)$ imposes restrictions on the "colors" (users) that can be assigned to step s.

[4] For more information on parameterized algorithms and complexity, see monographs [9,10,14].

Crampton *et al.* introduced new techniques for analyzing WSP$(=, \neq)$ and significantly improved the complexity of FPT algorithms to solve the problem [6]. The approach of Crampton *et al.* is based on partitioning the set of steps and, for each block of steps in the partition, assigning a user to that block, where the user was authorized for each step in the block. The existence of such a partition and allocation of users to blocks demonstrates that a workflow specification is satisfiable. This method assumes that the allocation of a user to one particular block is independent of the allocation of users to other blocks: this assumption holds for separation- and binding-of-duty constraints; however, it does not hold for seniority constraints because the choice of a senior user for one block may limit the choices of user available for other blocks.

Constraints of the form (ρ, s', s'') have been called Type 1 constraints [6], and were formally introduced by Crampton [8]. Wang and Li introduced Type 2 constraints [15], which have the form (ρ, s', S'), where $S' \subseteq S$ and the constraint is satisfied by plan π if there exists $s'' \in S'$ such that $(\pi(s'), \pi(s'')) \in \rho$. Finally, Crampton *et al.* defined Type 3 constraints [6], which have the form (ρ, S', S''), where $S', S'' \subseteq S$ and the constraint is satisfied if there exist $s' \in S'$ and $s'' \in S''$ such that $(\pi(s'), \pi(s'')) \in \rho$.

Crampton *et al.* [7] showed that it is possible to rewrite a workflow specification containing Type 2 or Type 3 constraints as a collection of workflow specifications, each containing Type 1 constraints only. Moreover, the number of workflow specifications is determined by k (the number of steps) only, which means that the existence of an FPT algorithm for Type 1 constraints can be used to establish the existence of an FPT algorithm for specifications containing any combination of Type 1, 2 or 3 constraints. In this paper, we demonstrate the existence of an FPT algorithm for Type 1 constraints containing the $<$ relation provided the reduced graph D is of bounded treewidth. The prior work of Crampton *et al.* [7] enables us to construct an FPT algorithm for Type 2 and 3 constraints.

3 Tree Decompositions and Treewidth

Tree decompositions provide a means of representing a (directed) graph using a tree. Subsets of the graph's vertices form the nodes of the tree, in such a way that a subtree containing a particular vertex is connected and the subtrees associated with the end-points of an edge in the graph have nonempty intersection. The treewidth of a graph G is a measure of the minimum number of vertices that are required in each node of a tree in order to construct a tree decomposition of G. Treewidth is known to be an important parameter when considering the complexity of graph-related problems that are NP-hard in general. As we will see, treewidth plays an important role in the complexity of the workflow satisfiability problem when we define a transitive relation $<$ on U and define workflow constraints in terms of $<$.

Definition 1. *A* tree decomposition *of a (directed) graph $G = (V, E)$ is a pair $(\mathcal{T}, \mathcal{X})$, where $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is a tree and $\mathcal{X} = \{\mathcal{B}_i : i \in V_\mathcal{T}\}$ is a collection of subsets of V called* bags*, such that*

1. $\bigcup_{i \in V_\mathcal{T}} \mathcal{B}_i = V$.
2. *For every edge (arc) $xy \in E$, there exists $i \in V_\mathcal{T}$ such that $\{x, y\} \subseteq \mathcal{B}_i$.*
3. *For every $x \in V$, the set $\{i : x \in \mathcal{B}_i\}$ induces a connected subtree of $\mathcal{T}$.*

The width *of $(\mathcal{T}, \mathcal{X})$ is $\max_{i \in V_\mathcal{T}} |\mathcal{B}_i| - 1$. The* treewidth *of a graph G is the minimum width of all tree decompositions of G.*

To distinguish between vertices of G and $\mathcal{T}$, we call vertices of $\mathcal{T}$ *nodes*. We will often speak of a bag $\mathcal{B}$ interchangeably with the node it corresponds to in $\mathcal{T}$. Thus, for example, we might say two bags $\mathcal{B}, \mathcal{B}'$ are neighbors if they correspond to nodes in $\mathcal{T}$ which are neighbors. We define the *descendants* of a bag $\mathcal{B}$ as follows: every child of $\mathcal{B}$ is a descendant of $\mathcal{B}$, and every child of a descendant of $\mathcal{B}$ is a descendant of $\mathcal{B}$. At the same time, we will say $\mathcal{B} = \mathcal{B}'$ if $\mathcal{B}, \mathcal{B}'$ contain the same vertices, while still treating them as different bags.

It is well-known that a connected graph is of treewidth 1 if and only if it is a tree with at least one edge [12]. Every tree T with at least one edge has the following tree decomposition $\mathcal{T}$ of width 1: for every vertex x of T let $\{x\}$ be a bag of $\mathcal{T}$ and for every edge xy of T let $\{x, y\}$ be a bag of $\mathcal{T}$. Two bags are adjacent in $\mathcal{T}$ if one of them is a proper subset of the other. For the graph depicted in Fig. 1 (b) there is a tree decomposition of width 2: it has two bags $\{\mathsf{AppC}, \mathsf{PrepC}, \mathsf{PrepP}\}$ and $\{\mathsf{AppC}, \mathsf{AppP}, \mathsf{PrepP}\}$ connected by an edge.

The graph of Fig. 2 has a tree decomposition of width 3, as shown in Fig. 3. The graph of Fig. 2 can be extended as follows to more fully reflect the Royal Holloway management hierarchy. Each faculty at Royal Holloway has several academic departments each led by Head of Department (HoD) and so we may add HoD's, each with an arc to the corresponding Dean, and non-HoD members of staff, each with an arc to the corresponding HoD. This extension of the graph of Fig. 2 essentially adds just trees to the graph and it is not hard to check that the treewidth of the extended graph is still 3.

The Royal Holloway management hierarchy is not exceptional in the following sense: it is unlikely that a member of staff will have many line managers (quite often there is only one line manager). Thus, it does not seem unreasonable to expect the reduced graph of the corresponding partial order to have bounded treewidth and for the treewidth to be rather small. Moreover, our Royal Holloway example indicates that construction of (near-)optimal tree decompositions for such hierarchies may be not hard.

It is NP-complete to decide whether the treewidth of a graph G is at most r (when r is part of input) [2]. Bodlaender [4] obtained an algorithm with running time $O(f(r)n)$ for deciding whether the treewidth of a graph G is at most r, where n is the number of vertices in G and f is a function depending only on r. This algorithm constructs the corresponding tree decomposition with $O(n)$ nodes, if the answer is YES. Unfortunately, f grows too fast to be of practical

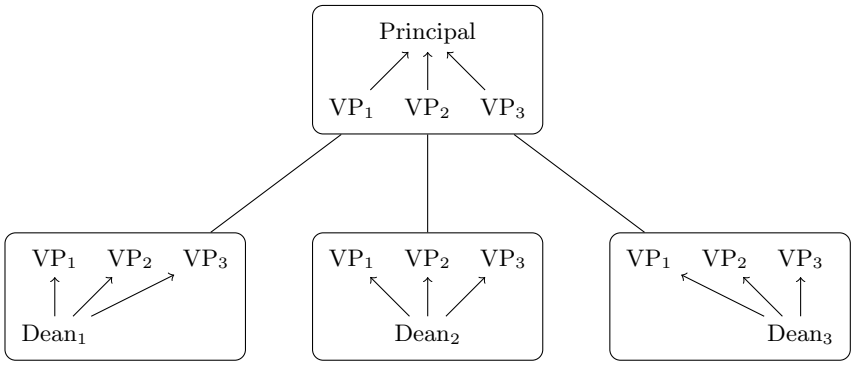


Fig. 3. Tree Decomposition of Royal Holloway management hierarchy

interest. However, there are several polynomial-time approximation algorithms and heuristics for computing the treewidth of a graph and the corresponding tree decomposition, see, e.g., [5].

We now describe a special type of tree decomposition that is widely used to construct dynamic programming algorithms for solving problems on graphs, called a *nice tree decomposition*. In a nice tree decomposition, one node in $\mathcal{T}$ is considered to be the root of $\mathcal{T}$, and each node $i \in V_{\mathcal{T}}$ is of one of the following four types:

1. a *join node* $\mathcal{B}$ has two children $\mathcal{B}'$ and $\mathcal{B}''$, with $\mathcal{B} = \mathcal{B}' = \mathcal{B}''$;
2. a *forget node* $\mathcal{B}$ has one child $\mathcal{B}'$, and there exists $u \in \mathcal{B}'$ such that $\mathcal{B} = \mathcal{B}' \setminus \{u\}$;
3. an *introduce node* $\mathcal{B}$ has one child $\mathcal{B}'$, and there exists $u \notin \mathcal{B}'$ such that $\mathcal{B} = \mathcal{B}' \cup \{u\}$;
4. a *leaf node* $\mathcal{B}$ is a leaf of $\mathcal{T}$.

The following useful lemma, concerning the construction of a nice tree decomposition from a given tree decomposition, was proved by Kloks [12, Lemma 13.1.3].

Lemma 1. *Given a tree decomposition with $O(n)$ nodes of a graph G with n vertices, we can construct, in time $O(n)$, a nice tree decomposition of G of the same width and with at most $4n$ nodes.*

Lemma 2. *Let D be a (directed) graph, $(\mathcal{T}, \mathcal{X})$ a tree decomposition of D, and let Y be a set of vertices in D such that $D[Y]$ is connected. Then the set of bags containing vertices in Y induces a connected subtree in $\mathcal{T}$.*

Proof. The proof is by induction on $|Y|$. The base case, $|Y| = 1$, follows from Definition 1. Let $y \in Y$ such that $D[Y \setminus \{y\}]$ is connected and suppose that the set of bags containing vertices in $Y \setminus \{y\}$ induces a connected subtree $\mathcal{T}'$ of $\mathcal{T}$. Let $z \in Y$ such that yz is an edge of D. By Definition 1, y and z belong to the

same bag $\mathcal{B}$ and observe that $\mathcal{B}$ is in $\mathcal{T}'$. Thus, the subtree of $\mathcal{T}$ induced by the bags containing y and $\mathcal{T}'$ intersect and so the set of bags containing vertices in Y induces a connected subtree in $\mathcal{T}$. □

Lemma 3. *Let D be the reduced graph for $(U, <)$. Let u, v be users and $\mathcal{B}$ a set of users such that $u \neq v$ and $u, v \notin \mathcal{B}$, and $\mathcal{B}$ separates u from v in D. Then $u < v$ if and only if there exists $w \in \mathcal{B}$ such that $u < w$ and $w < v$.*

Proof. By transitivity, if $u < w < v$ then $u < v$. For the other direction, suppose $u < v$. Then by the definition of D there must exist a directed path from u to v in D. Since $\mathcal{B}$ separates u and v, this path must contain a user w in $\mathcal{B}$. Therefore $u < w$ and $w < v$. □

4 FPT Algorithm for Bounded Treewidth

In this section, we consider the special case of the problem when the reduced graph D of $(U, <)$ is of bounded treewidth. In other words, in this section, we assume that the treewidth of D is bounded by a constant r. Note that D may have much smaller treewidth than the full graph H. For example, when $<$ is a linear order on U, then H is a tournament with treewidth $|U| - 1$, but D is a directed path with treewidth 1.

Theorem 1. *Let $\mathcal{I}$ be an instance of* WSP$(=, \neq, <)$ *and let D be the reduced graph of $(U, <)$. Given a tree decomposition of D of treewidth r and with $O(n)$ nodes, we can solve $\mathcal{I}$ in time $O(nk4^k(r + 2 + 3^{r+1})^k)$, where k is the number of steps and n is the number of users.*

By Lemma 1, assume we have a nice tree decomposition $(\mathcal{T}, \mathcal{X})$ of D of width r and with at most $4n$ nodes. Henceforth, we assume that we have constructed a nice tree decomposition for the instance $\mathcal{I}$.

Before proving the above result, we provide an informal insight into our approach. Dynamic programming is a well known technique that is used to solve a problem by systematically solving subproblems, each of which may contribute to the solution of other (typically larger or more complex) subproblems. For example, one might solve all subproblems of size i, and use these to solve all subproblems of size $i + 1$, or one might make use of structural graph properties, such as tree decompositions.

In the case of WSP$(=, \neq, <)$ we use dynamic programming techniques to compute solutions to restricted instances of the original problem instance, and for each of these restricted instances, we construct possible intermediate solutions for each bag in the nice tree decomposition. Working from the leaves of the decomposition back to the root, we extend intermediate solutions for child nodes to an intermediate solution for the parent node. The existence of an intermediate solution for the root node, implies the existence of a solution for the original problem instance (Lemma 4). Then, in Lemma 5, we establish the complexity of computing an intermediate solution, thereby completing the proof of Theorem 1.

Roughly speaking, for every subset T of the set S of steps, each bag $\mathcal{B}$ in the tree decomposition of D, and each step x in T, we keep track of which user in $\mathcal{B}$, if any, x is to be assigned to, and otherwise what relation the user assigned to x should have to the users in $\mathcal{B}$. Before proceeding further, we introduce some definitions and notation.

Let us say that $u > v$ if $v < u$, and $u \sim v$ if neither $u < v$ nor $v < u$. Define the *relation* of v to u, a function $\phi(v, u)$ from the set of all pairs of users to the set of three symbols $[<], [>], [\sim]$, as follows:

$$\phi(v,u) = \begin{cases} [<] \text{ if } v < u \\ [>] \text{ if } v > u \\ [\sim] \text{ if } v \sim u. \end{cases}$$

For each bag $\mathcal{B} = \{u_1, u_2, \ldots, u_p\}$ in $\mathcal{X}$, and each user $v \notin \mathcal{B}$, define the *relation* of v to $\mathcal{B}$, $\mathcal{R}(v, \mathcal{B})$ to be the ordered tuple $(\phi(v, u_1), \ldots, \phi(v, u_p))$.

Definition 2. *Given a workflow instance $\mathcal{I}$ with constraint graph $G = (S, E)$, a bag $\mathcal{B}$ in the nice tree decomposition of $(U, <)$, a set of steps T and a function $R : T \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$, we say $\pi : T \to U$ is a* $(\mathcal{B}, T, R)$-plan *if the following conditions are satisfied:*

1. *$\pi(x) \in L(x)$ for each $x \in T$;*
2. *if there is an edge between x and y in $G[T]$, then $\pi(x) \neq \pi(y)$;*
3. *if there is an arc from x to y in $G[T]$, then $\pi(x) < \pi(y)$;*
4. *for each step x, $\pi(x)$ is either a user in $\mathcal{B}$ or a user in a descendant of $\mathcal{B}$;*
5. *for any $x \in T$, $u \in \mathcal{B}$, $\pi(x) = u$ if and only if $R(x) = u$;*
6. *if $R(x) \notin \mathcal{B}$, then $\mathcal{R}(\pi(x), \mathcal{B}) = R(x)$.*

R provides a partial allocation of users in $\mathcal{B}$ to steps in T; where no user is allocated, R identifies the relationships that must hold between the user that is subsequently allocated to the task and those users in $\mathcal{B}$. The existence of a $(\mathcal{B}, T, R)$-plan means that we can extend R to a full plan π by traversing the nice tree decomposition.

We may now define the function that is central to our dynamic programming approach. For every bag $\mathcal{B}$ in the tree decomposition of D, every subset T of S, and every possible function $R : T \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$, define $F(\mathcal{B}, T, R) =$ TRUE if there exists a $(\mathcal{B}, T, R)$-plan and FALSE otherwise.

Lemma 4. *Let $\mathcal{B}_0$ be the root node in the nice tree decomposition of D. Then $\mathcal{I}$ is a* YES*-instance if and only if there exists a function $R : S \to \mathcal{B}_0 \cup \{[<], [>], [\sim]\}^{|\mathcal{B}_0|}$ such that $F(\mathcal{B}_0, S, R) =$ TRUE.*

Proof. By the first three conditions on $F(\mathcal{B}_0, S, R)$ being TRUE and the definition of the constraint graph G, it is clear that if $F(\mathcal{B}_0, S, R) =$ TRUE for some R then we have a YES-instance. So now suppose $\mathcal{I}$ is a YES-instance, and let $\pi : S \to U$ be a valid plan. Then for each $x \in S$, let $R(x) = \pi(x)$ if $\pi(x) \in \mathcal{B}_0$, and otherwise, let $R(x) = \mathcal{R}(\pi(x), \mathcal{B})$. Then observe that all the conditions on $F(\mathcal{B}_0, S, R)$ being TRUE are satisfied and therefore $F(\mathcal{B}_0, V, R) =$ TRUE.

Lemma 5. *We can compute $F(\mathcal{B}, T, R)$ for every bag $\mathcal{B}$ in $\mathcal{X}$, every $T \subseteq S$, and every $R : T \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$ in time $O(nk4^k(r + 2 + 3^{r+1})^k)$.*

Proof. We will start by constructing, in advance, a matrix $\mathcal{L} = [L_{s,u}]_{s \in S, u \in U}$ such that $L_{s,u} = 1$ if $u \in L(s)$ and $L_{s,u} = 0$, otherwise. This will take time $O(kn)$. Let $\mathcal{B}$ be in $\mathcal{X}$, T a subset of S, and R a function from T to $\mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$. Recall that every bag $\mathcal{B}$ is either a leaf node, a join node, a forget node or an introduce node. We will consider the four possibilities separately.

$\mathcal{B}$ is a leaf node. Since $\mathcal{B}$ has no descendants, $F(\mathcal{B}, T, R) =$ False unless $R(x) \in \mathcal{B}$ for every $x \in T$. So now we may assume $R(x) \in \mathcal{B}$ for all x. But then the only possibility for a $(\mathcal{B}, T, R)$-plan is one in which $\pi(x) = R(x)$ for all x. Therefore we may check, in time $O(k^2)$, whether this plan satisfies the (first three) conditions on $F(\mathcal{B}, T, R)$ being True. (Use matrix $\mathcal{L}$ to check that $\pi(x) \in L(x)$ for all $x \in T$.) If it does, $F(\mathcal{B}, T, R) =$ True, otherwise $F(\mathcal{B}, T, R) =$ False.

For the remaining cases, we may assume that $F(\mathcal{B}', T, R)$ has been calculated for every child of $\mathcal{B}'$ of $\mathcal{B}$ and every possible T, R.

$\mathcal{B}$ is a forget node. Let $\mathcal{B}' = \{u_1, u_2, \ldots, u_p\}$ be the child node of $\mathcal{B}$ and assume without loss of generality that $\mathcal{B} = \{u_1, u_2, \ldots, u_{p-1}\}$. For $i \in [p-1]$, let X_i be the set of steps in T with $R(x) = u_i$.

Suppose that π is a $(\mathcal{B}, T, R)$-plan. Then let $R' : T \to \mathcal{B}' \cup \{[<], [>], [\sim]\}^{|\mathcal{B}'|}$ be the function such that $R'(x) = \pi(x)$ if $\pi(x) \in \mathcal{B}'$, and $R'(x) = \mathcal{R}(\pi(x), \mathcal{B}')$ if $\pi(x) \notin \mathcal{B}'$. It is clear that $F(\mathcal{B}', T, R') =$ True. Now we show some properties of R.

Firstly, since π is a $(\mathcal{B}, T, R)$-plan, it must be the case that $\pi(x) = R(x)$ if $R(x) \in \mathcal{B}$ and therefore $R'(x) = R(x)$ if $R(x) \in \mathcal{B}$. Secondly, since π is a $(\mathcal{B}, T, R)$-plan and $u_p \notin \mathcal{B}$, it must be the case that $\pi(x) = u_p$ only if $R(x) = \mathcal{R}(u_p, \mathcal{B})$. Therefore $R'(x) = u_p$ only if $R(x) = \mathcal{R}(u_p, \mathcal{B})$. Finally, for $x \in T$ with $R'(x) \notin \mathcal{B}'$, let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_{p-1}})$ and let $R'(x) = (x'_{u_1}, x'_{u_2}, \ldots, x'_{u_p})$. Since π is a $(\mathcal{B}, T, R)$-plan and a $(\mathcal{B}', T, R')$-plan, we must have that $x_{u_i} = \phi(\pi(x), u_i) = x'_{u_i}$ for all $i \in [p-1]$. That is $R(x)$ and $R'(x)$ are the same except that $R'(x)$ has the extra co-ordinate x'_{u_p}. It follows that to obtain R' from R, we merely need to guess which x with $R(x) = \mathcal{R}(u_p, \mathcal{B})$ are assigned to u_p by R', and for all other x, what the value of x_{u_p} should be.

Therefore, in order to calculate $F(\mathcal{B}, T, R)$, we may do the following: Try every possible way of partitioning $T \setminus (X_1 \cup X_2 \cup \cdots \cup X_{p-1})$ into four sets $X_p, X_<, X_>, X_\sim$, subject to the constraint that $x \in X_p$ only if $R(x) = \mathcal{R}(u_p, \mathcal{B})$. For each such partition, construct a function $R' : T \to \mathcal{B}' \cup \{[<], [>], [\sim]\}^{|\mathcal{B}'|}$ such that

1. $R'(x) = R(x)$ if $R(x) \in \mathcal{B}$.
2. $R'(x) = u_p$ if $x \in X_p$.
3. For all other x, let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_{p-1}})$. Then $R'(x) = (x'_{u_1}, x'_{u_2}, \ldots, x'_{u_p})$, where $x'_{u_i} = x_{u_i}$ for all $i \in [p-1]$, and $x'_{u_p} = [<]$ if $x \in X_<$, $x'_{u_p} = [>]$ if $x \in X_>$, and $x'_{u_p} = [\sim]$ if $x \in X_\sim$.

and check the value of $F(\mathcal{B}', T, R')$.

By the above argument, if $F(\mathcal{B}, T, R) = \textsc{True}$ then it must be the case that $F(\mathcal{B}', T, R') = \textsc{True}$ for one of the R' constructed in this way. Therefore if $F(\mathcal{B}', T, R') = \textsc{False}$ for all such R', we know that $F(\mathcal{B}, T, R) = \textsc{False}$. Otherwise, if $F(\mathcal{B}', T, R') = \textsc{True}$ for some R', let π be a $(\mathcal{B}', T, R')$-plan, and observe that by construction of R' and X_p, π is a $(\mathcal{B}, T, R)$-plan as well. Therefore $F(\mathcal{B}, T, R) = \textsc{True}$.

Finally, observe that there are at most 4^k possible values of R' to check and each R' can be constructed in time $O(k)$, and therefore we can calculate $F(\mathcal{B}, T, R)$ in time $O(k4^k)$.

$\mathcal{B}$ is an introduce node. Let $\mathcal{B} = \{u_1, u_2, \ldots, u_p\}$, let $\mathcal{B}'$ be the child node of $\mathcal{B}$ and assume without loss of generality that $\mathcal{B}' = \{u_1, u_2, \ldots, u_{p-1}\}$. Let $X_p \subseteq T$ be the set of all $x \in T$ with $R(x) = u_p$, and let $T' = T \setminus X_p$. Define a function $R' : T' \to \mathcal{B}' \cup \{[<], [>], [\sim]\}^{|\mathcal{B}'|}$ as follows:

1. $R'(x) = R(x)$ if $R(x) \in \mathcal{B}'$.
2. For all other x, let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_p})$. Then set $R'(x) = (x'_{u_1}, x'_{u_2}, \ldots, x'_{u_{p-1}})$, where $x'_{u_i} = x_{u_i}$ for all $i \in [p-1]$.

We will now give eight conditions which are necessary for $F(\mathcal{B}, T, R) = \textsc{True}$. We will then show that these conditions collectively are sufficient for $F(\mathcal{B}, T, R) = \textsc{True}$. Since each of these conditions can be checked in time $O(k^2)$, we will have that $F(\mathcal{B}, T, R)$ can be calculated in time $O(k^2)$.

Condition 1: $L_{x,u_p} = 1$ for each $x \in X_p$. This condition is clearly necessary, as for every $(\mathcal{B}, T, R)$-plan π we have $\pi(x) = u_p$.

Condition 2: X_p is an independent set in G. Since in any $(\mathcal{B}, T, R)$-plan, all steps in X_p must be assigned the same user, any arc or edge between steps in X_p will not be satisfied.

Condition 3: If there exists $x \in X_p$, $y \notin X_p$ with an arc from y to x in G, then either $R(y) = u_i$ for some $u_i \in \mathcal{B}'$ with $u_i < u_p$, or $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [<]$. For if not, then any $(\mathcal{B}, T, R)$-plan will assign y to a user v such that $v > u_p$ or $v \sim u_p$, and the arc yx will not be satisfied.

Condition 4: If there exists $x \in X_p$, $y \notin X_p$ with an arc from x to y in G, then either $R(y) = u_i$ for some $u_i \in \mathcal{B}'$ with $u_i > u_p$, or $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $u_p = [>]$. The proof is similar to the proof of Condition 3.

Condition 5: If there exists $y \notin X_p$ such that $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [<]$, then there must exist $u_i \in \mathcal{B}'$ with $y_{u_i} = [<]$ and $u_i < u_p$. For suppose there is a $(\mathcal{B}, T, R)$-plan π, and let $v = \pi(y)$. Note that v must be in a descendant of $\mathcal{B}$ but not in $\mathcal{B}'$. Therefore $\mathcal{B}'$ separates v from u_p in D, for any v in a descendant of $\mathcal{B}$. (This follows from Lemma 2 where Y is the vertices of a path between v and u_p). Then by Lemma 3, as $v < u_p$ there exists $u_i \in \mathcal{B}'$ with $v < u_i < u_p$. Therefore $y_{v_i} = [<]$.

Condition 6: If there exists $y \notin X_p$ such that $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [>]$, then there must exist $u_i \in \mathcal{B}'$ with $y_{u_i} = [>]$ and $u_i > u_p$. The proof is similar to the proof of Condition 5.

Condition 7: If there exists $y \notin X_p$ such that $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [\sim]$, then there is no $u_i \in \mathcal{B}$ such that $y_{u_i} = [<]$ and $u_i < u_p$, or $y_{u_i} = [>]$ and $u_i > u_p$. For suppose there is a $(\mathcal{B}, T, R)$-plan π, and let $v = \pi(y)$. Suppose for a contradiction that there exists $u_i \in \mathcal{B}$ such that $y_{u_i} = [>]$ and $u_i > u_p$. (The case $y_{u_i} = [<]$ and $u_i < u_p$ is handled similarly). Then $v > u_i$ and so by transitivity, $v > u_p$. But this is a contradiction as $y_{u_p} = [\sim]$.

Condition 8: $F(\mathcal{B}', T', R') =$ True. For suppose π is a $(\mathcal{B}, T, R)$-plan. Then observe that by construction of R', π restricted to T' is a $(\mathcal{B}', T', R')$-plan.

It now remains to show that if Conditions 1-8 hold then $(\mathcal{B}, T, R) =$ True. Let π' be a $(\mathcal{B}', T', R')$-plan whose existence is guaranteed by Condition 8, and let π be the extension of π' to T in which $\pi(x) = u_p$ for all $x \in X_p = T \setminus T'$. Since π' is a $(\mathcal{B}', T', R')$-plan, $\pi(x) \in L(x)$ for all $x \in T'$, and by Condition 1, $\pi(x) \in L(x)$ for all $x \in X_p$. For every x with $R(x) \in \mathcal{B}$, we have that $\pi(x) = R(x)$ by the fact that π' is a $(\mathcal{B}', T', R')$-plan and $R(x) = u_p$ for all $x \in X_p$.

Now consider x with $R(x) \notin \mathcal{B}$. Then let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_p})$. By construction of R' and the fact that π' is a $(\mathcal{B}', T', R')$-plan, $\phi(\pi(x), u_i) = x_{u_i}$ for $i \in [p-1]$. Suppose $x_{u_p} = [<]$. Then by Condition 5, there exists $u_i \in \mathcal{B}'$ with $x_{u_i} = [<]$ and $u_i < u_p$. Therefore $\pi(x) < u_i$ and so $\pi(x) < u_p$. Therefore $\phi(\pi(x), u_i) = [<]$. Similarly, using Condition 6, if $x_{u_p} = [>]$ then $\phi(\pi(x), \mathcal{B}) = [>]$. If $\phi(\pi(x), u_i) = [\sim]$ then by Condition 7 there is no $u_i \in \mathcal{B}'$ with $\pi(x) > u_i > u_p$ or $\pi(x) < u_i < u_p$. Then by Lemma 3, $\pi(x) \sim u_p$ and so $\phi(\pi(x), u_p) = [\sim]$. In each case we have that $\phi(\pi(x), u_p) = x_{u_p}$ and so $\mathcal{R}(\pi(x), \mathcal{B}) = R(x)$.

It is clear that for each step x, $\pi(x)$ is either in $\mathcal{B}$ or in a descendant of $\mathcal{B}$. It remains to show that the arcs and edges in $G[T]$ are satisfied by π.

As π' is a $(\mathcal{B}', T', R')$-plan, every arc and edge in $G[T']$ is satisfied by π. By Condition 2 there are no edges and arcs within $G[X_p]$. It remains to show that the arcs and edges between X_p and T' are satisfied by π. Consider an edge between $x \in X_p$ and $y \in T'$. Since $\pi(x) = u_p$, and $\pi(y) \neq u_p$ (since u_p does not appear in $\mathcal{B}'$ or any descendant of $\mathcal{B}'$ by definition of a tree decomposition), this edge is satisfied. Now suppose there is an arc from $y \in T'$ to $x \in X_p$. By Condition 3, either $\pi(y) = R(y) = u_i$ with $u_i < u_p$, or $y_{u_p} = [<]$, in which case $\pi(y) < u_p$ (as we have shown $\phi(\pi(y, u_p) = y_{u_p})$. In either case $\pi(y) < \pi(x)$ and so the arc is satisfied. Similarly, if there is an arc from $x \in X_p$ to $y \in S'$, then by Condition 4 $\pi(y) > \pi(x)$ and the arc is satisfied.

Thus π satisfies all the conditions of a $(\mathcal{B}, T, R)$-plan and so $F(\mathcal{B}, T, R) =$ True.

$\mathcal{B}$ is a join node. Let $\mathcal{B}', \mathcal{B}''$ be the two child nodes of $\mathcal{B}$, and recall that $\mathcal{B}'$ and $\mathcal{B}''$ contain the same users as $\mathcal{B}$. Let X be the set of all $x \in T$ with $R(x) \in \mathcal{B}$.

Let π be a $(\mathcal{B}, T, R)$-plan. Then let X' be the set of all $x \in T \setminus X$ such that $\pi(x) = v$ for some v in a descendant of $\mathcal{B}'$, and let X'' be the set of all $x \in T \setminus X$ such that $\pi(x) = v$ for some v in a descendant of $\mathcal{B}''$. (Observe that X, X', X'' is a partition of T.) Let $T' = X \cup X'$ and let R' be the function R restricted to T'. Similarly let $T'' = X \cup X''$ and let R'' be the function R restricted to T''. Then observe that $F(\mathcal{B}', T', R') =$ True and $F(\mathcal{B}'', T'', R'') =$ True.

Now consider an arc from $x \in X'$ to $y \in X''$. Then $\pi(x) < \pi(y)$. Since $\mathcal{B}$ separates $\pi(x)$ from $\pi(y)$ (by Lemma 2 with Y the set of vertices on a path between $\pi(x)$ and $\pi(y)$), there must exist $u_i \in \mathcal{B}$ such that $\pi(x) < u_i < \pi(y)$. Therefore $x_{u_i} = [<]$ and $y_{u_i} = [>]$. Similarly, if there is an arc from $y \in X''$ to $x \in X'$ then there exists $u_i \in \mathcal{B}$ with $x_{u_i} = [<]$ and $y_{u_i} = [>]$.

We therefore have that if $F(\mathcal{B}, T, R) = \textsc{True}$, then there exists a partition X', X'' of $T \setminus X$ such that $F(\mathcal{B}', T', R') = \textsc{True}$ and $F(\mathcal{B}'', T'', R'') = \textsc{True}$ (where T', T'', R', R'' are as previously defined) and for any arc from $x \in X'$ to $y \in X''$, there exists $u_i \in \mathcal{B}$ with $x_{u_i} = [<]$ and $y_{u_i} = [>]$ (and similarly for arcs from $y \in X''$ to $x \in X'$). We now show that the converse is true.

Suppose these conditions hold, and let π' be a $(\mathcal{B}', T', R')$-plan and π'' a $(\mathcal{B}'', T'', R'')$-plan. Note that for all $x \in X$, $\pi'(x) = R(x) = \pi''(x)$. Let π be the assignment on S made by combining π' and π'', i.e. $\pi(x) = \pi'(x) = \pi''(x)$ for $x \in X$, $\pi(x) = \pi'(x)$ for $x \in X'$, and $\pi(x) = \pi''(x)$ for $x \in X''$.

Observe that by definition of π' and π'', $L_{x,\pi(x)} = 1$ for all $x \in T$, $\pi(x) = R(x)$ if $R(x) \in \mathcal{B}$, and otherwise $\mathcal{R}(\pi(x), \mathcal{B}) = R(x)$. Any edges and arcs in $G[X \cup X']$ are satisfied by π, by definition of π', and any edges and arcs in $G[X \cup X'']$ are satisfied by π, by definition of π''. It remains to consider the edges and arcs between X' and X''. Since the tasks in X' and X'' are assigned to disjoint sets of users (by Lemma 2), any edge between and X' and X'' is satisfied. If there is an arc from $x \in X'$ to $y \in X''$, then by our assumption there exists $u_i \in \mathcal{B}$ with $x_{u_i} = [<]$ and $y_{u_i} = [>]$. Therefore $\pi(x) < u_i < \pi(y)$, and therefore $\pi(x) < \pi(y)$, and so the arc is satisfied. A similar argument applies when there is an arc from $y \in X''$ to $x \in X'$.

Since there are at most $2^{|T|}$ possible ways to partition $T \setminus X$ into X' and X'', we can calculate $F(\mathcal{B}, T, R)$ in $O(2^k)$ time.

The above bounds show that, provided all the values for descendants of $\mathcal{B}$ have been computed, $F(\mathcal{B}, T, R)$ can be calculated in time $O(k4^k)$, for each possible $\mathcal{B}, T$ and R. It remains to count the number of possible values of $\mathcal{B}, T$ and R. There are at most $4n$ values of $\mathcal{B}$. Calculating $F(\mathcal{B}, T, R)$ for every T and R can be viewed as calculating F for every function $R^* : S \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|} \cup \{0\}$, T being defined as the set of steps not mapped to 0. Finally, for each step x in S there are $r + 2 + 3^{r+1}$ possible values for $R^*(x)$ and therefore $(r + 2 + 3^{r+1})^k$ possible values for R^*. Therefore the total number of possible values of $F(\mathcal{B}, T, R)$ is $O(n(r + 2 + 3^{r+1})^k)$, and so every value $F(\mathcal{B}, T, R)$ can be calculated in time $O(nk4^k(r + 2 + 3^{r+1})^k)$. □

5 Hardness

The main theorem of this section establishes a lower bound for the complexity of the workflow satisfiability problem. In fact, we show that in general, the trivial $O(n^k)$ algorithm is nearly optimal. Our result assumes the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [11]: that is, we assume that there is no $2^{o(n)}$-time algorithm for n-variable 3-SAT.

Theorem 2. WSP$(=, \neq, <)$ *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *unless* ETH *fails, where* f *is an arbitrary function,* k *is the number of steps and* n *is the number of users. This results holds even if the full graph of* $(U, <)$ *is 2-degenerate.*

The proof of Theorem 2 can be found in the appendix. It is well-known (see, e.g., [10]) that ETH is stronger than the widely believed complexity hypothesis W[1] $\neq$ FPT. Thus, we have the following:

Corollary 1. WSP$(=, \neq, <)$ *is not FPT unless W[1] = FPT. This results holds even if the full graph of* $(U, <)$ *is 2-degenerate.*

This corollary proves that while the class of treewidth bounded graphs is sufficiently special to imply an FPT algorithm, considering the more general class of graphs of bounded degeneracy does not make the problem any easier.

6 Concluding Remarks

The main contribution of this paper is the development of the first FPT algorithm for WSP$(=, \neq, <)$, where $<$ is a (transitive) relation on the set of users. Unlike WSP$(=, \neq)$ which is FPT in the general case, WSP$(=, \neq, <)$ is not FPT unless W[1]=FPT, which is highly unlikely. In fact, under a stronger hypothesis (ETH) we have shown that we even cannot have an algorithm significantly faster than the trivial brute-force algorithm. Thus, it is natural to identify special cases of WSP$(=, \neq, <)$ that are in FPT and of practical relevance. We have done this by restricting the reduced graph D of $(U, <)$ to lie in the class of graphs of bounded treewidth. We believe that this restriction on treewidth holds for many user hierarchies that arise in practice. On the other hand, we have also shown that the restriction of the reduced (or even full) graph to the class of 2-degenerate graphs does not reduce the complexity of the problem.

Our FPT algorithm is efficient for small values of the number of steps k and the treewidth r of D (we may view $k + r$ as a combined parameter). However, it is quite often the case that the first FPT algorithm for a parameterized problem is not efficient except for rather small values of the parameter, but subsequent improvements bring about an FPT algorithm efficient for quite large values of the parameter [10,14]. We believe that a more efficient FPT algorithm for WSP$(=, \neq, <)$ may be possible and we hope to be able to report progress in this area.

One natural extension of this work is to consider the preorder generated from an authorization policy, where $u \sqsubseteq u'$ iff the set of steps for which u is authorized is a subset of the set of steps for which u' is authorized. This ordering is weaker than that defined in Sec. 2 and used throughout the rest of the paper, which required that the set of steps for which u is authorized to be a strict subset of those for which u' is authorized. Hence, we may have $u \sqsubseteq u'$ and $u' \sqsubseteq u$ but $u \neq u'$. In fact, such an ordering defines sets of users that are *indistinguishable*, in the sense that they are authorized for the same set of steps. Hence, we might reasonably consider WSP$(=, \neq, \sqsubset, \sim, \nsim)$, where $u \sim u'$

if u and u' are indistinguishable. Of course, the graph of $\sqsubseteq$ is not acyclic, as cycles of length two will exist between any pair of indistinguishable users, so new techniques may be required to determine whether this problem is FPT or not.

Acknowledgment This research was partially supported by an International Joint grant of the Royal Society.

References

1. Aho, A., Garey, M., Ullman, J.: The transitive reduction of a directed graph. SIAM J. Comput. 1(2), 131–137 (1972)
2. Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods 8(2), 277–284 (1987)
3. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer-Verlag, London, 2nd edn. (2009)
4. Bodlaender, H.: A linear time algorithm for finding tree decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
5. Bodlaender, H., Koster, A.: Treewidth computations I. upper bounds. Information and Computation 208(3), 259–275 (2010)
6. Crampton, J., Gutin, G., Yeo, A.: On the parameterized complexity of the workflow satisfiability problem. In: Proceedings of 19th ACM Conference on Computer and Communications Security (2012), to appear: pre-print available at `http://arxiv.org/abs/1205.0852v1`
7. Crampton, J., Gutin, G., Yeo, A.: On the parameterized complexity and kernelization of the workflow satisfiability problem. arxiv (2012-10-11), 1205.0852v2
8. Crampton, J.: A reference monitor for workflow systems with constrained task execution. In: Ferrari, E., Ahn, G.J. (eds.) SACMAT. pp. 38–47. ACM (2005)
9. Downey, R., Fellows, M.: Parameterized complexity. Monographs in computer science, Springer (1999)
10. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science, Springer (2006)
11. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. 63(4), 512–530 (2001)
12. Kloks, T.: Treewidth. Computations and Approximations, Lecture Notes in Computer Science, vol. 842. Springer (1994)
13. Marx, D.: Can you beat treewidth? Theory of Computing 6(1), 85–112 (2010)
14. Niedermeier, R.: Invitation to Fixed Parameter Algorithms. Oxford Lecture Series in Mathematics And Its Applications, Oxford University Press, USA (2006)
15. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM Trans. Inf. Syst. Secur. 13(4), 40 (2010)

A Proof of Theorem 2

In order to prove Theorem 2, we first consider the following problem and prove the following lemma.

SubTDAG isomorphism	
Input:	Transitive acyclic digraphs $D = (V_D, A_D)$ and $R = (V_R, A_R)$, a subset $W_R = \{w_1, \ldots, w_{\lvert W_R \rvert}\}$ of V_R, and disjoint subsets $W_{D,1}, \ldots, W_{D,\lvert W_R \rvert}$ of V_D.
Parameter:	$\lvert V_R \rvert$
Question:	Is there an injection $\gamma : V_R \to V_D$ such that $\gamma(w_i) \in W_{D,i}$ for each $i \in [\lvert W_R \rvert]$, and for every $(u, v) \in A_R$, $(\gamma(u), \gamma(v)) \in A_D$?

Lemma 6. SubTDAG isomorphism *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *where* f *is an arbitrary function,* n *is the number of vertices in* D *and* k *is the number of vertices in* R*, unless* ETH *fails. This result holds even if* D *and* R *are 2-degenerate.*

To prove Lemma 6, we start by considering the following problem and a lemma by Marx [13].

Partitioned Subgraph Isomorphism (PSI)	
Input:	Undirected graphs $H = (V_H, E_H)$ and $G = (V_G = \{g_1, \ldots, g_l\}, E_G)$, and a partition of V_H into (disjoint) subsets $W_{H,1}, \ldots, W_{H,l}$.
Question:	Is there an injection $\phi : V_G \to V_H$ such that for every $i \in [l]$, $\phi(g_i) \in W_{H,i}$ and for every $(g_i, g_j) \in E_G$, $(\phi(g_i), \phi(g_j)) \in E_H$?

Lemma 7. (*Corollary 6.3,* [13]) Partitioned Subgraph Isomorphism *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *where* f *is an arbitrary function,* k *is the number of edges in* G *and* n *is the number of vertices in* H*, unless* ETH *fails.*

Proof of Lemma 6. The proof is by a reduction from the Partitioned Subgraph Isomorphism problem. We assume that we have an instance of PSI as described in the formulation of the problem above. We assume, without loss of generality, that there are no isolated vertices in G. Recall that the vertices of G are $g_1, \ldots, g_l$ and let $W_{H,1} = \{x(11), \ldots, x(1r_1)\}, \ldots, W_{H,l} = \{x(l1), \ldots, x(lr_l)\}$. We now construct an instance of SubTDAG isomorphism. The digraph R is obtained from G by subdividing every edge and orienting all edges towards the new vertices. The vertex subdividing an edge g_ig_j will be denoted by g_{ij} and so R will have arcs (g_i, g_{ij}) and (g_j, g_{ij}). Similarly, D is obtained from H by subdividing every edge and orienting all edges towards the new vertices. The vertex subdividing an edge $x(i\tau_i)x(j\tau_j)$ will be denoted by $x(i\tau_i, j\tau_j)$. It is easy

to verify that D and R are both 2-degenerate acyclic digraphs and both are transitive because they do not have directed paths of length 2. Let $W_R = V_G$ and $W_{D,i} = W_{H,i}$ for each $i \in [l]$. We claim that $(G, H, W_{H,1}, \ldots, W_{H,l})$ is a YES-instance of PSI if and only if $(D, R, W_R, W_{D,1}, \ldots, W_{D,l})$ is a YES-instance of SUBTDAG ISOMORPHISM.

Suppose that our instance of PSI is a YES-instance and let ϕ be the required injection. By definition, $\phi(g_i) = x(i\tau_i)$, where $\tau_i \in [r_i]$, for each $i \in [l]$. Let $\gamma : V_R \to V_D$ be defined as follows: $\gamma(g_i) = x(i\tau_i)$ for each $i \in [l]$ and $\gamma(g_{ij}) = x(i\tau_i, j\tau_j)$. Since $g_ig_j \in E_G$ implies $\phi(g_i)\phi(g_j) \in E_H$ and by the definition of γ, if $(g_i, g_{ij}) \in A_R$ then $(\gamma(g_i), \gamma(g_{ij})) \in A_D$. Thus, our instance of SUBTDAG ISOMORPHISM is a YES-instance, too.

Now suppose that the instance of SUBTDAG ISOMORPHISM is a YES-instance and $\gamma : V_R \to V_D$ is the corresponding injection such that $\gamma(g_i) = x(i\tau_i)$, where $\tau_i \in [r_i]$, for each $i \in [l]$. By definition of γ, $(g_i, g_{ij}) \in A_R$ implies $(x(i\tau_i)\gamma(g_{ij})) \in A_D$ and $(g_j, g_{ij}) \in A_R$ implies $(x(j\tau_j)\gamma(g_{ij})) \in A_D$. By the construction of D, the above implies that $\gamma(g_{ij}) = x(i\tau_i, j\tau_j)$. Now define an injection $\phi : G \to H$ as follows: $\phi(g_i) = \gamma(g_i) = x(i\tau_i)$ for each $i \in [l]$. The requirement that $g_ig_j \in E_G$ implies $\phi(g_i)\phi(g_j) \in E_H$ follows from the fact that $\gamma(g_{ij}) = x(i\tau_i, j\tau_j)$. Thus, the instance of PSI is a YES-instance, too.

Let k_G be the number of edges in G and n_H the number of vertices in H. Recall that k is the number of vertices in R and n is the number of vertices in D. By construction of R and D and the assumption that G has no isolated vertices, $k = |E_G| + |V_G| = \Theta(k_G)$ and $n = n_H + |A_H| = O(n_H^2)$.

An algorithm for SUBTDAG ISOMORPHISM running in time $f(k)n^{o(\frac{k}{\log k})}$ implies an algorithm running in time $f(k_G)n_H^{o(\frac{k_G}{\log k_G})}$ for PSI, which along with Lemma 7 completes the proof of the lemma. □

Proof of Theorem 2. The proof is by a reduction from the SUBTDAG ISOMORPHISM problem. Let $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$ be an instance of SUBTDAG ISOMORPHISM. We construct an instance of WSP$(=, \neq, <)$ as follows. We define the set U of users to be V_D and the set S of steps to be V_R. For every step $w_i \in W_R$, $L(w_i) = W_{D,i}$, and for every step $s \in S \setminus W_R$, $L(s) = U$.

We define the relation $<$ on U as follows. For every $x, y \in U$, $x < y$ if and only if $x \neq y$ and there is a arc from x to y in D. For every arc $(u, v) \in A_R$, we add a constraint $(<, u, v)$ and for every pair u, v of distinct non-adjacent vertices of R, we add a constraint $(\neq, u, v)$. Let the instance of WSP$(=, \neq, <)$ thus constructed be $\mathcal{I}$. We claim that $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$is a YES instance of SUBTDAG ISOMORPHISM iff $\mathcal{I}$ is a YES instance of WSP$(=, \neq, <)$.

Suppose that $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$ is a YES-instance of SUBTDAG ISOMORPHISM and let γ be a required injection for this instance. We define a plan π as $\pi(v) = \gamma(v)$ for every $v \in S$. It is easy to see that π is an valid plan for $\mathcal{I}$.

Conversely, suppose that $\mathcal{I}$ is a YES-instance of WSP$(=, \neq, <)$ and let π be a valid plan for this instance. We define a function $\gamma : V_R \to V_D$ as follows.

For every $u \in V_R$, we set $\gamma(u) = \pi(u)$. It remains to verify that γ is a required injection for the instance $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$. We first show that γ is an injection. Suppose this were not the case and let u and v be two distinct vertices such that $\gamma(u) = \gamma(v)$. This implies that $\pi(u) = \pi(v)$. But then this assignment satisfies neither the constraint $(\neq, u, v)$ nor the constraint $(<, u, v)$, which is a contradiction. Hence, we conclude that γ is indeed an injection. Now, consider an arc $(u, v) \in R$. Since π is a valid plan, $\pi(u) < \pi(v)$, which implies that $\gamma(u) < \gamma(v)$, which by definition is possible only if $(\gamma(u), \gamma(v)) \in A_D$. This completes the proof of correctness of the reduction.

It remains to apply Lemma 6 to complete the proof of the theorem. □