Dynamic Workflow Adjustment with Security Constraints

Haibing Lu[1], Yuan Hong[2], Yanjiang Yang[3], Yi Fang[1], and Lian Duan[4]

[1] Santa Clara University
{hlu,yfang}@scu.edu
[2] University at Albany
hong@albany.edu
[3] I2R Singapore
yyang@i2r.a-star.edu.sg
[4] New Jersey Institute of Technology
lian.duan@njit.edu

Abstract. Dynamic workflow adjustment studies how to minimally adjust existing user-task assignments, when a sudden change occurs, e.g. absence of users, so that all tasks are being attended and no constraint is violated. In particular, we study two key questions: (i) Will the workflow still be satisfiable given a change? (ii) If the answer is yes, how to find a satisfying assignment with the minimum perturbation to the old system? We consider various types of changes, including absence of a user, addition of a separation-of-duty constraint, addition of a binding-of-duty constraint, and revocation of a user-to-task authorization, study their theoretical properties and formulate them into the well-studied Boolean satisfiability problem, which enables a system engineer without much technical background to solve problems by using standard satisfiability solvers. A step further, towards more efficient solutions for our specific problems, we propose customized algorithms by adapting and tailoring the state-of-art algorithms inside standard solvers. Our work would have implications for business process management, staffing, and cost planning.

Keywords: workflow, security, dynamic, satisfiability.

1 Introduction

A workflow can be defined as a set of tasks and dependencies that control the coordination requirements among these tasks [2]. A workflow example is an information system for an online pharmacy store, which involves a set of tasks, e.g. order entry, medication assessment, billing, and shipping, a set of employees with different roles, e.g. pharmacist and non-technical staff, and a set of constraints, e.g. non-technical staff cannot perform medication assessment, and a person who does credit check cannot perform billing due to the fraud concern.

The problem of allocating users to tasks to comply with a given authorization policy and also fulfil the workflow requirement, is important in access control and has received considerable attention in the literature. However, there still lack studies from the dynamic perspective, despite a few papers looking into this problem, e.g. [17,5].

V. Atluri and G. Pernul (Eds.): DBSec 2014, LNCS 8566, pp. 211–226, 2014.

Indeed, many factors of a workflow are dynamically changing, e.g. a user is absent due to sickness, the right of a user to access a certain task is temporarily revoked due to frequent mistakes, and a user has to step way from a task due to an emerging conflict of interest. A poorly designed workflow, although might be working at the current moment, is vulnerable to future changes and may cause huge troubles both financially and operationally to an organization. So it is crucial to study the resilience and flexibility of a workflow system with respect to various types of changes. It is also important to investigate how much disruption to an existing system is necessary to make a satisfying workflow assignment when a sudden change occurs.

In this paper, we formulate and study dynamic workflow adjustment with various changes, which is to make a workflow assignment with the minimum perturbations to the current workflow system, while complying with the authorization policy and all associated constraints. Specific types of dynamic changes considered include absence of a user, addition of a separation-of-duty constraint, addition of a binding-of-duty constraint, and revocation of a user-to-task authorization. To tackle dynamic workflow adjustment with various changes, we provide Boolean satisfiability model formulations, which enable a workflow engineer without much technical background to solve the problems with standard solvers.

Due to the hardness nature of dynamic workflow adjustment, it is of practical importance to have efficient and customized algorithms, rather than resorting to standard solvers, which rarely take account of the characteristics of individual problems. A significant difference of dynamic workflow adjustment from the conventional workflow assignment problem is that a satisfying assignment for the system before the change is already available, which should be taken advantage of, rather than designing a new assignment from scratch like a standard solver does. More importantly, finding the satisfying workflow assignment, closest to the old assignment, is our ultimate goal, as a satisfying workflow assignment with much disruption to the old system is of no value in practice. We are thus motivated to customize the state-of-art algorithms for the Boolean satisfiability problem to our unique problems.

Our work would have practical implications on workflow planning, staffing, and cost budgeting. By studying the resilience of a workflow system against different types of changes, a system manager can plan ahead to minimize the expected loss. By studying the minimum required perturbation to the old assignment to make a satisfying assignment, the manager can have a better understanding of the strategic importance of a specific position or an employee and thus make a better and more flexible workflow system.

2 Problem Definitions

Many types of workflow assignment constraints have been studied in the literature, e.g. [5,17,9,4]. Following their research, we consider skill constraints, separation-of-duty constraints, binding-of-duty constraints, and performing constraints.

Skill constraints, also called static constraints, refer to that a task has to be performed by persons with necessary skills or credentials. Skill constraints are typically enforced by role-based access control (RBAC) [13] due to its various advantages, e.g.

low administrative cost and support of permission inheritance through role hierarchy. The basic idea of RBAC is to associate roles with tasks and then assign roles to users, so that a user is authorized to all tasks that are associated with the roles assigned to him/her. For instance, a pharmacist role is associated with tasks of ordering entry, credit check, fulfilling order, mediation assessment, shipping, and billing. So any employee with the pharmacist role can perform all associated tasks. The literature of RBAC, e.g. [10,11,16], typically denotes user-task assignments by UPA, user-role assignments by UA, and role-task assignment by PA, in which UA and PA can deduce UPA. Since this paper assumes roles are stable, to ease the modeling, we consider and denote user-task assignments by A, which is a binary matrix, i.e. $x_{ij} = 1$ means user i is permitted to perform task j; otherwise not. Note that in our paper, authorization is different from actual assignment. Authorization of $x_{ij} = 1$ only means user i has necessary credentials (or skills) to perform task j. For instance, a pharmacist is permitted to perform the billing task, but may not be assigned to the billing task.

A separation-of-duty (SoD) constraint is to distribute responsibilities to prevent from fraud and error. For instance, many companies require that a person can not perform both purchasing and billing tasks to avoid embezzlement. The conventional perception of a separation-of-duty constraint is a pair of conflicting tasks that no one can perform simultaneously. This paper considers a separation-of-duty constraint from a more general perspective by including conflict-of-interest constraints, as they both advocate decentralization of tasks. An example of conflict-of-interest constraint is that in order to provide an objective review a funding proposal reviewer is not allowed to review the proposal of the person whom he/she had worked with or supervised. So in our paper, a separation-of-duty constraint is defined and denoted by s_{ijkl}, which states that if user i performs task j, then user k is not allowed to perform task l. As such, the conventional definition of a pair of conflicting tasks j and l can be described as $\bigcup_{\forall i} s_{ijil}$.

A binding-of-duty (BoD) constraint specifies the binding relation of tasks. For instance, a person who changes a password must be the person who creates the password. In our paper, it is defined and denoted by b_{ijkl}, stating that if user i performs task j, then user k has to perform task l. Note that our definition is different from and more general than the conventional definition of a BoD constraint, which refers to that bound tasks have to be performed by the same subject. In some cases, a binding relation can be associated with multiple subjects. For instance, a company may specify that the manager who approves a project proposal must come from the same department as the proposal submitter due to the same knowledge background. By our BoD definition, a conventional BoD constraint on binding task j and l can be described as $\bigcup_{\forall i} b_{ijil}$.

A performing constraint specifies that every task needs to be performed by at least one user; in other words, no unattended task. The constraint can be represented by $\sum_i x_{ij} \geq 1, \forall j$.

In this paper, we consider and study four types of changes that may interrupt a running workflow system. They are: (1) absence of a user, (2) addition of a SoD constraint, (3) addition of a BoD constraint, and (4) revocation of an authorization.

A common obstruction to a workflow system is the change of users. Addition of a user does not cause constraint conflicts, although the system engineer needs to assign appropriate tasks to the new user, the study of which is out of the scope of this paper.

When user i is absent, a performing constraint may be violated, e.g. task j becomes unattended if user i was the only one performing the task. If another user is to replace the absent user, other types of constraint conflicts, e.g. SoD and BoD, may occur. So to prevent potential loss, a workflow designer has to plan ahead by investigating the workflow resilience to such a type of changes. Two questions are faced. First, will a workflow still be satisfiable after a change occurs? Second, which might be more important, what is the satisfying workflow assignment with the minimum disruption to the old system? By satisfying, we mean the workflow assignment does not cause any constraint conflict. The second question is more important because a satisfying workflow assignment with much disruption to the old system is of no practical value. To answer the two questions, we define the following problem.

Problem 1. Given users U, tasks T, user-task authorizations A, BoD constraints B, SoD constraints S, existing satisfying user-task assignment $\tilde{X}$, and a number δ, if user i' is absent, does there exist a satisfying workflow assignment X such that $\sum_{ij} |x_{ij} - \tilde{x}_{ij}| \leq \delta$?

δ is the threshold for the amount of disruption. When δ is greater than $\sum_{ij} |\tilde{x}_{ij}|$, problem 1 is the formulation of the first question. Indeed, problem 1 is the representation of the decision version of the second question. So by solving problem 1 multiple times with different values of δ, one can find the answer for the second question.

A BoD constraint may be added or deleted, when a user's responsibilities changed, user relations evolved, task characteristics are updated, etc. Addition of a BoD constraint may cause the existing workflow assignment to be unsatisfying, while deletion of a BoD constraint does not impact the satisfiability. Addition of a BoD constraint gives rise to the same two questions. First, will the addition of a SoD constraint make the workflow unsatisfiable? Second, how to find a satisfying workflow assignment without the minimum disruption to the old system? The two questions are formulated as problem 2.

Problem 2. Given users U, tasks T, user-task authorizations A, BoD constraints B, SoD constraints S, and existing satisfying user-task assignments $\tilde{X}$ and a number δ, if a BoD constraint $b_{i'j'k'l'}$ is added, does there exist a satisfying workflow assignment X such that $\sum_{ij} |x_{ij} - \tilde{x}_{ij}| \leq \delta$?

Addition of a SoD constraint $s_{i'j'k'l'}$ may also cause constraint conflicts if $\tilde{x}_{i'j'} = 1$ and $\tilde{x}_{k'l'} = 1$ both hold in the old workflow system. Will the change cause the system to a standstill? How much effort is required to make another satisfying workflow system? The two questions can be answered by solving the following problem.

Problem 3. Given users U, tasks T, user-task authorizations A, BoD constraints B, SoD constraints S, and existing satisfying user-task assignments X and a number δ, if a SoD constraint $s_{i'j'k'l'}$ is added, does there exist a satisfying workflow assignment X such that $\sum_{ij} |x_{ij} - \tilde{x}_{ij}| \leq \delta$?

Authorization might be added or revoked in the middle of a process. As addition of an authorization, e.g. a staff may be upgraded to the pharmacist role after getting the licence, does not cause any type of constraint conflicts studied in this paper, so in terms of change of authorization we only consider the revocation case. Revocation

of authorization may occur when a user becomes disqualified for certain tasks. For instance, a pharmacist is permitted to process shipment. But if he frequently makes mistakes, then his permission to that task may be suspended or revoked. The problem is that when his assignment to the shipment task is canceled, we have to find another person to replace that person, if that person is the only one assigned to that job in the old workflow system. There might be multiple persons with the authorization to the shipment task. But an assignment decision may cause other types of conflicts, e.g. SoD and BoD. In that case, we may have to make more changes to resolve cascaded conflicts. To investigate the satisfiability and resilience of the workflow to such a type of change, we define the following problem.

Problem 4. Given users U, tasks T, user-task authorizations A, BoD constraints B, SoD constraints S, existing satisfying user-task assignment $\tilde{X}$, and a number δ, if authorization $a_{i'j'}$ is revoked, does there exist a satisfying workflow assignment X such that $\sum_{ij} |x_{ij} - \tilde{x}_{ij}| \leq \delta$?

3 Theoretical Study

Finding a satisfying workflow assignment without any constraint conflict from scratch has been proven to be NP-hard [5]. The difference in our dynamic workflow adjustment problems is that there was a satisfying workflow assignment available. Intuitively, it should not be difficult to examine the workflow satisfiability under a change by tweaking the previous workflow assignment. But it turns out a dynamic workflow adjustment problem can be as difficult as the workflow design problem.

In this section, we will prove the dynamic workflow adjustment problem in the case of one user being absent is NP-complete based on some known results.

Statement 1. *The problem of determining when a planar map, i.e. it can be drawn on the plane in such a way that its edges intersect only at their endpoints, is three-colorable is NP-complete [15].*

Statement 2. *Every planar map is 4-colorable [1].*

Theorem 1. *Problem 1 is NP-complete.*

A decision problem is NP-complete if it belongs to NP and also can be reduced to a NP-complete problem.

Given a new workflow assignment, one can examine its difference from the old assignment and its satisfiability in polynomial time. So problem 1 belongs to NP.

Consider a special case of problem 1 with δ being a large number, so the decision problem asks whether a workflow is satisfiable when a user is absent. The satisfiability problem can be reduced to planar 3-colorability. Statements 1 and 2 show that any planar graph has a 4-coloring solution, but hard to find a 3-coloring solution. An instance of problem 1 can be represented by $\{U, T, A, C, B, X, U_i\}$, which denote users, tasks, authorizations, conflict-of-interest constraints, binding-of-duty constraints, previous assignments and the user who is absent. A planar map can be represented by regions $\{r_i\}$. We denote $col : r_i \rightarrow \{1, 2, 3, 4\}$ to be a 4-coloring solution, such that $col(r_i) \neq col(r_j)$ if r_i and r_j are adjacent. For each planar map instance, we can construct an equivalent instance of problem 1 as the follows:

- For U, let it be $\{u_1, u_2, u_3, u_4\}$;
- For T, create a task t_i to correspond to each region R_i of the map;
- For A, users are allowed to execute all tasks;
- For C, include $\{s_{1i1j}, s_{2i2j}, s_{3i3j}, s_{4i4j}\}$, i.e. separation-of-duty constraint on t_i and t_j, if r_i and r_j are adjacent;
- For B, let it be empty;
- For X, let x_{ij} be 1 if $col(r_j) = i$ so that X are feasible assignments to the above constraints C;
- For u_i, let it be any user.

When a user is absent, the constructed instance of problem 1 becomes equivalent to finding a 3-coloring solution to a planar map. So problem 1 is NP-complete. □

4 Model Formulation

To tackle the dynamic workflow adjustment problems, one straightforward approach is to formulate them with well-studied models and then take advantage of existing solvers, which can save much effort for a system engineer. We find that the dynamic workflow adjustment problems can be modeled as Boolean satisfiability problems, which are well studied and have many good algorithms as well as available public/commercial software packages. Boolean satisfiability problem, commonly abbreviated as SAT, is probably one of the most studied problems in computer science, and has a range of applications in electronic design automation and artificial intelligence. SAT is historically notable as the first problem proven to be NP-complete. However, SAT is widely used because conflict-driven clause learning (CDCL) SAT solvers [14] are so effective in practice.

The dynamic workflow adjustment problems can be formulated as Boolean satisfiability problems, which enable one to adopt exiting SAT solvers. Before we provide their SAT formulations, we firstly examine the constraints.

A BoD constraint b_{ijkl} requires user k to perform task l when user i performs task j. In other words, if x_{ij} is TRUE, x_{kl} has to be TRUE, which can be expressed by:

$$(\neg x_{ij} \bigvee x_{kl}).$$

When x_{ij} is TRUE, $\neg x_{ij}$ is FALSE. Then in order to make the clause to be TRUE, x_{kl} has to be TRUE.

A SoD constraint s_{ijkl} forbids user k from performing task l when user i performs task j. In other words, one of x_{ij} and x_{kl} has to be FALSE, which can be expressed by the clause:

$$(\neg x_{ij} \bigvee \neg x_{kl}),$$

because in order to make the clause to be TRUE, the negation of one of x_{ij} and x_{kl} has to be TRUE.

A performing constraint requires that each task t_j needs to be performed by at least one person. In other words, one of x_{ij} has to be TRUE, which can be expressed by the clause:

$$(\bigvee_{i \in A_j} x_{ij}),$$

where A_j denotes the set of users with the authorization to task t_j.

Before any change happens, the workflow is satisfied by the current user-task assignments $\{\tilde{X}_{ij}\}$, which means the following CNF expression is TRUE:

$$(\bigwedge_{\forall s_{ijkl}} (\neg\tilde{X}_{ij} \bigvee \neg\tilde{X}_{kl})) \bigwedge (\bigwedge_{\forall b_{ijkl}} (\neg\tilde{X}_{ij} \bigvee \tilde{X}_{kl})) \bigwedge (\bigwedge_{\forall j} (\bigvee_{i \in A_j} \tilde{X}_{ij})).$$

Consider problem 1, which essentially tries to answer two questions: whether the workflow is satisfiable when user i' is absent? what is the satisfying assignment with the minimum perturbation to the old assignment?

The first question can be formulated as the SAT problem of finding an assignment of TRUE and FALSE values to variables $\{x_{ij}\}$ to satisfy the CNF expression:

$$E_1 = (\bigwedge_{\forall s_{ijkl} | i \neq i'} (\neg x_{ij} \bigvee \neg x_{kl})) \bigwedge (\bigwedge_{\forall b_{ijkl} | i \neq i'} (\neg x_{ij} \bigvee x_{kl})) \bigwedge (\bigwedge_{\forall j} (\bigvee_{i \in A_j | i \neq i'} x_{ij})),$$

which contains the clause representation of all workflow assignment constraints.

The second question can be described as the weighted MAX-SAT problem with the CNF expression:

$$E_2 = E_1 \bigwedge (\bigwedge_{\forall i | i \neq i'} ((x_{ij} \bigvee \neg x_{ij}) \bigwedge (\tilde{x}_{ij} \bigvee \neg x_{ij}) \bigwedge (x_{ij} \bigvee \neg \tilde{x}_{ij}) \bigwedge (\tilde{x}_{ij} \bigvee \neg \tilde{x}_{ij}))).$$

Clauses $(x_{ij} \bigvee \neg x_{ij}) \bigwedge (\tilde{x}_{ij} \bigvee \neg x_{ij}) \bigwedge (x_{ij} \bigvee \neg \tilde{x}_{ij}) \bigwedge (\tilde{x}_{ij} \bigvee \neg \tilde{x}_{ij})$ are used to evaluate the equality of x_{ij} and $\tilde{x}_{ij}$, as the clauses are TRUE if and only if the value of x_{ij} is the same as $\tilde{x}_{ij}$. We let the weights on all clauses in E_1 be a sufficiently large number and the weights on the other clauses be 1. So to maximize such a weighted MAX-SAT problem, clauses in E_1 must be satisfied, as they carry significantly large weights. As such, we guarantee that the optimal solution to the weighted MAX-SAT problem corresponds to a satisfying workflow assignment. Therefore, the constructed weighted MAX-SAT problem is equivalent to the original minimal perturbation problem.

Consider problem 2, addition of a BoD constraint $s_{i'j'k'l'}$. Whether the workflow is satisfiable after the change can be formulated as a SAT problem with the expression:

$$E_3 = E_1 \bigwedge (\neg x_{ij} \bigvee x_{kl}).$$

The problem of finding a satisfying assignment with the minimum perturbation after the BoD constraint change can be formulated as a MAX-SAT problem of the CNF expression:

$$E_4 = E_3 \bigwedge (\bigwedge_{\forall i | i \neq i'} ((X_i \bigvee \neg X_i) \bigwedge (\tilde{X}_i \bigvee \neg X_i) \bigwedge (X_i \bigvee \neg \tilde{X}_i) \bigwedge (\tilde{X}_i \bigvee \neg \tilde{X}_i)))$$

with the weights on clauses of E_3 being a significantly large number and the weights on the other clauses being 1.

Consider problem 3, addition of a SoD constraint $s_{i'j'k'l'}$. Whether the workflow is satisfiable after the change can be formulated as a SAT problem with the expression:

$$E_5 = E_1 \bigwedge (\neg x_{i'j'} \bigvee \neg x_{k'l'}).$$

The problem of finding a satisfying assignment with the minimum perturbation after the SoD constraint change can be formulated as a MAX-SAT problem of the CNF expression:

$$E_6 = E_5 \bigwedge (\neg x_{i'j'} \bigvee \neg x_{k'l'}) \bigwedge (\bigwedge_{\forall i | i \neq i'} ((X_i \bigvee \neg X_i) \bigwedge (\tilde{X}_i \bigvee \neg X_i) \bigwedge (X_i \bigvee \neg \tilde{X}_i) \bigwedge (\tilde{X}_i \bigvee \neg \tilde{X}_i)))$$

with the weights on clauses of E_5 being a significantly large number and the weights on the other clauses being 1.

Consider problem 4, revocation of authorization $a_{i'j'}$. Whether the workflow is satisfiable after the change can be formulated as a SAT problem with the CNF expression:

$$E_7 = (\bigwedge_{\forall s_{ijkl}} (\neg \tilde{X}_{ij} \bigvee \neg \tilde{X}_{kl})) \bigwedge (\bigwedge_{\forall b_{ijkl}} (\neg \tilde{X}_{ij} \bigvee \tilde{X}_{kl})) \bigwedge (\bigwedge_{\forall j} (\bigvee_{i \in A'_j} \tilde{X}_{ij})).$$

The problem of finding a satisfying assignment with the minimum perturbation after the SoD constraint change can be formulated as a MAX-SAT problem of the CNF expression:

$$E_8 = E_7 \bigwedge (\bigwedge_{\forall i | i \neq i'} ((x_i \bigvee \neg x_i) \bigwedge (\tilde{X}_i \bigvee \neg x_i) \bigwedge (x_i \bigvee \neg \tilde{x_i}) \bigwedge (\tilde{x_i} \bigvee \neg \tilde{x_i})))$$

with the weights on clauses of E_7 and $(\neg x_{i'j'} \bigvee \neg x_{k'l'})$ being a significantly large number and the weights on the other clauses being 1.

5 Customized Algorithms

Resorting to existing optimization and SAT solvers is a common approach for the access control community to tackle encountered problems, e.g. [10,5,17]. However, a disadvantage of solvers is that they are designed as a universal platform for all feeded problems and thus disregard the properties of individual problems that could be used to design more efficient algorithms.

Unlike designing a workflow assignment from scratch, the dynamic workflow adjustment problems have an important piece of information available, the previous satisfying assignment. To make a satisfying workflow assignment, an intuitive way is to tweak the previous conflicting assignment instead of trying to make up the whole assignment from empty as a solver would do. In this section, we will present customized algorithms for the dynamic workflow adjustment problems. As they are based on the start-of-art algorithms for the SAT problem, so we firstly give a brief introduction on them.

5.1 State-of-Art SAT Algorithms

The state-of-art SAT algorithms are DPLL [7] and CDCL (a modern variant of DPLL) [14,12]. Both CDCL and DPLL algorithms are complete, backtracking-based, tree search algorithms for deciding the satisfiability of a CNF expression. At each step, the algorithms choose a variable, assign a value to it, simplify the formula and then check if the

simplified formula is satisfiable. If it is true, the original formula is satisfiable. Otherwise, assume the opposite value to the variable. If it is not satisfiable either, the algorithm backtracks to a higher level. The difference between CDCL and DPLL is that the DPLL algorithm backtracks to the next higher level, which is referred to as chronical backtracking, while the CDCL algorithm may go up more levels by using clause learning, which is referred to as non-chronical backtracking.

Both algorithms speed up the backtracking by the eager use of the unit propagation rule at each step.

Unit propagation. If a clause is a *unit clause*, i.e. it contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true. Thus, no choice is necessary. In practice, it would lead to deterministic cascades of units and could avoid a large part of the naive search space. For example, consider the expression of $(x_1 \bigvee x_2) \bigwedge (\neg x_2 \bigvee x_3)$. If the literal x_1 has assumed FALSE, then $(x_1 \bigvee x_2)$ becomes a unit clause and x_2 has to assume TRUE to make the clause TRUE. As a cascade effect, $(\neg x_2 \bigvee x_3)$ becomes a unit clause and x_3 has to assume TRUE to make the clause TRUE.

5.2 Basic Algorithm

Instead of designing an algorithm for each presented problem, we introduce and study a basic problem, sharing the commonality with all dynamic workflow adjustment problems, and its algorithm can be applied to all problems with simple problem-specific configurations. The basic problem is defined as follows.

Problem 5. Given a list of perturbations $PList$ to a previously satisfying workflow assignment $\tilde{X}$, which leads to performing constraint conflicts only and no other types of constraint conflicts, does there exist a satisfying workflow assignment X with $\sum_{ij} |x_{ij} - \tilde{x}_{ij}| \leq \delta$ and without changing any given perturbation in $PList$?

Note that problem 5 limits existing conflicts to performing constraint conflicts only. So to resolve such a problem, at the beginning we only need to focus on how to find users to cover the unattended tasks. For each unattended task, there might be multiple users with rights to access. If we randomly pick and assign a user to an unattended task, other types of constraint conflicts, e.g. SoD and BoD, might be triggered. If one tries to fix a cascaded constraint conflict, more constraint conflicts could be generated. In the worse cases, we may end up in an infinite loop. So to effectively solve the problem, we need a strategy on how to make perturbations at each step.

Inspired by the DBLL and CDCL algorithms, we present a complete, backtracking-based, tree search algorithm, stated in Algorithm 1. The basic idea is that at each step we pick an unattend task and then select an authorized user to cover it. If the selected perturbation causes SoD and BoD constraint conflicts, we make further perturbations to resolve those conflicts, as the unit propagation does in the CDCL and DPLL algorithms. In particular, if user i is assigned to the unattended task j and a SoD constraint s_{ijil} is violated because user i was assigned to task l, we perturb both x_{il} and x_{kl} from 1 to 0. If a BoD constraint b_{ijil} is violated because user i was not assigned to task l, then we perturb both x_{il} and x_{kl} from 0 to 1 as well. The algorithm is written in a

Algorithm 1. BasicPerturb(X, A, B, S, PList, δ, DLevel)

```
1: if DLevel==0 then
2:     return UNSATISFIABLE
3: end if
4: NewPerturbations←PickPerturbation(X, A, B, C, PList);
5: if NewPerturbation==∅ then                          ▷ No satisfying assignment
6:     DLevel←DLeve-1;
7:     Backtrack();                                    ▷ Chronological Backtracking
8: else
9:     NewPerturbations←Propagate(X, A, B, C, PList, NewPerturbations);
10:    if IsConflict()==TRUE then
11:        NeighborSearch();                           ▷ Search neighboring nodes
12:    else if IsSatisfied()==TRUE then
13:        return SATISFIABLE
14:    else
15:        DLevel←DLevel+1;
16:        PList←PList⋃ NewPerturbations;
17:        Perturb(X, A, B, C, PList, DLevel);
18:    end if
19: end if
```

recursive form as the *BasicPerturb()* function and stated in Algorithm 1. The arguments of the *BasicPerturb()* function are X, the original user-task assignments, A, the user-task authorizations, B, BoD constraints, S, SoD constraints, $PList$, the given list of perturbations, δ, the maximal amount of allowed perturbations, and $DLevel$, the level of searching. The algorithm description is as follows.

- Lines 1-3 state that if $DLevel$ becomes 0, the problem is determined unsatisfiable. The value of $DLevel$ indicates the searching level and is set to 1 when the algorithm starts. So if $DLevel$ becomes 0, it means the whole searching space has been traversed and no satisfying solution has been found.
- At line 4, the *PickPerturbation()* function is to select an unattend task first and then assign an authorized user to it. Note that if the selected user causes unsolvable conflicts, we will keep trying to find another authorized user to the unattend task without considering other unattended tasks. If no feasible user exists, then the whole search at the current level fails and has to move back to the next parent node. The selection of a task and a user is called branching. There are many branching heuristics. Our general rule is to select the task with more restrictions (e.g., more associated constraints, less authorized users) and the user with more freedom (e.g, less assignments, more authorizations). Branching rules indeed play an important role in a search tree algorithm. We will have a more detailed discussion later.
- Lines 5-8 state that the algorithm backtracks, if no satisfying assignment can be found. Again, note that at each level only one task is selected. If we failed in finding a user for it, we do not consider other unattend task at this level, because the search at the current level is doomed to fail. In such a case, the search moves back to the next parent node. Note that this papers only uses the chronological backtracking strategy, which is also used in the DBLL algorithm. A non-chronological

backtracking strategy, which allows to jump to a higher level by learning the traversed route and has been used in the modern SAT solvers, might be used to reduce the searching time. Non-chronological backtracking may be beneficial to our problems. But we leave the study in the future work.

- At line 9, the *Propagate()* function propagates the picked perturbation. As *PickPerturbation()* is to fix a performing constraint conflict, i.e. assigning an authorized user to an unperformed task, such a perturbation decision may cause BoD and SoD constraint conflicts. In such cases, the algorithm makes more perturbations to fix the cascaded BoD and SoD constraint conflicts. If a BoD (or SoD) constraint b_{ijkl} (or s_{ijkl})is violated, the assignments of $x_{ij} = 1$ and $x_{kl} = 1$ are canceled. Note that the *Propagate()* function does not resolve further cascaded performing constraint conflicts. The *Propagate* function is similar to the unit propagation procedure used in the SAT solvers.
- At lines 10-12, the *IsConflict()* function checks whether the new perturbations cause a conflict. In particular, we check two types of conflicts. One is the number of the total perturbations made so far. If it exceeds the maximum accepted number δ, then the search along this route has failed. The other one is that whether any new perturbation changes a previous perturbation decision. If so, then the search has failed also, because a loop has occurred. When either case happens, *NeighborSearch()* is called to find another authorized user to the picked unattended task.
- At lines 13-14, the *IsSatisfied()* function checks whether the new perturbations make a satisfying workflow management system.
- Lines 16-18 are executed when the evaluation of the *IsSatisfied()* function is FALSE. $PList$ is updated by including new perturbations and then the search continues.

5.3 Problem-Specific Configurations

All studied dynamic workflow adjustment problems can adopt algorithm 1 with some problem-specific configurations. To demonstrate how it works, we will run it on a toy online pharmacy example. There are 7 task, T1 - T7: order entry, credit check, fulfil order, medication assessment, shipping, billing, and update legers respectively. There are four roles: P (pharmacist), T (technical staff), N (non-technical staff), and A (accountant). The task-role relation and personnel assignments are written in Table 1. There are 4 employees: David, Sam, John, and Eva. There are a SoD constraint on T2 and T6, a SoD constraint on T3 and T6, and a BoD constraint on T6 and T7. All these are depicted in Figure 1.

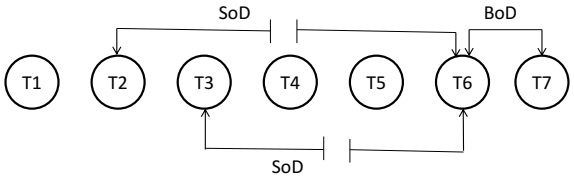

Fig. 1. An Order Fulfillment Process Diagram for an Online Pharmacy

Absence of User. For problem 1, we can directly apply algorithm 1 by setting the starting perturbation list $PList$ as $\{\forall j, X_{ij} : 1 \rightarrow 0\}$, where i is the absent user. $PList$ is a set of assignment deletion and does not cause any other types of constraint conflicts, except performing constraint conflicts. So $PList$ satisfies the algorithmic requirement. For illustration, suppose Eva is absent and the maximum accepted number of perturbations is 3. An algorithm execution example is shown in Figure 2. At level 1, the assignment of T6 and T7 are empty, because Eva is absent and she is the only user who was performing them. At level 2, we pick the unattended task T7 first and then assign John it. As a result, T6 has to be assigned to John due to the BoD constraint on T6 and T7. As it propagates, the assignment of John to T3 is deleted, because of the SoD constraint on T6 and T3. At this point, other than performing constraint conflicts, there is no other type of constraint conflict. So it proceeds to level 3. By selecting T3 and assigning Sam to it, we find a satisfying workflow assignment, as all tasks are attended by at least one authorized employee, no constraint conflict exists, and the number of total perturbations is 3.

Table 1. Task-Role Relation and Personnel Assignment

	T1	T2	T3	T4	T5	T6	T7
Associated Roles	PTNA	PT	PTA	P	PTNA	PTNA	PTA
Assignment	David (N)	Sam (T)	John (P)	John (P)	David (N)	Eva (A)	Eva (A)

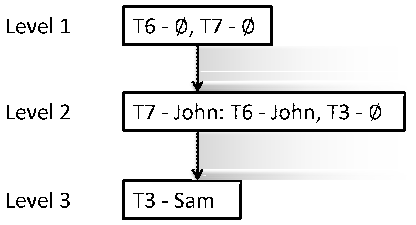

Fig. 2. Absence of Eva

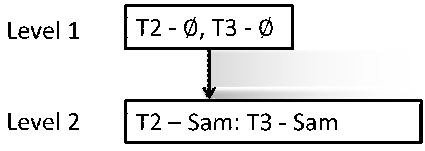

Fig. 3. Addition of a BoD Constraint (T2, T3)

Addition of BoD Constraint. Algorithm 1 can also be applied to the case when a BoD constraint is added. A BoD constraint b_{ijkl} requires $x_{kl} = 1$ if $x_{ij} = 1$. b_{ijkl} causes a conflict when $x_{ij} = 1$ and $X_{kl} = 0$ in the old system. To resolve a BoD constraint conflict, a simple way is to delete both assignments. It may lead to performing constraint conflicts. If so, we can simply call algorithm 1 by making $PList$ as $\{x_{ij} : 1 \rightarrow 0, x_{kl} : 1 \rightarrow 0\}$, which only causes performing constraint conflicts.

Suppose a BoD constraint on T2 and T3 is added to Figure 1. An algorithm execution example is shown in Figure 3. At level 1, the assignments of T2 and T3 are set as empty. At level 2, select T2 and assign Sam to T2, as Sam has authorization to T2. Due to the BoD constraint on T2 and T3, Sam is assigned to T3 as well, which constitutes a satisfying workflow system. Note that a branching rule indeed is very important for the algorithm efficiency. As if we assign other authorized user to T2, it may cause conflicts and then have to return to make another selection. We will discuss branching rules and how to use them to improve algorithm efficiency later.

Addition of SoD Constraint. Algorithm 1 can also apply to the case when a SoD constraint is added with some simple configurations. A SoD constraint s_{ijkl} requires $x_{ij} = 1$ and $x_{kl} = 1$ cannot both hold. So addition of s_{ijkl} may cause a SoD constraint conflict if $X_{ij} = 1$ and $X_{kl} = 1$ both hold in the old system. To resolve the conflict, we simply delete assignments of $x_{ij} = 1$ and $x_{kl} = 1$, and then directly adopt algorithm 1 by making $PList$ as $\{x_{ij} : 1 \to 0, X_{kl} : 1 \to 0\}$.

To illustrate it, suppose a SoD constraint on T3 and T4 is added to Figure 1 and the maximum accepted number of perturbations is 3. An algorithm execution example is shown in Figure 4. At level 1, the assignments of T3 and T4 are set as empty. At level 2, John is assigned to the unattended task T4. At level 3, Eva is assigned to T3 first. As a result, the assignment of Eva to T6 is deleted because of the SoD constraint on T3 and T6, and then the assignment of Eva to T7 is deleted because of the BoD constraint. Since the number of the currently total perturbations exceeds 3, the assignment of Eva to T3 fails and the searching goes back. Alternatively, Sam is assigned to T3, which constitutes a satisfying workflow system.

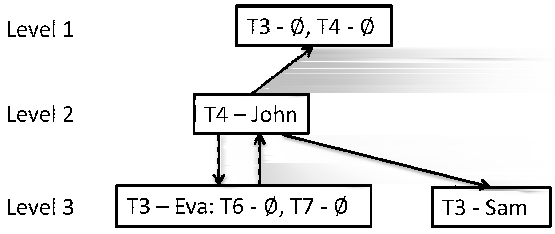

Fig. 4. Addition of a SoD Constraint (T3, T4)

Revocation of Authorization. Problem 4 can directly adopt algorithm 1 by making $PList$ as $x_{ij} : 1 \to 0$, when the authorization of $a_{ij} = 1$ is revoked, which only causes performing constraint conflicts.

To illustrate it, suppose John is forbidden from accessing T3, due to his frequent mistakes on fulling orders, despite his role allows him to access T3. Again, we assume the maximum accepted number of perturbations is 3. An algorithm execution example is shown in Figure 5. At level 1, the assignment of T3 is set as empty due to the authorization revocation. At level 2, Eva is picked and assigned to T3. As a result, the assignment

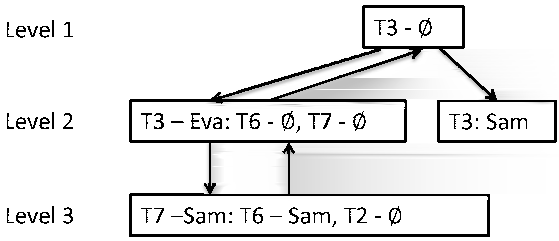

Fig. 5. Revocation of John - T3

of Eva to T6 becomes illegal and is deleted due to the SoD constraint on T3 and T6. Furthermore, the assignment of Eva to T7 has to be deleted due to the BoD constraint on T6 and T7. Then the execution proceeds to level 3. As T6 and T7 are unattended, Sam is selected and assigned to T7. As a result, Sam is assigned to T6 due to the BoD constraint on T6 and T7, and then the assignment of Sam to T2 has to be deleted due to the SoD constraint on T2 and T7. At this point, the number of total perturbation has exceeds 3. So the assignment of Sam to T2 fails. Since the assignment of Eva to T7 was changed at the upper level and is tabued at the current level, so there is no feasible assignment for T7. Then the search moves back. Then, Sam is selected and assigned to T3, which constitutes a satisfying workflow system.

5.4 Branching Heuristics

Branching plays an important role in a search tree algorithm. Indeed, CDCL and DPDL usually refer to a set of algorithms with different branching heuristics. A good branching rule may find a satisfying solution quickly. For instance, consider the example of Figure 5. If we assign Sam to T3 in the first place, then we obtain a satisfying workflow assignment immediately without wasting time on searching other branches. By playing with some synthetic data sets, we find that that in order to find a satisfying solution quickly, a good branching strategy should prioritize an unattended task with more restrictions and a user with more freedom. In particular, we sum up the following experiences.

- *Pick a task with less authorized users.* The motivation is to narrow search space. If there are a few authorized users, then we only need to consider those few options. For instance, considering Figure 2, when Eva is absent and leaves T6 and T7 unattend, we pick T7 over T6. As there are fewer assignment options available for T7, it will be quick for us to reach a conclusion whether the current branch is satisfiable.
- *Pick a task with more associated constraints.* The motivation is also to narrow search space. If we pick a task with more associated constraints, when a user is assigned to the picked task, those associated constraints may be triggered and thus many other personnel assignments are determined through propagation. So we can quickly conclude whether this rout is satisfying.

- *Pick a user with less assignments.* The motivation is to increase the chance of constituting a satisfying solution. By picking a user with less assignments, it would be less likely that the assignment causes conflicts later.
- *Pick a user with less associated constraints.* The motivation is the same as above. By picking a user with less associated constraints, we can assign that person more freely and expect less conflicts later.

Note that algorithm 1 can also be applied to the complex cases that many types of changes occur the same time. As Algorithm 1 requires the beginning perturbation list $PList$ causes performing constraint conflicts only, so we only need to resolve other types of constraint conflicts firstly, e.g. for addition of s_{ijkl} (or b_{ijkl}) delete assignments of $x_{ij} = 1$ and $x_{kl} = 1$.

6 Related Work

The work most related to ours is Basin et al. [5], which studies the optimal workflow adjustment problem. One main difference is that we study the optimal workflow adjustment problem with respect to each specific type of changes, e.g. SoD and BoD, and consider the minimum perturbation to the old workflow system as the objective. Another difference is in the approach to solve the problem. We not only provide SAT model formulations for the studied problems, but also present customized algorithms by modifying the state-of-art SAT algorithms, which should have more practical importance than model formulations, in particular given that the workflow adjustment problems are NP-hard in nature. Another work that is related to ours is the workflow resiliency problem introduced by Wang and Li [17]. They study whether a workflow can be executed successfully if a given number of users is unavailable. Their problem can be viewed as a special case of our dynamic adjustment problem, as they consider only one specific type of changes and have no constraint on the amount of perturbations. In [6], Crampton was the first to study the decision problem whether an allocation of users to tasks exists for a given workflow such that an authorization policy is satisfied . In [17], Wang and Li call it the workflow satisfiability problem and prove it is NP-complete for their authorization model. Some papers, e.g. [3], consider the different delegation models for workflows, which allow the assignment of access rights available to a user to another user. Some papers study the characteristics of SoD and BoD and their impacts the design of an access control system, e.g. [8].

7 Conclusion

In this paper, we formulated and studied the optimal dynamic workflow adjustment problems with respect to various type of changes, including absence of a user, addition of a SoD constraint, addition of a BoD constraint, and revocation of an authorization. We provide SAT model formulations to the studied problems. In addition, we provide customized algorithms inspired by the state-of-art SAT algorithms.

References

1. Appel, K., Haken, W.: Every planar map is four colorable. Illinois Journal of Mathematics 21(3), 429–567 (1977)
2. Atluri, V., Chun, S.A., Mazzoleni, P.: A chinese wall security model for decentralized workflow systems. In: Proceedings of the 8th ACM Conference on Computer and Communications Security, CCS 2001, pp. 48–57. ACM, New York (2001)
3. Atluri, V., Warner, J.: Supporting conditional delegation in secure workflow management systems. In: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, SACMAT 2005, pp. 49–58. ACM, New York (2005)
4. Bai, X., Gopal, R., Nunez, M., Zhdanov, D.: On the prevention of fraud and privacy exposure in process information flow. INFORMS J. on Computing 24(3), 416–432 (2012)
5. Basin, D., Burri, S.J., Karjoth, G.: Optimal workflow-aware authorizations. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, SACMAT 2012, pp. 93–102. ACM, New York (2012)
6. Crampton, J.: A reference monitor for workflow systems with constrained task execution. In: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, SACMAT 2005, pp. 38–47. ACM, New York (2005)
7. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
8. Li, N., Tripunitara, M.V., Bizri, Z.: On mutually exclusive roles and separation-of-duty. ACM Trans. Inf. Syst. Secur. 10(2) (May 2007)
9. Li, N., Tripunitara, M.V., Wang, Q.: Resiliency policies in access control. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, pp. 113–123. ACM, New York (2006)
10. Lu, H., Vaidya, J., Atluri, V.: Optimal boolean matrix decomposition: Application to role engineering. In: IEEE 24th International Conference on Data Engineering, pp. 297–306 (2008)
11. Lu, H., Vaidya, J., Atluri, V., Hong, Y.: Constraint-aware role mining via extended boolean matrix decomposition. IEEE Transactions on Dependable and Secure Computing 9(5), 655–669 (2012)
12. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: Proceedings of the 38th Annual Design Automation Conference, DAC 2001, pp. 530–535. ACM (2001)
13. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer 29(2), 38–47 (1996)
14. Silva, J.A.P.M., Sakallah, K.A.: Grasp: a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, pp. 220–227. IEEE Computer Society (1996)
15. Stockmeyer, L.: Planar 3-colorability is polynomial complete. SIGACT News 5(3), 19–25 (1973)
16. Vaidya, J., Atluri, V., Guo, Q., Lu, H.: Edge-rmp: Minimizing administrative assignments for role-based access control. Journal of Computer Security 17(2), 211–235 (2009)
17. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM Trans. Inf. Syst. Secur. 13(4), 40:1–40:35 (2010)