AUTHORIZATION QUERY PROBLEM

constraint, genetic algorithm

The proposed algorithm, which is especially fit for the case that the computational complexity of different subcases of WUAQ, and show that many instances in each subcase are intractable. In particular, inspired by the idea of the genetic algorithm, we propose an algorithm to approximate solve an intractable subcase of the WUAQ problem. An important observation is that this algorithm can be efficiently modified to handle the other subcases of the WUAQ problem. The experimental results show the advantage of the proposed algorithm, which is especially fit for the case that the computational overhead is even more important than the accuracy in a large-scale RBAC system.

keywords: role-based access control, user authorization query, weight, constraint, genetic algorithm

1. Introduction

The user authorization query (UAQ) problem for role based access control (RBAC), introduced by Zhang et al. [1], is to determine whether there exists an optimum set of roles that should be activated to provide a particular set of permissions requested by a user, which has been the subject of considerable research in recent years, and is widely accepted as a key issue related to efficiently handling users’ access requests [2], [3].

The concept of UAQ was originally proposed by Du et al. [4] with the name of inter-domain role mapping (IDRM) problem. However, the definition of IDRM is basic and incomplete, as an unique minimal set of roles that exactly covers the requested permissions may not exist. Chen et al. [4], [5] redefined the IDRM problem by ensuring two aspects of constraints: (i) all of the requested permissions should be available; (ii) the principle of least privilege should be observed. Zhang et al. [1] generalized IDRM as UAQ, and divided it into three subcases: the exactly match, the availability match, and the least privilege match. Wickramaarachchi et al. [6] extended the UAQ problem to a more general definition, in which the number of permissions granted is restricted to both a lower bound and an upper bound. One optimization objective is to prefer a set of roles that has permissions as close to the lower bound as possible, and the other prefers a set of permitted permissions as close to the upper bound as possible. The above specifications for UAQ have considered only the optimization objective for the number of permissions, while the optimization objective for the number of roles is also equally important, which has been largely ignored. Mousavi et al. [7] gave a formulation of UAQ as a joint optimization objective for both of the number of roles and the number of permissions. However, they only considered the optimization objective for extra permissions, while the missing permissions is omitted. Hence, Lu et al. [2] proposed a comprehensive definition of the UAQ problem by considering the optimization objectives for the number of permissions as well as the number of roles, where the UAQ problem includes two components: the Core-UAQ problem and the Constrained-UAQ problem by introducing the irreducibility, role-cardinality, and permission-cardinality constraints to Core-UAQ. A typical RBAC system may optionally include any Core-UAQ or Constrained-UAQ.

A key limitation of existing works for the specification of UAQ rely on the assumption that there is no difference between permissions, i.e., they did not consider the different nature and importance of each permission, or treated the permissions evenly. However, this is not always true in practice. For example, the permission that writes to the student achievement may be more important than the permission reads to the student achievement. Unfortunately, the existing works of UAQ simply ignores this difference. Here, we present an example to motivate the new features of the notation about the weight of permission/role to optimize the UAQ problem. Let us assume that the requested permission set is \( \{p_2, p_3, p_5\} \), permissions \( p_1 \) and \( p_2 \) belong to \( r_1 \), permissions \( p_3 \) and \( p_4 \) belong to \( r_2 \), permissions \( p_2, p_4 \) and \( p_5 \) belong to \( r_3 \). It is obvious that both \( \{r_1, r_2\} \) and \( \{r_2, r_3\} \) are the solutions for the available match of UAQ, since each of them has only one extra permissions \( p_1 \) and \( p_5 \), respectively. However, it may not make any sense for choosing the solution \( \{r_1, r_2\} \), if the permission \( p_1 \) is more critical than the permission \( p_5 \), such as \( p_1 \) denotes as the permission that writes to the student achievement, while \( p_5 \) represents as the permission that reads to the student achievement. In such
case, the solution \( \{r_2, r_3\} \) is a better choice if we consider
the weight of the extra permissions. Therefore, the weight
of permission/role is a value attached to a permission/role
representing its importance \([8],[9]\), which is very important
rather than trivial extension that should be introduced to the
specification of the UAQ problem, and we name it as the
weighted user authorization query (WUAQ) problem.

As discussed in \([2]\), we have studied the computational
complexity of the UAQ, and shown that many instances
in each subcase with additional constraints are intractable.
Since the UAQ problem is only a subcase of WUAQ as it as-
sumes that each permission has the same impact of weight,
it is obvious that the WUAQ problem is also intractable.
In this work, we continue our research into the computa-
tional complexity of the UAQ problem by considering the
weights of permissions and roles. A more important issue is
how to design efficient algorithms to resolve the intractable
instances of the WUAQ problem. In previous, several re-
searchers have proposed exhaustive algorithms to compute
a solution for the UAQ problem. Zhang \textit{et al.} \([1]\) proposed a
two-step algorithm for the UAQ problem. However, this
algorithm may falsely reject some legal success requests.
Wickramaarachchi \textit{et al.} \([6]\) introduced two approaches to
address the UAQ problem: (i) developing algorithm that us-
ing the backtracking-based search techniques; (ii) reducing
the problem to the MAXSAT problem. As pointed out by
Armando \textit{et al.} \([11]\), the first approach is exponential-time
in design, thus it does not seem to scale to larger RBAC poli-
cies. The second approach is unsound, inefficient and offers
only limited support for the joint optimization of the number
of roles and extra permissions, as pointed out by Mousavi
\textit{et al.} \([7]\). Lu \textit{et al.} \([2]\) proposed an approach to solve the
intractable cases of the UAQ problem by employing static
pruning, preprocessing and depth-first search based algo-
rithm to reduce its running time. However, the above ex-
haustive algorithms may not fit with the WUAQ problem
due to two major reasons: (i) the previous algorithm fo-
cus on the numbers of roles and permissions, rather than the
weights of them; (ii) under the premise of ensuring the accu-
ricy, the computational overhead may be even more im-
portant \([8]\). Thus, an efficient approximate approach is urgently
needed for WUAQ.

Briefly, the main contributions of this paper can be
summarized as follows.

- We propose a comprehensive definition of the WUAQ
  problem, by introducing the weight of permission/role
to UAQ, and consider the role-weighted-cardinality and
  permission-weighted-cardinality constraints.
- We study the computational complexity of different
  subcases of WUAQ problems, and show that many in-
  stances in each subcase are intractable.
- We propose an algorithm to approximately solve the
  intractable cases of the WUAQ problem, an im-
  portant observation is that this algorithm can be efficiently
  modified to handle the other subcases of the WUAQ
  problems.

The rest of this paper is organized as follows. In Sect. 2,
we give the formal definition of the WUAQ problem, and
study the computational complexity of its variants subcases.
Section 3 presents an approximate algorithm to solve the in-
tractable cases of the WUAQ problem. In Sect. 4, we imple-
ment the proposed algorithm, and make a comparison with
other work. We conclude this paper in Sect. 5.

2. The Weighted User Authorization Query Problem

Ma \textit{et al.} have introduced a formal definition of the weight
of permission/role, and given methods of calculating them.
For more details, please refer to \([8],[9]\). In this section, we
assume that the weights are given by the system, and in-
troduce two types of constraints: role-weighted-cardinality
and permission-weighted-cardinality to the WUAQ prob-
lem. Continuing the specification style of UAQ in \([2]\), the
WUAQ problem is defined as follows.

\textbf{Definition 1. (The WUAQ Problem)} Given a set \( R \) of all
roles, a set \( P \) of all permissions, and a set \( P_{req} \) of permis-
sions requested by a user \( u \), the WUAQ problem is to iden-
tify a role set \( R' = R \) that can be activated by \( u \) while may
optionally satisfy the following constraints:

- \textbf{Role-weighted-cardinality:} A role-weighted-cardinal-
  ity constraint is denoted as \( rwc(R, O_r) \), where \( R \subseteq R \),
  \( O_r \in \{k, \infty^+, \infty^-\} \), and \( k \) is a positive number. We say
  that \( rwc(R, O_r) \) is satisfied if and only if the following
  conditions hold:
    - \( W(R) \leq k \), if \( O_r = k \), where \( W(R) \) denotes the
      weight of the role set \( R \);
    - \( W(R) \) is maximized if \( O_r = \infty^+ \);
    - \( W(R) \) is minimized if \( O_r = \infty^- \);

- \textbf{Permission-weighted-cardinality:} A permission-wei-
  ghted-cardinality constraint is denoted as \( pwc(R, O_p) \),
  where \( R \subseteq R \), \( O_p \in \{t^+, t^-, t^0, 0^+, 0, 0^\} \), and \( t \) is a
  positive number. We say that \( R \) satisfies \( pwc(R, O_p) \) if
  and only if the following conditions hold:
    - \( Perm(R) \supseteq P_{req} \) and \( W(Perm(R) - P_{req}) \leq t \) if
      \( O_p = t^+ \), where \( Perm(R) \) maps a role set \( R \) onto a
      set of all available permissions, \( W(P) \) denotes the
      weight of the permission set \( P \);
    - \( Perm(R) \subseteq P_{req} \) and \( W(P_{req} - Perm(R)) \leq t \) if
      \( O_p = t^- \);
    - \( W(P_{req} \cup Perm(R) - P_{req} \cap Perm(R)) \leq t \) if \( O_p = t^0 \);
    - \( Perm(R) \supseteq P_{req} \) and for all \( R' \subseteq R \), \( Perm(R') \supseteq
      P_{req} \) that \( W(Perm(R') - P_{req}) \geq W(Perm(R) - P_{req}) \)
      if \( O_p = 0^+ \);
    - \( Perm(R) \subseteq P_{req} \) and for all \( R' \subseteq R \), \( Perm(R') \subseteq
      P_{req} \) that \( W(P_{req} - Perm(R')) \geq W(P_{req} - Perm(R)) \)
      if \( O_p = 0^- \);

  A permission-weighted-cardinality constraint specifies
  an optimization requirement on the weight of the permis-
ions that can be acquired by the requesting user. The parameter $t^*$ and $t^+$ specify the threshold values that the weight of extra and missing permissions that the system can be able to tolerate, respectively, while $t^-$ specifies the threshold values that the weight of both extra and missing permissions. Similarly, the parameters $0^*$, $0^-$ and $0^+$ prefer to minimize the weight of extra permissions, missing permissions and the union of them. For instance, when the system can be able to tolerate missing permissions that the total weight of them is 0.5, then $t$ can be set to 0.5 for the case $O_p = t^-$. When the total weight of extra permissions that more than 1.0 may bring the intolerable risk to the system then $t$ can be set to 1.0 for the case $O_p = t^+$. Similarly, if the system want to minimize the total weight of extra and missing permissions, then the case $O_p = 0^+$ can satisfy such requirement. In another aspect, a role-weighted-cardinality constraint specifies an optimization requirement on the weight of activated roles. The parameter $k$ specifies the threshold values that the weight of selected roles by the system, while the parameters $\infty^+$ and $\infty^-$ prefer to maximize and minimize the weight of selected roles, respectively. It may be useful when security constraints make some selected roles to be unavailable, while minimizing the weight of selected roles activated in a user’s session may allow an administrator to more efficiently manage the system. For example, when the system can be able to tolerate a set of roles whose total weight is no more than 1.5 that activated in a session, then $k$ can be set to 1.5 for the case $O_t = k$, while the case $O_t = \infty^-$ can satisfy the requirement that the system want to maximize the selected roles.

To specify a subcase of the WUAQ problem, we write it followed by the list of constraints within a pair of braces. For instance, $WUAQ(rwc : k + pwc : t^*)$ denotes the subcase that finds the role set $R$ to satisfy not only $rwc(R, k)$, but also $pwc(R, t^*)$. Note that all the permission-weighted-cardinality constraints can be combined with the role-weighted-cardinality constraints, there may be conflicts between these two types of constraints, we hence simply assign higher priority to the permission-weighted-cardinality constraints than the role-weighted-cardinality to solve such conflicts.

The UAQ problem essentially is a subcase of WUAQ, since it just assumes that each permission and role has the same impact of weight. In this case, determining the computational complexity of the WUAQ problem is a challenge work. In the following, Theorem 1 derives the complexity of different subcases of WUAQ problems.

**Theorem 1.** The computational complexity of different subcases of WUAQ problems are as shown in Table 1.

**Proof.** The proof for Theorem 1 consists of three parts:

(i) We show that $WUAQ(rwc : k)$ is NP-complete by proving that it is both NP-hard and in NP. Firstly, it can see that $WUAQ(rwc : k)$ is in NP, this is because if one correctly guess a subset $\mathcal{R}$ of $\mathcal{R}$ as a solution for $WUAQ(rwc : k)$, verifying whether $W(\mathcal{R}) \leq k$, which can be done in polynomial time. Secondly, we prove that $WUAQ(rwc : k)$ is NP-hard by reducing the NP-complete subset sum problem [12] to it. The subset sum problem can be depicted as follows: Given a set $A = \{a_i : 1 \leq i \leq n\}$ of positive integers and a positive integer $M$, the goal is to determine whether there exists a subset of $A$ sum of whose elements equal to a given integer $M$. The reduction is as follows: Given $A$ and $M$, construct $WUAQ(rwc : k)$ as follows: Let each element in $A$ map to a role weight $W(r) \mid r \in R$ in the problem, and let $M$ equal to $k$. Next, we construct an RBAC state as follows: For each corresponding role in the element of $A$, create a single permission $p$ to which the role is covered, and assign the value of $W(r)$ as its weight. The resulting solution $R$ can be found with respect to $W(R) \leq k$, if and only if, there exists a subset of $A$ sum of whose elements equal to $M$.

(ii) We study the complexities of WUAQ problems for the subcases $O_p = t^*$. In fact, the UAQ problem can be regarded as a special case of the WUAQ problem. For example, $UAQ(available + pc : t^*)$ in [2] is equal to $WUAQ(pwc : t^*)$ when we assume that each permission has the same weight. In this case, the complexity of $WUAQ(pwc : t^*)$ is at least as hard as $UAQ(available + pc : t^*)$, because any solution for the former problem is also a solution for the later one. It obviously that $WUAQ(pwc : t^*)$ is in NP, since verifying whether a given role set is a solution for $WUAQ(pwc : t^*)$ can be done in polynomial time. Similarly, the remainder subcases for $O_p = t^+$, and the WUAQ problems for $O_p = t^-$, $O_p = 0^+$ and $O_p = 0^-$ can be derived.

(iii) Since the complexities of WUAQ problems for the subcases $O_p = t^+$ are at least as hard as both of the cases $O_p = t^+$ and $O_p = 0^+$, as any solution for one of the latter two is also a solution for the former one. Similarly, we can derive the complexity of WUAQ problems for the cases $O_p = 0^+$.

Other results in Table 1 can be implied from the proved cases. \qed

<table>
<thead>
<tr>
<th>WUAQ</th>
<th>Role-weighted-cardinality</th>
<th>None</th>
<th>$O_t = \frac{\infty^*}{k}$</th>
<th>$O_t = \infty^+$</th>
<th>$O_t = \infty^-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>P</td>
<td>$NP$-complete</td>
<td>P</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>$O_p = t^-$</td>
<td>$NP$-complete</td>
<td>$NP$</td>
<td>$NP$-hard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_p = t^+$</td>
<td>$NP$-hard</td>
<td>P</td>
<td>$NP$-hard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_p = 0^-$</td>
<td>$NP$-hard</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O_p = 0^+$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Computational complexities of different subcases of WUAQ problems.
An Efficient Approximate Algorithm for the Weighted User Authorization Query problem

The fact that WUAQ is intractable, as shown in Theorem 1, means that there exist difficult problem instances that take exponential time in the worst case. However, many instances that will be encountered in practice may still be efficiently solvable. For example, WUAQ\((pwc : 0^*)\) is NP-hard as shown in Table 1. In order to solve such intractable subcase, we first propose a binary evolutionary (BE) algorithm to approximate solve it, which is inspired by the idea of the genetic algorithm [13], [14]. Next, we show that our algorithm can be efficiently modified to handle the other subcases of the WUAQ problems.

3.1 The Binary Evolutionary Algorithm

In the BE algorithm, we first generate a population of role sets at random, and evaluate their fitness. Next, we generate a new population based on mutation and crossover with a probability distribution. Finally, the algorithm will stop when iteration times are over the threshold value, and output a solution that approximate solve WUAQ\((pwc : 0^*)\). The steps of the BE algorithm are summarized as follows, and the pseudo code is given in Algorithm 1. This algorithm has a time complexity of \(O(lmn)\), where \(l, m, n\) denote the number of iteration times, the size of population, the number of all available roles, respectively. The main notations used in this paper are shown in Table 2.

(i) **Preprocess**: Remove roles in \(R\) that do not have at least one permission in \(P_{req}\).

(ii) **Coding**: Convert each solution of the WUAQ problem to a corresponding chromosome, that use \(n\)-bit string \(y_1, \cdots, y_n\) where each bit \(y_i\) is either 1 or 0 to represent whether the \(i\)th role is selected in the solution.

(iii) **Start**: Select a population of \(m\) points \(x_1, \cdots, x_m\) to represent the roles set at random, and set \(l = 0\).

(iv) **Compute fitness**: Compute the fitness of the role set using the evaluation function.

(v) **Replacement**: Sort the \(m\) points according to the value of their fitness from large to small, and replace the latter half part by the front half.

(vi) **Regeneration**: If all the points in the first half of the population have the same fitness, set \(l = l + 1\), save \(x_1\), and go to step (iii).

(vii) **Mutate**: For each point \(x_i\) that \(\frac{m}{2} < i \leq m\) in the population and for each bit in \(x_i\), with probability \(p_{muta}\), alter its value.

(viii) **Crossover**: For each \(y_j\) in the pair points \(x_i\) and \(x_{(i+1)}\) from the \(x_1, \cdots, x_m\), with probability \(p_{cross}\), exchange \(x_i.y_j\) with \(x_{(i+1)}.y_j\).

(ix) **Stop**: Set \(l = l + 1\), if \(l\) is equal to the threshold value, stop. Otherwise go to step (iv).

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>The BE Algorithm for WUAQ((pwc : 0^*))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>:</td>
<td>(R, P, P.R, P_{req}, W(p), p_{muta}, p_{cross}, E)</td>
</tr>
<tr>
<td><strong>Output</strong>:</td>
<td>(O(R))</td>
</tr>
<tr>
<td><strong>1:</strong></td>
<td>(\text{for each } r \in R \text{ do} )</td>
</tr>
<tr>
<td></td>
<td>(\text{if } Perm(r) \cap P_{req} = \emptyset \text{ then} )</td>
</tr>
<tr>
<td></td>
<td>(R \leftarrow R[r] )</td>
</tr>
<tr>
<td></td>
<td>(\text{end if} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end for} )</td>
</tr>
<tr>
<td><strong>2:</strong></td>
<td>(\text{Rand}(E[m]) / \ast \text{ generate a random population} /)</td>
</tr>
<tr>
<td></td>
<td>(\text{while } + l &lt; \text{threshold do} )</td>
</tr>
<tr>
<td></td>
<td>(\text{for each } i \in [0, n] \text{ do} )</td>
</tr>
<tr>
<td></td>
<td>(E[i].fi t \leftarrow p_{size} - w_{ep} - 100w_{mp} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end for} )</td>
</tr>
<tr>
<td></td>
<td>(\text{Sort}(E[m]) / \ast \text{From big to small sort} /)</td>
</tr>
<tr>
<td></td>
<td>(\text{if } O.f i t &lt; E[0].fi t \text{ then} )</td>
</tr>
<tr>
<td></td>
<td>(O \leftarrow E[0] )</td>
</tr>
<tr>
<td></td>
<td>(\text{end if} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end if} )</td>
</tr>
<tr>
<td></td>
<td>(\text{for each } i &gt; \frac{n}{2} \text{ do} )</td>
</tr>
<tr>
<td></td>
<td>(\text{for each } j \in [0, n] \text{ do} )</td>
</tr>
<tr>
<td></td>
<td>(E[i] \leftarrow E[i] )</td>
</tr>
<tr>
<td></td>
<td>(\text{if rand()} &lt; p_{muta} \text{ then} )</td>
</tr>
<tr>
<td></td>
<td>(E[i].R[j] \leftarrow E[i].R[j] )</td>
</tr>
<tr>
<td></td>
<td>(\text{end if} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end for} )</td>
</tr>
<tr>
<td></td>
<td>(\text{for each } i %2 == 0 \text{ do} )</td>
</tr>
<tr>
<td></td>
<td>(\text{for each } j \in [0, n] \text{ do} )</td>
</tr>
<tr>
<td></td>
<td>(E[i].R[j] \leftarrow E[i].R[j] )</td>
</tr>
<tr>
<td></td>
<td>(\text{if rand()} &lt; p_{cross} \text{ then} )</td>
</tr>
<tr>
<td></td>
<td>(E[i].R[j] \leftarrow E[i+1].R[j] )</td>
</tr>
<tr>
<td></td>
<td>(\text{end if} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end for} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end for} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end for} )</td>
</tr>
<tr>
<td></td>
<td>(\text{end while} )</td>
</tr>
</tbody>
</table>

3.2 Handling the Other Subcase of the WUAQ Problems

The algorithm described in the previous subsection address a subcase WUAQ\((pwc : 0^*)\). An important observation is that this algorithm can handle the other subcases of the
WUAQ problems by efficiently modifying the evaluation function of fitness. The evaluation function of fitness used in the BE algorithm is defined as follows.

**Definition 2.** The evaluation function of fitness is defined as: $fit = \alpha \times p_{size} + \beta \times r_{size} + \gamma \times w_{ep} + \delta \times w_{mp} + \varepsilon \times w_{sr},$ where $\alpha, \beta, \gamma, \delta$ and $\varepsilon$ are parameters used to adjust the relative importance about $p_{size}, r_{size}, w_{ep}, w_{mp}$ and $w_{sr}$.

Table 3 summarizes the differences among the components used for the different subcases of the WUAQ problem with role weighted-cardinality and permission weighted-cardinality constraints. For example, in order to deal with the subcase $WUAQ(pwc : 0^+)$, the evaluation function of fitness is defined as $fit = p_{size} - w_{ep} - 100 w_{mp},$ where we set $\alpha = 1, \beta = 0, \gamma = -1, \delta = -100$ and $\varepsilon = 0$. In particularly, $100 \times w_{mp}$ can be regarded as a strict penalty for cases where the chosen roles do not cover all the requested permissions, while $1 \times w_{ep}$ can be regarded as a less penalty for choosing extra permissions. This algorithm will search for the least weight of extra permissions, which is a solution for $WUAQ(pwc : 0^+)$.

**4. Experimental Results**

In order to show the advantage of the proposed BE algorithm, we have implemented it and performed several experiments using randomly generated instances. We make a comparison of our BE algorithm with the Depth-First Search (DFS) algorithm in [2] for solving the subcase $WUAQ(pwc : 0^+)$ for two major reasons: Firstly, the comparison of the DFS with the Backtracking-Based Search (BBS) algorithm proposed in [6] has shown the effectiveness of the DFS algorithm. Secondly, both BE and DFS algorithm can be efficiently modified to handle the other subcases of the WUAQ/UAQ problems. The implementation of both BE and DFS algorithms were written in C. All the experiments have been carried out on a standard desktop PC with an Intel(R) Core(TM) i7-4790 CPU running at 3.60GHz, and with DDR3 16GB 1600MHz memory, running Microsoft Windows 7 Ultimate Editions. For each instance, 10 randomly generated test cases are run, the averages of the test results are used to generate the graphs.

**4.1 Effectiveness of Mutation and Crossover**

In order to verify the accuracy of the solutions obtained. We first implemented the DFS algorithm to generate an optimal solution $\mathcal{R}_{DFS}$ for reasonable size problems. We next ran the BE algorithm to get a role set $\mathcal{R}_{BE}$, and computed the accuracy of the BE algorithm, which was defined as follow.

**Definition 3.** The accuracy of the BE algorithm is defined as: $\left(1 - \frac{W(\text{Perm(\mathcal{R}_{DFS})}) - W(\text{Perm(\mathcal{R}_{BE})})}{W(\text{Perm(\mathcal{R}_{DFS})})}\right) \times 100\%.$

Figure 1 shows the average CPU times and accuracy under different probability of crossover and mutation for the test case: $P_{req} = 30$, $R = 60$, $P = 600$, the size of population $m = 120$, and the number of iteration $l = 100$. The $x$-axis denotes the probability of mutation which we fix its value as $\frac{1}{r_{cross}}, \frac{3}{r_{cross}}$, and $\frac{4}{r_{cross}}$ respectively. It can be clearly seen from Fig. 1 (a) that the average CPU times is least when we choose the parameter $p_{muta} = \frac{2}{r_{cross}}$, for the fixed $p_{cross}$, and the average CPU times increases with the maximal $p_{cross}$ for
Fig. 2  Running time and accuracy for Binary Evolutionary (BE) algorithm and Depth First Search (DFS) algorithm.
the fixed \( p_{\text{muta}} \). This is reasonable because the BE algorithm depends on the generated data and the size of the role set \( R \), hence the less \( P_{\text{cross}} \) will save the CPU times. But the impact of \( p_{\text{muta}} \) for CPU times is less. As shown in Fig. 1 (b), the average accuracy increases with the maximal \( P_{\text{cross}} \) for the fixed \( p_{\text{muta}} \). This is because it is easy to find an optimal solution if we assign a big value for \( P_{\text{cross}} \). However, we can’t get the optimal solution when the value of \( P_{\text{muta}} \) is either maximum or minimum. Together with the observation, we choose the parameters \( p_{\text{muta}} = \frac{1}{r_{\text{max}}} \) and \( P_{\text{cross}} = 0.6 \) for the remainder experiments.

4.2 Comparison of the BE Algorithm with the DFS Algorithm

Figure 2 shows the results of running the experiments for the four test case (1) \( P_{\text{req}} : R : P = 1 : 1 : 2 \); (2) \( P_{\text{req}} : R : P = 1 : 1 : 10 \); (3) \( P_{\text{req}} : R : P = 1 : 2 : 4 \); (4) \( P_{\text{req}} : R : P = 1 : 2 : 20 \). In particularly, we generate four different instances: \( \text{BE} \ 50-60 \), \( \text{BE} \ 50-280 \), \( \text{BE} \ 180-60 \), \( \text{BE} \ 180-280 \), the first parameter denotes the size of the population, and the second parameter denotes the number of iteration in this algorithm. The runtime and accuracy of these two algorithms depend on the total number of the requested permissions \( P_{\text{req}} \), available roles \( R \) and available permissions \( P \).

In Figs. 2 (a), (c), (e) and (g), both of these two algorithms produce comparable results when the number of requested permissions is small. However, as the number of requested permissions increases, the overall CPU time taken increases exponentially, this makes the DFS algorithm impractical for implementation in dynamic systems. However, the BE algorithm with different instances take a few seconds, even for a larger number of roles, permissions and requested permissions. The reason is that the BE algorithm will stop when iteration times are over the threshold value. It is worth noting that the BE algorithm turns out to be more effective when both of the population \( m \) and iteration \( l \) are small. Such as \( \text{BE} \ 50-60 \) always has the least CPU times, and \( \text{BE} \ 180-280 \) always has the largest CPU times compare with the other two subcases of BE. However, the accuracy of BE is close to the DFS algorithm when we choose large number of \( m \) and \( l \). As the number of requested permissions increases, the accuracy of the BE algorithm decreases. For example, the accuracy of \( \text{BE} \ 50-60 \) is less than 75% for a bad instance, as shown in Fig. 2 (h).

Consequently, for the case that the accuracy is not very critical, we can make the accuracy of the BE algorithm in an acceptable extent by enlarging the value of population size and iteration. The BE algorithm is able to efficiency approximate solve the WUAQ problem even though the number of permissions in a larger scale RBAC.

5. Conclusion

In this paper, we introduced the concept of permission/role weight, and gave a formal definition for the WUAQ problem by considering two types of constraints: role-weighted-cardinality and permission-weighted-cardinality. In fact, WUAQ is a more comprehensive definition that includes the UAQ problem. Furthermore, we studied the computational complexity analysis of various subcases of the WUAQ, and showed that most instances of WUAQ problem are intractable. In particular, we proposed a BE algorithm to efficiently approximate solve an instance of WUAQ, and showed how it can be efficiently modified to handle the other subcases. The comparison between the BE algorithm and DFS algorithm shown the efficiency and accuracy of the proposed BE algorithm. This algorithm is especially fit for the case that the computational overhead is even more important than the accuracy.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant 61402418, 61503342, 61672468, 61602418. Social development project of Zhejiang provincial public technology research under Grant 2017C33054, MOE (Ministry of Education in China) Project of Humanity and Social science under Grant 12YJCZH142, Zhejiang Provincial Natural Science Foundation of China under Grant LY13F020017, LY15F020013, LY16F030002.

References

[10] L. Chen and J. Crampton, “Set cover problems in role-based access


Jianfeng Lu is an associate professor in the School of Mathematics-Physical and Information Engineering at Zhejiang Normal University. He received his B.S. degree in the School of Computer Science and Technology at Wuhan University of Science and Technology in 2005, and the Ph.D. degree in the School of Computer Science and Technology at Huazhong University of Science and Technology in 2010. In 2013, he was a visiting researcher with the University of Pittsburgh, Pittsburgh, USA. His currently research interests include access control and game theory.

Zheng Wang is currently a graduate student in the School of Mathematics-Physical and Information Engineering at Zhejiang Normal University. He is expected to graduate in June 2017. His research interests include access control and optimization algorithm.

Dewu Xu is an associate professor in the School of Economics and Management at Zhejiang Normal University. He received his M.S. degree in School of Information Science and Technology at East China Normal University in 2005. His research interests include cryptography and distributed system security.

Changbing Tang is an assistant professor in the School of Mathematics-Physical and Information Engineering at the Zhejiang Normal University. He received his B.S. and M.S. degrees in the mathematics and applied mathematics from Zhejiang Normal University, in 2004 and 2007, respectively, and the Ph.D. degree in the Department of Electronic Engineering at Fudan University in 2014. His research interests include game theory and its applications.

Jianmin Han is a professor in the School of Mathematics-Physical and Information Engineering at Zhejiang Normal University. He received his B.S. degree in the department of Computer Science and Technology at Daqing Petroleum institute in 1992, and the Ph.D. degree in the department of Computer Science and Technology at East China University of Science and Technology in 2009. His research interests include privacy preservation and game theory.