Algorithmic Problems in Access Control

by

Nima Mousavi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Access control is used to provide regulated access to resources by principals. It is an important and foundational aspect of information security. Role-Based Access Control (RBAC) is a popular and widely-used access control model, that, as prior work argues, is ideally suited for enterprise settings. In this dissertation, we address two problems in the context of RBAC.

One is the User Authorization Query (UAQ) problem, which relates to sessions that a user creates to exercise permissions. UAQ's objective is the identification of a set of roles that a user needs to activate such that the session is authorized to all permissions that the user wants to exercise in that session. The roles that are activated must respect a set of Separation of Duty constraints. Such constraints restrict the roles that can be activated together in a session. UAQ is known to be intractable (**NP**-hard). In this dissertation, we give a precise formulation of UAQ as a joint-optimization problem, and analyze it. We examine the manner in which each input parameter contributes to its intractability. We then propose an approach to mitigate its intractability based on our observation that a corresponding decision version of the problem is in **NP**. We efficiently reduce UAQ to Boolean satisfiability in conjunctive normal form (CNF-SAT), a well-known **NP**-complete problem for which solvers exist that are efficient for large classes of instances. We also present results for UAQ posed as an approximation problem; our results suggest that efficient approximation is not promising for UAQ. We discuss an open-source implementation of our approach and a corresponding empirical assessment that we have conducted.

The other problem we consider in this dissertation regards an efficient data structure for distributed access enforcement. Access enforcement is the process of validating an access request to a resource. Distributed access enforcement has become important with the proliferation of data, which requires access control systems to scale to tens of thousands of resources and permissions. Prior work has shown the effectiveness of a data structure called the Cascade Bloom Filter (CBF) for this problem. In this dissertation, we study the construction of instances of the CBF. We formulate the problem of finding an optimal instance of a CBF, where optimality refers to the number of false positives incurred and the number of hash functions used. We prove that this problem is **NP**-hard, and a meaningful decision version is in **NP**. We then propose an approach to mitigate the intractability of the problem by reducing it to CNF-SAT, that allows us to use a SAT solver for instances that arise in practice. We discuss an open-source implementation of our approach and an empirical assessment based on it.

Acknowledgements

I would like to express my deepest appreciation to Prof. Mahesh Tripunitara for generously offering his time, support and guidance throughout my PhD. Your advice on both research as well as on my career have been priceless.

I am grateful to Prof. Sylvia Osborn, Prof Jochen Koenemann, Prof. Stephen Smith, Prof Shreyas Sundaram, and Prof. Nick Wormald for serving on my PhD Committee. Your suggestions helped me to improve my dissertation.

I owe my deepest gratitude to my parents. Words cannot express how grateful I am to you for all of your love and support.

Lovingly dedicated to my parents.

Table of Contents

List of Figures

Chapter 1

Introduction

Access control is a fundamental security mechanism that is used extensively in computing systems. It is concerned with regulating actions such as read and write, by users to resources. An entity that initiates access is called a subject or a user. The entity to which access is requested is called an object or a resource. The manner in which a subject desires to access an object (e.g., read, write or execute) is called an access right. An access control system makes its decision based on a protection state. The protection state is expressed using some syntax. A natural syntax is an access matrix [22, 25]. Figure 1.1 shows an example of an access matrix with rows of subjects and columns of objects. The entry (s, o) in the matrix specifies the rights that subject s has to object o. The term permission is often used for an access right and an object pair.

	Budget	Hire	Layoff	Pay	Invoice
Alice	R,W			R,W	R
Bob	R,W	R,W	R,W	R,W	

Figure 1.1: An example of access matrix.

1.1 Role Based Access Control

Role Based Access Control (RBAC) is a syntax for expressing an access control policy. It introduces a layer of roles between users and permissions. Roles typically represent responsibility and authority within a given enterprise. Users acquire all permissions associated with the roles to which they are authorized. An intuition behind RBAC is that roles are relatively stable, while users and permissions may change rapidly, and so RBAC can provide a stable and flexible access control policy.

RBAC is one of the most popular access control models [21, 31, 51] and a de-facto standard for access control in enterprise settings. According to the Research Triangle Institute, the majority of users in enterprises of 500 or more use RBAC [42]. Many IT vendors, including IBM, Microsoft, Oracle, Sybase, and Siemens have developed products based on RBAC.

Roles in RBAC can be related via a hierarchy which defines inheritance semantics using a role-role relation [36, 51, 47]. An RBAC policy is characterized by the triple $\langle UA, PA, RH\rangle$. UA is the assignment of users to roles, and PA is the assignment of roles to permissions. RH, the role hierarchy, is a partial order of roles. If $\langle r_1, r_2\rangle \in RH$, we say that r_1 inherits r_2. The semantics of inheritance can vary (see Section 2.3 in Chapter 2). The simplest semantics is that every user that is authorized to r_1 is authorized also to r_2. Every permission to which r_2 is authorized is also authorized to r_1. Figure 1.2 shows an example of RBAC.

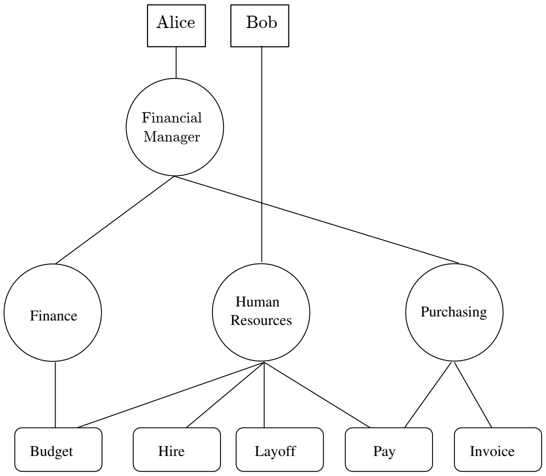


Figure 1.2: An example of RBAC for two users, Alice and Bob. Rectangles are users, circles are roles, and rounded rectangles are permissions. Alice is authorized to role *Financial Manager* and Bob is authorized to role *Human Resources*. The RBAC is characterized by the triple $\langle UA, PA, RH\rangle$, where $UA = \{\langle Alice, FinancialManager\rangle, \langle Bob, HumanResources\rangle\}$, $PA = \{\langle Finance, Budget\rangle, \langle HumanResources, Budget\rangle, \langle HumanResources, Hire\rangle, \langle HumanResources, Layoff\rangle, \langle HumanResources, Pay\rangle, \langle Purchasing, Pay\rangle, \langle Purchasing, Invoice\rangle\}$, and $RH = \{\langle FinancialManager, Finance\rangle, \langle FinancialManger, Purchasing\rangle\}$

1.2 The User Authorization Query problem

In RBAC, a user exercises permissions by activating the roles to which he is authorized. However, there is a cost associated with the activation of each set of roles. The cost is related to the number of roles that need to be activated so the user indeed acquires the permissions he requires, and the extra permissions that the user is forced to acquire upon activating that set of roles. There are also Separation of Duty (SoD) constraints, which restrict the set of roles that are allowed to be activated in the same session [16, 53, 7, 55, 21]. Given a user and a set of permissions, User Authorization Query (UAQ) is the problem of finding an optimal subset of roles to which the user is authorized such that he acquires all the permissions he wants and respects all the SoD constraints. We define UAQ formally in Section 2.1. UAQ is the first problem we study in this dissertation.

1.3 The Cascade Bloom Filter for Access Enforcement

In RBAC, access enforcement is the process by which a trusted entity, called a reference monitor, allows or rejects an access request to a resource by a user. Efficient access enforcement is a meaningful and established problem in access control. Prior work has shown that an approach to access enforcement based on a data structure called the cascade Bloom filter is fast and space-efficient. However, such prior work leaves open the problem of constructing instances of the data structure. We give a formal definition for this construction problem in Section 3.3. There are two natural trade-offs in the construction of a cascade Bloom filter: the number of false positives it incurs, and the number of hash functions it employs. An optimal cascade bloom filter is a cascade Bloom filter which minimizes the number of false positives and the number of hash functions. In this dissertation, we focus on designing optimal cascade Bloom filters for the purpose of access enforcement.

1.4 Thesis Statements

- Although UAQ is computationally hard and admits no efficient approximation algorithm (assuming that $\mathbf{P} \neq \mathbf{NP}$), there exist effective ways to mitigate its intractability in practice.

  Our approach to proving the above thesis is by construction — we show how each parameter contributes to the computational complexity of UAQ and present algorithms that solve UAQ in polynomial time given some conditions on the input parameters. We present also a general solution to UAQ based on an efficient reduction to CNF-SAT.

- The problem of constructing optimal Cascade Bloom Filters for access enforcement of RBAC policies is **NP**-hard, and a corresponding decision version is in **NP**. Large classes of problem instances that arise in practice can be solved efficiently.

  Our approach to proving the above assertions is to present corresponding proofs, and by construction. We present an algorithm for finding an optimal CBF based on a reduction to CNF-SAT.

1.5 Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we address UAQ. We present new results on the computational complexity of UAQ and show how the intractability of UAQ can be mitigated. In Chapter 3 we present new results on the computational complexity of finding optimal Cascade Bloom Filters. We show also how the intractability of CBF can be mitigated in practice. We conclude with Chapter 4.

Chapter 2

The User Authorization Query Problem in RBAC

In this chapter, we study the User Authorization Query problem (UAQ) in Role-Based Access Control (RBAC) [18, 59, 64]. UAQ arises in the context of RBAC sessions [21, 51], and is known to be intractable [12, 18]. We first present a comprehensive definition for UAQ in Section 2.1. We then present new results on the complexity of UAQ in Section 2.3. We address how the intractability of UAQ can be mitigated (Section 2.4). Finally, we discuss an empirical evaluation of our approach (Section 2.5).

2.1 Introduction

In RBAC, permissions are not assigned directly to users. Rather, a user is assigned to roles, which in turn are assigned permissions. A user is authorized to those permissions that are assigned to the roles to which he is authorized. In Figure 2.1, for example, a user is assigned to three roles, and is authorized to five permissions via those roles.

If a user wants to exercise a permission, he must first create a session [21, 51]. He specifies the roles he would like associated with a session when creating it. A set of Separation of Duty (SoD) constraints[§], D, may be imposed on him. Each such constraint is of the form $\langle R', t \rangle$, where R' is a set of roles and t is an integer.

[§] Some prior work [32] calls these “mutually exclusive role constraints” to clearly distinguish the security objective (separation of duty) from the enforcement mechanism. We call them SoD constraints to be consistent with prior work on UAQ [18, 59, 64].

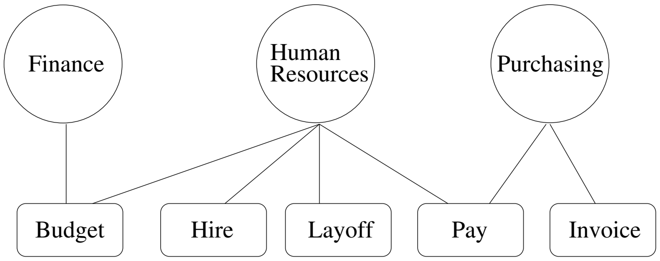


Figure 2.1: An example of an RBAC policy for a user. Circles are roles and rectangles are permissions.

The semantics of such a constraint is that t or more roles from the set R' are not allowed to be activated in any session. In the example in Figure 2.1, the constraint $\langle\{\text{Finance}, \text{Human Resources}, \text{Purchasing}\}, 3\rangle$ precludes the user from activating all three roles in the same session.

UAQ is an optimization problem with two distinct objectives. The user has a set of permissions to which he wants the session to be authorized. However, a set of roles that give him exactly the permissions he seeks may not exist [64]. Consequently, the permissions that a user wants are specified as two sets [59]: a lower bound set P_{lb} and an upper bound set $P_{ub} \supseteq P_{lb}$. The permissions in P_{lb} are those to which the session must be authorized, and $P_{ub} - P_{lb}$ is a "slack" or extra set of permissions to which the session may be authorized. The user may wish to either minimize or maximize the number of extra permissions to which his session is authorized. His choice depends on which of two security objectives, safety or availability, he wants to prioritize over the other. If he prioritizes safety, then he would want to minimize the number of permissions from $P_{ub} - P_{lb}$. If he prioritizes availability, he would want to maximize the number of permissions from $P_{ub} - P_{lb}$. In the latter case, he still has safety in that the permissions cannot exceed P_{ub}.

Apart from the number of extra permissions that is an optimization objective as we discuss above, the number of roles that are activated is another optimization objective. We may want, for example, to minimize the number of roles that are activated in the session. This may be an important consideration for the system; minimal sets of roles in sessions may be more efficient for the system to support. It has been argued [64] that we may instead want to maximize the number of roles to which a session is authorized. As prior work [12] points out, from the standpoint of computational complexity, allowing both options of minimization and maximization is no more difficult than allowing minimization only. (We point out, however, that the assertion in [12] that constraints do not impact the computational complexity of UAQ is not true. For example, with the introduction of constraints, if **P** $\neq$ **NP**, the subcase of UAQ that maximizes the number of extra permissions is intractable.)

UAQ Specification An instance of UAQ is specified by the following inputs.

- An RBAC policy, ρ, for a user, where $\rho = \langle R, P, RH, RP \rangle$, where RH is a role-hierarchy that relates roles in a partial order, and RP is an assignment of roles to permissions. The role hierarchy RH is reflexive, i.e., given a role r that appears in ρ, $\langle r, r \rangle \in RH$. We denote the set of all roles in ρ as R, and the set of all permissions as P.

- A set of SoD constraints, D, each of the form $\langle R_c, t_c \rangle$, where $t_c \in [1, |R_c|]$, $R_c \subseteq R$.

- Two sets of permissions, P_{lb} and P_{ub}, with $P \supseteq P_{ub} \supseteq P_{lb}$.

- An optimization objective for roles, $o_r \in \{\text{min}, \text{max}, \text{none}\}$. The option "none" means that we do not care to optimize the number of roles in a solution.

- An optimization objective for extra permissions, $o_p \in \{\text{min}, \text{max}, \text{none}\}$. If $P_{ub} = P_{lb}$, we assume $o_p = \text{none}$.

- A priority, $pri \in \{n_r, n_p\}$; pri indicates which of the two optimization objectives, roles or extra permissions, we prioritize over the other. If $pri = n_r$ ($pri = n_p$ resp.), we prioritize optimization of the number of roles (number of permissions resp.). The need for this arises from our two distinct optimization objectives. As we point out below in Example 2, without this parameter, there can exist two optimal solutions that are incomparable to one another. This parameter is meaningful only if $o_r \neq$ none and $o_p \neq$ none. If $o_r =$ none and $o_p \neq$ none, then $pri = n_p$, and if $o_p = \text{none}, o_r \neq$ none, then $pri = n_r$. If both are none, then this input is ignored.

Prior work [64] also considers what are called cardinality constraints as an input. However, as subsequent work [59] points out, these can be addressed prior to formulating a UAQ instance. An instance may be associated with no valid solutions, or one or more valid solutions. A valid solution is $\langle R_s \rangle$, where $R_s \subseteq R$

is a set of roles that satisfies every constraint in D, and the set of permissions $P_s \subseteq P$, to which the roles in R_s are authorized, is such that $P_{ub} \supseteq P_s \supseteq P_{lb}$. If $o_r =$ min (max), then $|R_s|$ is the minimum (maximum) possible number of roles. If $o_p =$ min (max), then $|P_s - P_{lb}|$ is the minimum (maximum) possible number of extra permissions.

More than one valid solution can exist.

Example 1. *Consider the RBAC policy for a user, ρ, from Figure 2.1. Let $D = \{\langle\{\mathit{Human\ Resources}, \mathit{Purchasing}\}, 2\rangle\}$; that is, the only SoD constraint is that the roles Human Resources and Purchasing are not allowed to be activated in the same session. Let $P_{lb} = \{\mathit{Pay}\}$. We consider several example requests with different values for the other parameters.*

- *$o_p = \mathit{min}, P_{ub} = P_{lb}$ and any values for the other inputs: has no valid solutions because any choice for R_s results in more permissions than Pay only.*

- *$o_p = \mathit{min}, P_{ub} = P_{lb} \cup \{\mathit{Hire}, \mathit{Invoice}\}$ and any values for the other inputs: the only valid solution is $R_s = \{\mathit{Purchasing}\}$, with one extra permission, Invoice.*

- *$o_p = \mathit{max}, o_r = \mathit{min}, pri = n_p, P_{ub} = P_{lb} \cup \{\mathit{Budget}, \mathit{Hire}, \mathit{Layoff}, \mathit{Invoice}\}$: the only valid solution is $R_s = \{\mathit{Human\ Resources}\}$.*

- *If we change pri to n_r in the previous example request, the valid solution remains $R_s = \{\mathit{Human\ Resources}\}$. We choose Human Resources over Purchasing, even though both result in only 1 role, because of the secondary requirement of maximizing extra permissions.*

Example 2. *This example illustrates the difference that the input pri makes. Suppose ρ is the policy from Figure 2.1, $P_{lb} = \{\mathit{Budget}, \mathit{Pay}\}$, P_{ub} is all the permissions from the figure, $D = \emptyset$, $o_p = \mathit{min}$ and $o_r = \mathit{min}$.*

- *If $pri = n_r$, then our solution is $\{\mathit{Human\ Resources}\}$. We have only 1 role and 2 extra permissions (Hire and Layoff).*

- *If $pri = n_p$, then our solution is $\{\mathit{Finance}, \mathit{Purchasing}\}$. We have 2 roles and only 1 extra permission (Invoice).*

If we do not have the input pri, the solutions would both be valid, but incomparable to one another.

Layout In the next section, we discuss related work. In that context, we discuss also soundness, efficiency and joint-optimization issues with prior approaches. In Section 2.3, we present our results on the computational complexity of UAQ. In Section 2.4, we discuss how the intractability of UAQ can be mitigated. In Section 2.4.1, we view UAQ from the standpoint of efficient approximation, and present related results. In Section 2.4.3, we reduce UAQ to CNF-SAT. In Section 2.4.4, we discuss an algorithm for UAQ that is fixed-parameter polynomial-time. Finally, we present empirical results in Section 2.5.

2.2 Related Work

To our knowledge, UAQ was first posed by Du and Joshi [18]. That work shows also that the subcase of optimizing roles is **NP**-hard and the corresponding decision version is in **NP**. That work does not consider the SoD constraints or the optimization of extra permissions. Zhang and Joshi [64] generalize UAQ by introducing the problem of optimizing the number of extra permissions in addition to the number of roles. Wickramaarachchi et al. [59] propose two approaches for mitigating the intractability of UAQ. That work also proposes a different combination of mixing the optimization of number of roles and permissions than Zhang and Joshi [64]. Chen and Crampton [12] consider the subcase of UAQ that is the optimization of permissions as a problem that is related to a variant of the well-known set cover problem [24]. Chen and Crampton's work [12] establishes that the minimization version of the subcase it considers is **NP**-hard and the maximization version is in **P**. It does not consider SoD constraints or the optimization of roles.

To our knowledge, the pieces of prior work that consider the mitigation of the intractability of UAQ are those of Du and Joshi [18], Zhang and Joshi [64], Wickramaarachchi et al. [59] and Armando et al. [3]. The issues with the approaches of Du and Joshi [18] and Zhang and Joshi [64] are identified by Wickramaarachchi et al. [59] and we refer the reader to that work for a comprehensive discussion. We provide a summary here. The work of Du and Joshi [18] considers only the problem of role minimization in the absence of constraints. The work of Zhang and Joshi [64] first proposes a greedy algorithm that is not sound [59]. Their approach to dealing with the unsoundness renders the algorithm inefficient; it is exponential-time in its design (i.e., their reduction of UAQ to CNF-SAT is exponential).

The work of Armando et al. [3] points out, based on empirical observations, that the approach of Wickramaarachchi et al. [59] is inefficient. They then propose a complementary approach of caching session information for greater efficiency. That work does not identify the inefficiency inherent to the approach of Wickramaarachchi et al. [59] as we do below. Their work is complementary to our work. The work of Wickramaarachchi et al. [59] proposes two approaches to mitigate the intractability of UAQ. Both leverage prior work on Boolean satisfiability (SAT). The first approach is an algorithm similar to an earlier algorithm for deciding SAT instances. The algorithm is exponential-time in its design. This analytical observation is validated by their empirical observations, which show that this first approach of theirs is inefficient in comparison to the second. The approach also does not address the joint optimization.

The second approach proposes a mapping to CNF-SAT (SAT instances in Conjunctive Normal Form). The problem of deciding whether a Boolean expression in CNF is satisfiable is known to be **NP**-complete [24]. Unfortunately, this second approach is unsound, inefficient and is limited in the manner in which the joint optimization is addressed. The unsoundness arises from the manner in which the RBAC policy ρ is mapped. Suppose $r \in R$ is a role such that $\{\langle r, r_1 \rangle, \ldots, \langle r, r_k \rangle\} \subseteq RH$, and $\{\langle r, p_1 \rangle, \ldots, \langle r, p_m \rangle\} \subseteq RP$. Then the approach proposes that we encode this as a clause $r \leftrightarrow r_1 \wedge \ldots \wedge r_k \wedge p_1 \wedge \ldots \wedge p_m$, where "$\leftrightarrow$" is "if and only if" and "$\wedge$" is conjunction. (We abuse notation slightly in our use of r_i, p_j as both roles and permissions, and Boolean variables that correspond to them.)

The problem with this is that there can exist UAQ instances for which the approach incorrectly determines that there is no valid solution. Given a constraint $\langle R, t \rangle$, if a permission $p \in P_{lb}$ is assigned to t or more roles in R, then the approach would deem that there is no valid solution, which is incorrect. This problem is quite general, and not some small corner-case. For every constraint $\langle R, t \rangle$, there exists an infinite number of UAQ instances for which the approach is unsound.

One may be tempted to adopt a "quick fix" to address the above problem. For example, one may change the "if and only if" to an "only if" only. This will not work. The reason is that the manner in which the joint optimization is addressed by Wickramaarachchi et al. [59] relies on the "if and only if." We discuss this further below in the context of the limited support for joint optimization in that work.

The approach of Wickramaarachchi et al. [59] is inefficient (other than the inherent inefficiency due to the **NP**-hardness of UAQ). The inefficiency arises from the manner in which constraints are encoded. Given a constraint $\langle R, t \rangle$, the approach first enumerates every t-sized subset of R. Suppose $\{r_1, \ldots, r_t\}$ is such a subset. The approach encodes the subset as a clause $\neg r_1 \vee \ldots \neg r_t$. The problem with this is that it results is an exponential blowup in its design. As with unsoundness, there is an infinite number of classes of UAQ instances, with infinite members in each, for which this approach causes an exponential blowup. An example is when t is polynomial in $|R|$, e.g., $t = \sqrt{|R|}$.

Finally, the approach offers only limited support for the joint optimization of the numbers of roles and extra permissions. This issue is related to the unsoundness above. The manner in which joint optimization is supported is using the notion of relaxable and non-relaxable clauses. A clause is said to be non-relaxable if in a truth-assignment, that clause must evaluate to 1; otherwise it is relaxable. There exist SAT solvers that, in addition to meeting the constraint on non-relaxable clauses, maximize the number of relaxable clauses that evaluate to 1 in a truth-assignment.

The idea in Wickramaarachchi et al. [59] to minimize the number of roles, then, is to add a clause $\neg p$ for each $p \in P_{ub} - P_{lb}$ and specify that every such clause is relaxable. To maximize the number of roles, we add a clause p (rather than $\neg p$) and specify those to be relaxable. In conjunction with the "if and only if" clauses we discuss above in the context of unsoundness, this ensures that the number of roles is maximized if the number of extra permissions is maximized, and the number of roles is minimized if the number of extra permissions is minimized. Unfortunately, there is no obvious way to address other combinations, for example, minimize the number of roles and maximize the number of extra permissions.

2.3 Complexity Results

In this section, we discuss the computational complexity of four subcases of UAQ: MIN-UAQ-P (UAQ in which we seek to minimize the number of extra permissions authorized by the solution-set of roles only), MAX-UAQ-P (maximize extra permissions), MIN-UAQ-R (minimize number of roles in the solution-set), and MAX-UAQ-R (maximize roles). We identify the input parameters that contribute to the complexity of each. We show how the complexity of each subcase depends on the input parameters. Our results not only suggest efficient algorithms for some subcases of UAQ, but also give a better understanding of the sources of complexity of UAQ.

Role hierarchy Before we present our complexity results, we discuss different semantics of the role hierarchy RH in an RBAC policy. We explain in Section 2.3.5 how the semantics of the role hierarchy affects the complexity of MAX-UAQ-R.

The semantics of role hierarchy affects the set of roles that a user can activate, and the set of permissions that each role has. We recall that a role r_i is senior to a role r_j if $\langle r_i, r_j \rangle \in RH$. Joshi et al [27] introduce three types of role hierarchies:

1. *permission-inheritance* hierarchy (P-hierarchy)[¶].

2. *activation-inheritance* hierarchy (A-hierarchy).

3. *permission-activation-inheritance* hierarchy (PA-hierarchy)[‖].

In the P-hierarchy, a senior role inherits all the permissions of its junior roles. Under the P-hierarchy, a user assigned to a senior role is not allowed to activate any of its junior roles, unless the user is assigned to them directly. Under the A-hierarchy a user assigned to a senior role is allowed to activate any of its junior roles, but the permissions of the junior roles are not propagated to the senior role. In the PA-hierarchy, a senior role inherits all the permissions of its junior roles, and a user assigned to a senior role is allowed to activate any of its junior roles.

The above three different semantics for the role hierarchy RH can be handled when an instance of UAQ is generated [59]. That is, the set of the roles that the user can activate, and the set of permissions available to each role are modified as follows. The set of roles that the user can activate is the set of roles to which he is directly assigned as well as their junior roles if the A-hierarchy or PA-hierarchy is adopted as the semantics of the role hierarchy. The set of permissions available to each role is all the permissions to which it is authorized directly as well as the permissions of its junior roles if the P-hierarchy or PA-hierarchy is chosen. We assume that the issue with different semantics for role hierarchy is already handled in the generation of the UAQ instance.

Li et al. [31] introduce another semantics for the role hierarchy, which we call the *automatic-activation* hierarchy. In the automatic-activation hierarchy, the activation of a senior role in a session implies the activation of all of its junior roles in that session. The automatic-activation hierarchy can be seen as a fix for the problem of *implicit SoD constraints* in the P-hierarchy and PA-hierarchy. We explain the problem with an example from [59]. Assume that there is an SoD

[¶] Joshi et al. [27] call this the I-hierarchy.

[‖] Joshi et al. [27] call this the IA-hierarchy.

constraint c that does not allow two roles r_1 and r_2 to be activated in the same session, i.e., $c = \langle\{r_1, r_2\}, 2\rangle$. However, the constraint c does not prevent role r_3, a senior role to r_1, and role r_4, a senior role to r_2, from being activated in the same session. Under the P-hierarchy or the PA-hierarchy, a user assigned to r_3 and r_4 can obtain all the permissions of r_1 and r_2 by activating r_3 and r_4 without violating any constraint. Indeed, the constraint c implies three other constraints: $\langle\{r_1, r_4\}, 2\rangle$, $\langle\{r_2, r_3\}, 2\rangle$, and $\langle\{r_3, r_4\}, 2\rangle$.

We adopt the automatic-activation hierarchy as the semantics of the role hierarchy in the remainder of this chapter. We assume also that the user is allowed to activate any role that is junior to a role to which he is authorized. We define two functions $\delta_{RH} : 2^R \to 2^R$ and $\pi_{RH} : 2^R \to 2^R$ to account for the role hierarchy RH.

For any $R' \subseteq R$, $\delta_{RH}(R')$ is the set of all roles that are activated upon activation of the roles in R'. In other words, $\delta_{RH}(R')$ is the set of all roles in R' and their junior roles in RH. The function δ_{RH} is defined formally below.

Definition 1. *For role hierarchy RH and any $R' \subseteq R$, the set $\delta_{RH}(R')$ is the smallest subset of R that satisfies the two following conditions:*

1. *if $r \in R'$, then $r \in \delta_{RH}(R')$.*
2. *if $r \in \delta_{RH}(R')$ and $\langle r, r'\rangle \in RH$, then $r' \in \delta_{RH}(R')$.*

We define $\pi_{RH}(R')$ to be the set of roles in R' and all senior roles of any role in π_{RH}. The function π_{RH} is defined formally below.

Definition 2. *For role hierarchy RH and any $R' \subseteq R$, the set $\pi_{RH}(R')$ is the smallest subset of R that satisfies the two following conditions:*

1. *if $r \in R'$, then $r \in \pi_{RH}(R')$.*
2. *if $r \in \pi_{RH}(R')$ and $\langle r', r\rangle \in RH$, then $r' \in \pi_{RH}(R')$.*

2.3.1 Four NP-hard Problems

In this section, we first give a formal definition for three known **NP**-hard problems: 3-CNF SAT, Vertex Cover, and Set Cover. Then we define the Minimum K-Coverage problem and prove that it is **NP**-hard. The complexity proofs in the following sections rely on these problems.

3-CNF SAT. An instance of 3-CNF SAT is a Boolean formula ϕ that is a conjunction (AND) of clauses, where a clause is disjunction (OR) of three literals. An example for ϕ is $(x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$. The formal language for the 3-CNF SAT problem is

$$\begin{aligned}\text{3-CNF-SAT} = \{\langle \phi \rangle :\ & \phi \text{ is a boolean formula in 3-CNF such that} \\ & \text{there exists a truth assignment for } \phi \\ & \text{that causes every clause to evaluate to } 1\}.\end{aligned}$$

Theorem 1. *[14] 3-CNF-SAT is **NP**-hard.*

Vertex Cover. A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that every edge in E has at least one endpoint in V'. The Vertex Cover problem is to find a vertex cover of minimum size. The formal language for the corresponding decision version is

$$\text{VERTEX-COVER} = \{\langle G, k \rangle : \text{there exists a vertex cover of size at most } k \text{ for } G\}.$$

Theorem 2. *[14] Vertex Cover is **NP**-hard.*

Set Cover. Given a finite universe $\mathcal{U}$ of elements, $e_1, e_2, \ldots, e_n$, and a family $\mathcal{F}$ of subsets, $S_1, S_2, \ldots, S_m \subseteq \mathcal{U}$, a set cover is a $S \subseteq \mathcal{F}$ that covers every element of $\mathcal{U}$, that is, $\bigcup_{S_i \in S} S_i = \mathcal{U}$. The Set Cover problem is to find a set cover of minimum size. The formal language for the decision version of Set Cover is

$$\text{SET-COVER} = \{\langle \mathcal{U}, \mathcal{F}, k \rangle : \text{there exists a set cover of size at most } k\}.$$

Theorem 3. *[14] Set Cover is **NP**-hard.*

Minimum K-Coverage. An instance of Minimum K-Coverage consists of a finite universe $\mathcal{U}$ of elements, $e_1, e_2, \ldots, e_n$, a family $\mathcal{F}$ of subsets, $S_1, S_2, \ldots, S_m \subseteq \mathcal{U}$, and an integer k. The Minimum K-Coverage problem is to find a subset $S \subseteq \mathcal{F}$ such that the size of S is at least k and S minimizes the number of covered elements, i.e., $\bigcup_{S_i \in S} S_i$. The formal language for the corresponding decision problem is

$$\begin{aligned}\text{MIN-K-COVERAGE} = \{\langle \mathcal{U}, \mathcal{F}, k_1, k_2 \rangle :\ & \text{there exists a subset } S \text{ of } \mathcal{F} \text{ such} \\ & \text{that } |S| \geq k_1 \text{ and } |\bigcup_{S_i \in S} S_i| \leq k_2\}.\end{aligned}$$

We prove that Minimum K-Coverage problem is **NP**-hard by proving that a special case of it, Minimum K-Vertex Cover, is **NP**-hard. We define Minimum K-Vertex Cover below.

Minimum K-Vertex Cover is the optimization problem of finding a subset of size at least k of vertices of a given graph $G(V, E)$ that minimizes the number of covered edges. A formal definition for the corresponding decision problem is

$$\text{MIN-K-VERTEX-COVER} = \{\langle G, k_1, k_2 \rangle : \text{there exists a subset } V' \subseteq V \text{ such that the size of } V' \text{ is at least } k_1 \text{ and } |\bigcup_{v \in V'} E_G(v)| \leq k_2, \text{ where } E_G(v) \text{ is the set of edges connected to } v\}.$$

The following theorem proves that an efficient algorithm for MIN-K-VERTEX-COVER is unlikely to exist.

Theorem 4. *Minimum K-Vertex Cover is **NP**-hard.*

Proof. We prove that VERTEX-COVER $\leq_p$ MINIMUM-K-VERTEX-COVER. Let $\phi = \langle G, k \rangle$ be an instance of Vertex Cover. We construct an instance of Minimum K-Vertex Cover, $\psi = \langle G', k_1, k_2 \rangle$, such that ϕ is true if and only if ψ is true. We construct ψ by computing the complement graph $\bar{G}$. The output of the reduction is $\psi = \left\langle \bar{G}, k, \binom{k}{2} + k(|V| - k) - |E| \right\rangle$.

Suppose that G has a vertex cover $V' \subseteq V$ of size (at most) k. We show that V' covers at most $\binom{k}{2} + k(|V| - k) - |E|$ edges in $\bar{G}$, and therefore is a solution for ψ. First note that the total number of edges covered by any subset of vertices in G and $\bar{G}$ is constant, that is, for any $T \subseteq V$, we have

$$\left| \bigcup_{v \in T} E_G(v) \right| + \left| \bigcup_{v \in T} E_{\bar{G}}(v) \right| = \binom{|T|}{2} + |T|(|V| - |T|)$$

where $E_G(v)$ denotes the set of edges connected to v in G. Since V' is a vertex cover in G, we have $|\bigcup_{v \in V'} E_G(v)| = |E|$. Thus, $|\bigcup_{v \in V'} E_{\bar{G}}(v)| \leq \binom{k}{2} + k(|V| - k) - |E|$. Conversely, if ψ is true, then there exists a subset $V' \subseteq V$ of size k that covers at most $\binom{k}{2} + k(|V| - k) - |E|$ edges in $\bar{G}$. Therefore, $|\bigcup_{v \in V'} E_G(v)| \geq \binom{k}{2} + k(|V| - k) - (\binom{k}{2} + k(|V| - k) - |E|) = |E|$ and thus V' is a vertex cover in G. □

Theorem 5. *Minimum K-Coverage is **NP**-hard.*

Proof. Minimum Coverage is **NP**-hard because it is a generalization of Minimum K-Vertex Cover. □

2.3.2 Complexity Results for MIN-UAQ-P

MIN-UAQ-P problem, a subcase of UAQ, is to find a set of roles that minimizes the number of extra permissions while satisfying all the constraints and requirements. As a decision problem, an instance of MIN-UAQ-P is specified by the following inputs.

- RBAC policy, ρ, for a user, where $\rho = \langle R, P, RH, RP \rangle$, where R is the set of all roles in the RBAC policy, P is the set of all permissions in the RBAC policy, RH is a role hierarchy that relates roles in a partial order, RP is an assignment of roles to permissions,
- A set of SoD constraints, D, each of the form $\langle R_c, t_c \rangle$, where $t_c \in [1, |R_c|]$, $R_c \subseteq R$. The semantics of $\langle R_c, t_c \rangle$ is that t_c or more roles from the set R_c are not allowed to be activated at the same time.
- A set of permissions $P_{lb} \subseteq P$
- A set of permissions $P_{ub} \subseteq P$
- $k \in \mathbb{N}$

We say a set of roles R' activates a permission p if there exists at least a role in R' that activates p. The formal language for the corresponding decision problem is

$$\begin{aligned} \text{MIN-UAQ-P} = \{\langle \rho, D, P_{lb}, P_{ub}, k \rangle :\ & \text{there exists a set of roles } R' \subseteq R \text{ such that} \\ & \delta_{RH}(R') \text{ satisfies every constraint in } D, \\ & \text{activates all permissions in } P_{lb}, \\ & \text{no permission from } P - P_{ub}, \\ & \text{and at most } k \text{ permissions from } P_{ub} - P_{lb}\}. \end{aligned}$$

Theorem 6. *[12] MIN-UAQ-P is **NP**-hard.*

We take a finer look at the sources of complexity of MIN-UAQ-P. We show how the size of P_{lb}, the number of SoD constraints ($|D|$), and the role hierarchy (RH) affect the computational complexity of MIN-UAQ-P.

Algorithms 1 Before we present the assertion on the hardness of MIN-UAQ-P, we discuss Algorithm 1 in Figure 2.3 on which some of our proofs in this section

```
CheckD Routine
Input: $\langle \rho, D, S \rangle$
Output: true/false
foreach $\langle R_c, t_c \rangle \in D$ do
    if $|S \cap R_c| \geq t_c$ then
        return false
return true
```

Figure 2.2: Auxiliary routine *checkD* takes inputs: an RBAC policy ρ, a set of SoD constraints D, and a subset of roles S. *checkD* returns 'true' if the S satisfies all the constraints and 'false' otherwise. The time complexity of *checkD* is $O(|D||R||S|)$.

```
Algorithm 1
Input: MIN-UAQ-P$\langle \rho, D, P_{lb}, P_{ub}, k \rangle$
Output: true/false

foreach $p_i \in P_{lb}$ do
    $R_i = \{r : r$ is authorized to $p_i\}$
    if $R_i = \emptyset$ then
        return false
$F = \{\{r_1, r_2, \ldots, r_{|P_{lb}|}\} : r_i \in R_i$ for all $i\}$
foreach $S \in F$ do
    $R_{actv} = \delta_{RH}(S)$
    $P_{actv} = \{p : R_{actv}$ activates $p\}$
    if $P_{lb} \subseteq P_{actv} \subseteq P_{ub}$ and $|P_{actv} - P_{lb}| \leq k$ then
        if checkD($\delta_{RH}(S)$) then return true
return false
```

Figure 2.3: Algorithm 1 takes as instance of MIN-UAQ-P and returns 'true' if there is a solution and 'false' otherwise. The time complexity of Algorithm 1 is $O(|R|^{|P_{lb}|+2}(|P|^2 + |D|))$. Algorithm 1 is used in the proof of Theorem 6.

and the following sections rely. Algorithm 1 takes as input a decision instance of MIN-UAQ-P and returns 'true' if there is a solution and 'false' otherwise. Algorithm 1 relies on the routine *checkD* in Figure 2.2 that checks whether a set of roles S satisfies the SoD constraints in D.

In Algorithm 1, for each permission in P_{lb}, a role from the set of roles to which they are authorized is picked (Line 1-5). For each such set of roles (Line 6), we account for the role hierarchy, RH (Line 7). That is, we assume that activation of a senior role implies activation of all its junior roles. Therefore, we need to ensure that all roles (junior and senior) satisfy the constraints in D. We then compute the set of all permissions, P_{actv}, to which at least a role in that set is authorized (Line 8). We then check whether P_{actv} includes all the permissions in P_{lb} and no permission out of P_{ub}, and the number of permissions from $P_{ub} - P_{lb}$ (the "extra permissions") in P_{actv} is at most k (Line 9). If yes, then all that remains to be checked is satisfaction of the constraints in D, which we do in Line 8. Algorithm 1 runs in time $O(|R|^{|P_{lb}|+2}(|P|^2 + |D|))$.

MIN-UAQ-P can be solved in polynomial time using Algorithm 1 when $|P_{lb}|$ bounded by a constant. However, if the size of P_{lb} is unbounded, MIN-UAQ-P is **NP**-hard (see [12]). In the following, we show how the computational complexity of MIN-UAQ-P changes as the size of P_{lb} increases.

Theorem 7. *MIN-UAQ-P is **NP**-hard even if* $|P_{lb}| = |R|^{O(1)}$.

Proof. Let $|P_{lb}| \leq c_1|R|^{c_2}$ for some positive constants c_1 and c_2. We show that there is a reduction from SET-COVER (see Section 2.3.1 for a definition of SET-COVER.) to MIN-UAQ-P that transforms an instance $\phi = \langle \mathcal{U}, \mathcal{F}, k\rangle$ of Set Cover into an instance $\psi = \langle \rho, D, P_{lb}, P_{ub}, k'\rangle$ of MIN-UAQ-P such that $P_{lb} \leq c_1|R|^{c_2}$ and ψ is true if and only if ϕ is true.

Let m and n denote the size of $\mathcal{U}$ and $\mathcal{F}$ respectively. For each set S_i in $\mathcal{F}$, we define a role r_i. For each element e_j in the universe $\mathcal{U}$, we define a permission p_j in P_{lb}. Each role r_i for $1 \leq i \leq n$ is authorized to p_j if S_i includes e_j. We also introduce m new permissions $p'_1, \cdots, p'_m$ in P_{lb} and $(2c_1^{-1}m)^{1/c_2}$ new roles $r_{n+1}, \cdots, r_{n+(2c_1^{-1}m)^{1/c_2}}$ authorized to non-empty subsets of the new permissions, including a role r_{esp} authorized to all the new permissions (assuming that m is sufficiently large, i.e., $(2c_1^{-1}m)^{1/c_2} \leq 2^m$). Each role r_i for $1 \leq i \leq n + (2c_1^{-1}m)^{1/c_2}$ is additionally authorized to a permission $p''_i \in P_{ub} - P_{lb}$.

The number of permissions in P_{lb} is $2m$, and the number of roles is $n + (2c_1^{-1}m)^{1/c_2} \geq (c_1^{-1}|P_{lb}|)^{1/c_2}$. The role hierarchy RH consists of tuples of the form $\langle r_i, r_i\rangle$ for roles defined above. There are no SoD constraints for the roles. We set $k' = k + 1$.

Suppose ϕ has a set cover $S \subseteq \mathcal{F}$ of size at most k. It is easy to verify that the set of roles $R_s = \{r_i : S_i \in S\} \cup \{r_{esp}\}$ activates all the permissions in P_{lb} and at most $k+1$ permissions from $P_{ub} - P_{lb}$. Thus, R_s is a solution to ψ. Conversely, assume that the set of roles R_s activates all permissions in P_{lb}, and at most $k+1$ permissions from $P_{ub} - P_{lb}$. The set R_s must consist of at most of $k+1$ roles; otherwise it activates more than $k+1$ permissions from $P_{ub} - P_{lb}$. Thus, $S = \{S_i : r_i \in R_s \text{ and } i \leq n\}$ is a set cover, and the size of S is at most k since $|S| \leq |R_s| - 1$.

□

Theorem 7 proves that MIN-UAQ-P is **NP**-hard even if $|P_{lb}|$ is as small as $O(|R|^\epsilon)$ for any constant $\epsilon > 0$. For finding a limit on the size of P_{lb} such that MIN-UAQ-P is no longer **NP**-hard (assuming **P** $\neq$ **NP**), we use the following well known conjecture.

Conjecture 1 (Exponential Time Hypothesis (ETH) [26])**.** *There is no $2^{o(n)}$-time algorithm for 3-CNF-SAT with n variables.*

Corollary 1. *Assuming ETH, there is no $2^{o(n)}$-time algorithm for any **NP**-hard problem, where n is the size of the problem.*

Proof. Follows from the fact that there exists a polynomial time reduction from 3-CNF-SAT to any **NP**-hard problem and the fact that that the class of $2^{o(n)}$-time problems is closed under composition with polynomials. □

Assuming that ETH is true, we are able to prove that MIN-UAQ-P is not **NP**-hard if the number of lower bound permissions is as small as $|R|^{o(1)}$ (e.g. $|P_{lb}| = O(\log |R|)$).

Theorem 8. *Assuming ETH, MIN-UAQ-P is not **NP**-hard if $|P_{lb}| = |R|^{o(1)}$.*

Proof. If $P_{lb} = |R|^{o(1)}$, then Algorithm 1 in Figure 2.3, can decide MIN-UAQ-P in $O(2^{|R|^{o(1)}}(|P|^2 + |D|) = O(2^{n^{o(1)}})$ where n is the size of MIN-UAQ-P, contradicting ETH. □

Impact of D and RH on the hardness of MIN-UAQ-P

We observe from the proof of Theorem 7 that the set of SoD constraints, D, and role hierarchy, RH, do not contribute to the computational complexity of MIN-UAQ-P in the sense that MIN-UAQ-P is **NP**-hard even if there is no SoD constraint and role hierarchy (i.e., $D = \emptyset$ and $RH = \{\langle r_i, r_i \rangle : r_i \in R\}$).

2.3.3 Complexity Results for MAX-UAQ-P

MAX-UAQ-P problem, a subcase of UAQ, is to find a subset of roles that maximizes the number of extra permissions granted while satisfying all the SoD constraints and requirements. A decision instance of MAX-UAQ-P is specified with the following inputs.

- RBAC policy, ρ, for a user, where $\rho = \langle R, P, RH, RP \rangle$, where R is the set of all roles in the RBAC policy, P is the set of all permissions in the RBAC policy, RH is a role hierarchy that relates roles in a partial order, RP is an assignment of roles to permissions,
- A set of SoD constraints, D, each of the form $\langle R_c, t_c \rangle$, where $t_c \in [1, |R_c|]$, $R_c \subseteq R$. The semantics of $\langle R_c, t_c \rangle$ is that t_c or more roles from the set R_c are not allowed to be activated at the same time.
- A set of permissions $P_{lb} \subseteq P$
- A set of permissions $P_{ub} \subseteq P$
- $k \in \mathbb{N}$

The formal language for the corresponding decision problem is

$$\begin{aligned}\text{MAX-UAQ-P} = \{\langle \rho, D, P_{lb}, P_{ub}, k \rangle : &\text{ there exists a set of roles } R' \subseteq R \text{ such that} \\ &\delta_{RH}(R') \text{ satisfies every constraint in } D, \\ &\text{activates all permissions in } P_{lb}, \\ &\text{no permission from } P - P_{ub}, \\ &\text{and at least } k \text{ permissions from } P_{ub} - P_{lb}\}.\end{aligned}$$

Theorem 9. *MAX-UAQ-P is **NP**-hard.*

Proof. We show that MAX-UAQ-P is **NP**-hard by reducing SET-COVER to MAX-UAQ-P (see Section 2.3.1 for a definition of SET-COVER.) Let $\phi = \langle \mathcal{U}, \mathcal{F}, k \rangle$ be an instance of SET-COVER. We construct the MAX-UAQ-P instance $\psi = \langle \rho, D, P_{lb}, P_{ub}, k' \rangle$ that is true if and only if ϕ is true.

For each element e_j in $\mathcal{U}$, we define a permission p_j in $P_{ub} - P_{lb}$. For each subset S_i in $\mathcal{F}$, we define a role r_i, which is authorized to permission p_j if and

```
Algorithm 2
Input: MAX-UAQ-P$\langle \rho, D, P_{lb}, P_{ub}, k \rangle$
Output: true/false
$R_{free} = R - \bigcup_{c_i \in D} \pi_{RH}(R_{c_i})$
foreach $c_i \in D$ do
    $F_i \leftarrow \{R' : R' \subseteq \pi(R_{c_i})$ and $|R'| < t_{c_i}\}$
$F \leftarrow \{\{R_1 \cup R_2 \cup \cdots \cup R_{|D|}\} : R_i \in F_i$ for all $i\}$
foreach $S \in F$ do
    $R_{val} = \{r :\ r$ is authorized to only permisssions in $P_{ub}\}$
    $R_{actv} = (\delta_{RH}(S) \cup R_{free}) \cap R_{val}$
    $P_{actv} \leftarrow \{p :\ R_{actv}$ activates $p\}$
    if $P_{lb} \subseteq P_{actv}$ and $|P_{actv} - P_{lb}| \geq k$ then
        if checkD($R_{actv}$) then return true
return false
```

Figure 2.4: Algorithm 2 takes an instance of MAX-UAQ-P and returns ‘true’ if there is a solution and ‘false’ otherwise. Algorithm 2 relies on the routine *checkD* in Figure 2.2 that checks whether a set of roles S satisfies the SoD constraints in D. Algorithm 2 runs in time $O(|R|^{|D|t+2}(|P|^2 + |D|))$. Algorithm 2 is used in the proof of Theorem 9.

only if $e_j \in S_i$. The set of SoD constraints, D, consists of a single constraint $\langle R, k+1 \rangle$. The set of lower bound permissions, P_{lb}, is empty. We set $k' = |P_{ub}|$. There is no role hierarchy for the roles, i.e., RH consists of only pairs of the form $\langle r_i, r_i \rangle$

Suppose ϕ has a solution $S \subseteq F$ of size at most k. The set of roles $R_s = \{r_i : S_i \in S\}$ satisfies every SoD constraint and activates all the permissions in P_{ub}: R_s is a solution for ψ. Conversely, if ψ has a solution R_s, then $S = \{S_i : r_i \in R_s\}$ is a set cover of size at most k for ϕ. □

Two following Theorems show how the size of P_{ub} impacts the computational complexity of MAX-UAQ-P.

Theorem 10. *MAX-UAQ-P is* **NP***-hard even if* $P_{ub} = |R|^{O(1)}$.

Proof. Assume $P_{ub} \leq c_1|R|^{c_2}$ for some constant positives c_1 and c_2. In this case, we show that MAX-UAQ-P is **NP**-hard by reducing 3-CNF SAT problem to it. (see Section 2.3.1 for a definition of 3-CNF SAT.) Given a 3-CNF formula ϕ of n variables and m clauses, we construct an instance $\psi = \langle \rho, D, P_{lb}, P_{ub}, k \rangle$ of MAX-UAQ-P such that ψ is true if and only if ϕ is satisfiable.

We define a role r_i for each literal x_i in ϕ, and a constraint $\langle \{r_i, r_j\}, 2)\rangle$ in D for each negated literals x_i and x_j ($x_j = \neg x_i$). For each clause $c = (x_i \vee x_j \vee x_k)$ in ϕ, we define a permission $p_c \in P_{ub} - P_{lb}$ to which r_i, r_j, and r_k are authorized. We also define m additional permissions $p'_1, \cdots, p'_m$ in P_{ub}, and $(2c_1^{-1}m)^{1/c_2}$ additional roles, each of which is authorized to a non-empty subset of the additional permissions, including a role r_{esp} authorized to all p'_i's. We set $k = 2m$. The set of lower bound permissions P_{lb} is an empty set. The role hierarchy relation, RH, only includes pairs of the form $\langle r_i, r_i\rangle$. It is clear that the reduction is polynomial in the size of the formula ϕ. The number of roles is $n + (2c_1^{-1}m)^{1/c_2} \geq (c_1^{-1}|P_{ub}|)^{1/c_2}$.

Suppose that there exists a valid assignment for the literals in ϕ that makes ϕ evaluate to true. The set of roles $R_s = \{r_i : x_i \text{ is true in the assignment}\} + \{r_{esp}\}$ is a solution to ψ since the set of roles R_s ($\delta_{RH}(R_s)$) satisfies every SoD constraint in D and is authorized to all permissions in P_{ub}. Conversely, let R_s be a solution to ψ. It is easy to see that the assignment in which a literal x_i is true if and only if r_i is in R_s, is a valid assignment and makes the formula evaluate to true. □

Theorem 11. *Assuming ETH, MAX-UAQ-P is not **NP**-hard if* $P_{ub} = |R|^{o(1)}$.

Proof. We can decide MAX-UAQ-P in $O(|R|^{P_{lb}+2}2^{|P_{ub}|}(|P|^2+|D|))$ using Algorithm 1 in Figure 2.3. We return true if Algorithm 1 accepts $\langle \rho, P_{lb} \cup P_s, P_{ub}, |P_{ub}|\rangle$ for at least a choice of P_s, where P_s is a subset of size k of $P_{ub} - P_{lb}$; otherwise, we return false. Therefore, when $P_{ub} = |R|^{o(1)}$, MIN-UAQ-R can be decided in $O(2^{|R|^{o(1)}}(|P|^2 + |D|)$, contradicting ETH. □

Impact of D and RH on the hardness of MAX-UAQ-P

Algorithms 2 Before we show how the computational complexity of MAX-UAQ-P is related to the SoD constraints, we discuss algorithm 2 in Figure 2.4. Algorithm 2 takes as input a decision instance of MAX-UAQ-P and returns 'true' if there is a solution and 'false' otherwise.

In Algorithm 2, we first compute R_{free}, which is the set of roles that are not in any SoD constraint (Line 1). For each constraint $c_i = \langle R_{c_i}, t_{c_i}\rangle$ in D, we compute every subset of size at most $t_{c_i} - 1$ of $\pi_{RH}(R_{c_i})$ (Line 2-3), where $\pi_{RH}(R_{c_i})$ is defined in Definition 2. For each constraint c_i, a subset in F_i is picked (Line 4). For each such set of roles S (Line 5), we compute the set of active roles, R_{actv}, which consists of all roles in S (as well as their junior roles) and free roles (R_{free}) (Line 6-7). We then check whether P_{actv} includes all the permissions in P_{lb} and the number of permissions from $P_{ub} - P_{lb}$ (the "extra permissions") in P_{actv} is at least k (Line 9). If yes, then we return true if the set of active roles satisfies all the constraints.

Algorithm 2 runs in time $O(|R|^{|D|t+2}(|P|^2 + |D|))$ where t is the maximum value that an integer second component in any constraint can take, i.e., $\max t_i$ for $\langle R_{c_i}, t_i \rangle \in D$. Therefore, Max-UAQ-P is in **P** if the number of constraints and t are bounded by constant.

Corollary 2. *MAX-UAQ-P is in **P** if $|D| = O(1)$ and $t = O(1)$.*

From the proofs of Theorem 9 and 10, we conclude that MAX-UAQ-P is **NP**-hard if the number of constraint or integer second component in any constraint is unbounded.

Corollary 3. *MAX-UAQ-P is **NP**-hard if any of t or $|D|$ is unbounded.*

We also observe from the proof of 9 that role hierarchy does not contribute to the complexity of MAX-UAQ-P in the sense that MAX-UAQ-P is **NP**-hard even if there is no role hierarchy, i.e., if $RH = \{\langle r_i, r_i \rangle : r_i \in R\}$.

2.3.4 Complexity Results for MIN-UAQ-R

MIN-UAQ-R problem is to find a subset of roles of minimum size that satisfies all the SoD constraints and all the requested permissions. Stating MIN-UAQ-R as a decision problem, an instance of MIN-UAQ-R is specified by the following inputs.

- RBAC policy, ρ, for a user, where $\rho = \langle R, P, RH, RP \rangle$, where R is the set of all roles in the RBAC policy, P is the set of all permissions in the RBAC policy, RH is a role hierarchy that relates roles in a partial order, RP is an assignment of roles to permissions,
- A set of SoD constraints, D, each of the form $\langle R_c, t_c \rangle$, where $t_c \in [1, |R_c|]$, $R_c \subseteq R$. The semantics of $\langle R_c, t_c \rangle$ is that t_c or more roles from the set R_c are not allowed to be activated at the same time.
- A set of permissions $P_{lb} \subseteq P$
- A set of permissions $P_{ub} \subseteq P$
- $k \in \mathbb{N}$

The formal language for the corresponding decision problem is

$$\begin{aligned}\text{MIN-UAQ-R} = \{\langle \rho, D, P_{lb}, P_{ub}, k\rangle : {} & \text{there exists a set of roles } R' \subseteq R \text{ such that} \\ & \text{the size of } \delta_{RH}(R') \text{ is at most } k, \\ & \delta_{RH}(R') \text{ satisfies every constraint in } D, \\ & \text{activates all permissions in } P_{lb}, \\ & \text{but no permission from } P - P_{ub}\}.\end{aligned}$$

Theorem 12. *[18] MIN-UAQ-R is* ***NP****-hard.*

We now take a finer look at the sources of complexity of MIN-UAQ-R. We show how the number of permissions in P_{lb}, the set of SoD constraint D, and role hierarchy RH contribute to the computational complexity of MIN-UAQ-R.

Theorem 12 proves that MIN-UAQ-R is **NP**-hard if the size of P_{lb} , the number of lower bound permissions, is unbounded. However, MIN-UAQ-R can be solved in polynomial time if $|P_{lb}|$ is bounded by a constant. For each permission in P_{lb}, we pick a role that activates it (if such role does not exist, we return false), and check if the selected roles together with their junior roles satisfies SoD constraint and activates no permission out of P_{ub}. In the following, we show how the computational complexity of MIN-UAQ-R changes as the size of P_{lb} increases. Following theorem proves that MIN-UAQ-R is still **NP**-hard if $|P_{lb}| = O(|R|^{\epsilon})$ for any $\epsilon > 0$.

Theorem 13. *MIN-UAQ-R is* ***NP****-hard even if* $|P_{lb}| = |R|^{O(1)}$.

Proof. Assume $|P_{lb}| \leq c_1|R|^{c_2}$ for some positive constants c_1 and c_2. We prove it by showing a reduction that transforms an $\phi = \langle \mathcal{U}, \mathcal{F}, k\rangle$ of SET-COVER (see Section 2.3.1 for a definition of SET-COVER.) to an instance $\psi = \langle \rho, D, P_{lb}, P_{ub}, k'\rangle$ of MIN-UAQ-R such that $P_{lb} \leq c_1|R|^{c_2}$.

Let m and n denote the size of $\mathcal{U}$ and $\mathcal{F}$ respectively. For each element e_j in $\mathcal{U}$, we define a permission p_j in P_{lb}. For each subset S_i in $\mathcal{F}$, we define a role r_i, which is authorized to the permissions corresponding to the elements in that subset, i.e., p_j's that $e_j \in S_i$. We also define m additional permissions $p'_1, \cdots, p'_m$ in P_{lb}, and $(2c_1^{-1}m)^{1/c_2}$ roles $r_{n+1}, \cdots, r_{n+2c_1^{-1}m^{1/c_2}}$ authorizing to non-empty subsets of the additional permissions, including a role r_{esp} authorized to all p'_i's. The number of roles is $n + (2c_1^{-1}m)^{1/c_2} \geq (c_1^{-1}|P_{lb}|)^{1/c_2}$.

Suppose that $S \subseteq \mathcal{F}$ is a set cover of size at most k. The set of roles $R_s = \{r_i : S_i \in S\} \cup \{r_{esp}\}$ is a solution of size $k+1$ for ψ. Conversely, if R_s is a solution for ψ of size k', then $S = \{S_i : r_i \in R_s \text{ and } i \leq n\}$ is a set cover of size at most $k' - 1$. □

We can prove a lower bound for the size of P_{lb} such that MIN-UAQ-R is **NP**-hard using ETH (see Conjecture 1).

Theorem 14. *Assuming ETH, MIN-UAQ-R is not **NP**-hard if* $|P_{lb}| = |R|^{o(1)}$.

Proof. Algorithm 1 in Figure 2.3 can be used to decide MIN-UAQ-R, if instead of checking the number of extra permissions in Line 9, we check whether the number of activated roles, i.e., R_{actv} is less than k. Thus, when $P_{lb} = |R|^{o(1)}$, MMIN-UAQ-R can be decided in time $O(2^{|R|^{o(1)}}(|P|^2 + |D|)) = O(2^{n^{o(1)}})$ where n is the size of MIN-UAQ-R instance, contradicting ETH. □

For example MIN-UAQ-R when $|P_{lb}| = O(\log^k |R|)$ (k is constant) satisfies the condition of Theorem 14, and therefore is not **NP**-hard (unless ETH is wrong). We show that if we assume additionally that the number of constraints is constant, then there exists a polynomial algorithm that can decide MIN-UAQ-R.

Theorem 15. *MIN-UAQ-R can be solved in* $poly(|R|^{|D|}, 2^{|P_{lb}|})$*, where* $poly(.)$ *is a polynomial function of its arguments.*

Proof. We present an algorithm based on dynamic programming that decides MIN-UAQ-R. We define $T(i, X, u_1, \cdots, u_{|D|})$ to be one if there exists a subset of roles $S \subseteq R$ such that

- $|\delta_{RH}(S)| \leq i$,
- $\delta_{RH}(S)$ activates all permissions in X,
- $\delta_{RH}(S)$ activates no permissions not included in P_{ub}, and
- $|\delta_{RH}(S) \cap R_{c_i}| = u_i$ for all $\langle R_{c_i}, t_i \rangle \in D$

and zero otherwise, for $i \in \{0, \cdots, k\}$ and $X \subseteq P_{lb}$. A subproblem $T(.)$ can be expressed in terms of smaller subproblems as below.

$$T(i, X, u_1, \cdots, u_{|D|}) = \max_{r_j \in R} T(i - \delta_{RH}(r_j), X', u'_1, \cdots, u'_{|D|})$$

where $X' = X - \{p : \delta_{RH}(r_j) \text{ activates } p\}$, $u'_k = u_k - |\delta_{RH}(r_j) \cap R_{c_k}|$, and $R_{valid} = \{r : \delta_{RH}(r) \text{ activates no } p \in P - P_{ub}\}$. The algorithm is shown in Figure 2.5.

□

```
Algorithm 3
Input: MIN-UAQ-R$\langle \rho, D, P_{lb}, P_{ub}, k \rangle$
Output: true/false

forall $i \in \{-|R|, \cdots, 0\}$ do
    forall $u_1, \cdots, u_{|D|} \in \{-|R|, \cdots, |R|\}$ do
        $T(i, \emptyset, u_1, \cdots, u_{|D|}) \leftarrow 0$
$T(0, \emptyset, 0, \cdots, 0) \leftarrow 1$
$R_{valid} = \{r_i : \delta_{RH}(r_i)$ activates no $p \in P - P_{ub}\}$
for $i = 1 \rightarrow k$ do
    forall $X \subseteq P_{lb}$ do
        forall $u_1, \cdots, u_{|D|} \in \{0, \cdots, |R|\}$ do
            $T(i, X, u_1, \cdots, u_{|D|}) = \max_{r_j \in R_{valid}} T(i - \delta_{RH}(r_j), X', u'_1, \cdots, u'_{|D|})$
            where $X' = X - \{p : \delta_{RH}(r_j)$ activates $p\}$ and
            $u'_k = u_k - |\delta_{RH}(r_j) \cap R_{c_k}|$.
foreach $u_1 \in \{0, t_1 - 1\}, \cdots, u_{|D|} \in \{0, \cdots, t_{|D|} - 1\}$ do
    if $T(k, P_{lb}, u_1, \cdots, u_{|D|}) = 1$ then
        return true
return false
```

Figure 2.5: Algorithm 3 takes an instance of MIN-UAQ-R and returns ‘true’ if there is a solution and ‘false’ otherwise. The time complexity of Algorithm 3 is $O(|R|^{|D|+4}\, 2^{|P_{lb}|}(|P|^2 + |D|)))$.

Impact of D and RH on the hardness of MIN-UAQ-R

We observe from the proof of Theorem 13 that the SoD constraints and role hierarchy do not contribute to the computational complexity of of MIN-UAQ-R in the sense that MIN-UAQ-R is **NP**-hard even if there is no SoD constraint or role hierarchy (i.e., $D = \emptyset$ and $RH = \{\langle r_i, r_i\rangle : r_i \in R\}$). However, it remains open as to whether the existence of SoD constraints and role hierarchy can change the computational complexity of subcases of MIN-UAQ-R in P (e.g., MIN-UAQ-R when $|P_{lb}| = O(\log^k |R|)$ and $|D| = O(1)$)

2.3.5 Complexity Results for MAX-UAQ-R

MAX-UAQ-R problem is to find a subset of roles of maximum size that satisfies all the SoD constraints and all the permissions in P_{lb}. Stating MAX-UAQ-R as a decision problem, an instance of MAX-UAQ-R is specified by the following inputs.

- RBAC policy, ρ, for a user, where $\rho = \langle R, P, RH, RP\rangle$, where R is the set of all roles in the RBAC policy, P is the set of all permissions in the RBAC policy, RH is a role hierarchy that relates roles in a partial order, RP is an assignment of roles to permissions,
- A set of SoD constraints, D, each of the form $\langle R_c, t_c\rangle$, where $t_c \in [1, |R_c|]$, $R_c \subseteq R$. The semantics of $\langle R_c, t_c\rangle$ is that t_c or more roles from the set R_c are not allowed to be activated at the same time.
- A set of permissions $P_{lb} \subseteq P$
- A set of permissions $P_{ub} \subseteq P$
- $k \in \mathbb{N}$

The formal language for the corresponding decision problem is

$$\begin{aligned}\text{MAX-UAQ-R} = \{\langle \rho, D, P_{lb}, P_{ub}, k\rangle :\ & \text{there exists a set of roles } R' \subseteq R \text{ such that}\\ & \text{the size of } \delta_{RH}(R') \text{ is at most } k,\\ & \delta_{RH}(R') \text{ satisfies every constraint in } D,\\ & \text{activates all permissions in } P_{lb},\\ & \text{but no permission from } P - P_{ub}\}.\end{aligned}$$

Theorem 16. *MAX-UAQ-R is **NP**-hard.*

Proof. We prove it by showing a reduction from 3-CNF-SAT to MAX-UAQ-R. (see Section 2.3.1 for a definition of 3-CNF-SAT.) Given an instance ϕ of 3-CNF-SAT, we construct an instance of MAX-UAQ-R, $\psi = \langle \rho, D, P_{lb}, P_{ub}, k \rangle$ as follows.

Assume without loss of generality that the negation of each literal $x \in \phi$ is in ϕ too (if not we add the trivial clause $(x \vee \neg x \vee x)$ to ϕ). We define a role r_i for each literal x_i in ϕ. The permission set P is empty, and the role hierarchy relation $RH = \{\langle r_i, r_i \rangle : r_i \in R\}$, where R is the set of roles defined. The set of SoD constraints, D, consists of a constraint $\langle \{r_i, r_j\}, 2 \rangle$ for each negated literals x_i and x_j, where $x_j = \neg x_i$, and a constraint $\langle \{r_i, r_j, r_k\}, 3 \rangle$ for each clause $(x_i \vee x_j \vee x_k)$ in ϕ. We claim that ϕ is satisfiable if and only if ψ has a solution of size at least $m/2$ where m is the number of literals in ϕ.

If ϕ is satisfiable, then the set of roles corresponding to those literals with false value in the satisfying assignment is a solution for ψ. That is, the set of roles $R_s = \{r_i : x_i \text{ is false in the assignment}\}$ satisfies all the SoD constraints, and is of size $m/2$. Conversely, suppose that R_s is a solution to ψ. It is easy to verify that the assignment in which a literal is false if and only if the corresponding role is in R_s, makes ϕ evaluate to true. □

The set of permissions in the proof of Theorem 16 is empty, so we can conclude the following corollary.

Corollary 4. *MAX-UAQ-R is **NP**-hard even if* $|P_{lb}| = O(1)$.

Impact of D and RH on the hardness of MAX-UAQ-R

We observe in the proof of Theorem 16 that the hardness of MAX-UAQ-R is related to the number of SoD constraints. However, the following theorem shows that MAX-UAQ-R is **NP**-hard even if there exists only one SoD constraint.

Theorem 17. *MAX-UAQ-R is **NP**-hard even if* $|D| = 1$ *and* $|P_{lb}| = O(1)$.

Proof. We prove that MIN-K-COVERAGE $\leq_p$ MAX-UAQ-R. (see Section 2.3.1 for a definition of MIN-K-COVERAGE.) Let $\phi = \langle \mathcal{U}, \mathcal{F}, k_1, k_2 \rangle$ be an instance of MIN-K-COVERAGE where $\mathcal{U}$, is a finite set of elements; $\mathcal{F}$, a family of subsets of $\mathcal{U}$; k_1 and k_2, positive integers. The instance ϕ is true if there exists a subset $S \subseteq \mathcal{F}$ such that $|S| \leq k_1$ and S covers at most k_2 elements of $\mathcal{U}$, i.e., $\left|\bigcup_{S_i \in S} S_i\right| \leq$

k_2. (see Section 2.3.1 for a proof that MIN Coverage problem is **NP**-hard.) We construct an instance of MAX-UAQ-R, $\psi = \langle \rho, D, P_{lb}, P_{ub}, k \rangle$ as follows.

The role set, R, consists of a senior role r_{e_j} for each element $e_j \in \mathcal{U}$, and a junior role r_i for each set $S_i \in F$. A senior role r_{e_j} is "senior" to a junior role r_i if and only if S_i contains e_j. That is, we have a pair $\langle r_{e_j}, r_i \rangle$ in RH for every e_j and S_i that $e_j \in S_i$. Let R_J and R_S denote the set of junior and senior roles respectively. The set of SoD constraints D consists of a single constraint, $\langle R_J, |R_J| - k_1 + 1 \rangle$, which limits the number of junior roles that can be activated at the same time to $|R_J| - k_1$. The set of permissions, P, is empty. We claim that ϕ is true if and only if there exists a set of roles R' such that the size of $\delta_{RH}(R')$ is at least $|R| - k_1 - k_2$, and $\delta_{RH}(R')$ satisfies the SoD constraint.

If ϕ is true, then there exists a subset $S \subseteq \mathcal{F}$ of size k_1 that covers at most k_2 elements of $\mathcal{U}$. The set of roles $R_s = \{r_i : S_i \notin S\} \cup \{r_{e_j} : e_j \notin \bigcup_{S_i \in S} S_i\}$ is a solution to ψ since $\delta_{RH}(R_s)$ satisfies the SoD constraint and is of size at least $|R| - k_1 - k_2$. Note that by activating the roles in R_s, no other (junior) roles not in R_s is activated. That is, $\delta_{RH}(R_s)$ equals to R_s.

Conversely, assume that ψ has a solution R_s such that the size of $\delta_{RH}(R_s)$ is at least $|R| - k_1 - k_2$. We claim that $S = \{S_i : r_i \notin R_s\}$ is a solution to ϕ. Since $\delta_{RH}(R_s)$ satisfies the SoD constraint, $\delta_{RH}(R_s)$ has at most $|R_J| - k_1$ junior roles. We assume without loss of generality that $\delta_{RH}(R_s)$ includes exactly $|R_J| - k_1$ junior roles; otherwise, we can add some more junior roles to R_s and still have the SoD constraint satisfied. So, $\delta_{RH}(R_s)$ includes at least $|R_S| - k_2$ senior roles. Under the automatic-activation assumption, we know that no senior role r_{e_j} in R_s is "senior" to any junior role r_i not in R_s; otherwise, the SoD constraint on junior roles is violated. Thus, S, the set of S_i's corresponding to the junior roles not in R_s, does not cover at least $|\mathcal{U}| - k_2$ elements of $\mathcal{U}$. It proves that S is a solution for ϕ because S is of size k_1 and covers at most k_2 elements of $\mathcal{U}$.

□

However, MAX-UAQ-R can be decided in polynomial time if the number of SoD constraints, D, and the maximum value that an integer second component in any constraint can take, i.e., $\max\{t_i\}$ for $\langle R_{c_i}, t_i \rangle \in D$, are constants. To decide MAX-UAQ-R in that case, we use Algorithm 2 in Figure 2.4 with a modification in Line 9. In Line 10 of Algorithm 2, we check for two conditions: (1) P_{actv} is a superset of P_{lb}; (2) the number of activated roles, R_{actv}, is at least k. The algorithm runs in $O(|R|^c(|P|^2 + |D|))$, where c is a constant such that $|D|t \leq c - 2$.

```
Algorithm 4
Input: MAX-UAQ-R$\langle \rho, D, P_{lb}, P_{ub}, k \rangle$
Output: true/false
$R_{val} = \{r : r$ is authorized to only permissions in $P_{ub}\}$
foreach $C \subseteq D$ do
    $R_{C_i} = (\bigcap_{c \in C_i} R_c - \bigcup_{c \notin C_i} R_c) \cap R_{val}$
foreach $p_i \in P_{lb}$ do
    $R_i = \{r : r$ is authorized to $p_i$ and $r \in R_{val}\}$
    if $R_i = \emptyset$ then
        return false
$F_p \leftarrow \{\{r_1, r_2, \ldots, r_{|P_{lb}|}\} : r_i \in R_i$ for all $i\}$
foreach $S \in F_p$ do
    foreach $(k_1, \cdots, k_{2^{|D|}})$ such that $k_i \in \{0, 1, \cdots, |R_C|\}$ do
        $R_{actv} \leftarrow S \cup M(R_{C_1}, k_i) \cup \cdots \cup M(R_{C_{2^{|D|}}}, k_{2^{|D|}})$
        if checkD($R_{actv}$) and $|R_{actv}| \geq k$ then
            return true
return false
```

Figure 2.6: Algorithm 4 takes an instance of MAX-UAQ-R and returns 'true' if there is a solution and 'false' otherwise. It returns a correct answer assuming that the activation of a senior roles in RBAC does not imply activation of any junior rolest. The time complexity of Algorithm 4 is $O(|R|^{2^{|D|}+|P_{lb}|+2}(|P|^2 + |D|))$

The computational complexity of MAX-UAQ-R changes as we assume different semantics for role hierarchy RH. If we assume that the activation of a senior roles does not imply activation of any junior roles, then MAX-UAQ-R with constant number of constraints (i.e., when $|D| = O(1)$ and $P_{lb} = O(1)$ is no longer **NP**-hard. In that case Algorithm 4 in Figure 2.6 solves MAX-UAQ-R in polynomial time.

Algorithm 4: Algorithm 4 takes as input an instance of MAX-UAQ-R and returns 'true' if there is a solution and 'false' otherwise. It assumes that activation of a senior role does not imply activation of its junior roles. It relies on the routine *checkD* in Figure 2.2 that checks whether the input set of roles satisfies the SoD constraints in D.

In Algorithm 4, we first compute the set of valid roles, i.e., the roles not authorized to any permission in $P - P_{ub}$. Then for each subset C_i of SoD constraints, D, we compute R_{C_i} that is the set of valid roles that appear in every constraint in C_i (Line 2-3). For each permission in P_{lb}, a role from the set to which they are authorized is picked (Line 4-8). For each such set of roles S (Line 9), we test all different ways that we can extend S (Line 10-13). $M(R_{C_i}, k_i)$ represents any subset of size k_i of R_{C_i}. We return true if we are able to extend S to at least k roles such that it satisfies all SoD constraint (Line 12). Algorithm 4 runs in time $O(|R|^c(|P|^2 + |D|))$, where c is a constant such that $2^{|D|} + |P_{lb}| \leq c - 2$.

Corollary 5. *MAX-UAQ-R is in **P** when $|D| = O(1)$ and $P_{lb} = O(1)$, if there is no role hierarchy or the activation of senior roles does not imply activation of junior roles.*

2.4 Mitigating the intractability of UAQ

The results in Section 2.3 show that, in general, all the four subcases of UAQ are **NP**-hard. Therefore, unless **P** = **NP**, there are no efficient algorithms to find an optimal solution to UAQ. By an efficient algorithm, we mean the one that runs in time polynomial in its input size. In this section, we consider how the intractability of UAQ can be mitigated. That is, how we can efficiently address instances of UAQ that may arise, notwithstanding its worst-case intractability.

The most common approach to dealing with the intractability of **NP**-hard optimization problems is to relax the requirement of finding an optimal solution. An example of this approach is *Approximation Algorithms* that efficiently find a solution that approximates the optimal solution. We discuss this approach in Section 2.4.1.

Another common approach is to relax the requirement of efficiency for all problem instances. In Section 2.4.2, we discuss an example of this approach, where we reduce UAQ to SAT and use a SAT Solver.

2.4.1 Approximation Algorithms

A standard approach to mitigate the intractability of NP-hard problems is to look for efficient algorithms that find approximate solutions, i.e. approximation algorithms. Approximation algorithms are algorithms to find a solution that is guaranteed to be within some factor of the optimum. However, an approximation algorithm cannot be defined for instances of UAQ that have no solution. Therefore, we confine ourselves to those instances of UAQ that have at least a solution. We show

how hard it is to approximate UAQ efficiently by establishing inapproximability results for each subcase of UAQ. These results give a better understanding of the hardness of UAQ in terms of how well it can be approximated. In this section, we discuss the hardness of the approximate variants of MIN-UAQ-P, MAX-UAQ-P, MIN-UAQ-R, and MAX-UAQ-R.

MIN-UAQ-P A solution to the optimization version of MIN-UAQ-P is a set of roles $R' \subseteq R$ such that $\delta_{RH}(R')$ satisfies every SoD constraint, activates all permissions in P_{lb}, but no permission out of P_{ub}. The cost associated with each solution R' is the number of permissions in P_{ub} to which $\delta_{RH}(R')$ is authorized. In the optimization version of MIN-UAQ-P, the goal is to find a solution of minimum cost.

Theorem 18. *Unless* **P** $=$ **NP**, *MIN-UAQ-P cannot be approximated within* $O(n^{1-\epsilon})$ *for any constant* $\epsilon > 0$*, where* n *is the number of permissions in* P_{ub}*.*

Proof. Given an instance ϕ of 3-CNF-SAT, we construct an instance ψ of MIN-UAQ-P such that for any $\epsilon > 0$

- If ϕ is satisfiable, then OPT(ψ) = $O(n^{\epsilon})$.
- If ϕ is unsatisfiable, then OPT(ψ) = $\Theta(n)$

where n is the size of P_{ub}. Therefore, the MIN-UAQ-P cannot be approximated within $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$; otherwise, the approximation algorithm for MIN-UAQ-P can be used to decide 3-CNF-SAT efficiently, which contradicts the assumption that **P** $\neq$ **NP**.

We construct ψ as follows. The set of roles consists of a main role r_i for each literal x_i in ϕ, and an auxiliary role a_j for each clause c_j in ϕ. The set of permissions P_{lb} consists of a permission p_j for each clause c_j of ϕ. Each main role r_i is authorized to permission p_j if and only if clause c_j contains x_i. Each auxiliary role a_j is authorized to permissions $p_j \in P_{lb}$ as well as $p'_{j,1}, \cdots, p'_{j,K} \in P_{ub} - P_{lb}$. The set of SoD constraints consists of a constraint $\langle\{r_i, r_j\}, 2\rangle$ for each negated literals x_i and x_j (i.e., $x_i = \neg x_j$) in ϕ, and a constraint $\langle\{r_i, a_j\}, 2\rangle$ for any pair of main and auxiliary roles, r_i, a_j. There is no role hierarchy, i.e., $RH = \{\langle r_i, r_i\rangle : r_i \in R\}$.

If ϕ is true, then there exists an assignment for the literals in ϕ that makes every clause evaluate to true. The set of roles corresponding to the literals with true value in the assignment, satisfies all SoD constraints, activates all permissions in P_{lb}, but no permission out of P_{lb}. The set of auxiliary roles, A, is another solution to ψ, which activates all permissions in P_{ub}, and therefore its cost is $O(Km)$ where m is number of clauses in ϕ. It is easy to verify that A is the only solution to ψ, if ϕ is unsatisfiable. By setting $K = m^{1/\epsilon - 1}$, the cost of the optimal solution to ψ is $O(m)$ if ϕ is satisfiable, and $O(m^{1/\epsilon})$ otherwise. □

MAX-UAQ-P A solution to the optimization version of MAX-UAQ-P is a set of roles $R' \subseteq R$ such that $\delta_{RH}(R')$ satisfies every SoD constraint, and activates all permissions in P_{lb}, but no permission out of P_{ub}. The cost associated with each solution is the number of permissions to which $\delta_{RH}(R')$ is authorized. In the optimization version of MAX-UAQ-P, the goal is to find a solution of maximum cost.

Theorem 19. *Unless **P** $=$ **NP**, MAX-UAQ-P cannot be approximated within $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$, where n is the number of permissions in P_{ub}.*

Proof. Given an instance of 3-CNF-SAT, ϕ, we construct an instance ψ of MAX-UAQ-P such that

- If ϕ is satisfiable, then OPT(ψ) = $O(n)$.
- If ϕ is unsatisfiable, then OPT(ψ) = $O(n^{\epsilon})$

where n is the size of P_{ub}.

We construct ψ as follows. The set of role consists of a main role r_i for each literal x_i in ϕ, an auxiliary role a_j for each clause c_j. The set of permissions P_{lb} consists of a permission p_j for each clause c_j in ϕ. The set of permissions $P_{ub} - P_{lb}$ consists of K permissions $p'_{j,1}, \cdots, p'_{j,K}$ for each clause c_j. Each main role r_i is authorized to permissions $p_j, p'_{j,1}, \cdots, p'_{j,K}$ if and only if clause c_j contains literal x_i. Each auxiliary role a_j is authorized to permission $p_j \in P_{lb}$. For each negated literals, x_i and x_j (i.e., $x_i = \neg x_j$), we define a SoD constraint $\langle\{r_i, r_j\}, 2\rangle$. We also add a SoD constraint $\langle\{r_i, a_j\}, 2\rangle$ to the set of constraints for any pair of auxiliary and main roles, r_i and a_j. There is no hierarchy for the roles, i.e., $RH = \{\langle r_i, r_i\rangle : r_i \in R\}$.

If ϕ is true, then there exists an assignment for variables that satisfies all the clauses in ϕ. The set of roles corresponding to the literals with true value in the assignment, is a solution for the ψ, which activates all the permissions in P_{ub}. The set of auxiliary roles, A, is also a solution for ψ, which activates no extra permissions. It is easy to verify that A is the only solution for ψ, if ϕ is not satisfiable. We set $K = m^{1/\epsilon - 1}$. The cost of the optimal solution to ψ is of $O(m^{1/\epsilon})$ if ϕ is satisfiable, and $O(m)$ otherwise.

□

MIN-UAQ-R A solution to the optimization version of MIN-UAQ-R is a set of roles $R' \subseteq R$ such that $\delta_{RH}(R')$ satisfies every SoD constraint and activates all permissions in P_{lb}, but no permission out of P_{ub}. The cost of each solution R' is the size of activated roles, i.e, $|\delta_{RH}(R')|$. In the optimization version of MIN-UAQ-R, the goal is to find a solution with the minimum number of activated roles.

In the following theorem, we prove that it is **NP**-hard to approximate MIN-UAQ-R within $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$.

Theorem 20. *Unless* **P** = **NP***, MIN-UAQ-R cannot be approximated within* $O(n^{1-\epsilon})$ *for any constant* $\epsilon > 0$*, where is* n *is the number of roles.*

Proof. We prove that for any $\epsilon > 0$ there is a gap introducing reduction from 3-CNF-SAT to MIN-UAQ-R that transforms a 3-CNF-SAT instance, ϕ, to a MIN-UAQ-R instance, ψ, such that

- if ϕ is satisfiable, then OPT(ψ) $= O(n^{\epsilon})$, and
- if ϕ is not satisfiable, then OPT(ψ) $= \Theta(n)$,

where n is the size of R. Therefore, if there exists a $O(n^{1-\epsilon})$-approximation algorithm for MIN-UAQ-R, then it can be used to solve 3-CNF-SAT in polynomial time, which contradicts the assumption that **P** $\neq$ **NP**.

We construct ψ as follows. The set of permissions P_{lb} consists of K permissions $p_{j,1}, \ldots, p_{j,k}$ for each clause c_j in ϕ. The set of roles consists of a main role r_i for each literal x_i in ϕ, and an auxiliary role $a_{j,k}$ for each permission $p_{j,k}$ in P_{lb}. Each main role r_i is authorized to all permissions $p_{j,1}, \ldots, p_{j,K}$ for any clause c_j that contains x_i. Each auxiliary role $a_{j,k}$ is authorized to $p_{j,k}$. The set of SoD constraints consists of a constraint $\langle \{r_i, r_j\}, 2 \rangle$ for each negated literals x_i and x_j, and a constraint $\langle \{a_{j,k}, r_i\}, 2 \rangle$ for each pair of auxiliary and main roles. The role hierarchy has only trivial pairs of form $\langle r_i, r_i \rangle$ for any role r_i. The set of permissions P_{ub} is equal to P_{lb}.

Assume that there exists an assignment to the literals in ϕ that satisfies every clauses. Therefore, the set of roles corresponding to those literals with true value in that assignment, is a solution for ψ, which has the cost of $O(m)$, where m is the number of clauses in ϕ. On the other hand, if ϕ is not satisfiable, then the only solution to ψ is the set of auxiliary roles, which has a cost of Km. By letting $K = m^{1/\epsilon-1}$, the optimal solution to ψ is of cost of $O(m)$, if ϕ is satisfiable, and $m^{1/\epsilon}$ otherwise. The size of ψ is $\Theta(Km^2)$, which is polynomial in the size of ϕ.

□

MAX-UAQ-R A solution to the decision version of MAX-UAQ-R is a set of roles $R' \subseteq R$ such that $\delta_{RH}(R')$ satisfies every SoD constraints and activates all permissions in P_{lb}, but no permission out of P_{ub}. The cost associated to each solution R' is the number of activated roles, i.e., $|\delta_{RH}(R')|$. In the optimization version of MAX-UAQ-R, the goal is to find a solution with the maximum number of activated roles.

In the following theorem, we prove that it is **NP**-hard to approximate MAX-UAQ-R within $O(n)$ where n is the size of role set, R.

Theorem 21. *Unless **P** = **NP**, MAX-UAQ-R cannot be approximated within $O(n)$, where n is the number of roles.*

Proof. We prove that there exists a gap introducing reduction from 3-CNF-SAT to MAX-UAQ-R that transforms a instance ϕ of 3-CNF-SAT to an instance ψ of MAX-UAQ-R such that

- if ϕ is satisfiable, then OPT(ψ) $= O(n)$, and
- if ϕ is not satisfiable, then OPT(ψ) $= 1$,

Where n is the number of roles. Therefore, any approximation algorithm that approximates MAX-UAQ-R within a factor of $O(n)$, can decide 3-CNF-SAT efficiently-contradicting the assumption that **P** $\neq$**NP**.

We construct ψ as follows. Assume without loss of generality that the negation of each literal $x \in \phi$ is in ϕ (if not we add clause $(x \vee \neg x \vee x)$ to ϕ). For each literal x_i in ϕ, we define a role r_i. For each clause c_j of ϕ, we define a permission $p_j \in P_{lb}$ to which a role r_i is authorized if and only if the literal x_i is in that clause c_j. We define an auxiliary role a, which is authorized to all permissions in P_{lb}. The auxiliary role cannot be activated with any role r_i at a same time. So, we have a SoD constraint $\langle\{r_i, a\}, 2\rangle$ for any role r_i. For each negated literals, x_i and x_j, we define a SoD constraint $\langle\{r_i, r_j\}, 2\rangle$. Finally, the set of upper bound permissions P_{ub} is equal to P_{lb}, and the role hierarchy includes only trivial pairs of the form $\langle r_i, r_i\rangle$ for each role r_i.

If ϕ is true, then there exists an assignment for the literals that satisfies every clause in ϕ. It is easy to verify that the set of roles corresponding to the literals with true values in the assignment, is a solution for ψ. The number of roles in the solution is half of the total number of roles (i.e., the number of literals in ϕ), hence the cost of the solution is $O(n)$. The auxiliary role, r_a, is also a solution for ψ. It is easy to verify that r_a is the only solution for ψ, if ϕ is not satisfiable.

□

Theorems 18, 19, 20, and 21 suggest that approximation algorithms is not a promising approach to mitigate the intractability of UAQ. In the following sections, we show how the intractability of UAQ can be mitigated by first reducing the optimization version of UAQ to its decision version (Section 2.4.2) and then solving the decision version (Sections 2.4.3 and 2.4.4).

2.4.2 Reduction to the decision version of UAQ

In this section, we design an oracle for the decision version of UAQ, and show how we can use it for the optimization version. The decision version of UAQ that corresponds to the optimization version defined in Section 2.1 is specified with the following inputs.

- An RBAC policy, ρ, for a user, where $\rho = \langle RH, RP \rangle$, where RH is a role-hierarchy that relates roles in a partial order, and RP is an assignment of roles to permissions. For convenience, we assume that RH is reflexive, i.e., given a role r that appears in ρ, $\langle r, r \rangle \in RH$. We denote the set of all roles in ρ as R, and the set of all permissions as P.

- A set of SoD constraints, D, each of the form $\langle R_c, t_c \rangle$, where $t_c \in [1, |R|]$, $R_i \subseteq R$.

- Two sets of permissions, P_{lb} and P_{ub}, with $P \supseteq P_{ub} \supseteq P_{lb}$.

- $k_r \in \{\leq, \geq\} \times [1, |R|]$, where R is the set of roles in the policy ρ. This indicates the number of roles we seek in a solution, and whether that number is an upper or lower bound.

- $k_p \in \{\leq, \geq\} \times [0, |P_{ub} - P_{lb}|]$. k_p is the extra-permissions analogue to k_r.

A decision instance is either true or false. It is true if there exists a set of roles that satisfies k_r and k_p, and false otherwise. For example, a decision UAQ instance with $k_r = (\leq, k_1)$ and $k_p = (\geq, k_2)$ is true, if there exist a subset of roles R' such that the total number of activated roles upon its activation, i.e., $\delta_{RH}(R')$, is of size at most k_1, and $\delta_{RH}(R')$ satisfies all the constraints, and the set of permission P' to which $\delta_{RH}(R')$ is authorized includes P_{lb} and at least k_2 permissions from $P_{ub} - P_{lb}$.

From the decision to the optimization version The decision and optimization versions of UAQ are related closely. There exists a polynomial-time Turing reduction [11] from the optimization version to the decision version. That is, given an oracle Ω for the decision version, we can efficiently solve the optimization version. A concrete approach is two-dimensional binary search. For example, for the case that $pri = n_p$ (i.e., prioritize permissions over roles), we first fix k_r at $\langle \geq, 1 \rangle$; i.e., we accept a solution with any number of roles. We then perform a binary search for the optimal number of permissions with $O(\log |P_{ub} - P_{lb}|)$ invocations to Ω. Once we find the optimal number of permissions, π, we then perform a binary search with $O(\log |R|)$ invocations to Ω with k_p adjusted to π (i.e., k_p set to $\langle \leq, \pi \rangle$ or $\langle \geq, \pi \rangle$ depending on the optimization objective for permissions).

The algorithm is shown in Figure 2.7. It can be seen as a polynomial-time Turing reduction [11] from the optimization version to the decision version. We assume that an oracle, Ω, to the decision version is given. The oracle takes as input $\rho, D, P_{lb}, P_{ub}, k_r, k_p$ under *UAQ Specification*.

Given an oracle for the decision version, a polynomial-time Turing reduction [11] can give us a certificate (i.e., set of roles) that is a solution. Indeed, a certificate is a by-product of tools such as SAT solvers along with the 'satisfiable' or 'unsatisfiable' output. We can then repeatedly look for additional solution sets of roles by excluding a role from R that is in a prior solution.

In summary, the optimization problem, the problem of finding pareto solution, and the problem of identifying one or multiple sets of roles that are valid solutions all rely on the construction of Ω, an approach to the decision version. Consequently, we focus on the decision version of UAQ in the remainder of this section.

The following theorem establishes an upper bound on the complexity of UAQ; the decision version remains in **NP**.

Theorem 22. *The decision version of UAQ $\in$ **NP**.*

Proof. It suffices to show that for every instance of UAQ, there is a polynomial-sized certificate that can be verified in polynomial time. Let R be the set of all

```
Algorithm 5
Input: $\langle \rho, D, P_{lb}, P_{ub}, k \rangle$
Output: A subset of roles, $R_s$
if $o_r = \textit{none and } o_p = \textit{none}$ then
    return $\Omega(\rho, D, P_{lb}, P_{ub}, \langle \geq, 1 \rangle, \langle \geq, 0 \rangle)$
if $o_r = \textit{none}$ then $pri = n_p$
else if $o_p = \textit{none}$ then $pri = n_r$
if $pri = n_p$ then
    $R_s = \emptyset$; $\alpha_p \leftarrow 0$; $\beta_p \leftarrow |P_{ub} - P_{lb}|$
    $c_p \leftarrow \lfloor (\beta_p - \alpha_p)/2 \rfloor$
    while $\alpha_p \leq \beta_p$ do
        if $o_p = \textit{max}$ then
            $R'_s \leftarrow \Omega(\rho, D, P_{lb}, P_{ub}, \langle \geq, 1 \rangle, \langle \geq, c_p \rangle)$
            if $R'_s = \emptyset$ then $\beta_p \leftarrow c_p - 1$
            else $R_s \leftarrow R'_s$; $\alpha_p \leftarrow c_p + 1$
        else
            $R'_s \leftarrow \Omega(\rho, D, P_{lb}, P_{ub}, \langle \geq, 1 \rangle, \langle \leq, c_p \rangle)$
            if $R'_s = \emptyset$ then $\alpha_p \leftarrow c_p + 1$
            else
                $R_s \leftarrow R'_s$; $\beta_p \leftarrow c_p - 1$
                if $o_p = \textit{none}$ then break
        $c_p \leftarrow \lfloor (\beta_p - \alpha_p)/2 \rfloor$
if $pri = n_p$ then
    if $R_s = \emptyset$ then return $\emptyset$
    Let $P_s$ be all permissions authorized to $R_s$
    $\alpha_r \leftarrow 1$; $\beta_r \leftarrow R$; $c_r \leftarrow \lfloor (\beta_r - \alpha_r)/2 \rfloor$
    while $\alpha_r \leq \beta_r$ do
        if $o_r = \textit{max}$ then
            $R'_s \leftarrow \Omega(\rho, D, P_{lb}, P_{ub}, \langle \geq, c_r \rangle, \langle =, |P_s| \rangle)$
            if $R'_s = \emptyset$ then $\beta_r \leftarrow c_r - 1$
            else $R_s \leftarrow R'_s$; $\alpha_r \leftarrow c_r + 1$
        else
            $R'_s \leftarrow \Omega(\rho, D, P_{lb}, P_{ub}, \langle \leq, c_r \rangle, \langle =, |P_s| \rangle)$
            if $R_s = \emptyset$ then $\alpha_r \leftarrow c_r + 1$
            else
                $R_s \leftarrow R'_s$; $\beta_r \leftarrow c_r - 1$
                if $o_r = \textit{none}$ then break
        $c_r \leftarrow \lfloor (\beta_r - \alpha_r)/2 \rfloor$
else {Similar to the "if" case, for roles first.}
return $R_s$
```

Figure 2.7: Algorithm 5 is a two-dimensional binary search for the optimal numbers of roles and extra permissions using an oracle, Ω, for the decision version of UAQ.

roles in the RBAC policy, ρ. A certificate is a set of roles R', where $R' \subseteq R$. This certificate is clearly polynomial (linear) in the size of the instance. Let P' be the set of permissions to which $\delta_{RH}(R')$ is authorized. An algorithm to verify the certificate first generates P' and then checks that $R' \subseteq R$, $|\delta_{RH}(R')|$ satisfies k_r, $P_{lb} \subseteq P' \subseteq P_{ub}$, $|P' - P_{lb}|$ satisfies k_p, and $\delta_{RH}(R')$ satisfies every constraint in D. This algorithm is polynomial-time in the instance. □

Therefore, **NP** is an upper bound for the inefficiency of any approach for mitigating the intractability of UAQ. In particular, it is known that **NP** $\subseteq$ **PSPACE** $\subseteq$ **EXPTIME** [**], and both containments are thought to be strict [4]. An approach that causes an exponential blowup in its design, like the one proposed in [59], is inefficient unless **NP** $=$ **PSPACE** $=$ **EXPTIME**.

2.4.3 Efficient Reduction to CNF SAT

In this section, we discuss our oracle for the decision version of UAQ – a polynomial time many-one reduction [4] to CNF-SAT[††] . The main technical challenges are to capture the SoD constraints and multiple objectives (roles and extra permissions).

An instance of CNF-SAT is a set of clauses that are a conjunction. A clause comprises one or more literals that are a disjunction. A literal is a variable or its negation. For each role $r_i \in R$ and permission $p_j \in P_{ub}$, we define a Boolean variable which is true if and only if r_i or p_j is activated, respectively. A satisfying assignment corresponds to a valid solution. If a SAT solver discovers that an input instance is satisfiable, as auxiliary output, it provides an assignment to the variables, which immediately tells us the set of roles to be activated. (Even if it does not, it is easy for us to construct one using a polynomial-time Turing reduction [4].)

The RBAC policy, ρ, and permission sets, P_{lb} and P_{ub} We adopt the approach of prior work [59], with a correction for their soundness issue, to capture ρ and P_{lb} in CNF-SAT. That is, for each $\langle r, p \rangle \in RP$ we capture it as a clause $r \rightarrow p$, where "$\rightarrow$" is "implies." Also, for every permission p, if $r_1, \ldots, r_n$ are the roles

[**] **PSPACE** is the class of decision problems that can be solved given space polynomial in the size of the input. **EXPTIME** is the class of decision problems that can be solved given time exponential in the size of the input.

[††] CNF-SAT is the problem of deciding whether a Boolean expression in Conjunctive Normal Form (CNF) is satisfiable. It is a well-known **NP**-complete problem [24].

to which it is authorized in ρ, we add a clause $p \rightarrow r_1 \vee \ldots \vee r_n$. These capture our intent that if r is activated, p is activated, and for p to be activated, at least one of the roles to which it is authorized must be activated. We also capture each $\langle r_1, r_2 \rangle \in RH$ with a clause $r_1 \rightarrow r_2$. This encodes the requirement that a junior role must be activated if a senior role is. This is optional — Wickramaarachchi et al. [59] provide a discussion of how we can deal with RH in the context of UAQ. We refer the reader to that work for a discussion.

We add a clause p for each $p \in P_{lb}$, i.e., a clause with the single variable p. This captures our intent that every permission in P_{lb} must be activated. For every permission in $P - P_{ub}$, we add the clause $\neg p$. This captures our intent that only permissions in P_{ub} are allowed to be activated. (We can alternatively simply remove those permissions and any role that is authorized to any of those permissions before formulating our UAQ instance.)

SoD constraints This is one of the technical challenges with reducing UAQ to SAT. We adopt an approach similar to that of Sinz [48]. We first encode each constraint as a Boolean circuit. That is, we first reduce constraint-satisfaction to CIRCUIT-SAT (the satisfiability problem for Boolean circuits), and then adopt a "textbook" reduction from CIRCUIT-SAT to CNF-SAT [14].

We first describe Boolean circuits that we call Bit-Sum, Sum and Compare that are building blocks. Then, we propose a circuit that we call Max-Circuit. It takes input n bits and an integer k, and outputs 1 if and only if at most k of the n bits are 1. All inputs to a Boolean circuit are bits; an integer input is encoded in binary. Our encoding of a constraint $\langle R_c, t \rangle \in D$ is as a Max-Circuit with input $\langle r_1, \ldots, r_n \rangle$ and t, where $R_c = \{r_1, \ldots, r_n\}$. We show an example of encoding the constraint $\langle \{r_1, r_2, r_3\}, 2 \rangle$ in Figure 2.8. In the following, with each description of a circuit, we mention its size. The size of a circuit is characterized by the number of gates in it.

- **Bit-Sum**:
  - Input: $\langle y_n, y_{n-1}, \ldots, y_1 \rangle$, x
  - Output: $\langle z_n, z_{n-1}, \ldots, z_1 \rangle$, such that $\sum_{i=1}^{n} z_i 2^{i-1} = \sum_{i=1}^{n} y_i 2^{i-1} + x$.
  - Let $c_1, \ldots, c_{n-1}$ be carry variables. Then, $z_1 = x \oplus y_1$, $c_1 = x \wedge y_1$ and $z_i = c_{i-1} \oplus y_i$, $c_i = c_{i-1} \wedge y_i$ for all $i \in \{2, \ldots, n\}$. The number of gates in a Bit-Sum circuit is $2n$.
- **Sum**:

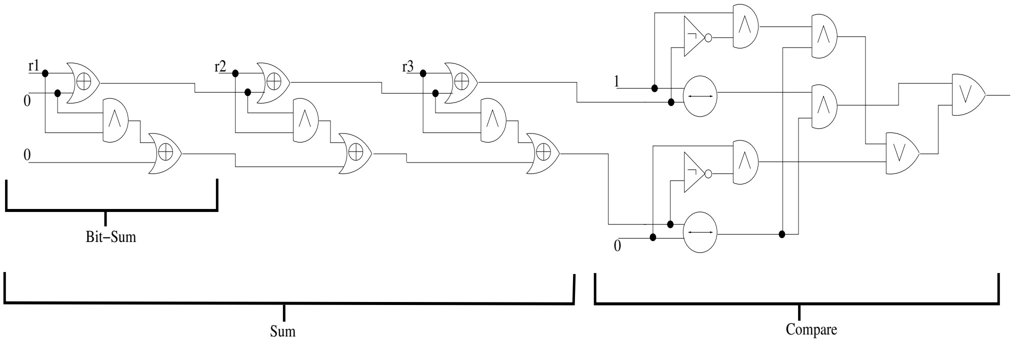


Figure 2.8: An example of Max-Circuit that we build for the constraint $\langle\{r_1, r_2, r_3\}, 2\rangle$ using the Bit-Sum, Sum and Compare building blocks. We use gates that correspond to $\wedge, \vee, \neg, \leftrightarrow$ and $\oplus$. We point out that the last two can be reduced in linear time to the first three: $x \leftrightarrow y = (\neg x \vee y) \wedge (x \vee \neg y)$, and $x \oplus y = (x \vee y) \wedge (\neg x \vee \neg y)$.

  - Input: $\langle x_n, x_{n-1}, \dots, x_1\rangle$
  - Output: $\langle z_m, z_{m-1}, \dots, z_1\rangle$, such that $\sum_{i=1}^{m} z_i 2^{i-1} = \sum_{i=1}^{n} x_i$, $m = \lceil \log n \rceil$.
  - The Sum circuit uses n modules of Bit-Sum, each of m gates. The total number of gates in Sum is $2nm = 2n\lceil \log n \rceil$.
    Sum $=$ Bit-Sum$(x_n,$ Bit-Sum$(x_{n-1}, \dots,$ Bit-Sum$(x_1, y_m, y_{m-1}, \dots, y_1) \dots))$, where $\langle y_m, y_{m-1}, \dots, y_1\rangle = \langle 0, 0, \dots, 0\rangle$.

- **Compare**
  - Input: $(x_n, x_{n-1}, \dots, x_1)$, $(y_n, y_{n-1}, \dots, y_1)$
  - Output: z, such that $z = 1$ if and only if $\sum_{i=1}^{n} x_i 2^{i-1} \leq \sum_{i=1}^{n} y_i 2^{i-1}$.
  - Let $e_1, \dots, e_n$ and $l_1, \dots, l_n$ be equality and less-ness variables, respectively. Then $e_n = x_n \leftrightarrow y_n$, $l_m = \neg x_n \wedge y_n$ and $e_i = e_{i+1} \wedge (x_i \leftrightarrow y_i)$, $l_i = ((\neg x_i \wedge y_i) \wedge e_{i+1}) \vee l_{i+1}$ for all $i \in \{1, \dots, n-1\}$. Then $z = l_1 \vee e_1$. The number of gates in Compare is $5n - 2$.

- **Max-Circuit**
  - Input: $(x_n, x_{n-1}, \dots, x_1)$, $k \leq n$

  - Output: z, such that z equals to 1 if and only if the number of true variables in $x_n, x_{n-1}, \ldots, x_1$ is at most k.
  - Let $m = \lceil \log n \rceil$ and $y_m, \ldots, y_1$ be such that $\sum_{i=1}^{m} y_i 2^{i-1} = k$. Then $z = \text{Compare}(\text{Sum}((x_n, x_{n-1}, \ldots, x_1)), (y_m, \ldots, y_1))$. The total number of gates in Max-Circuit(n) is at most $(2n + 5)\lceil \log n \rceil$.

Joint optimization This is the other technical challenge with UAQ. We use Boolean circuits for this as well. We first introduce an additional circuit that we call Min-Circuit.

- **Min-Circuit**
  - Input: $(x_n, x_{n-1}, \ldots, x_1)$, $k \leq n$
  - Output: z, such that z equals to 1 if and only if the number of true variables in $x_n, x_{n-1}, \ldots, x_1$ is at least k.
  - $z = \text{Compare}((y_m, \ldots, y_1), \text{Sum}((x_n, x_{n-1}, \ldots, x_1)))$ and $\sum_{i=1}^{m} y_i 2^{i-1} = k$. The total number of gates in Min-Circuit(n) is at most $(2n + 5)\lceil \log n \rceil$.

To encode $k_r = \langle \leq, c \rangle$, i.e., that we want a set of roles of size at most c, we employ Max-Circuit with inputs R (the set of all roles in the RBAC policy) and c. To encode $k_r = \langle \geq, c \rangle$, we employ Min-Circuit with inputs R and c. To encode $k_r = \langle =, c \rangle$, we employ both. That is, each of Max- and Min-Circuit is eventually encoded as clauses in CNF-SAT, which are then conjuncted, thus giving us the semantics we seek with "=." We can similarly encode k_p, the decision version of the optimization objective for extra permissions.

Cost of the reduction Each gate of Max-Circuit and Min-Circuit is converted to 4 clauses. If $d(p) = |\{r_i : r - i \text{ is authorized to } p\}|$, the total number of clauses is $|P_{lb}| + \sum_{p \in P_{ub} - P_{lb}} (d(p) + 1) + \sum_{\langle R_c, t \rangle \in D} (8|R_c| + 20) \log |R_c| + (8|P_{ub} - P_{lb}| + 20) \log |P_{ub} - P_{lb}|$ which is $O(|P_{ub}|(|R| + \log |P_{ub}|) + |D||R| \log |R|)$.

That is, if the UAQ instance is of size n, our reduction outputs $O(n^2 \log n)$ clauses. We infer this from the term $|D|\,|R| \log |R|$, which dominates in the expression above — both D and R are input to the problem. The running time of our reduction, and the size of the output CNF-SAT instance is $O(n^2 \log n)$.

Algorithm 6
Input: $\langle \rho, D, P_{lb}, P_{ub}, k_p, k_r \rangle$
Output: A subset of roles

```
1  foreach $p_i \in P_{lb}$ do
2      $R_i \leftarrow \{r : r$ is authorized to $p_i\}$
3  foreach $p_i \in P_{ub} - P_{lb}$ do
4      $R'_i \leftarrow \{r : r$ is authorized to $p_i\} \cup \{r_\epsilon\}$
5  $F \leftarrow \{\{r_1, \ldots, r_{|P_{lb}|}, r'_1, \ldots, r'_{|P_{ub}-P_{lb}|}\} : r_i \in R_i$ and $r'_i \in R'_i$ for all $i\}$
6  foreach $S \in F$ do
7      Let $\delta_{RH}(S)$ be the set of roles such that $r \in \delta_{RH}(S)$ if and only if either
       $r \in \delta_{RH}(S)$, or $r$ is junior to some $r' \in \delta_{RH}(S)$
8      $P_{actv} \leftarrow \{p :\ \delta_{RH}(S)$ activates $p\}$
9      if $P_{actv}$ satisfies $k_p$ and $\delta_{RH}(S)$ satisfies $k_r$ then
10         if checkD($\delta_{RH}(S)$) then return $\delta_{RH}(S)$
11 return $\emptyset$
```

Figure 2.9: Algorithm 6 takes an instance of the decision version of UAQ. It returns a set of roles that is a valid solution. The algorithm is exponential in $|P_{ub}|$, but is polynomial-time if $|P_{ub}|$ is bounded by a constant.

2.4.4 Fixed-Parameter Polynomial Algorithm

In this section, we devise the oracle for the decision version of UAQ using an algorithm that is efficient as long as the number of permissions is bounded by some constant. We call it FPP algorithm, where FPP stands for Fixed Parameter Polynomial Algorithm. A problem is said to be *fixed-parameter polynomial* with respect to parameter k if there exists an algorithm that solves it in time $O(n^{ck})$, where n is the input size and c is some constant [54].

Algorithm 6. Before we make our next assertion, we discuss algorithm 6 in Figure 2.9. For each permission in P_{lb}, we compute the set of roles authorized to it in Line 1-2. For each permission in $P_{ub} - P_{lb}$, we compute the set of roles containing dummy role r_ϵ and those roles authorized to the permission. The dummy role r_ϵ is authorized to no permission. We assume that the RBAC policy ρ has the dummy role, otherwise, we add it to the ρ in the preprocessing phase. In line 5-6, we choose a role for each permission from the corresponding set computed earlier. In Line 7, we take care of the role hierarchy, that is, we assume activation of a senior role implies activation of all its junior roles. We assume access to an auxiliary routine, *checkD* in Figure 2.2, that checks whether its input set of roles satisfies every constraint in D. $\delta_{RH}(S)$ is needed only if we seek to activate all junior roles of a role that is activated. If not, we need S only, and the check in Line 9 is for S, not $\delta_{RH}(S)$. If we seek all solutions, then rather than returning in Line 9, we continue to process all sets in F.

The algorithm imposes an upper bound of $|P_{ub}|$ on the number of roles in a solution set R_s. For each permission in P_{ub}, it picks a role as specified in Line 1–4. F contains all such sets of roles. As we adopt at most one role per permission, there are at most $(|R| + 1)^{|P_{ub}|}$ sets in F. If we impose the constraint that $|R_s| \leq |P_{ub}|$, F contains every possible solution set of roles. In Lines 5–11, for each set of roles in F, we check if it satisfies the other parameters for a solution, namely k_r, k_p and D.

Theorem 23. *Suppose the solution we seek, R_s, is such that the $|R_s| \leq |P_{ub}|$. Then, UAQ is fixed-parameter polynomial in $|P_{ub}|$.*

Proof. The algorithm in Figure 2.9 is correct, and runs in time $O(n^{|P_{ub}|+2})$, where n is the size of the input other than P_{ub}. The reason is that $|F| \leq (|R| + 1)^{|P_{ub}|} = O(n^{|P_{ub}|})$, and the processing of each set in F (e.g., the check against each constraint in D in Line 11) takes at worst quadratic time. Therefore, the algorithm is polynomial-time if $|P_{ub}| \leq c$ for some constant c. □

We argue that the precondition in the assertion of the above theorem, that $|R_s|$ is bounded by $|P_{ub}|$, is reasonable, particularly when we seek to minimize the number of activated roles. For each permission, we expect to acquire it with one role. If $|R_s| > |P_{ub}|$, then we know that we have some redundant roles that can be removed from the solution without affecting the permissions to which the session is authorized. The theorem and corresponding algorithm are of interest because part of the motivation for UAQ is the principle of least privilege. There may be cases in which it is reasonable to assume that $|P_{ub}|$ is small. We include an oracle based on this algorithm in our empirical assessment in the next section.

2.5 Empirical Evaluation

We have implemented both our reduction to CNF-SAT (Section 2.4.2) and fixed parameter polynomial (Section 2.4.4) approaches. For the former, we produce input for the zChaff SAT solver [63]. For the optimization version, we have implemented the two-dimensional binary search that we discuss in Section 2.4.2. All our implementations are available for public download [37]. We are not aware of any benchmark for UAQ. Consequently, we have done what prior work [18, 59, 64] does — generate several different kinds of test cases that exercise various parameters to the problem. We created a program that we have made publicly available [37] to generate test cases consistent with prior work. Our test program allows us to specify:

- minimum and maximum number of roles
- minimum and maximum number of permissions
- number of roles authorized to each permission
- minimum and maximum number of constraints
- minimum and maximum number of roles in each constraints
- number of permissions in P_{lb}
- number of permissions in P_{ub}
- depth of role-hierarchy, RH

Within those parameters that the user specifies, we generate the input randomly. For example, if the user specifies 3 for the number of roles authorized to each permission, we randomly pick role-permission assignment (i.e., each permission is connected to 3 random roles).

All data points in our graphs represent a mean across at least 10 different inputs generated randomly, that correspond to the value in the horizontal axis, each of which was run 10 times. That is, each data point is an average across 100 runs. We also computed a 95% confidence interval which we show in the graphs with a vertical line segment at the data point. (These may be barely visible because the intervals are small.) All the CPU times that we report are for full joint-optimization. As we perform binary search, the corresponding decision instances are about an order of magnitude faster. Our evaluations were conducted on a standard desktop PC with an Intel Dual Core E8400 CPU, each of which clocks at 3 GHz and has a 6 MB cache. The machine runs the 32-bit Ubuntu Linux 10.04 LTS operating system and has a 4GB RAM.

Overall observations Our approaches are orders of magnitude faster than prior approaches for which empirical results have been reported [59]. Furthermore, we are able to push our implementations far beyond prior approaches. For example, we have tried for up to $|R| = 200$; prior work [59] goes only up to fewer than 75. Our results are not surprising to us. They are a consequence of the efficiency inherent to our reduction (in the case of the CNF-SAT approach), and the fact that the fixed-parameter polynomial-time algorithm indeed demonstrates efficient (polynomial-time) behaviour so long as $|P_{ub}|$ is bounded by a somewhat small constant.

We were given access to the decision version of the CNF-SAT approach of [59]. We tried several inputs for comparison. We discovered that its exponential behaviour becomes quickly apparent. For an RBAC policy of only 14 roles and a single SoD constraint in D, the approach from that work takes 1.5 minutes. For the same policy, our approach takes 0.005 seconds.

Specific observations We report CPU times along 6 different axes for both our approaches.

- Figures 2.10 and 2.11 show our performance for different number of roles (i.e., $|R|$) and depth of role-hierarchy. We observe that both our approaches are resilient to an increase in either parameter. We have tried up to a somewhat unrealistic role-hierarchy depth of 10; prior benchmarks [30] suggest that the maximum role-hierarchy in enterprise settings is 5.

- Figures 2.12 and 2.13 show our performance for different number of constraints (i.e., $|D|$) and roles in a constraint (i.e., the first component of a constraint). Both approaches show a slow, linear worsening of performance in both cases. However, even for up to $|D| = 200$, the CNF-SAT approach takes less than 0.2 seconds. We have also tried up to a somewhat unrealistic number of roles in a constraint of 90. The CPU time for both our approaches remains less than 1 second.

- Figures 2.14 and 2.15 show our performance for different values of the integer (second component) of constraints, and $|P_{lb}|$. We point out that as $P_{lb} \subseteq P_{ub}$, and therefore $|P_{lb}|$ is a lower-bound for $|P_{ub}|$. Both our approaches remain highly resilient to different values of the integer in constraints. For increasing $|P_{lb}|$, however, the fixed-parameter polynomial-time algorithm starts to demonstrate exponential behaviour once the parameter

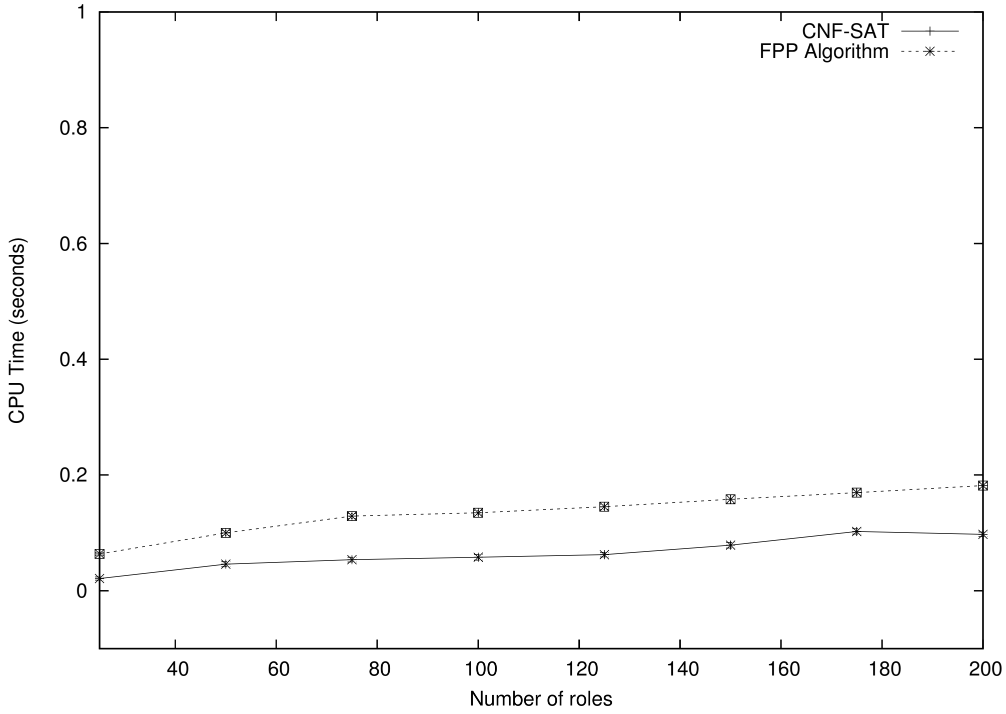


Figure 2.10: Performance of the optimization versions of our approaches for different values of $|R|$.

crosses a particular small threshold value. This is completely expected; the algorithm is exponential in $|P_{ub}|$, and therefore demonstrates polynomial-time behaviour only if $|P_{ub}|$ is bounded. In our implementation, this bound is approximately 7. The CNF-SAT approach, on the other hand, remains highly efficient for the larger values of $|P_{lb}|$.

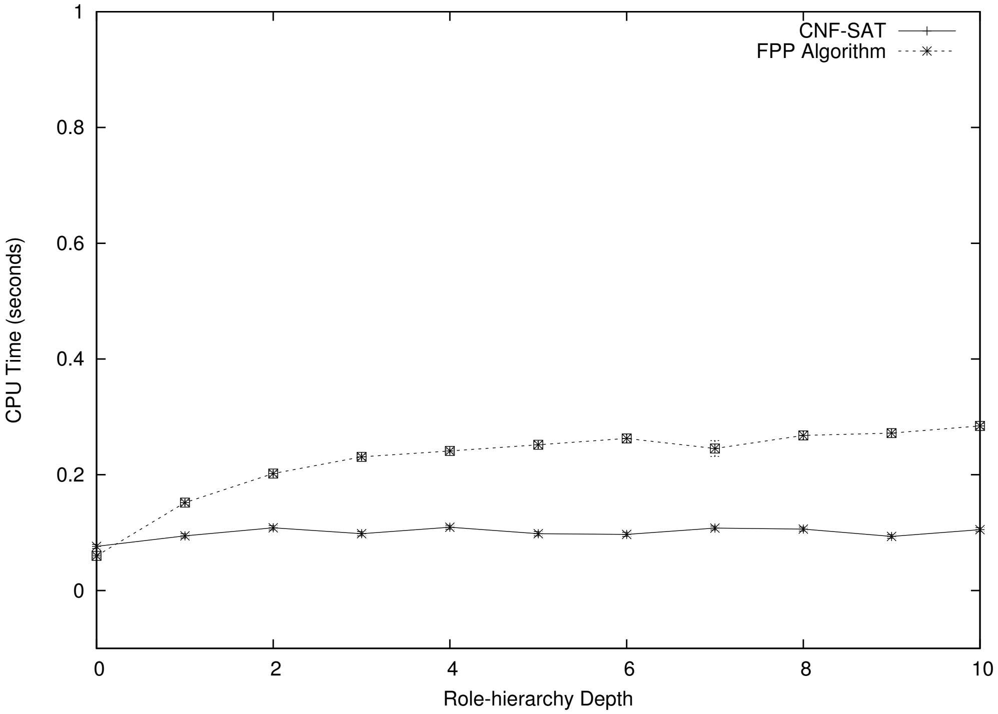


Figure 2.11: Performance of the optimization versions of our approaches for different values of depth of the role hierarchy.

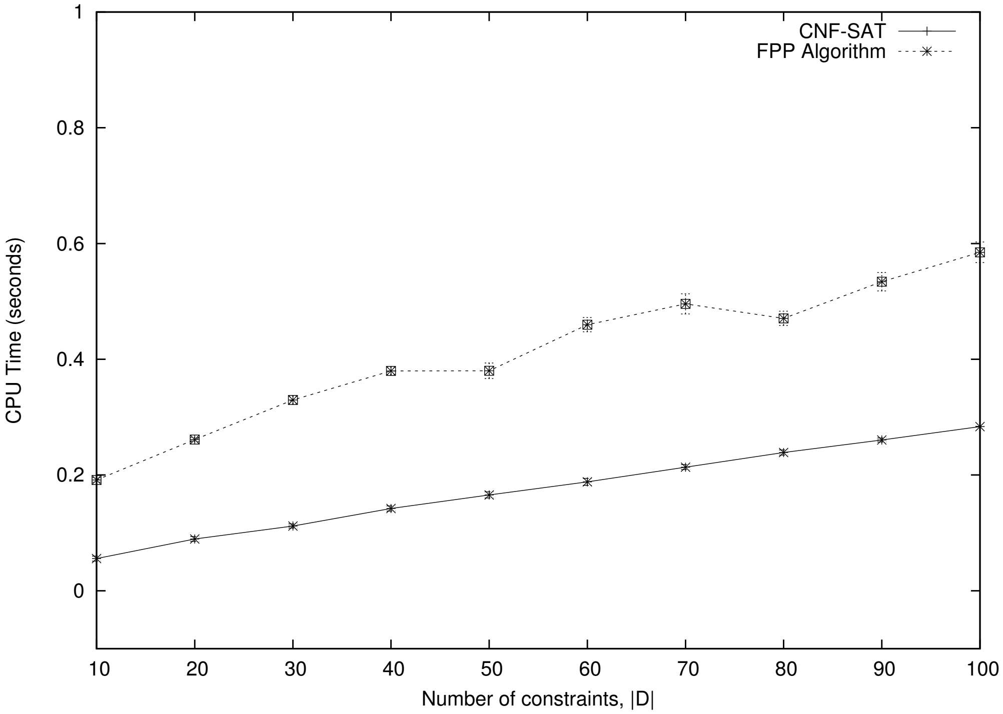


Figure 2.12: Performance of the optimization versions of our approaches for different numbers of constraints.

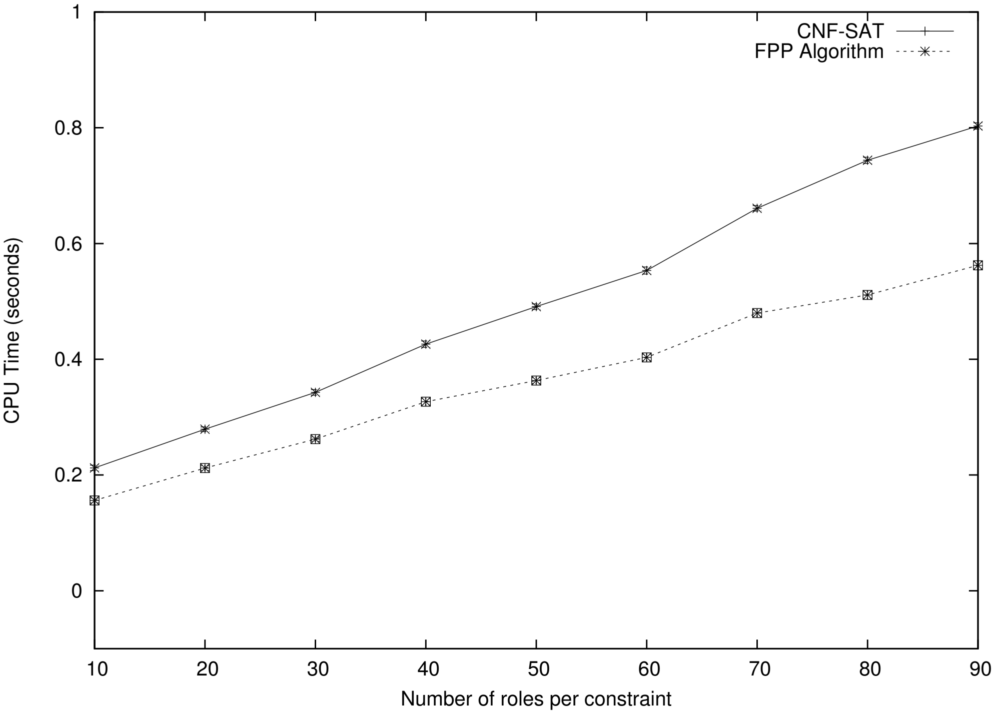


Figure 2.13: Performance of the optimization versions of our approaches for different number of roles in constraints.

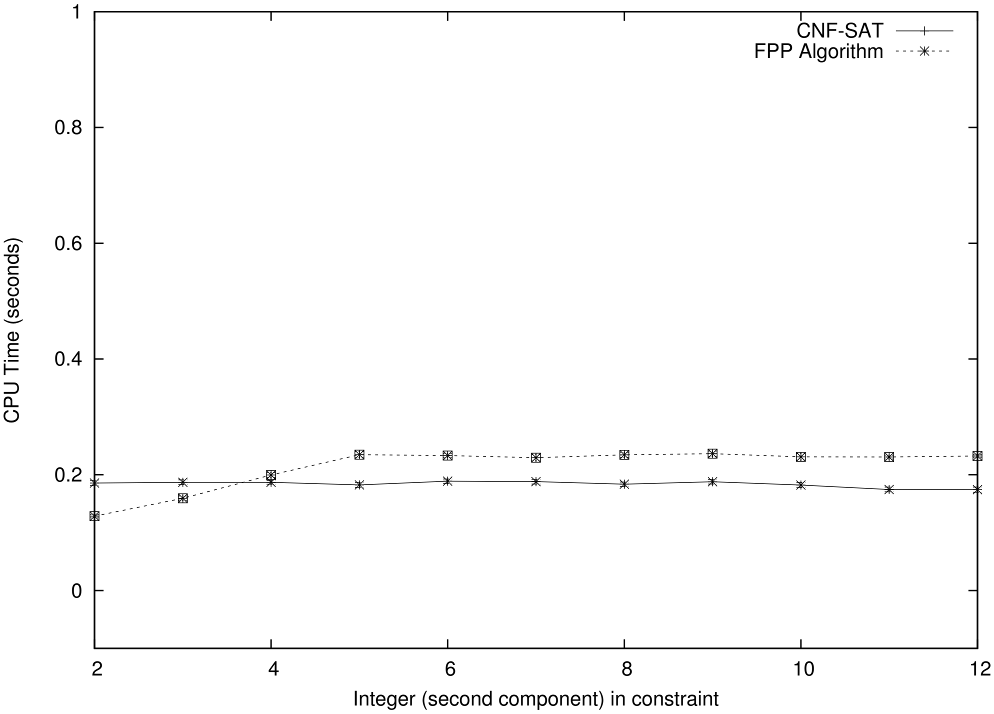


Figure 2.14: Performance of the optimization versions of our approaches for different values of the integer (second component) of constraints.

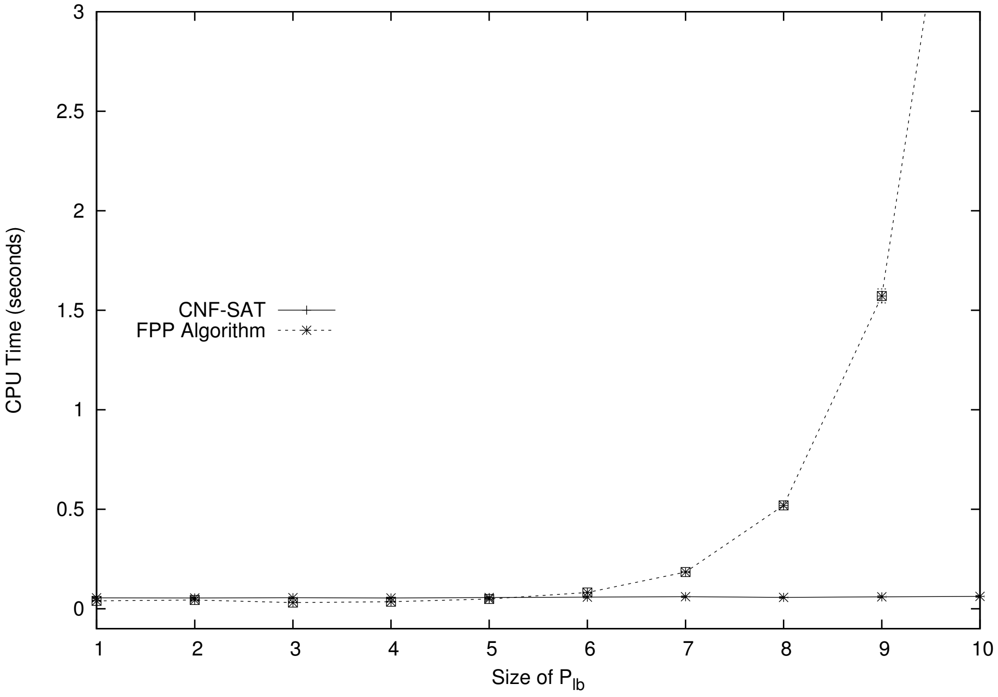


Figure 2.15: Performance of the optimization versions of our approaches for different values of $|P_{lb}|$.

2.6 Hard Instances of UAQ

The results of Section 2.5 suggest that the instances of UAQ in the benchmark we adopt are easy for our CNF-SAT algorithm. This leads one to ask: what are some characteristics of hard instances of UAQ? Assuming **P** $\neq$ **NP**, we know that such hard instances exist. In this section we propose a method for systematically generating hard instances of UAQ and then test our algorithm that is based on reduction to CNF-SAT with such instances. We study also the structural properties of such hard instances.

Constraint Satisfaction Problem Before we explain our method of generating hard instances of UAQ, we define Constraint Satisfaction Problems (CSP). A constraint satisfaction problem comprises a set of variables $x_1, x_2, \cdots, x_n$, where the variable x_i takes a value from a nonempty domain d_i, and a set of constraints $C_1, C_2, \cdots, C_m$. Each constraint C_i is an associative relation specifying the allowable values for the variables involved in C_i. A solution to a CSP is an assignment of values to all variables such that it satisfies all the constraints. A CSP is satisfiable if and only if there is a solution to it.

A model for generating hard instances of CSP, called RD, has been proposed by Xu et al. [60, 62]. Our method for generating hard UAQ instances is based on their model, which is defined below.

Definition 3. *[60, 62] A class of random CSP instances of model RD is denoted by $RD(k, n, \alpha, r, p)$ where:*

- $k \geq 2$ *denotes the arity of each constraint,*
- n *denotes the number of variables,*
- $\alpha > 0$ *determines the domain size of each variable that is* n^{α}*,*
- $r > 0$ *determines the number of constraints that is* $rn \log n$*,*
- $1 > p > 0$ *denotes the tightness of each constraint*

An instance of $RD(k, n, \alpha, r, p)$ is constructed as follows. Given n variables, we first construct $rn \ln n$ constraints. Each constraint is formed by selecting k random variables. For each constraint, each tuple of possible values is selected to be incompatible with probability p.

Xu et al. [60, 62] have shown that for $p = 1 - e^{-\alpha/r}$, instances generated by model RD, $RD(k, n, \alpha, r, p)$, are hard if $\alpha > 1/k$ and $p \leq \frac{k-1}{k}$.

Hard UAQ Instances We now explain our method for generating hard instances of UAQ, which is based on model RD. To create an instance of UAQ, we first construct an RBAC policy ρ as follows.

- Generate n sets of roles $R_1, R_2, \cdots, R_n$, where each set comprises n^α distinct roles.
- Set $RH = \{\langle r, r \rangle : r \in R\}$ where $R = \bigcup\limits_i R_i$.
- For every pair of roles in the same set, $r_k^i, r_l^i \in R_i$, define a permission to which only r_k^i and r_l^i are authorized.
- Randomly select two sets of roles R_i and R_j, and for each pair of roles $r_k^i \in R_i$ and $r_l^j \in R_j$, define a permission to which only r_k^i and r_l^j are authorized. Perform this step $rn \ln n$ times.

We create an instance of UAQ in which we seek the minimum number of roles that activates all the permissions. In other words, we set $P_{lb} = P_{ub} = P$ (where P is the set of all permissions defined above), $D = \emptyset$, $o_r =$ min and $o_p =$ none.

We show how an instance of UAQ with $k_r = n^{\alpha+1} - n$ generated as above is an instance of $RD(k, n', \alpha, r, p)$ when $k = 2$ and $n' = n$. We observe that if there exists a set of $n^{\alpha+1} - n$ roles that activates all the permissions, then the remaining n roles, denoted by R', are such that no two roles are authorized to the same permission. Since every pair of roles in the same role-set R_i shares a permission, R' must have exactly one role from each set. That is, $|R' \cap R_i| = 1$ for all i.

An instance of UAQ that we generate can be seen as an instance of model RD, i.e., $RD(2, n, \alpha, r, p)$, by defining a variable x_i for each set of roles R_i. The domain of each variable x_i consists of n^α values $v_1, v_2, \cdots, v_{n^\alpha}$. For each permission p to which $r_k^i \in R_i$ and $r_l^j \in R_j$ are authorized, there is a constraint that disallows values v_k and v_l for x_i and x_j respectively. There is a satisfying assignment for $x_1, x_2, \cdots, x_n$ if and only if there exist a set of $n^{1+\alpha} - n$ roles that activates all the permissions in P_{lb}.

We generated hard UAQ instances with $\alpha = 0.5, p = 0.5, r = 0.72$, and $n \in \{25, 26, \cdots, 50\}$, and used the CNF-SAT algorithm from Section 2.4.3 to

Table 2.1: CPU time for hard instances of UAQ

n	CPU Time (sec)	$\|R\|$	$\|P_{lb}\|$	n	CPU Time (sec)	$\|R\|$	$\|P_{lb}\|$
25	34	125	963	38	208	228	2241
26	9	130	1002	39	291	234	2364
27	37	135	1011	40	259	240	2371
28	44	140	1130	41	454	246	2521
29	25	145	1137	42	233	252	2623
30	43	150	1167	43	588	301	3540
31	125	186	1792	44	293	308	3704
32	260	192	1877	45	472	315	3811
33	166	198	1946	46	524	322	3980
34	87	204	2001	47	825	329	4029
35	110	210	2139	48	737	336	4102
36	134	216	2134	49	598	343	4199
37	130	222	2209	50	1135	350	45075

Table 2.2: CPU time for UAQ instances from Section 2.5

UAQ instance	CPU Time (sec)	$\|R\|$	$\|P\|$	$\|P_{lb}\|$
uaq-R	0.097	200	500	7
uaq-RH	0.105	200	500	5
uaq-D	0.283	100	500	7
uaq-RPC	0.803	300	1000	7
uaq-t	0.174	100	500	6
uaq-Plb	0.060	100	500	11

solve them. Table 2.1 shows CPU time for each instance as well as the size of each instance.

The CPU times in Table 2.1 are significantly larger than the CPU time for the UAQ instances from the benchmark in Section 2.5. For the purpose of comparison, we selected six UAQ instances from the benchmark used for generating graphs in Section 2.5 (one for each graph). Table 2.2 shows the CPU Time as well as the size of each such instance.

2.6.1 The Structure of Hard UAQ Instances

In this section, we study the structural property of the generated hard UAQ instances. We take three approaches: (1) reducing UAQ to CSP, and studying the structure of the corresponding CSP, (2) studying the structural property of two graphs associated to UAQ instances, that we call the RPG and RRP graphs, and (3) reducing UAQ to CNF-SAT, and studying the structural property of the corresponding CNF-SAT formula.

Reducing to CSP Given a UAQ instance, we construct a CSP instance as follows. We define a variable for each role and each permission. Each variable is either zero or one, denoting whether the corresponding role or permission is active. Let v_r and v_p denote the variables corresponding to $r \in R$ and $p \in P$. We define the following constraints.

- for each permission $p \in P$, we define a constraint involving v_p and all v_r's corresponding to the roles authorized to p. The constraint allows only assignments in which v_p is one if and only if at least one of the v_r's is one.
- for each SoD constraint $\langle R_i, t_i \rangle \in D$, we define a constraint involving variables v_r's for all $r \in R_i$ that allows at most $t_i - 1$ of the variables to be one.
- for each $\langle r_i, r_j \rangle \in RH$, we define a constraint where the allowable combination values for (r_i, r_j) are $\{(one, one), (zero, one), (zero, zero)\}$.
- for each $p \in P_{lb}$, the only allowed value for v_p is one.
- for each $p \in P - P_{ub}$, the only allowed value for v_p is zero.
- a constraint involving variables v_p for $p \in P_{ub} - P_{lb}$ that allows at most k_p of the variables to be one.
- a constraint involving variables in $\{v_r : r \in R\}$ that allows at most k_r of the variables to be one.

A CSP instance can be further represented by a constraint graph where there is a vertex for each variable and each constraint. There is an edge between a variable-vertex and a constraint-vertex if and only if the corresponding variable occurs in the corresponding constraint.

Table 2.3: Tree width of the corresponding CSP of the generated hard UAQ's

n	CPU Time (sec)	Tree width (lb)	Tree width (ub)	n	CPU Time (sec)	Tree width (lb)	Tree width (ub)
25	34	17	29	38	208	24	71
26	9	19	41	39	291	25	71
27	37	19	47	40	259	24	76
28	44	18	44	41	454	25	83
29	25	18	45	42	233	25	84
30	43	18	43	43	588	30	98
31	125	22	56	44	293	32	97
32	260	24	63	45	472	30	105
33	166	25	60	46	524	31	98
34	87	25	70	47	825	33	117
35	110	26	59	48	737	32	115
36	134	24	71	49	598	31	122
37	130	26	73	50	1135	32	117

It can be shown that each CSP problem is solvable in $O(nd^{w+1})$-time where n is the number of variables, d is the maximum domain size of each variable, and w is the tree width of the corresponding constraint graph [46]. Therefore, tree width is a structural property of the CSP problem that is closely related to its hardness. In fact, the tree width provides an upper bound for the difficulty of a CSP problem. For each instance of hard UAQ's, we first reduce it to a CSP instance as explained above, and then compute the upper bound and lower bound of the tree width of the associated constraint graph. We do this because, in general, computing the tree width for a CSP instance is itself **NP**-hard. Table 2.3 shows a lower bound and upper bound for the tree width of the constraint graph corresponding to each hard UAQ instance generated in Section 2.6. We use the Minor-Min-Width algorithm [23] for computing the lower-bounds and the Greedy-Fill-In algorithm [29] for the upper-bounds.

RPG and RRG graphs We associate two graphs to every instance of UAQ: (1) Role-Permission Graph (RPG) and (2) Role-Role Graph (RRG). RPG Graph is a bipartite graph with a vertex for each role, a vertex for each permission in P_{lb}, and an edge connecting a role-vertex to a permission-vertex if the corresponding role is authorized to the corresponding permission. RRG graph has vertices representing roles. Two vertices are connected if and only if there is a permission in P_{lb} to which both corresponding roles are authorized. For both graphs, we compute the average degree and the average diameter. The diameter for each vertex is the shortest distance to the farthest vertex from it. We compute also the average clustering coefficient. For each vertex the clustering coefficient is the number of edges between its neighbours, including the vertex itself divided by the maximum number of possible edges. Thus, the average clustering coefficient gives a measure of the extent to which each vertex belongs to a clique. Table 2.4 shows the computed values for all the hard UAQ instances we generated. The parameters used in Table 2.4 are defined in Table 2.11.

Figures 2.16 and 2.17 show CPU time versus parameters of the RPG and RRG graphs. We observe from Figure 2.16 that CPU time is correlated exponentially to the number of roles, number of permissions in P_{lb} and average degree of role-vertices in RPG graph, while there is a small correlation between CPU time and each of average degree of permission-vertices, average diameter of role-vertices in the RPG graph, and average diameter of permission-vertices in the RPG graph. From Figure 2.17, we observe that CPU time exponentially increases as the average degree in the RRG graph increases. The average clustering coefficient in the RPG and RRG graphs decreases slightly as the CPU time increases. The CPU time has a small correlation with the average diameter of vertices in the RRG graph, and the average clustering coefficient in the RPG graph. The Pearson product-moment correlation coefficient [43] between the logarithm of the CPU time and each parameter is shown in Table 2.5. We observe from Table 2.5 that the size of P_{lb}, the average degree of role-vertices in RPG graph, and average degree in RRG graph have the most correlation with the CPU time.

Table 2.4: UAQ Structural Parameters for Hard UAQ instances

n	$\|R\|$	$\|P_{lb}\|$	RPG-R-Degree	RPG-P-Degree	RPG-R-Diameter	RPG-P-Diameter	RPG-R-Clustering	RPG-P-Clustering	RRG-Degree	RRG-Diameter	RRG-Clustering
25	125	963	15.40	2	9.34	9.73	0.14	0.67	15.40	4.69	0.52
26	130	1002	15.41	2	9.51	9.85	0.14	0.67	15.41	4.83	0.51
27	135	1011	14.97	2	9.94	10.44	0.15	0.67	14.97	5.24	0.52
28	140	1130	16.14	2	9.65	9.78	0.14	0.67	16.14	4.83	0.53
29	145	1137	15.68	2	10.11	10.27	0.13	0.67	15.68	5.08	0.53
30	150	1167	15.56	2	10.14	10.63	0.14	0.67	15.56	5.11	0.52
31	186	1792	19.26	2	9.924	10.05	0.11	0.67	19.26	4.98	0.50
32	192	1877	19.55	2	10.02	10.28	0.11	0.67	19.55	5.07	0.49
33	198	1946	19.65	2	9.34	9.80	0.12	0.67	19.65	4.76	0.52
34	204	2001	19.61	2	9.50	9.77	0.11	0.67	19.61	4.79	0.48
35	210	2139	20.37	2	9.29	9.45	0.10	0.67	20.37	4.66	0.47
36	216	2134	19.75	2	9.63	10.06	0.11	0.67	19.75	4.88	0.48
37	222	2209	19.90	2	9.63	9.84	0.11	0.67	19.90	4.84	0.48
38	228	2241	19.65	2	9.91	10.25	0.11	0.67	19.65	4.97	0.47
39	234	2364	20.20	2	9.88	10.06	0.11	0.67	20.20	4.98	0.48
40	240	2371	19.75	2	9.80	10.12	0.11	0.67	19.75	4.93	0.49
41	246	2521	20.49	2	9.05	9.47	0.10	0.67	20.49	4.57	0.47
42	252	2623	20.81	2	10.27	10.56	0.10	0.67	20.81	5.15	0.47
43	301	3540	23.52	2	9.97	10.19	0.09	0.67	23.52	5.01	0.48
44	308	3704	24.05	2	9.50	10.00	0.10	0.67	24.05	4.78	0.48
45	315	3811	24.19	2	9.76	9.92	0.09	0.67	24.19	4.91	0.45
46	322	3980	24.72	2	9.59	9.82	0.09	0.67	24.72	4.84	0.46
47	329	4029	24.49	2	9.50	9.88	0.09	0.67	24.49	4.79	0.45
48	336	4102	24.41	2	9.64	9.97	0.09	0.67	24.41	4.91	0.47
49	343	4199	24.48	2	10.24	10.71	0.09	0.67	24.48	5.13	0.46
50	350	4507	25.75	2	9.48	9.77	0.08	0.67	25.75	4.75	0.44

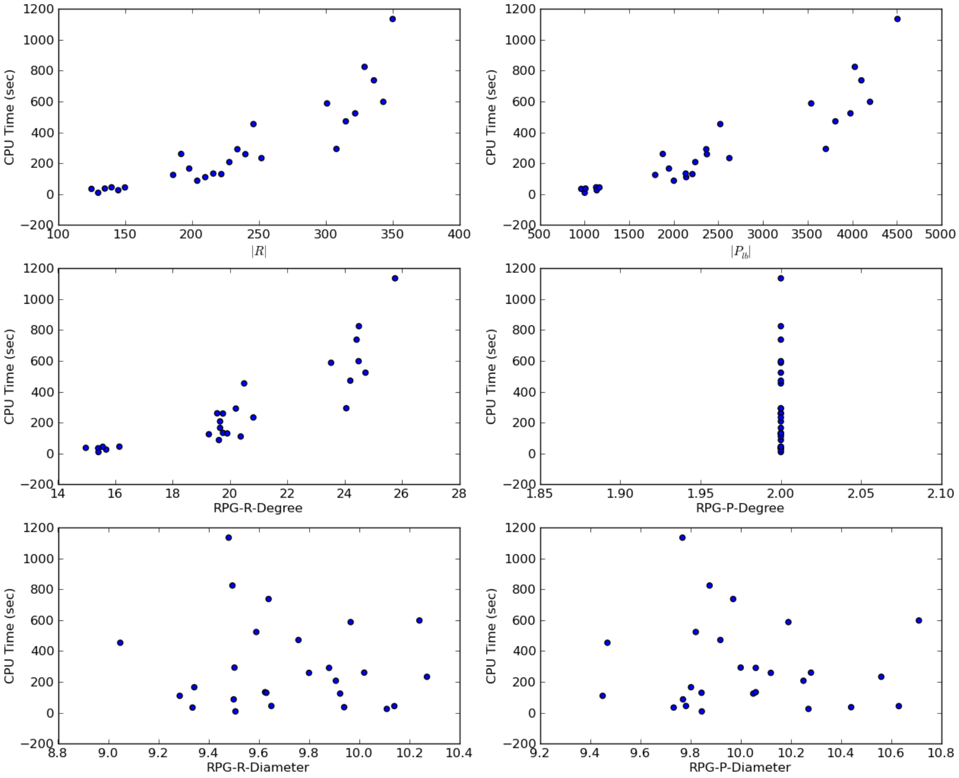


Figure 2.16: CPU time versus six different UAQ structural features: (1) $|R|$; number of roles, (2) $|P_{lb}|$; number of permissions in P_{lb}, (3) RPG-R-Degree; average degree of role-vertices in RPG graph, (4) RPG-P-Degree; average degree of permission-vertices in RPG graph, (5) RPG-R-Diameter; average diameter of role-vertices in RPG graph, and (6) RPG-P-Diameter; average diameter of permission-vertices in RPG graph.

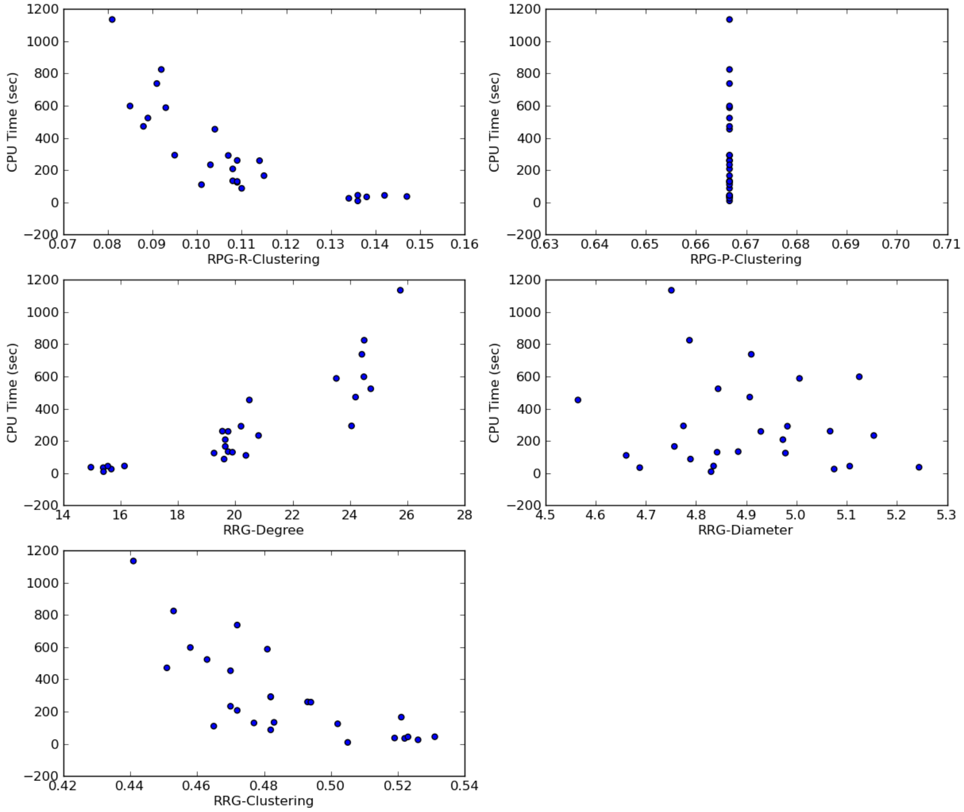


Figure 2.17: CPU time versus five different UAQ structural features: (1) RPG-R-Clustering; average clustering coefficient of role-vertices in RPG graph, (2) RPG-P-Clustering; average clustering coefficient of permission vertices in RPG graph, (3) RRG-Degree;average degree of vertices in RRG graph, (4) RRG-Diameter; average diameter of vertices in RRG graph, and (5) RRG-Clustering; average clustering coefficient of vertices in RRG graph.

Reduction to CNF-SAT In Section 2.4.3, we proposed a solution for UAQ based on reduction to CNF SAT. We now study the structural properties of the CNF-SAT formula generated for the UAQ instances under that reduction. Nudelman et al. [41] have proposed several parameters for a SAT formula, and have

Table 2.5: Pearson correlation coefficient between CPU Time and UAQ structural parameters

Parameter	Correlation Coeff.	Parameter	Correlation Coeff.
P_{lb}	0.9233	RPG-R-Clustering	-0.9084
R	0.9061	RPG-P-Clustering	0
RPG-R-Degree	0.9260	RRG-Degree	0.9260
RPG-P-Degree	0	RRG-Diameter	-0.1189
RPG-R-Diameter	-0.0407	RRG-Clustering	-0.7963
RPG-P-Diameter	-0.0491		

shown that they can be used to effectively predict the CPU time taken for solving the SAT formula [41, 61]. We compute 16 different parameters for the CNF-SAT formula that corresponds to each hard UAQ instance under our reduction. All the parameters we have selected are amongst the parameters proposed in [41, 61] that are directly related to the structure of a SAT formula. The Variable-Clause Graph (VCG) is a bipartite graph with a vertex for each variable and each clause in the SAT formula. A variable-vertex is connected to a clause-vertex if the corresponding variable occurs in the corresponding clause. The Clause Graph (CG) has vertices that represents clauses and an edge between two vertices if the corresponding clauses share a negated literal. The Variable Graph (VG) has vertices for each variable and an edge between variables that occur in at least one clause. Our results are presented in Table 2.6 and 2.7 and a definition for the parameters is presented in Table 2.12.

From Table 2.6 and 2.7, we observe that the value of three structural parameters, i.e., VCG-Variable-Degree, VG-Degree, and VG-Diameter, vary considerably for different hard instances of UAQ. Figure 2.18 shows CPU time versus the VCG-Variable-Degree, VG-Degree, and VG-Diameter. The correlation between these three parameters and logarithm of the CPU time is presented in Table 2.8 from which we observe that average degree of variable-vertices in VCG graph and average degree of vertices in VG graph have the most correlation with the CPU time.

We also observe from Table 2.6 that Clause-Variable-Ratio of all instance are very close to 4.26, which is roughly the clause-variable ratio for hard random 3-CNF SAT formulas [52].

Table 2.6: CNF-SAT Structural Parameters for Hard UAQ instances

n	Variables	Clauses	Clauses-Variable-Ratio	Unary-Clause	Binary-Clause	Ternary-Clause	Pos-Neg-Clauses-Ratio	Pos-Neg-Variable-Ratio	Horn-Variable	Horn-Clause-Fraction
25	1921	8106	4.22	15	978	8106	0.49	0.18	5.49	0.44
26	2263	9487	4.19	16	1019	9487	0.48	0.16	5.56	0.45
27	2348	9816	4.18	17	1028	9816	0.48	0.16	5.57	0.45
28	2433	10255	4.21	17	1147	10255	0.48	0.16	5.56	0.45
29	2518	10582	4.20	17	1154	10582	0.48	0.16	5.56	0.45
30	2603	10932	4.20	16	1185	10932	0.48	0.16	5.57	0.45
31	3215	13861	4.31	17	1809	13861	0.49	0.18	5.52	0.44
32	3317	14330	4.32	17	1894	14330	0.49	0.18	5.51	0.44
33	3419	14783	4.32	16	1964	14783	0.49	0.18	5.51	0.44
34	3521	15222	4.32	16	2018	15222	0.49	0.19	5.51	0.44
35	3623	15744	4.35	16	2157	15744	0.50	0.19	5.50	0.43
36	3725	16123	4.33	16	2151	16123	0.49	0.18	5.51	0.44
37	3827	16582	4.33	17	2226	16582	0.49	0.18	5.51	0.43
38	3929	16998	4.33	16	2259	16998	0.49	0.18	5.51	0.44
39	4031	17505	4.34	17	2381	17505	0.50	0.19	5.50	0.43
40	4133	17896	4.33	17	2389	17896	0.49	0.18	5.51	0.44
41	4235	18430	4.35	17	2538	18430	0.50	0.19	5.50	0.43
42	4337	18916	4.36	17	2641	18916	0.50	0.19	5.49	0.43
43	5779	25399	4.40	18	3560	25399	0.50	0.19	5.54	0.43
44	5912	26067	4.41	18	3724	26067	0.50	0.19	5.53	0.43
45	6045	26678	4.41	18	3830	26678	0.50	0.19	5.52	0.43
46	6178	27351	4.43	18	3999	27351	0.50	0.19	5.52	0.43
47	6311	27904	4.42	19	4048	27904	0.50	0.19	5.52	0.43
48	6444	28481	4.42	18	4121	28481	0.50	0.19	5.52	0.43
49	6577	29082	4.42	19	4219	29082	0.50	0.19	5.52	0.43
50	6710	29894	4.46	19	4526	29894	0.50	0.20	5.49	0.43

Table 2.7: CNF-SAT Structural Parameters for Hard UAQ instances

n	VCG-Clause-Degree	VCG-Variable-Degree	CG-Degree	CG-Clustering	VG-Degree	VG-Diameter
25	2.88	12.97	13.66	0.44	1.55	15.55
26	2.89	12.71	13.69	0.44	1.48	17.06
27	2.89	12.65	13.69	0.44	1.47	18.15
28	2.89	12.84	13.71	0.44	1.50	17.09
29	2.89	12.77	13.71	0.44	1.49	17.25
30	2.89	12.75	13.71	0.44	1.49	18.35
31	2.87	13.55	13.77	0.44	1.64	17.10
32	2.87	13.63	13.78	0.44	1.66	17.23
33	2.87	13.66	13.78	0.44	1.67	18.21
34	2.87	13.65	13.78	0.44	1.66	16.95
35	2.86	13.87	13.80	0.43	1.71	16.85
36	2.87	13.70	13.79	0.44	1.67	18.26
37	2.86	13.74	13.79	0.44	1.68	16.99
38	2.87	13.67	13.79	0.44	1.67	17.20
39	2.86	13.83	13.80	0.44	1.70	17.30
40	2.87	13.71	13.79	0.44	1.67	18.21
41	2.86	13.93	13.81	0.43	1.72	18.38
42	2.86	14.04	13.81	0.43	1.74	17.37
43	2.86	14.23	13.87	0.43	1.76	18.58
44	2.86	14.40	13.87	0.43	1.80	20.28
45	2.86	14.46	13.88	0.43	1.81	18.72
46	2.85	14.65	13.88	0.43	1.85	18.52
47	2.85	14.56	13.88	0.43	1.83	19.48
48	2.85	14.54	13.88	0.43	1.83	19.67
49	2.85	14.57	13.88	0.43	1.83	18.65
50	2.85	15.07	13.90	0.43	1.94	17.96

Table 2.8: Pearson correlation coefficient between CPU Time and CNF structural parameters

Parameter	Correlation Coeff.	Parameter	Correlation Coeff.
Variables	0.898	VG-Degree	0.9323
Clauses	0.899	VG-Diameter	0.5894
VCG-Variable-Degree	0.934		

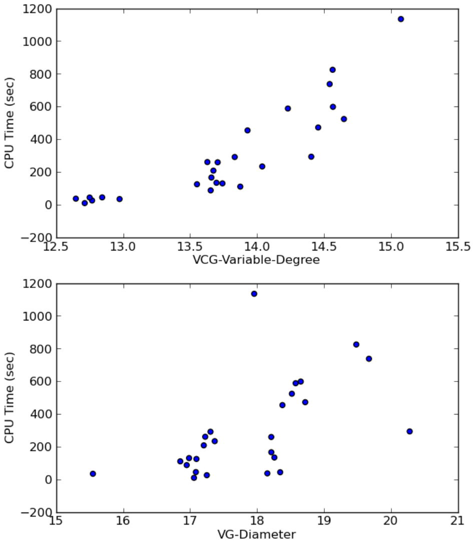


Figure 2.18: CPU time versus three CNF-SAT structural parameters:(1) VCG-Variable-Degree; average degree of variable-vertices in VCG graph, (2) VG-Degree; average degree of vertices in VG graph, and (3) VG-Diameter; average diameter of vertices in VG graph.

Comparison with benchmark Table 2.9 presents the values of various parameters for the 6 instances from the benchmark in Section 2.5. We observe from Table 2.9 that although the instances in the benchmark have large tree width, the corresponding CPU times are small. Therefore, tree width cannot explain the difference between the instances from the benchmark and the hard instances we generated. We recall that tree width only gives an upper bound for the computational complexity of an instance. We also observe that the value of parameters

VG-Degree, VG-Diamter and RRG-Degree cannot explain the CPU time difference between the hard instances and the instances in the benchmark. However, the value of RRG-R-degree and VCG-Variable-Degree, and the size of P_{lb} are significantly smaller for the UAQ instances from Section 2.5 compared to the hard UAQ instances. We increased the size P_{lb} to be as large as the size of P, and observed that the CPU time increases in most cases. Table 2.10 shows the CPU time for the new instances.

Table 2.9: Structural parameters for six UAQ instances from Section 2.5

n	CPU Time (sec)	$\lvert R\rvert$	$\lvert P_{lb}\rvert$	Tree width (lb)	Tree width (ub)	RPG-R-Degree	RRG-Degree	VCG-Variable-Degree	VG-Degree	VG-Diameter
uaq-R	0.097	200	7	24	116	7.5	14.30	11.56	1.28	23.34
uaq-RH	0.105	200	5	86	188	7.5	14.53	15.63	1.49	15.11
uaq-D	0.283	100	7	36	90	15	25.82	11.28	1.27	15.84
uaq-RPC	0.803	300	5	48	206	10	19.36	11.89	1.26	19.78
uaq-t	0.174	100	6	31	95	20	25.82	11.70	1.20	17.18
uaq-Plb	0.060	100	11	27	77	10	25.88	11.70	1.31	17.82

Table 2.10: CPU time for UAQ instances from Section 2.5 with $P_{lb} = P$

UAQ instance	CPU Time (sec)	$\lvert R\rvert$	$\lvert P\rvert$	$\lvert P_{lb}\rvert$
uaq-R	16.121	200	500	500
uaq-RH	0.126	200	500	500
uaq-D	0.287	100	500	500
uaq-RPC	5.326	300	1000	1000
uaq-t	> 1000	100	500	500
uaq-Plb	0.136	100	500	500

Table 2.11: UAQ structural parameters description

Feature	Description
RPG-R-Degree	Average degree of role-vertices in RPG graph
RPG-P-Degree	Average degree of permission-vertices in RPG graph
RPG-R-Diameter	Average diameter of role-vertices in RPG graph
RPG-P-Diameter	Average diameter of permission-vertices in RPG graph
RPG-R-Clustering	Average clustering coefficient of role-vertices in RPG graph
RPG-P-Clustering	Average clustering coefficient of permission-vertices in RPG graph
RRG-Degree	Average degree of vertices in RRG graph
RRG-Diameter	Average diameter of vertices in RRG graph
RRG-Clustering	Average clustering coefficient of vertices in RRG graph

Table 2.12: CNF-SAT structural parameters Description

Feature	Description
Variables	Number of variables in SAT formula
Clauses	Number of clauses in SAT formula
Clause-Variable-Ratio	Number of clauses divided by number of Variables
Unary-Clauses	Number of Clauses of size at most one
Binary-Clauses	Number of Clauses of size at most two
Ternary-Clauses	Number of Clauses of size at most three
Pos-Neg-Clause-Ratio	Average ratio of positive to negative literals in clauses
Pos-Neg-Variable-Ratio	Average ratio of positive to negative occurrence of variables
Horn-Variable	Average number of occurrence in a Horn clause of variables
Horn-Clause	Fraction of Horn clauses
VCG-Clause-Degree	Average degree of clause-vertices in VCG graph
VCG-Variable-Degree	Average degree of variable-vertices in VCG graph
CG-Degree	Average degree of vertices in CG graph
CG-Clustering	Average clustering coefficient of vertices in CG graph
VG-Degree	Average degree of vertices in VG graph
VG-Diameter	Average diameter of vertices in VG graph

Chapter 3

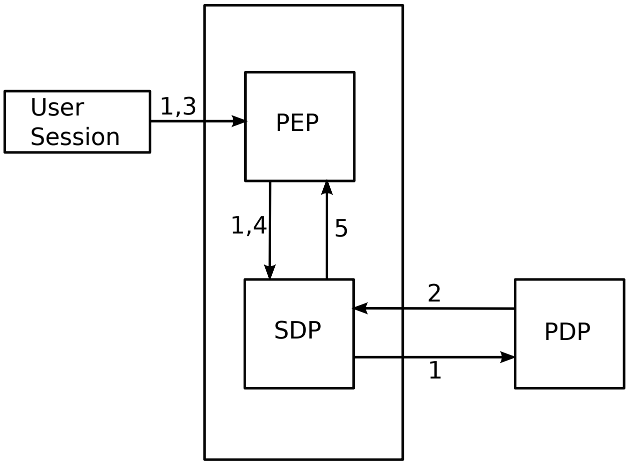
Cascade Bloom Filter for Distributed Access Enforcement

In this chapter, we study the problem of finding an optimal cascade Bloom filter. Cascade Bloom filter is a data structure proposed by [56] for distributed access enforcement applications. It has been validated empirically in [56] that the access enforcement approach based on a cascade Bloom filter is fast and space efficient. We formalize the problem of finding an optimal Cascade Bloom filter (CBF), and present our results on the computational complexity of the CBF problem. We address how the intractability of CBF can be mitigated, and discuss an empirical evaluation of our approach.

3.1 Introduction

In Role Based Access Control (RBAC) users acquire permissions via roles. A user creates a session to exercise a set of permissions. In the session, the user activates a subset of the roles to which he is authorized. When the user requests a permission in the session, the enforcement mechanism needs to check whether there exists an active role associated with the session that is authorized to that permission. If such a role exists, the request by the user is accepted. Access enforcement is the process in which an entity, called a reference monitor accepts or rejects the access request by a user (principal).

Distributed enforcement is an important setting in access control [1, 2, 8, 10, 20, 28]. An approach to the problem of efficient, and scalable access enforcement is to distribute enforcement across several reference monitors [33, 56, 58]. Wei et



1: Session initiation request

2: Access enforcement structure

3: Access request

4: Validated/translated access request

5: Access decision

Figure 3.1: An architecture for distributed access enforcement in RBAC. It is reproduced from prior work [58, 56]

al. [58] have proposed an architecture for distributed enforcement (see Figure 3.1). In Figure 3.1, the user first creates a session by issuing a request to the Policy enforcement Point (PEP) in which he specifies a list of roles that he wants to activate in that session. The PEP sends the request to the Secondary Decision Point (SDP), which forwards it to the Policy Decision Point (PDP). The PDP, which holds the RBAC policy, checks if the user is authorized to the roles that he wants to activate. If the answer is "yes", the PDP sends a data structure to SDP that can be used for the access enforcement by SDP. The SDP accepts or rejects an access request from the user using the data structure received from PDP.

An access request for a permission p in a session s is represented by the pair $\langle s, p \rangle$. Let S and P denote the set of active sessions and the set of permissions respectively. The set of valid requests $VRQ = \{\langle s, p \rangle : s \text{ is authorized to } p\}$ is a subset of $S \times P$. In the distributed access enforcement application, a data structure is used to represent the set VRQ. Prior work [56] proposed a data structure called cascade Bloom filter for the access enforcement application, and has shown empirically that the cascade Bloom filter is effective in practice. In this chapter, we study the problem of constructing an optimal cascade Bloom filter.

Layout In the next section, we discuss the Bloom filter. In Section 3.3, we discuss the Cascade Bloom filter and the problem of finding an optimal cascade Bloom filter (CBF).

3.2 Bloom Filter

A Bloom filter is a data structure proposed by Bloom [6] for the applications where the amount of memory required to store a data set with any error-free methods is impractically large. Since then, there has been a lot of work on Bloom filters, for example [9, 13, 15, 19, 34, 44, 45, 50]. A Bloom filter is a space efficient data structure that is used to represent a subset A of elements in a universe $\mathcal{U}$. We say that the Bloom filter *represents* A *against* $\mathcal{U} - A$; the Bloom filter supports membership queries, that is, whether an element of $\mathcal{U}$ is in A or not. A Bloom filter is an array M of m bits associated with a set H of hash functions $h : \mathcal{U} \rightarrow \{0, 1, \ldots, m-1\}$, and is represented by a tuple $\langle M, H \rangle$. All bits in the array are initially set to zero. To add an element $a \in A$ into the Bloom filter, the indices $h(a)$ for all $h \in H$ are computed and the corresponding bits in the array are set to one. To query for an element a, the indices $h(a)$ for all $h \in H$ are computed, and if any of the indices for a is not set to one, the element is surely not in A. If all are one, then the element is reported to be in A. It is possible that some elements not in A pass the membership query by coincidence. Such elements are called *false positives*.

For the purpose of access enforcement, we are required to respond to a membership query with no false positive errors. However, this requirement can be satisfied by keeping all the false positives in a list E [56]. An element is in A if and only if it tests positive with the Bloom filter and it is not in the list E.

Assuming that each hash function maps every element of U uniformly to an index in $\{0, 1, \ldots, m\}$, a formula can be derived for the false positive rate, i.e., the probability that an element not in A is a false positive. The formula enables one to minimize the false positive rate for a given A and M, by choosing the following value for the number of uniform hash functions.

$$k = \frac{m}{n} \ln 2$$

Where, k, m, and n are the size of H, M and A respectively. However, the uniformity assumption is not always true in practice. Thus, we study the problem of finding an optimal Bloom filter for a given set of elements $A \subseteq \mathcal{U}$, and a set of available hash functions, $\mathcal{H}$; we do not make any assumption for the hash functions in $\mathcal{H}$.

We can associate two types of costs to any Bloom filter that represents a set A with a bit array M and hash functions in H: (1) the memory cost of storing all the false positives in the list E, and (2) the computational cost associated with each membership query, which corresponds to the number of hash functions in H. Therefore, our goal is to find Bloom filters with fewer false positives and hash functions. In the following, we define the BF problem that is the optimization problem of finding an optimal Bloom filter with respect to two objectives: the number of false positives, and the number of hash functions used.

BF Specification. An optimization the BF problem is specified by the following inputs.

- $\mathcal{U}$: A finite universe of elements
- A : A subset of $\mathcal{U}$
- M : An array of m bits
- $\mathcal{H}$: A set of hash functions that map each element of $\mathcal{U}$ to a non-negative integer
- pri : $pri \in \{n_{fp}, n_h\}$ indicates which of the two optimization objectives, the number of false positives or the number of hash functions, we prioritize over the other. Without this parameter, there can exist two optimal solutions that are incomparable to one another.

Assuming that we prioritize the number of false positives over the number of hash functions, i.e., $pri = n_{fp}$, a solution to the optimization the BF problem is a Bloom filter $\langle M, H \rangle$ with a minimum number of false positives such that the size of $H \subseteq \mathcal{H}$ is minimum over all the solutions with the minimum number of false positives. A decision version of BF that corresponds to the above optimization version does not take the input pri. Instead two input integers:

- k_{fp}: this indicates the maximum number of false positives that we seek in a solution
- k_h: this indicates the maximum number of hash functions that a solution can use

A decision instance is either true or false. It is true if there exists a Bloom filter $\langle M, H\rangle$ that represents A against $\mathcal{U} - A$ with at most k_{fp} false positives and k_h hash functions (i.e., $|H| \leq k_h$).

From the decision version to the optimization version. There exists a polynomial-time Turing reduction [11] from the optimization versions of BF to the decision version. That is, given an oracle Ω for the decision version, we can solve the optimization version in polynomial time. An approach is performing a two-dimensional binary search for the number of false positives and the number of hash functions. For example, for the case that $pri = n_{fp}$, we first fix k_h at the total number of hash functions, $|\mathcal{H}|$; that is, we accept a solution with any number of hash functions. We then perform a binary search for the optimal number of false positives with $O(\log|\mathcal{U} - A|)$ invocations to Ω. Once we find the optimal number of false positives, opt_{fp}, we search for the optimal number of hash functions with $O(\log|\mathcal{H}|)$ invocations to Ω, while k_{fp} is set to opt_{fp}.

3.2.1 Complexity of the BF problem

In this section we discuss the computational complexity of the BF problem. The formal language for the corresponding decision problem is

$$\begin{aligned}\text{BF} = \{\langle \mathcal{U}, A, M, \mathcal{H}, k_{fp}, k_h\rangle : \ & \text{there exists a Bloom filter } \langle M, H\rangle \text{ that}\\ & \text{represents } A \text{ against } \mathcal{U} - A \text{ such that}\\ & \text{the number of false positives is at most } k_{fp},\\ & \text{and the size of } H \subseteq \mathcal{H} \text{ is at most } k_h\}.\end{aligned}$$

The following theorem shows that an efficient algorithm for the BF problem is unlikely to exist.

Theorem 24. *The BF problem is* ***NP****-hard.*

Proof. We prove it by showing that SET-COVER $\leq_p$ BF. (see Section 2.3.1 for a definition of SET-COVER.) Given an instance of SET-COVER, $\phi = \langle \mathcal{U}, \mathcal{F}, k\rangle$, we construct an instance ψ of BF such that ψ is true if and only if ϕ is true.

We construct $\psi = \langle \mathcal{U}', A, M, \mathcal{H}, k_{fp}, k_h\rangle$ as follows. The universe $\mathcal{U}'$ consists of an element x_i for each e_i in $\mathcal{U}$, as well as an additional element x_{n+1}, where n denotes the size of $\mathcal{U}$. The set A contains only the element x_{n+1}. That is, $\mathcal{U}' = \{x_1, x_2, \ldots, x_n, x_{n+1}\}$ and $A = \{x_{n+1}\}$. The set of hash functions, $\mathcal{H}$, consists of a hash function h_j for each subset S_j in F. The bit array, M, consists of two bits, and each hash function maps an element to either index zero or one. Each hash function h_j is defined as below.

$$h_j(x_i) = \begin{cases} 1 & \text{if } i \neq n+1 \text{ and } e_i \text{ is in } S_j \\ 0 & \text{otherwise} \end{cases}$$

We set k_{fp} to zero, and k_h to k.

Suppose that $S' \subseteq \mathcal{F}$ is a set cover of size k. The Bloom filter $\langle M, H \rangle$ where $H = \{h_j : S_j \in S'\}$ represents A with no false positives because each x_i in $\mathcal{U}' - A$ is mapped by at least one hash function in H to index one, which is not set in M. Conversely, assume that there exists a Bloom filter $\langle M, H \rangle$ that represents A with no false positives and $|H| \leq k$. The bit zero is the only bit in M that is set to one. The fact that the number of false positives is zero implies that there exists a hash function in H for each x_i in $\mathcal{U} - A$ that maps x_i to index one. Therefore, the set $S' = \{S_j : h_j \in H\}$ is a set cover of size at most k.

□

The reduction in the proof of Theorem 24 shows that the hardness of the BF problem is related to minimizing the second objective, i.e., the number of hash functions. However, the following theorem shows that optimizing the number of false positives is also **NP**-hard.

Theorem 25. *The BF problem is **NP**-hard even if* $k_h = |\mathcal{H}|$.

Proof. We prove it by showing that SET-COVER $\leq_p$ BF. (see Section 2.3.1 for a definition of SET-COVER.) Given an instance of SET-COVER instance, $\phi = \langle \mathcal{U}, \mathcal{F}, k \rangle$, we construct an instance of BF, $\psi = \langle \mathcal{U}', A, M, \mathcal{H}, k_{fp}, k_h \rangle$ as follows. Let n and m denote the size of $\mathcal{U}$ and F in ϕ respectively. The universe $\mathcal{U}'$ consists of $(n+2)m+1$ elements $x_1, x_2, \ldots, x_{(n+2)m+1}$ from which the first m elements, and the last element are in A, i.e., $A = \{x_1, x_2, \ldots, x_m, x_{(n+2)m+1}\}$. Bit array M is an array of $m+1$ bits. The set of hash functions, $\mathcal{H}$, consists of a hash function h_j for each set S_j in $\mathcal{F}$, where h_j is defined as below.

$$h_j(x_i) = \begin{cases} j+1 & \text{if } i \leq m \\ i+1-m & \text{if } m+1 \leq i \leq 2m \\ 1 & \text{if } 2m+1 \leq i \leq (n+2)m \text{ and } e_k \text{ is in } S_j, \\ & \text{where } k = \lceil \frac{i-2m}{m} \rceil \\ 0 & \text{o.w.} \end{cases}$$

We set k_{fp} to k, and k_h to $|\mathcal{H}|$. We claim that the BF instance is true if and only if the SET-COVER instance is true.

Suppose that there exists a set cover S' of size at most k. Then we show that $\langle M, H\rangle$ where $H = \{h_j : S_j \in S'\}$ is a solution for ψ. After adding all elements of A using hash functions in H, the array M has $|H| + 1$ bits set to one: bit zero since any hash function maps $x_{(n+2)m+1}$ to zero, and bit $j + 1$ for any hash function h_j in H. Let I denote the indices in M that are set to one, i.e., $I = \{j + 1 : h_j \in H\} \cup \{0\}$. Since S' is a set cover, any $x_i \in \mathcal{U}' - A$ for $i \geq 2m + 1$ is mapped to one by at least one hash function in H; none of $x_{2m+1}, x_{2m+2}, \ldots, x_{(n+2)m}$ is false positive. Any of $x_{m+1}, x_{m+2}, \ldots, x_{2m}$ is false positive if and only if $i + 1 - m \in I$. Therefore, the number of false positives is at most k. Conversely, assume that $\langle M, H\rangle$ is a solution to ψ. We show that $S' = \{S_j : h_j \in H\}$ is a set cover. Assume toward a contradiction that there exists an element e_k that S' does not cover. So, all of $x_{(k+1)m+1}, \ldots, x_{(k+2)m}$ are false positives since any hash function in H maps them to zero, which is set to one. Now, we show that the size of H is at most k, so is the size of S'. Assume toward a contradiction that there are at least $k + 1$ hash functions in H. The array M has at least $k + 2$ bits set to one after adding all elements of A, because each hash function h_j in H sets two bits to one: bit zero, and bit $j+1$. Each nonzero bit of index $j \neq 0$ in M makes x_{m+j-1} to be a false positive. Therefore, the number of false positives is at least $k + 1$, which contradicts the assumption that H is a solution to ψ. □

Theorem 24 gives a lower bound for the hardness of BF problem, i.e., BF $\in$ **NP**-hard. However, the following theorem establishes an upper bound for the hardness of the BF problem is in **NP**.

Theorem 26. *The BF problem is in **NP**.*

Proof. We prove that there exists a polynomial certificate for BF, which can be verified in polynomial time. A certificate for BF is a subset of size at most k_h of $\mathcal{H}$. An algorithm to verify the certificate first constructs a M by adding all elements of A using hash functions in H. Then, it checks for each element of $\mathcal{U} - A$ whether it is a false positive. The algorithm accepts the certificate if the number of false positives is at most k_{fp}. The verification algorithm runs in time $O(|\mathcal{U}||H|T_h)$ where T_h is the time complexity of computing each hash function. We assume T_h is polynomial in $|\mathcal{U}|$ and $|M|$, and therefore the verification algorithm is polynomial in the size of the instance. □

Theorem 26 suggests a way for mitigating the intractability of the Bloom Filter problem — efficient reduction to CNF-SAT. We discuss this approach in the next section.

3.2.2 Efficient Reduction to CNF-SAT

In this section, we discuss how the intractability of the BF problem can be mitigated. We investigate a standard approach in which we reduce the BF problem to CNF-SAT for which solvers exist that are efficient for large classes of instances. The fact that BF is in **NP** implies that there exists a polynomial-time many-one reduction from BF to CNF-SAT. We present an efficient reduction, which is based on designing a circuit that decides BF. Our reduction to CNF-SAT involves reducing BF first to a circuit SAT problem, and then reducing the circuit SAT to CNF-SAT.

Let n denote the size of the universe $\mathcal{U}$. We encode the set A with n binary variables $\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{n}$ in our circuit, where circuit variable x_i is one if and only if element x_i is in A. We say $\mathrm{X} = \{\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{n}\}$ encodes A. Similarly, we encode $\mathcal{U} - A$ with n variables $\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n} \in \mathrm{Y}$. The set of circuit variables $\mathrm{K} = \{\mathrm{k}_1, \mathrm{k}_2, \ldots, \mathrm{k}_\mathrm{m}\}$ is a binary encoding for k if $m = \lceil \log k \rceil$, and $\sum_i \mathrm{k}_\mathrm{i} 2^i = k$.

The circuit has five inputs: X, which is the encoding of A; Y, the encoding of $\mathcal{U} - A$; W, an encoding of the size of M; K_h, the binary encoding of k_h; and K_fp, the binary encoding of k_{fp}. The output is one if and only if there exists a solution to the corresponding BF instance. As shown in Figure 3.2, the circuit consists of five components, which we explain in the following.

Hash Valid. A hash function in $\mathcal{H}$ may map an element of $\mathcal{U}$ to an index greater than $|M|$. Hash Valid is a module that determines if a hash functions is valid to be used in the Bloom filter. A Hash Valid module is shown in Figure 3.3. The first input to Hash Valid is W, which is the binary encoding of the size of the bit array M. Hash Valid uses a compare module from Section 2.4.3 to decide whether a hash function is valid; a hash function is valid if the maximum to which it maps an element of the universe is less than m. For each hash function h_j, we have a circuit variable $\mathrm{h}_\mathrm{j,val}$ that is one if and only if h_j maps every element of the universe to an integer in $\{0, 1, \ldots, m - 1\}$. Each variable $\mathrm{h}_\mathrm{j,val}$ is an input to the AND gate whose output $\mathrm{h}_\mathrm{j,sel}$ determines whether the hash function h_j is selected. $\mathrm{H}_\mathrm{sel} = \{\mathrm{h}_\mathrm{1,sel}, \mathrm{h}_\mathrm{2,sel}, \ldots, \mathrm{h}_{|\mathcal{H}|,\mathrm{sel}}\}$ is the output of Hash Valid, which is also an input to Filter Array and False Positives modules. Hash Valid consists

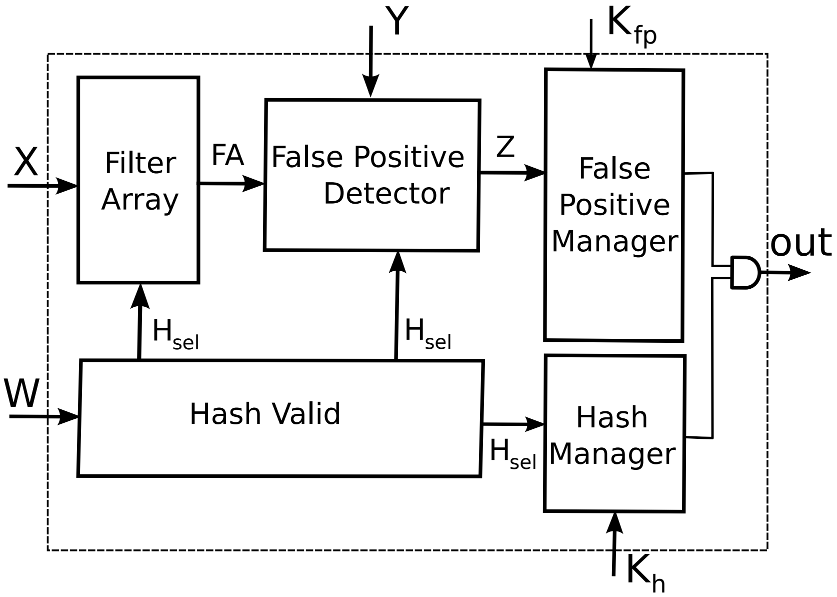


Figure 3.2: A circuit to decide the BF problem.

of $|\mathcal{H}|$ compare modules and $|\mathcal{H}|$ AND gates. Thus, the total number of gates in Hash Valid is $O(|\mathcal{H}| \log |M|)$.

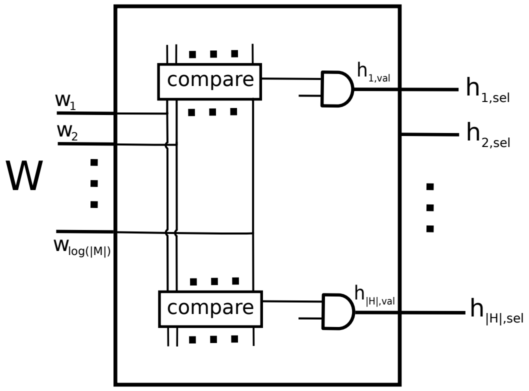


Figure 3.3: The Hash Valid module.

Filter Array. The first input to Filter Array is X, the encoding of A. The second input to the Filter Array is $\mathrm{H}_{\mathrm{sel}}$, Hash Valid's output. The output of Filter Array is the set of circuit variables $\mathrm{FA} = \{\mathrm{f_1}, \mathrm{f_2}, \ldots, \mathrm{f_{|M|}}\}$ that encodes the bits in the array M. We have the following relation between each $\mathrm{f_k}$ and input variables.

$$\mathrm{f_k} = \bigvee_{(\mathrm{x_i}, \mathrm{h_{j,sel}}) \in D} (\mathrm{x_i} \wedge \mathrm{h_{j,sel}}) \qquad \text{where } \ D = \{(\mathrm{x_i}, \mathrm{h_{j,sel}}) : h_j(x_i) = k\}$$

Filter Array has $O(|\mathcal{U}||\mathcal{H}|)$ gates.

False Positive Detector. Three inputs to a False Positives Detector module are Y, the encoding of $\mathcal{U} - A$, $\mathrm{H_{sel}}$, and FA. The output is $\mathrm{Z} = \{\mathrm{z_1}, \ldots, \mathrm{z_{|\mathcal{U}|}}\}$ that encodes the set of false positives. A circuit variable $\mathrm{z_k}$ is one if and only if element x_k is a false positive. An element $x_k \in \mathcal{U} - A$ passes a hash function h_j if either h_j is not selected or the bit to which x_k is mapped by h_j is set to one. An element x_k is a false positive if it is in $\mathcal{U} - A$ and it passes every hash function. We have the following relation between each circuit variable $\mathrm{z_k}$ and input variables.

$$\mathrm{z_k} = \bigwedge_{\mathrm{j}} (\mathrm{y_k} \wedge (\mathrm{h_{j,sel}} \wedge \mathrm{f_l})) \qquad \text{where } \ l = h_j(x_k)$$

False Positive Detector consists of $O(|\mathcal{U}||\mathcal{H}|)$ gates. False Positive Detector is shown in Figure 3.4.

Hash Manager. Hash Manager checks whether the total number of hash functions selected is less than k_h. Two inputs to Hash Manager are $\mathrm{H_{sel}}$ and $\mathrm{K_h}$, an binary encoding of integer k_h. The Hash Manager module consists of a Max-Circuit module from Section 2.4.3. The output is one if the total number of hash functions selected is less than the k_h. Hash Manager consists of $O(|\mathcal{H}| \log |\mathcal{H}|)$ gates.

False Positive Manager. False Positive Manager checks whether the total number of false positives is less than k_{fp} using a Max-circuit module from Section 2.4.3. False Positive Manager has two inputs: Z, the output of a False Positive Detector module; and $\mathrm{K_{fp}}$, an binary encoding of k_{fp}. The total number of gates in False Positive Manager is $O(|\mathcal{U}| \log(|\mathcal{U}|)$.

We adopt a "textbook" reduction from CIRCUIT-SAT to CNF-SAT [14] to generate the CNF formula that corresponds to the circuit. The total number of the gates in the circuit that decides an instance of BF is $O(n^2)$ where n is the size of the instance. Thus, the CNF-SAT formula is of size $O(n^2)$. This proves that the proposed reduction to CNF-SAT is efficient.

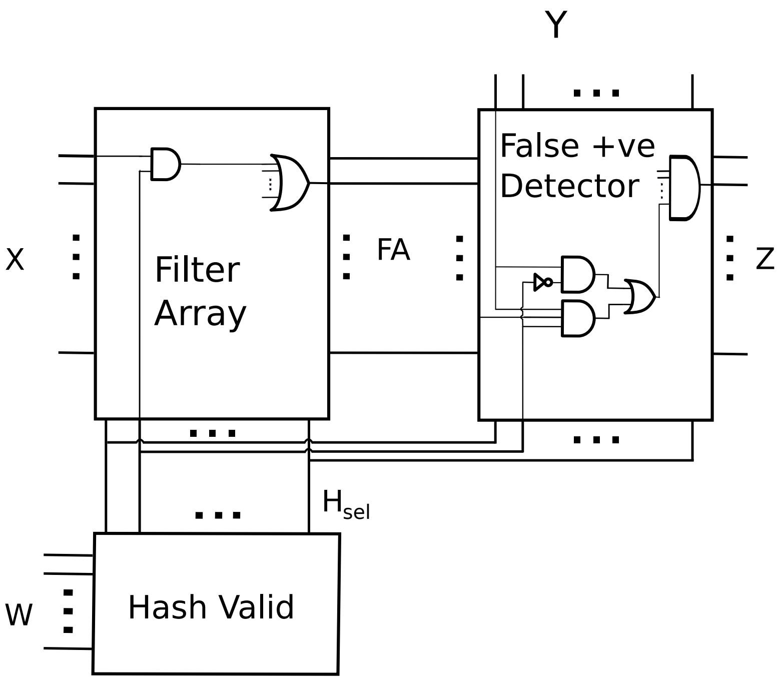


Figure 3.4: Filter Array, False Positives, and Hash Valid modules.

3.3 Cascade Bloom Filter

In this section, we discuss the cascade Bloom filter, a generalization of the Bloom filter, which is proposed by [56]. A Cascade Bloom filter represents a set A against $\mathcal{U} - A$ by employing a cascade of Bloom filters. The basic idea of cascading multiple Bloom filters is to use the next Bloom filter to distinguish between two sets that the previous Bloom filter failed to, i.e., the false positives of the previous Bloom filter and the set that the previous Bloom filter represents.

A cascade Bloom filter is specified with d Bloom filters $\mathcal{BF}_1, \mathcal{BF}_2, \ldots, \mathcal{BF}_d$, where Bloom filter $\mathcal{BF}_1$ represents A against $\mathcal{U} - A$, and Bloom filter $\mathcal{BF}_i$ represents FP_{i-1}, the false positives in the previous level, against A_i. Similar to the traditional Bloom filter, the false positives in the last Bloom filter, FP_d, are stored in an explicit list E in order to perform membership query with no error.

The idea of using multiple Bloom filters has been proposed [13, 15]. The Cascade Bloom filter is an adapted version of the Bloomier filter [13] suited to the purpose of access checking [56]. The Authors in [56] give an example that shows how a cascade Bloom filter with only two levels outperforms a Bloom filter by 33% less false positives, while it uses the same number of hash functions and allocates the same amount of memory to filter arrays.

Definition 4. *(Cascade Bloom Filter)[56]*
A cascade Bloom filter is $\langle \mathcal{B}, E \rangle$, where $\mathcal{B} = \mathcal{BF}_1, \mathcal{BF}_2, \ldots, \mathcal{BF}_d$ is a list of Bloom filters and $E \subseteq \mathcal{U}$ is a set of elements from a universe $\mathcal{U}$. Each $l = 1, \ldots, d$ is called a level and d is called the depth of the cascade. Each $\mathcal{BF}_i = \langle M_i, H_i \rangle$ represents a set $A_i \subseteq \mathcal{U}$ against a set $B_i \subseteq \mathcal{U}$, such that for $i = 2, \ldots, d$, A_i is the set of false positives in $\mathcal{BF}_{i-1}$ and B_i is equal to A_{i-1}, with $A_1 = A$ and $B_1 = \mathcal{U} - A$. The set E is the set of false positives in BF_d. We say that the cascade Bloom filter represents A against $\mathcal{U} - A$.

The *total memory size* of a cascade Bloom filter is the sum of the array size at each level, i.e., $\sum_i m_i$. The *total number of hash functions used* in a cascade Bloom filter is the sum of the number of hash functions used at each level, i.e., $\sum_i |H_i|$. Before defining the problem of finding an optimal cascade Bloom filter, we discuss two examples. The first example shows how a cascade Bloom filter can reduce the number of false positives when a traditional Bloom filter can not. It also explain why the filter BF_i, which is to distinguish between FP_{i-1} and A_{i-1}, should represent FP_{i-1}, not A_{i-1}.

Example 3. *Assume we want to represent $A = \{x_1, x_2\}$ against $\mathcal{U} - A = \{x_3\}$. The set of available hash functions is $\mathcal{H} = \{h_1, h_2\}$ where h_1 and h_2 are binary hash functions. $h_1(x) = 1$ if and only if $x = x_1$, and $h_2(x) = 0$ if and only if $x = x_2$. The element x_3 is a false positive for any Bloom filter $\langle M, H \rangle$. However, we can have zero false positive using a cascade Bloom filter with three Bloom filters: $\mathcal{BF}_1 = \langle M_1, H_1 \rangle$, where $|M_1| = 0$ and $H_1 = \emptyset$; $\mathcal{BF}_2 = \langle M_2, H_2 \rangle$, $|M_2| = 2$ and $H_2 = \{h_1\}$; and $\mathcal{BF}_3 = \langle M_3, H_3 \rangle$, $|M_3| = 2$ and $H_3 = \{h_2\}$.*

The following example shows that the number of false positives may not increase monotonically as the number of levels increase in the cascade Bloom filter.

Example 4. *Let $\mathcal{U} = \{x_1, x_2, \ldots, x_{n+1}\}$, $A = \{x_1, x_2, \ldots, x_n\}$. The set of available hash functions is $\mathcal{H} = \{h_1, h_2, \ldots, h_n\}$, where each hash function h_i is defined as below.*

$$h_j(x_i) = \begin{cases} j \mod 2 & i = j \\ j - 1 \mod 2 & \textit{o.w.} \end{cases} \tag{3.1}$$

If $Min_{fp}(d)$ denotes the minimum number of false positives that can be achieved with a cascade Bloom filter of depth d, we have the following.

$$Min_{fp}(d) = \begin{cases} 1 & \textit{if } 0 < d < 2n-1 \textit{ and } d \textit{ is odd} \\ n-k & \textit{if } 0 < d < 2n-1 \textit{ and } d \textit{ is even} \\ 0 & \textit{if } d \geq 2n-1 \end{cases}$$

The minimum number of false positives can be achieved by choosing h_j for the Bloom filter at level $2j$ and any hash function for the Bloom filter at level $2j-1$ for $j = 1, \ldots, \lceil d/2 \rceil$. The size of the bit array at each level is 2. Figure 3.5 shows $Min_{fp}(d)$ as d increases for $n = 10$.

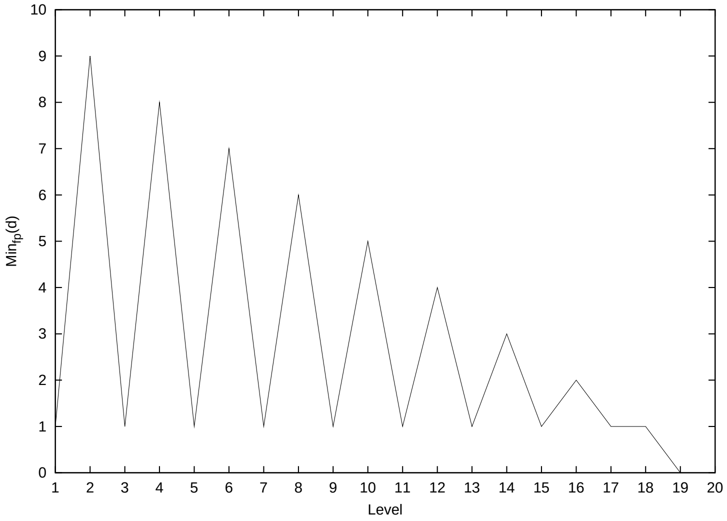


Figure 3.5: The minimum number of false positives that can be achieved with a cascade Bloom filter of depth d for $U = \{x_1, x_2, \ldots, x_{11}\}$, $A = \{x_1, x_2, \ldots, x_{10}\}$, and $H = \{h_j : j = 1, 2, \ldots, 10\}$ defined in Equation 3.1

Similar to the traditional Bloom filter, we define the problem of finding an optimal cascade Bloom filter of depth d with respect to two objectives: the number of false positives, and total number of hash functions used. It is a two dimensional optimization problem, which we define formally below.

CBF Specification. An optimization Cascade Bloom Filter problem (CBF) is specified by the following inputs.

- $\mathcal{U}$: A finite universe of elements
- A : A subset of $\mathcal{U}$
- d : A positive integer
- M : An array of m bits
- $\mathcal{H}$: A set of hash functions that map each element of $\mathcal{U}$ to a non-negative integer
- pri : $pri \in \{n_{fp}, n_h\}$ indicates which of the two optimization objectives, the number of false positives or the number of hash functions, we prioritize over the other.

A solution to the optimization version of the CBF problem is a cascade Bloom filter of depth d with total memory of m that minimize the number of false positives and hash functions used. A decision version of the CBF problem that corresponds to the above optimization version takes all inputs of the optimization version, but the input pri, as well as two input integers:

- k_{fp}: this indicates the maximum number of false positives that we seek in a solution
- k_h: this indicate the maximum number of hash function that a solution can use

The decision and optimization versions of Cascade Bloom Filter problem are related closely. Given an oracle Ω for the decision version, we can solve the optimization version using two dimensional binary search approach.

3.3.1 Complexity of the CBF problem

The formal language for the corresponding decision version is

$$\begin{aligned}\text{CBF} = \{\langle \mathcal{U}, A, d, M, \mathcal{H}, k_{fp}, k_h\rangle : &\text{ there exists a cascade Bloom filter } \langle B, E\rangle \text{ of}\\ &\text{depth } d \text{ that represents } A \text{ against } \mathcal{U} - A \text{ such}\\ &\text{that the number of false positives is at most } k_{fp},\\ &\text{the total hash functions used is at most } k_h,\\ &\text{and the total memory size is at most } |M|\}.\end{aligned}$$

The BF problem is a special case of the CBF problem with $d = 1$. Since the Bloom Filter Problem is **NP**-hard, it is unlikely that there exists an efficient algorithm for the Cascade Bloom Filter.

Theorem 27. *The CBF problem is **NP**-complete.*

Proof. CBF is **NP**-hard because it generalizes the BF problem, which is proved to be **NP**-hard in Theorem 24.

We show that CBF $\in$ **NP**. A certificate for an instance of CBF is a list of d Bloom filters, $\mathcal{BF}_1, \mathcal{BF}_2, \ldots, \mathcal{BF}_d$. The size of certificate is $O(d(|\mathcal{H}| + |\mathcal{M}|))$, which is polynomial in the size of the instance since $d = O(|M|)$.

The verification algorithm first checks whether $\sum_i |H_i| \leq k_h$ and $\sum_i m_i \leq |M|$. It then computes the sets A_i and B_i for each $i = 1, \ldots, d$, and checks whether hash functions selected for each level are valid; that is, $h \in H_i$ maps each elements of $A_i \cup B_i$ to an integer less than m_i. Finally, it computes the set of false positives, and checks if its size is less than k_{fp}. The verification algorithm runs in time $O(d|\mathcal{U}||\mathcal{H}|T_h)$, where T_h is the time complexity of computing each hash function. We assume T_h is polynomial in $|\mathcal{U}|$ and $|M|$. Therefore, the verification can be performed in polynomial time. □

Theorem 27 establishes an upper bound for the complexity of the Cascade Bloom Filter problem.

3.3.2 Efficient Reduction to CNF-SAT

In this section, we discuss how the intractability of the CBF problem can be mitigated. Our approach is to reduce an instance of the CBF problem to a CNF-SAT formula and solve the SAT formula using a SAT solver. Since CBF is in **NP**, there exists an efficient reduction from CBF to CNF-SAT. We find the efficient reduction by designing an efficient circuit that decides CBF and then adopting a "textbook" reduction from CIRCUIT-SAT to CNF-SAT [14].

The circuit that decides CBF is shown in Figure 3.6. It has five inputs: X, which is the encoding of A; Y, an encoding of $\mathcal{U} - A$; W, the encoding of $|M|$; $\mathrm{K_h}$, the binary encoding of k_h; and K_{fp}, the binary encoding of k_{fp}. The output of the circuit is one if and only if there exists a solution to the corresponding CBF instance. The circuit consists of d modules of Bloom filters, $\mathrm{BF_1}, \mathrm{BF_2}, \ldots, \mathrm{BF_d}$, a Hash Manager module, a Memory Manager module, and a False Positive Manager module. We explain each module in the following.

Bloom filter A Bloom Filter module is similar to the one described in the Section 3.2.2, except that it does not have False Positive Manager and Hash Manager modules (see Figure 3.7). A Bloom Filter module has two inputs: $\mathrm{X_i}$, the encoding of A_i; $\mathrm{Y_i}$, the encoding of B_i, and three outputs: $\mathrm{Z_i}$, the encoding of FP_i; $\mathrm{H_{i,sel}}$, the encoding of the set of hash functions selected at level i; $\mathrm{W_i}$, the binary encoding of m_i. A Bloom filter module consists of three modules: a Filter Array, a False Positives Detector, and a Hash Valid, which are described in the Section 3.2.2.

Hash Manager Hash Manager checks if the total number of hash functions used is at most k_h. It consists of a Max-circuit with inputs $\mathrm{H_{1,sel}}, \mathrm{H_{2,sel}}, \ldots, \mathrm{H_{d,sel}}$ and $\mathrm{K_h}$. A Hash Manager module consists of $O(d|\mathcal{H}| \log d|\mathcal{H}|)$ gates.

Memory Manager Memory Manager checks if the total memory size is at most $|M|$. The inputs to a Memory Manager module are the binary encoding of memory size of each level, i.e., $\mathrm{W_1}, \mathrm{W_2}, \ldots, \mathrm{W_d}$, and the binary encoding of the maximum memory size allowed, W. Memory Manager outputs one if and only if the total memory size is less than $|M|$. A Memory Manager module consists of a Max-circuit module from Section 2.4.3, and therefore has $O(d \log |M| \ (\log d + \log \log |M|))$ gates.

False Positive Manager False Positive Manager checks if the number of false positives is at most k_{fp}. The inputs to a False positive Manager module are $\mathrm{Z_d}$, the encoding of the false positives at level d, and $\mathrm{K_{fp}}$, the binary encoding of k_{fp}. False Positive Manager outputs one if and only if the number of false positives at most k_{fp}. A False Positives Manager module consists of a Max-circuit module from Section 2.4.3 , and therefore has $O(|U| \log |U|)$ gates.

The total number of gates in the circuit for CBF is $O(dn^2)$, where n is the size of an instant of CBF. Since $d = O(|M|)$, the total size of the circuit, and therefore the size of the corresponding CNF SAT formula is $O(n^3)$.

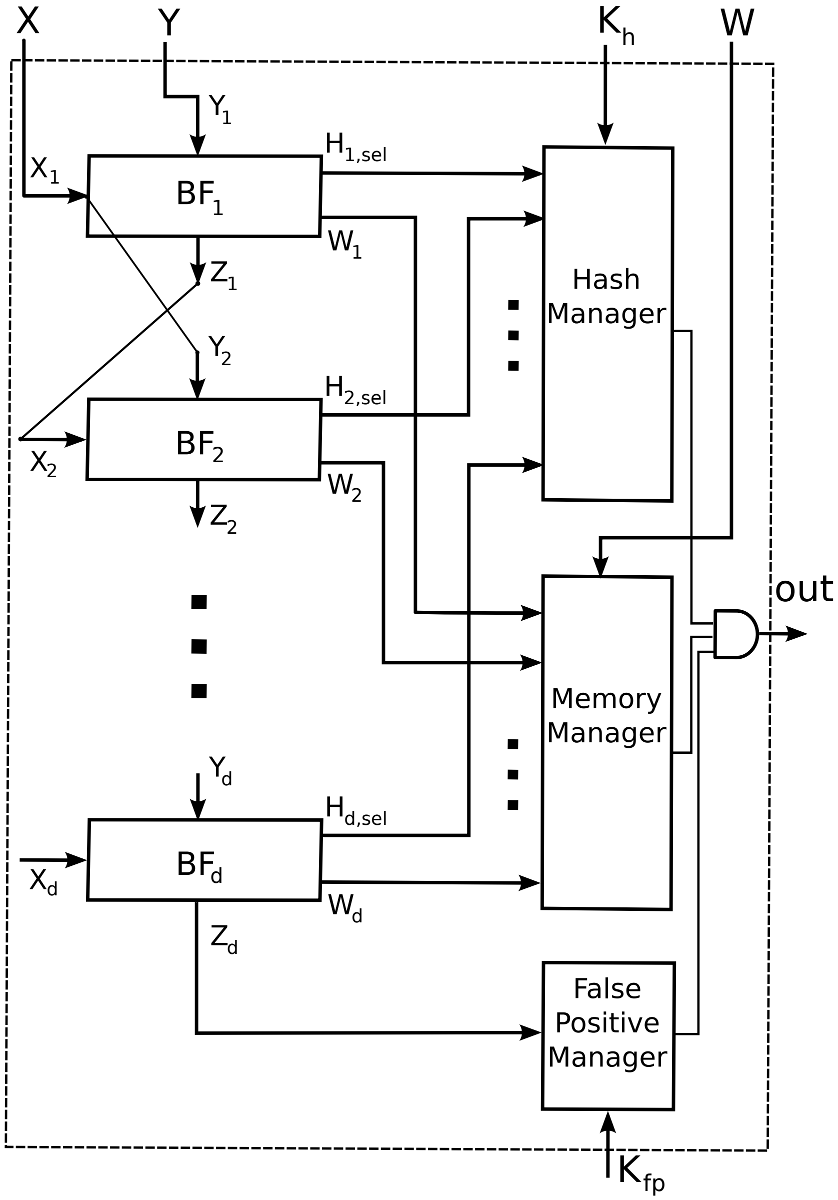


Figure 3.6: A circuit to decide the CBF problem.

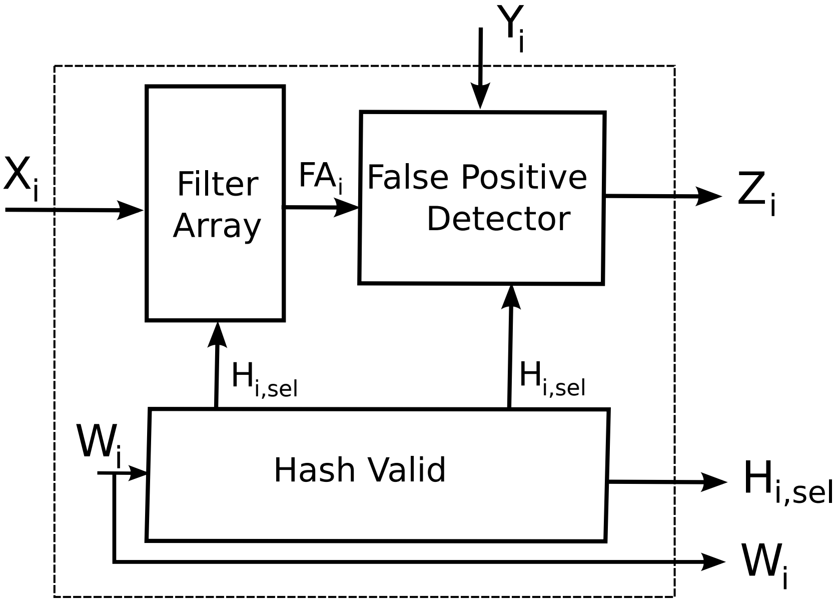


Figure 3.7: The Bloom filter module at level i, BF_i.

3.4 Empirical Evaluation

We have implemented our CNF SAT approach to the Cascade Bloom Filter problem. We used MiniSat [35], an open source SAT solver, to solve the SAT formula from the reduction in Section 3.3.2. The input for the empirical assessment has been generated based on the benchmark of RBAC instances in [30]. Each generated RBAC consists of two users, $2n$ roles, $10n$ permissions. Each user is assigned to a role with probability 0.5. Each permission is connected to k random roles where k is less than 8. There is also a role-hierarchy of depth 3, generated according to stanford model (see [30]). The generated RBAC policies are used to create CBF instances for our approach. For each RBAC instance with $2n$ roles, we create a session profile by instantiating n sessions. Each session profile comprises access pairs $\langle session\ id, permission\rangle$ for the permissions allowed in the sessions. In each session we choose the set of allowed permissions as below.

- Randomly pick one of the two users, as the user for the session (call it u)
- Randomly pick one role, r, from the roles to which u is authorized
- Collect all the junior roles S for r (including r itself)
- Collect all the permissions to which at least a role in S is authorized
- Output that set of permissions for that session

The sets A and $\mathcal{U}$ in the corresponding CBF instance are the set of all access pairs (i.e., $\langle session\ id, permission\rangle$) in the session profile and the set of all possible access pairs respectively. Each generated CBF instance is represented with the quantity "Problem size", which is the number of different sessions in the corresponding session profiles (i.e., n). All data points in our graphs represent a mean across at least 10 different inputs generated randomly.

All our implementations are available for public download [39]. Our empirical evaluations were conducted on a desktop PC with an Intel Dual Core E8400 CPU, each of which clocks at 3 GHz and has a 6 MB cache. The machine runs the 32-bit Ubuntu Linux 10.04 LTS operating system and has a 4 GB RAM.

Overall Observations. Our approach results in cascade Bloom filters with significantly fewer false positives than prior work. We observe that a cascade Bloom filter with more levels produces fewer false positives, and in some cases requires fewer hash functions to achieve the same number of false positives. Furthermore, our implementation can be adapted to trade off between efficiency (CPU time) and the quality of the solution, i.e., the number of false positives and hash functions used in a solution.

We present our specific observations in the following sections.

3.4.1 Comparison with the prior approach

We were given access to the implementation of the prior work [56]. We first compared the performance of two approaches in optimizing the number of false positives when there is no constraint for the number of hash functions used. Figure 3.8 shows the result of that comparison. We observe that our approach is resilient to the size of the problem, and is always able to find a cascade Bloom filter with a small number of false positives as the problem size increases.

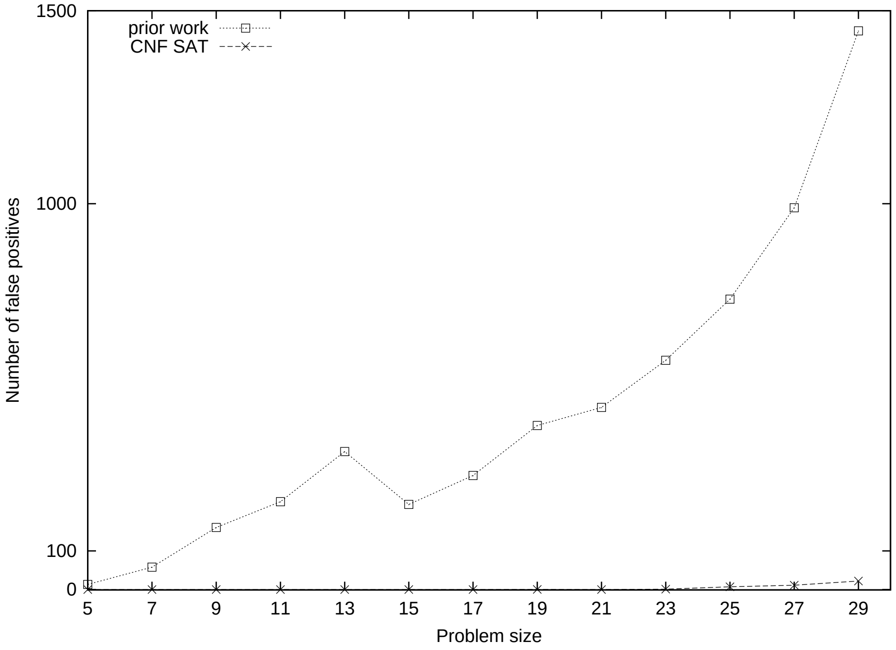


Figure 3.8: Performance comparison of our approach and prior approach in terms of the number of false positives for different problem sizes. The graph shows the number of false positives in the optimal solutions returned by two approaches.

To compare the performance of the two approaches in minimizing the number of hash functions, we set a constraint for the number of false positives to be no more than half of the maximum number of false positives, i.e., $|\mathcal{U} - A|/2$ where $\mathcal{U}$ is the universe and A is the set to be represented by the cascade Bloom filter. However, We were not be able to compare two approaches because the prior work fails for most instances. Figure 3.9 shows the success rates for the two approaches.

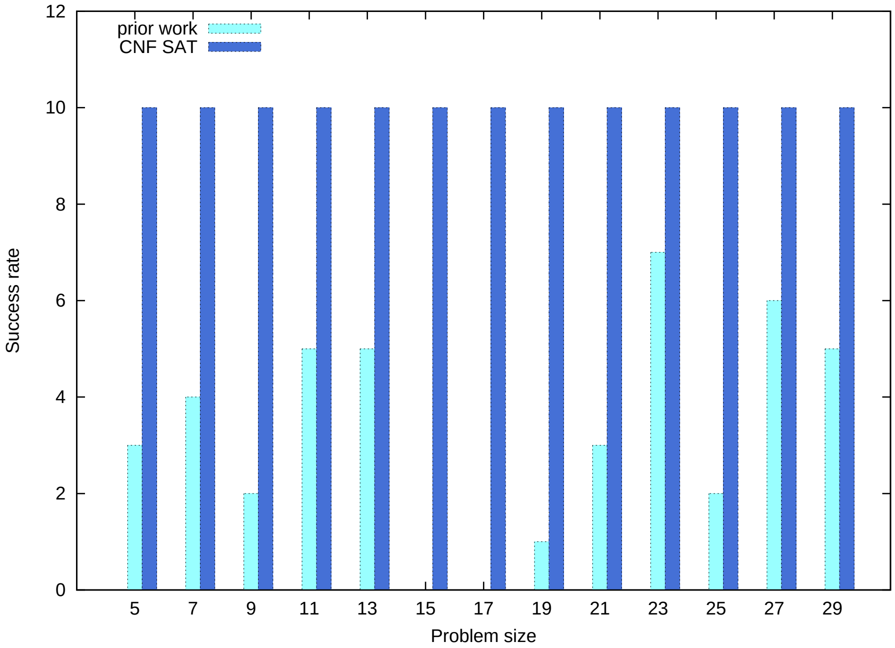


Figure 3.9: Comparison of the success rates of our approach and the prior approach for different problem size. Each bar represents the number of instances out of 10 instances for which each approach returns a solution.

3.4.2 Efficiency of our approach

We observe that the prior work always returns in few seconds, while it takes more time for our approach to find the optimal solution. However, our empirical results in the previous section show that the prior approach does not find an optimal solution, and it is not even complete; it may not return a solution for hard instances.

Figure 3.10 shows the CPU time for our approach to find a cascade Bloom filter with minimum number of false positives as the problem size increases. Figure 3.11 shows the CPU time for our approach to find a cascade Bloom filter with minimum number of hash functions. We observe that it takes few minutes for our approach to find the optimal cascade Bloom filter for large problem size. We are able to trade off between the CPU time and the quality of solution, i.e., the number of false positives and the number of hash functions, by introducing a time limit for each invocation to the Sat solver, MiniSat, in our binary search for finding the optimal solution. If MiniSat does not return an answer within the time limit, the approach would deem that the instance provided to MiniSat is unsatisfiable. Indeed, the time limit affects the quality of the solution as there might be the case that the instance deemed unsatisfiable is just a hard satisfiable instance for MiniSat. Figure 3.12 shows how the time limit affects the number of false positives in the solution that our approach returns.

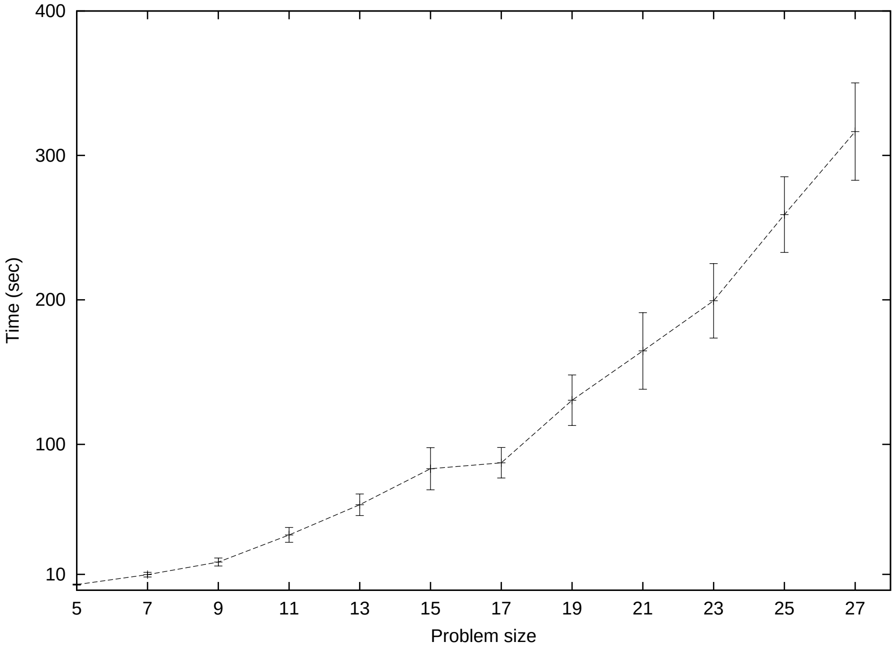


Figure 3.10: CPU time for our approach to find a solution with minimum number of false positives for different problem size. Each data point represents an average across 100 different inputs of the same size. The vertical line segment at each data point shows the 95% confidence interval.

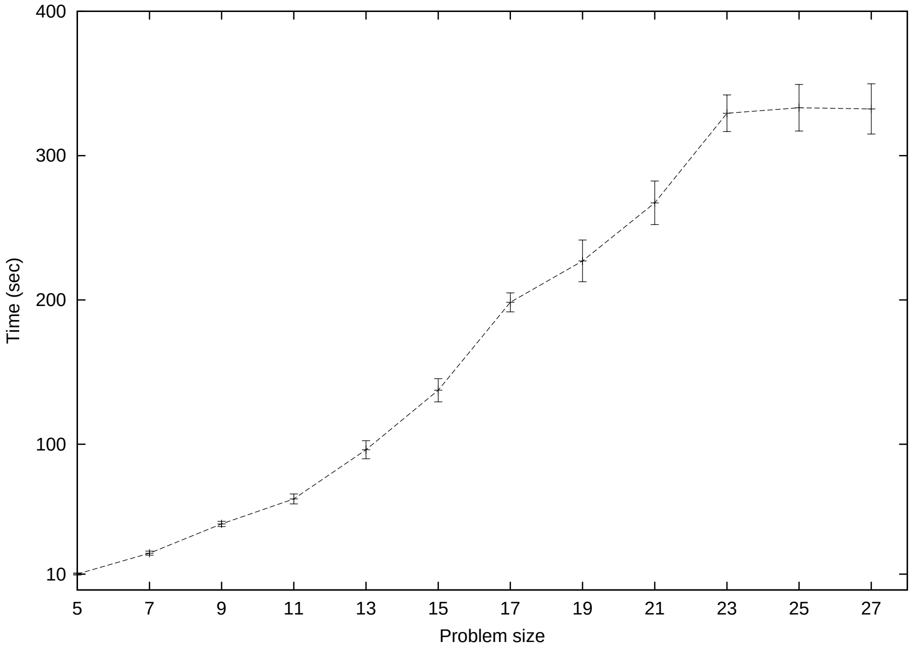


Figure 3.11: CPU time for our approach to find a solution with minimum number of hash functions for different problem size. Each data point represents an average across 100 different inputs of the same size. The vertical line segment at each data point shows the 95% confidence interval.

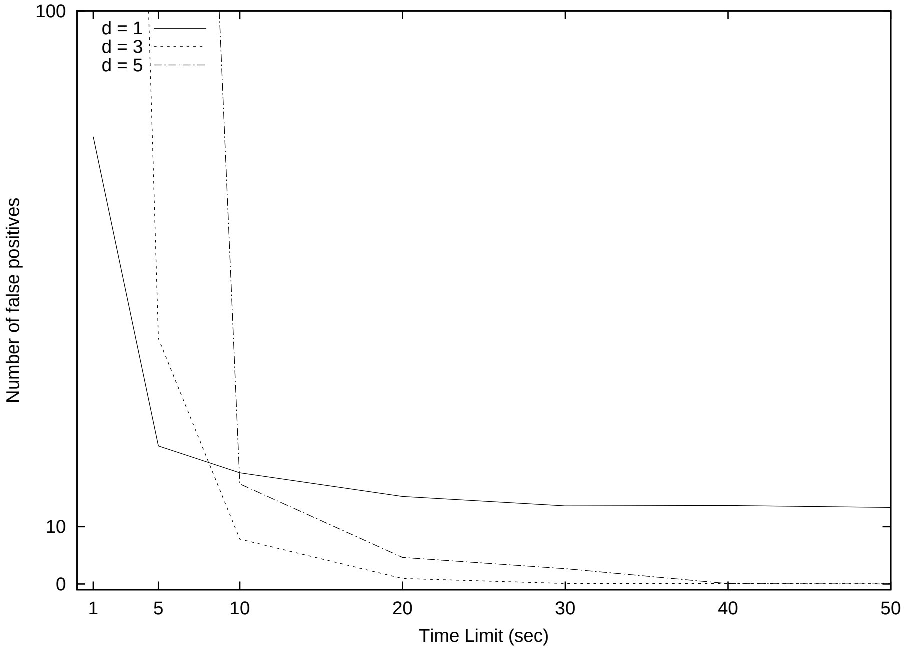


Figure 3.12: Minimum number of false positive that can be achieved for different values of the time limit, where d is the maximum number of levels.

3.4.3 Effect of levels on the performance of the cascade Bloom filter

Figure 3.13 shows how the number of false positives changes as the number of levels increases. We observe that the number of false positives may increase first, but eventually decrease as the number of level increases. It is discussed in Example 4 and Figure 3.5. Figure 3.14 shows how the number of false positives changes for different problem sizes. We observe that regarding the number of false positives, cascade Bloom filter is more resilient to an increase in the problem size than the traditional Bloom filter. Figure 3.15 shows how the number of hash functions required to achieve a k false positives decrease as k increases. We observe that for large value of k, a cascade Bloom filter with fewer levels uses fewer hash functions. However, for smaller value of k, a cascade Bloom filter with more levels uses fewer hash functions.

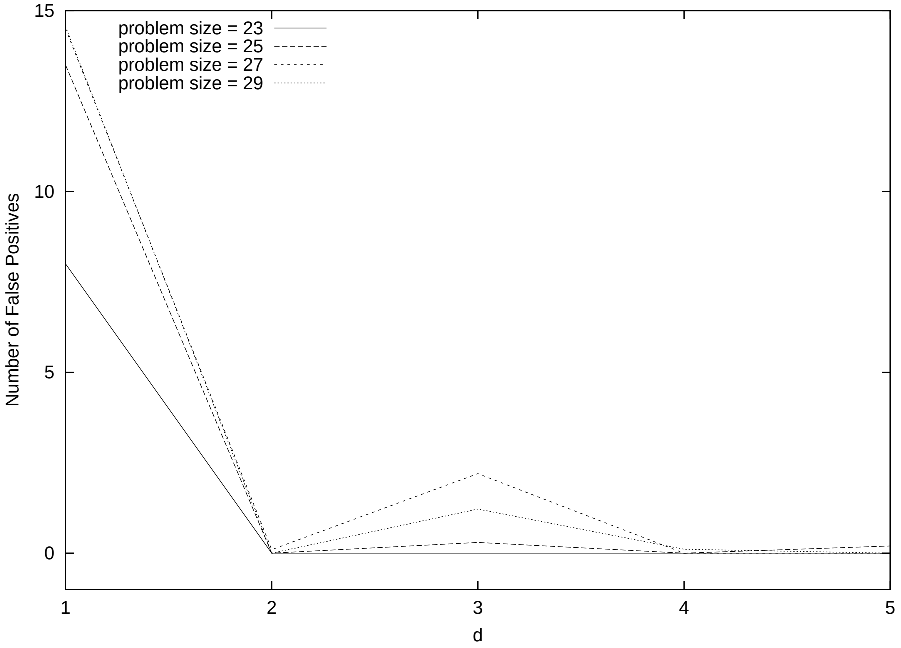


Figure 3.13: Minimum number of false positives that can be achieved for different values of depth.

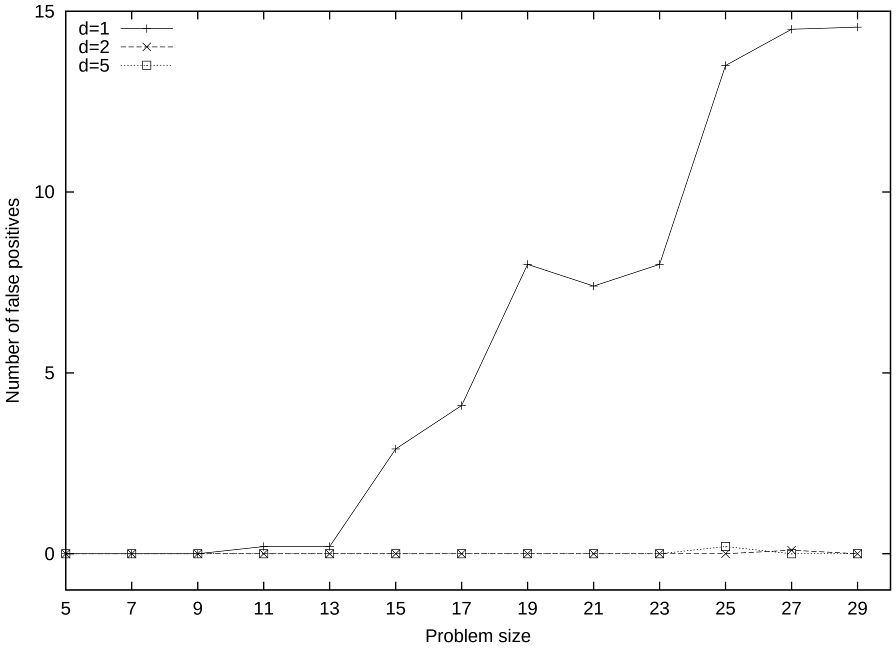


Figure 3.14: Minimum Number of false positives that can be achieved with a cascade Bloom filter for different problem size.

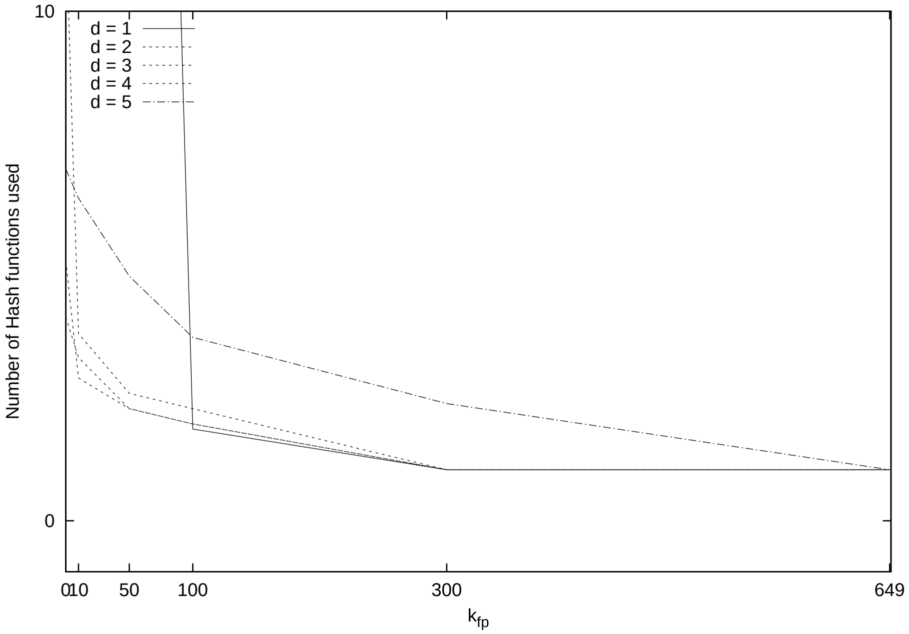


Figure 3.15: Minimum number of hash functions required for different values of k, the number of of false positives allowed. The maximum number of false positives is 650.

Chapter 4

Conclusions

In this dissertation, we focused on two problems in the context of Role-Based Access Control: the User Authorization Query (UAQ) Problem and the Cascade Bloom Filter (CBF) problem.

In Chapter 2, we formulated UAQ as a joint optimization problem of identifying optimal roles and extra permissions for an RBAC session. We studied the computational complexity of four subcases of UAQ. For each subcase, we identified the manner in which the input parameters contribute to computational complexity. We showed how hard it is to approximate each subcase. Our results show that approximation algorithms are not a promising approach to UAQ. We have identified several issues with prior work on this problem related to soundness, efficiency and limited support for the joint optimization. We have adopted a systematic approach based on the observations that the decision version of the general case remains in **NP**, and an effective approach to the decision version gives us an effective approach to the optimization version.

We have then investigated a standard, theoretically well-founded approach to addressing the decision version, reduction to CNF SAT. Our approach permits us to leverage existing SAT solvers, which are efficient for a large class of instances. We have implemented our approach, and provided an empirical assessment that validates our analytical insights. We observe that our approach is orders of magnitude faster than prior approaches. We have identified also that the benchmark that we and prior work on UAQ have adopted for empirical assessment is limited in that it appears to comprise easy instances only. We have systematically studied how to generate hard instances of UAQ, and the structural properties that such hard instances have.

In Chapter 3, we formulated the problem of finding an optimal Cascade Bloom

Filter (CBF) as a joint optimization problem of identifying the optimal number of false positives and number of hash functions used. Our formulation does not rely on the uniformity assumption for hash functions, an assumption that is customarily made in part work on the analysis of the Bloom filter data structure. We prove that CBF is **NP**-complete— a result that prevented us from finding an efficient algorithm for CBF. We have then proposed an approach for mitigating the intractability of CBF, that is an efficient reduction to CNF-SAT. We have implemented our approach, and made several observations based on our empirical results. Our approach results in cascade Bloom filters with significantly fewer false positives. We showed also the manner in which the number of levels affects the performance of a cascade Bloom filter.

4.1 Future Work

We have shown in Chapter 2 that the UAQ problem is in **P** if some input parameters are bounded. Some of the input parameters are indeed bounded for many of the UAQ instances that arise in practice. For example, it is reasonable to assume that the number of lower bound permissions in an instance of UAQ is small. Our proofs for the existence of efficient algorithms for the subcases of UAQ are by giving polynomial time algorithms. It will be interesting to improve the proposed algorithms for the subcases of UAQ that are shown to be in **P**. It is possible, in tandem with the evolution of a benchmark for UAQ, that such approaches are deemed to be efficient in practice. Also of interest is the identification and evolution of a benchmark for UAQ as the basis for empirical assessments.

The framework that we developed in Chapter 3, i.e., cascade Bloom filter, is of interest even outside the context of access enforcement. Our observation has shown that a cascade Bloom filter with few levels outperforms a conventional Bloom filter. It will be interesting to prove this result formally. It will be interesting to study the effect of the number of levels on the performance of a cascade Bloom filter as well.

The efficiency of our approach for mitigating the intractability of the CBF relies on the performance of SAT solvers. The fact that SAT solvers have been engineered using many intelligent methods may explain why they are often effective for many combinatorial problems, even though they seem to be oblivious of the underlying structure of the problems. Investigating other approaches that make use of the properties of the CBF problem, and comparison with our approach is an interesting aspect to explore for mitigating the intractability of CBF.

In the formulation of the CBF problem, we primarily focused on optimizing the number of false positives and the number of hash functions. We considered the total memory of the cascade Bloom filter as an input parameter. A natural extension to our formulation is to add the total memory needed as an optimization objective.

Bibliography

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706734, Oct. 1993.

[2] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications (JSAC)*, 23(10), October 2005.

[3] A. Armando, S. Ranise, F. Turkmen, and B. Crispo. Efficient run-time solving of RBAC user authorization queries: Pushing the envelope. In *Proceedings of the ACM Conference on Data and Applications Security and Privacy (CODASPY'12)*. ACM, Feb. 2012.

[4] S. Arora and B. Barak. Computational Complexity: A Modern Approach. *Cambridge University Press*, 2009.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 5(8), 1998.

[6] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM,* 13(7):422426, 1970.

[7] R.W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proceedings., 1990 IEEE Computer Society Symposium on Research in Security and Privacy.* pages 116–132, 1990.

[8] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access- control systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 8195, 2005.

[9] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control and Computing,* pages 636646. ACM Press, 2002.

[10] K. Borders, X. Zhao, and A. Prakash. Cpol: High-performance policy evaluation. In *Proceedings of the 12th ACM Coference on Computer and Communications Security (SACMAT05)*, pages 147157, 2005.

[11] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[12] L. Chen and J. Crampton. Set covering problems in role-based access control. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 689–704, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] B. Chazelle, J. Kilian, R. Rubinfield, and A. Tal. The bloomier filter: An efficient data structure for static support lookup tables. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms (SODA),* pages 3039, 2004.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3 edition, Sept. 2009.

[15] S. Cohen and Y. Matias. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD),* pages 241252, 2003.

[16] Clark, David D., and David R. Wilson. A comparison of commercial and military computer security policies. *IEEE symposium on security and privacy.* Vol. 184. 1987.

[17] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[18] S. Du and J. B. D. Joshi. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, SACMAT '06, pages 228–236, New York, NY, USA, 2006. ACM.

[19] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache:A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking,* 8(3):281293, 2000.

[20] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. dR-BAC: Distributed Role-Based Access Control for Dynamic Coalition Environments. In *Proceedings of the International Conference on Distributed Computing Systems,* July 2002.

[21] D. F. Ferraiolo, R. S. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, Aug. 2001.

[22] G. S. Graham and P. J. Denning. Protection principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417-429. AFIPS Press, May 16-18 1972.

[23] V. Gogate, and R. Dechter. A Complete Anytime Algorithm for Treewidth. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*,UAI '04, 2004.

[24] M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., 1979.

[25] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*,19(8):461471, Aug. 1976.

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

[27] J. D. B. Joshi, E. Bertino, and A. Ghafoor. Temporal hierarchies and inheritance semantics for GTRBAC. In *Proceedings of the seventh ACM symposium on Access control models and technologies (SACMAT '02).* ACM, New York, NY, USA

[28] G. Karjoth. Access control with IBM Tivoli Access Manager. *ACM Transactions on Information and System Security,* 6(2):232257, May 2003.

[29] A. Koster, H. Bodlaender, and S. Van Hoesel. Treewidth: Computational experiments. *Technical report*, Universiteit Utrecht, 2001.

[30] M. Komlenovic, M. Tripunitara, and T. Zitouni. An empirical assessment of approaches to distributed enforcement in role-based access control (RBAC). In *Proceedings of the first ACM conference on Data and application security and privacy*, CODASPY '11, pages 121–132, New York, NY, USA, 2011. ACM.

[31] Ninghui Li, Ji-Won Byun, and Elisa Bertino. A Critique of the ANSI Standard on Role-Based Access Control. *IEEE Security & Privacy,* 5(6):41–49, 2007.

[32] N. Li, M. V. Tripunitara, and Z. Bizri. On mutually exclusive roles and separation-of-duty. *ACM Trans. Inf. Syst. Secur.*, 10, May 2007.

[33] Y. Liu, C. Wang, M. Gorbovitski, T. Rothamel, Y. Cheng, Y. Zhao, and J. Zhang. Core role-based access control: efficient implementations by transformations. IN *Proceedings of the ACM SIGPLAN symposium on Partial Evaluation and semantics-based Program Manipulation,* pp. 112120, May 2006.

[34] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking,* 10(5):604612, 2002.

[35] MiniSat. http://minisat.se/, Jan 2013.

[36] Moffett, Jonathan D. and Lupu, Emil C. The uses of role hierarchies in access control. In *Proceedings of the fourth ACM workshop on Role-based access control, RBAC '99*, pages 153–160, New York, NY, USA, 1999, ACM.

[37] N. Mousavi, and M.V. Tripunitara. Mitigating the intractability of the user authorization query problem in role-based access control (RBAC). In *Proceedings of the international conference on Network and System Security, NSS '12*, pages 516-529, 2012, Springer-Verlag.

[38] N. Mousavi, and M.V. Tripunitara. CNF-SAT and Fixed-Parameter Polynomial-Time Implementations for UAQ (April 2012), *https://ece.uwaterloo.ca/tripunit/uaq/*

[39] N. Mousavi, and M.V. Tripunitara. CNF-SAT Implementations for CBF (Jan 2013), *https://ece.uwaterloo.ca/tripunit/CBF/*

[40] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, Aug. 2009.

[41] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In *Proceedings of Principles and Practice of Constraint Programming*, CP 2004, 10th International Conference, 2004.

[42] A.C. O'Connor and R.J. Loomis. Economic Analysis of Role-Based Access Control. *Research Triangle Institute*. Dec, 2010.

[43] K. Pearson. Notes on regression and inheritance in the case of two parents. In *Proceedings of the Royal Society of London*, 58 : 240-242, 1985.

[44] A. Pagh, R. Pagh, and S. S. Rao. An optimal bloom filter replacement. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* pages 823829, 2005.

[45] M. V. Ramakrishna. Practical performance of bloom filters and parallel freetext searching. *Communications of the ACM,* 32(10):12371239, Oct. 1989.

[46] S. J. Russell and P. Norvig, Artificial Intelligence - A Modern Approach (3rd ed.), *Pearson Education*,2010.

[47] Ravi S. Sandhu. Role Activation Hierarchies. *ACM Workshop on Role-Based Access Control.* RBAC '98, pages 33-40, 1998, ACM.

[48] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *P. van Beek, editor, CP*, volume 3709 of LNCS, pages 827831. Springer, 2005.

[49] M. Sipser. Introduction to the Theory of Computation. *Thomson Course Technology,* 2006.

[50] K. Shanmugasundaram, H. Bronnimann, and N. Memon. Payload attribution via hierarchical bloom filters. *In Proceedings of the 11th ACM conference on Computer and communications security (CCS04),* pages 3141. ACM Press, 2004.

[51] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[52] B. Selman, D. Mitchell, and H. Levesque. Generating Hard Satisfiability Problems. *Artificial Intelligence*, 1996.

[53] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, 63(9):1278-1308, 1975.

[54] S. D. Stoller, P. Yang, C R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *Proceedings of the 14th ACM conference on Computer and communications security (CCS '07).* ACM, New York, NY, USA, 2007.

[55] T.C. Ting. A user-role based data security approach. In Database Security: Status and Prospects, pages 187-208, Annapolis, Maryland, USA, 1988.

[56] M. Tripunitara and B. Carbunar. Efficient Access Enforcement in Distributed Role-Based Access Control (RBAC) Deployments. In *Proceedings of the 14th ACM Symposium on Access Control, Models and Technologies (SACMAT'09)*. ACM, June, 2009.

[57] V. Vazirani. Approximation Algorithms. *Springer*, 2001.

[58] Q. Wei, J. Crampton, K. Beznosov, and M. Ripeanu, Authorization Recycling in RBAC Systems, In *Proceedings of the 13th ACM Symposium on Access Control, Models and Technologies (SACMAT08),* pp. 6372, 2008.

[59] G. T. Wickramaarachchi, W. H. Qardaji, and N. Li. An efficient framework for user authorization queries in RBAC systems. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 23–32, New York, NY, USA, 2009. ACM.

[60] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. 2005. A simple model to generate hard satisfiable instances. In *Proceedings of the 19th international joint conference on Artificial intelligence (IJCAI'05).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 337-342.

[61] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *J. Artif. Int. Res.* 32, 1 (June 2008), 565-606.

[62] K. Xu and W. Li. 2006. Many hard examples in exact phase transitions. *Theor. Comput. Sci.* 355, 3 (April 2006), 291-302.

[63] zChaff. http://www.princeton.edu/~chaff/zchaff.html, Apr 2012.

[64] Y. Zhang and J. B. D. Joshi. Uaq: a framework for user authorization query processing in RBAC extended with hybrid hierarchy and constraints. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 83–92, New York, NY, USA, 2008. ACM.