Efficient Graph Based Approach to Large Scale Role Engineering

Dana Zhang*, **Kotagiri Ramamohanarao***, **Rui Zhang***, **Steven Versteeg****

*The University of Melbourne, Australia.

**CA Inc. Melbourne, Australia.

E-mail: danaz@acm.org, rao@csse.unimelb.edu.au, rui@csse.unimelb.edu.au, steve.versteeg@ca.com

Abstract. Role engineering is the process of defining a set of roles that offer administrative benefit for Role Based Access Control (RBAC), which ensures data privacy. It is a business critical task that is required by enterprises wishing to migrate to RBAC. However, existing methods of role generation have not analysed what constitutes a beneficial role and as a result, often produce inadequate solutions in a time consuming manner. To address the urgent issue of identifying high quality RBAC structures in real enterprise environments, we present a cost based analysis of the problem for both flat and hierarchical RBAC structures. Specifically we propose two cost models to evaluate the administration cost of roles and provide a k-partite graph approach to role engineering. Existing role cost evaulations are approximations that overestimate the benefit of a role. Our method and cost models can provide exact role cost and show when existing role cost evaluations can be used as a lower bound to improve efficiency without effecting quality of results. In the first work to address role engineering using large scale real data sets, we propose *RoleAnnealing*, a fast solution space search algorithm with incremental computation and guided search space heuristics. Our experimental results on both real and synthetic data sets demonstrate that high quality RBAC configurations that maintain data privacy are identified efficiently by *RoleAnnealing*. Comparison with an existing approach shows *RoleAnnealing* is significantly faster and produces RBAC configurations with lower cost.

Keywords. Role Engineering, Role Based Access Control, Graph Optimization.

1 Introduction

Role based access control (RBAC) is an authorisation mechanism for protecting data privacy and enforcing security policies in enterprises that uses roles to manage user permissions [6, 13]. RBAC is the primary alternative in enterprise environments to traditional mandatory access control (MAC) and discretionary access control (DAC) methods due to the huge administrative benefits it offers [5]. These benefits include increased flexibility of authorisation management, robustness in privacy policy enforcement and decreased susceptibility to Trojan Horses.

*This work was done while Dana Zhang was a PhD student at The University of Melbourne.

In RBAC, data is protected by assigning permissions to users through roles. Since roles can be mapped to organisational functions, modification of an organisational function only requires modification of the roles that are effected. Role assignment and revocation to and from the user are also simplified under this model, allowing for easier management and enforcement data privacy. Due to these benefits, multinational software corporation and security software provider CA Technologies identifies RBAC as an integral part of any privacy enabling, identity and access management initiative [15].

Migration to RBAC requires *role engineering*, the definition of a role infrastructure that accurately reflects the internal functionalities of the working enterprise [2]. However, role engineering is very difficult and is the most costly component of the role life-cycle [7]. Existing approaches have used scenario based, use-case and process driven techniques to derive roles [4, 11, 12]. These processes require manual elicitation and as a result, are time consuming, error prone and costly. While manual role elicitation has been successful in some enterprises, most enterprises are uncertain about how to define roles and how to determine which roles should be deployed. Automated role engineering processes that can use existing permission assignments to assist with identification of potential roles are preferred. For example, customers of CA Technology's Identity Management software which provides an RBAC implementation for data privacy, often request automated role engineering assistance during the initial migration phase.

The problem of finding the optimum number of roles has been shown to be NP-complete and hard to approximate [3, 10, 14]. Subsequent studies have used heuristics and have focused on clustering techniques as well as graph based approaches [1, 3, 16, 17, 18]. Existing data mining approaches are capable of generating a large numbers of roles, often exceeding the number of users or permissions. Pruning or weighted selection is used to identify the most practical roles. Graph methods map users, roles and permissions to vertices and their relationships to edges. Greedy approaches to identify roles attempt to minimise the graph representation according to cost metrics.

However, a major limitation with existing approaches is they are prone to local minima. Existing approaches consider only the basic role relationships and not their effect on the global cost. There are role relationships that have not been explored. While simple role costs have been attempted, we show that these cost evaluations of individual roles are approximations that over estimate the benefit of a role and actually encourage local minima.

We are the first to formalise the cost components of the RBAC model using k-partite graphs. This is also the first work to provide an algorithm that produces high quality roles (in terms of minimising administration costs) *on large-scale real data sets*. In particular, we make the following contributions:

1. We propose to model the RBAC problem using a k-partite graph, which allows for the representation of both flat (tri-partite) and hierarchical roles (k-partite where $k > 3$). Existing representations do not allow for the effective representation of hierarchical roles in real enterprises.

2. We propose two schemes to evaluate the cost of managing roles in the enterprise. The first focuses on the assignments of each role, which should be used when the administration costs of the role are the most critical consideration of the enterprise. The second cost model focuses on the cost of modifying users and permissions after roles have been deployed, which should be used if there is dynamic change of users and permissions in the enterprise.

3. We propose *RoleAnnealing*, an algorithm in the spirit of simulated annealing to pro-

duce high quality RBAC configurations using k-partite role graph representations. To improve performance we propose the following optimisations: *guided solution search* to filter undesirable roles, *incremental computation* to reduce graph operations and speed up cost calculations and *role bounds* to reduce role cost computation.

4. We test on both real and synthetically generated data. The results show *RoleAnnealing* can efficiently produce exceptional results. *RoleAnnealing* is significantly more efficient than direct application of simulated annealing and greedy algorithms.

The paper is organised as follows. Section 2 describes related work. We present our model of role engineering using k-partite graphs in Section 3. Sections 4 and 5 will propose two different schemes for evaluating role graph cost and exact role cost. We introduce *RoleAnnealing* in Section 6 and use it to test and examine the cost models in Section 7, where we also compare *RoleAnnealing* with an existing approach and discuss our findings. Finally, Section 8 contains the conclusion and future work.

2 Related Work

Role Based Access Control (RBAC) is a security administration concept for maintaining data privacy where users are assigned permissions through roles [6, 13]. Role engineering is the process of identifying a set of roles that accurately reflect the internal functionalities of an enterprise [2].

Early works for role elicitation were scenario based, use-case and process driven [4, 11, 12]. Permissions required for a scenario or process were grouped into a role; permissions that were required in a particular use-case were placed into roles. However, automated and semi-automated approaches are prefered as they have been shown to produce a 60% cost saving during role development, a 50% cost saving during role maintenance [9].

From a theoretical point of view, it has been shown that finding the optimal number of descriptive roles along with its variations are NP-Complete by Vaidya et al. and Lu et al. [10, 14]. This important work has resulted in subsequent studies producing heuristics and greedy algorithms for solution identification as the problems have also been shown to be hard to approximate by Ene et al. [3].

Recent efforts have deployed data mining as well as graph based techniques to address the role engineering problem. Association mining and a role cost concept to identify roles was used by Colantonio et al. [1]. A large set of candidate roles is created using a lattice technique. After all possible roles are created, roles are pruned according to a role cost. However, roles are deleted using what we show to be the lower bound for the actual role cost, incorrectly evaluating the value of a role and leaving roles that are not beneficial.

FP-trees for fast frequent pattern mining to identify frequent permission sets as roles was proposed by Zhang et al. [17]. Roles are added based on a cost metric, which does not over estimate role benefit. However, only a partial cost component is used for their cost metric.

Graph based approaches for role engineering have also been proposed. Graph optimisation for minimising role nodes and edges with an iterative merging technique that is order dependent has been proposed by Zhang et al. [16]. The process begins by inserting users and permissions into the graph as nodes and user-to-permission assignments into the graph as edges. A role based on each user's permission assignment is created and inserted into the graph. Pairs of roles are then merged based on one of three scenarios. If the roles are the same, they are merged into one role. If two roles exhibit partial ordering relations, an edge is created between two roles. If two roles have a common set of permissions

and do not fall under the first two scenarios, a new role is created and partial ordering relationships are created between the new role and the existing two roles. These merges are performed iteratively until no more merges are possible. Only hierarchical RBAC is permitted with this approach. The work has since been extended into exact and greedy algorithms for graph optimisation of flat RBAC using cliques and graph theory by Ene et al. [3].

However, there is currently no formal method for cost evaluation, existing approaches have not fully analysed the effects of roles on the overall framework and exact role costs have not been computed.

3 Problem Modeling

This section contains the terminology that we will use throughout the rest of the paper and mappings of both flat and hierarchical RBAC concepts to matrix representation. We then propose k-partite graph representation and analyse the costs of the k-partite graph in relation to role engineering.

3.1 Notation

We use the following notations proposed by the National Institute of Standards and Technology (NIST) for RBAC [6]:

- *USERS*, *ROLES* and *PRMS*, the set of users, roles and permissions respectively where permissions represent allowable operations on objects within the system.
- *UA* $\subseteq$ *USERS* $\times$ *ROLES*, a many-to-many mapping of user-to-role assignments.
- *PA* $\subseteq$ *PRMS* $\times$ *ROLES*, a many-to-many mapping of permission-to-role assignments.
- *RH* $\subseteq$ *ROLES* $\times$ *ROLES*, a partial order on *ROLES* called the inheritance relation, written as $\succeq$, where $r_1 \succeq r_2$ only if all permissions of r_2 are also permissions of r_1.
- *RolePermissions(r)*, the set of all permissions either directly granted or indirectly inherited by role r.
- *UserPermissions(u)*, the set of all permissions user u gets through his or her directly assigned or indirectly inherited roles.

We will also discuss *UP* $\subseteq$ *USERS* $\times$ *PRMS*, the underlying many-to-many mapping of user-to-permission assignments which must be maintained to ensure data privacy.
|*UserPermissions(u)*| and |*RolePermissions(r)*| will also be referred to as the number of permissions of user u or role r.

The graph is a directed acyclic graph (DAG) represented as $G = (V, E)$ where $V = V(G)$ is the set of vertices and $E = E(G)$ is the set of edges. E is a set of ordered pairs; when mapped to RBAC components the direction of the edges are user-to-permission $u \rightarrow p$, user-to-role $u \rightarrow r$, role-to-permission $r \rightarrow p$ and role-to-role $r_1 \rightarrow r_2$ when $r_1 \succeq r_2$. We say $v \in V$ and $w \in V$ are connected if $vw \in E$ and $v \rightarrow w$, where v is the head, w is the tail. The in-degree of a vertex $v \in V$ will be denoted $\mathrm{d}^-(v)$ and the out-degree of a vertex $v \in V$ will be denoted $\mathrm{d}^+(v)$. The degree of vertex $v \in V$ is $\mathrm{d}(v) = \mathrm{d}^-(v) + \mathrm{d}^+(v)$. In RBAC, graph $G = (U, R, P, E_{UR}, E_{RR}, E_{RP})$ where U is the set of users, R is the set of roles, P is the set of permissions, E_{UR} are edges between U and R, E_{RR} are edges between R and R, and

E_{RP} are edges between R and P. In the absence of roles, E_{UP} represent edges between U and P.

These graphs can often be separated into k-partite graphs where vertices are placed into k levels and edges may only exist between levels.

In this paper, we do not consider sessions or separation of duty constraints. However, they can be incorporated into our framework through temporal graphs or additional contraints on roles.

3.2 Data Representation

Access control can be modeled as matrix A, a sparse matrix that shows permission assignments of every user. $A = [a_{ij}]$ represents *UP* where $a_{ij} = 1$ implies user i has permission j and $a_{ij} = 0$ implies user i does not have permission j.

In A, each row represents all the permissions assigned to a particular user and each column represents the users a particular permission has been assigned to. This matrix represents direct user-to-permission assignments. In RBAC, the intermediary concept of *ROLES* is added. This addition of roles inserts a layer of abstraction between permissions and users where roles are assigned to users (*UA*) and permissions are then assigned to roles (*PA*). This addition of roles can be represented as a decomposition of A into a user-to-role matrix B and a role-to-permission matrix C and a binary matrix multiplication operator $\circledast$.

$$A = B \circledast C \tag{1}$$

Where matrix $B = [b_{ij}]$ represents user-to-role assignments *UA*. $b_{ij} = 1$ implies user i has role j and $b_{ij} = 0$ implies user i does not have role j. Matrix $C = [c_{ij}]$ represents role-to-permission assignments *PA*. $c_{ij} = 1$ implies role i has been given permission j and $c_{ij} = 0$ implies role i has not been granted permission j. If $a_{ij} = 1$, the decomposition of A into B and C should contain a $b_{ik} = 1$ and $c_{kj} = 1$ for at least one role k. If $a_{ij} = 0$, $b_{ik} = 0$ and $c_{kj} = 0$ for all k. This ensures data privacy.

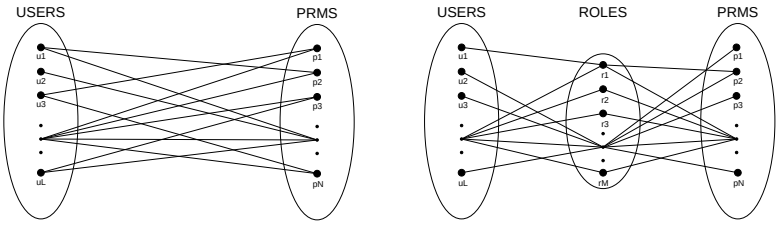
This single decomposition allows for the representation of one layer of roles between users and permissions. When the role hierarchy RH exists, an additional matrix decomposition can be used to represent relationships between roles. For example, the decomposition of A with a partial ordering hierarchy could be represented as $A = B \circledast B' \circledast C$ where B' represents role-to-role assignments. That is, $B' = [b'_{ij}]$ where $b'_{ij} = 1$ implies $r_i \succeq r_j$.

An example of $\circledast$ is boolean matrix multiplication or $a_{ij} = \bigvee_k b_{ik} \wedge c_{kj}$ where $\wedge$ represents the boolean *and* operation and $\bigvee$ represents the boolean *or* operator.

Matrices A, B and C can also be mapped to relations $R_A \subseteq U \times P$, $R_B \subseteq U \times R$ and $R_C \subseteq R \times P$ respectively. Doing so allows for the usage of the natural join operator: $R_A = R_B \bowtie R_C$. Substantial effort has been dedicated into the research of the natural join operation while limited need for an inverse join operation means the decomposition operation remains relatively unstudied. However, it can be seen that the role engineering problem is fundamentally an inverse join operation that needs to be performed according to some minimisation constraints.

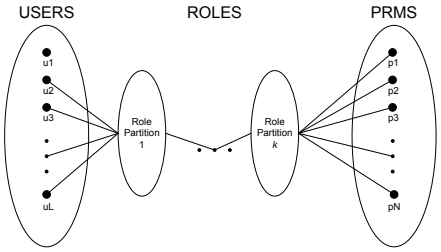
3.3 Modeling Role Engineering by k-Partite Graph

We map the role engineering problem to a k-partite graph, allowing for representation of hierarchical roles as well as flat roles while maintaining the original data privacy.



(a) Bipartite graph can represent user permission assignments

(b) Tripartite graph can represent user-to-role and role-to-permission assignments.



(c) k-partite graph can represent user-to-role, role-to-role and role-to-permission assignments.

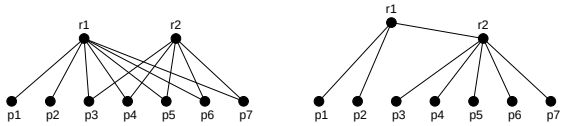
Figure 1: k-partite graph representations of access control

In the absence of a higher form of access control, user-to-permission assignments can be represented as a bipartite graph $G = (U, P, E_{UP})$ (Figure 1(a)). The graph is a bipartite graph ($k = 2$) with each user u and permission p represented as a vertex and each user permission assignment up represented as an edge.

In the presence of roles, the simplest form of RBAC can include a single layer of roles between users and permissions. This produces tripartite graph $G = (U, R, P, E_{UR}, E_{RP})$ as shown in Figure 1(b). In this configuration, permissions no longer have to be directly assigned to users. The insertion of roles does not disturb the existence of paths from users to permissions; the user-to-permission assignments represented by the graph are not modified and data privacy is maintained. Insertion of roles into the graph allows a tripartite graph ($k = 3$).

However, hierarchical RBAC cannot be modeled by this three tier representation. In hierarchical RBAC, partial ordering RH relationships exist between roles, complicating both the matrix and the graph representation. A partial ordering can be placed between any two roles $r_1 \succeq r_2$ if super role $r1$ contains all the permissions of subrole $r2$. This relationship means the super role of the partial ordering no longer has to be directly assigned to the permissions of the subrole; the permissions in the subrole are inherited by the super role. Figure 2 is an example of roles in a flat hierarchy versus a two level hierarchy.

Any subrole is also the potential super role of another role, adding to the complexity of role relationships. Multiple levels of roles could exist within the partial ordering, requiring consideration of a k-partite graph $G = (U, R, P, E_{UR}, E_{RR}, E_{RP})$ for full RBAC represen-



(a) Flat roles (b) Hierarchical roles

Figure 2: An example of flat roles vs. hierarchical roles in a partial ordering.

tation (Figure 1(c)). In this graph, vertices still represent *ROLES*, *USERS* and *PRMS* with edges now representing the role hierarchy *RH* as well as *UA* and *PA*. Users access permission assignments through roles, with roles now also capable of accessing permissions through other roles. This additional role-to-role relationship means G can potentially be separated into k levels depending on the number of levels of inheritance present in the graph.

The size of k will depend on $ROLES$. Any RBAC configuration will have at least $k = 3$ levels. In the presence of the role hierarchy RH, the number of levels (k) can only be as large as the number of permissions in the largest role.

Lemma 1. *The number of levels required to model any RBAC graph is between 3 and the number of permissions in the largest role* $+2$ *inclusive. That is, the range is* $[3, max(|(RolePermissions(r)|)+2]$ *where* $max(|(RolePermissions(r)|)$ *is the size of the largest role.*

Proof. When flat roles are inserted into the graph, RBAC is represented by a tripartite graph; one level for users, one level for roles and one level for the permissions ($k = 3$). When partial orderings are allowed, if $r \succeq r'$, r and r' belong to different levels. For role r_1 with size $|(\text{RolePermissions}(r_1)| = s$, a new role r_2 with size $|(\text{RolePermissions}(r_2)| = s - 1$ such that $r_1 \succ r_2$ can be created. Role r_3 can be created using r_2 such that $|(\text{RolePermissions}(r_3)| = |(\text{RolePermissions}(r_2)|-1 = s-2$ and $r_2 \succ r_3$. Using the same method, roles $r_1, r_2, \ldots r_s$ can be created such that $r_1 \succ r_2 \succ \cdots \succ r_s$ where $|(\text{RolePermissions}(r_s)| = 1$. A RBAC graph with these roles $r_1, r_2, \ldots r_s$ has $s+2$ levels, one for each of the roles in the connected partial ordering, one level for users and one level for permissions. At most, the largest number of role levels is equal to the size of the largest role and $k = \max(|(\text{RolePermissions}(r)|)+2$. □

However, it is very unlikely all of these roles exist. Usually $k << \max(|(\text{RolePermissions}(r)|)+2$. In some of the authors' software deployment enterprise experience, $6 \leq k \leq 7$.

3.4 Problem Definition

Since the purpose of RBAC is to reduce administration costs while maintaining data privacy, migrating from a bi-partite model of access control to a k-partite model for RBAC should reduce the administration required while maintaining the same underlying user-to-permission assignments. Reducing the complexity of the graph represents reducing the administration required on the configuration.

Finding the most administratively beneficial roles is equivalent to finding the RBAC graph with least administrative cost. However, the solution is not straight forward. When adding roles, simply minimising the number of roles does not always reduce the administration cost [3]. Reducing the number of edges, or role induced relationships, may increase the

number of roles [16]. While modifying the graph structure to reduce costs, the existence of paths between users and permissions must not be violated. This constraint ensures no additional permissions are granted and no existing permissions are removed from users.

Clearly more analysis of individual roles and their relationships needs to be performed when identifying cost models. Previously unconsidered is the effect of users and permissions in the graph. While their presence may not be modified, their relationship with roles should be studied. The presence of different roles dramatically effects how permissions are assigned to users and as a result, the effectiveness or standalone cost of roles.

Finally, the administrative maintenance cost of the model should also be considered. The average cost of modifying each user's permissions is just as important as the final cost of the model. After RBAC is established, it is expected the roles and role-to-permission assignments remain more static than users and their role requirements. As a result, RBAC modification during the addition and removal of users, permissions should be considered.

The goals of cost based role engineering graph optimisation can be formalised as follows:

1. *Given USERS and PRMS and ROLES, identify how administration cost can be measured, isolate different cost components and analyse their effect on the global infrastructure.*

2. *Given USERS and PRMS, find ROLES such that the graph representation of USERS, PRMS, ROLES and their relationships has minimal cost in accordance with an administration requirement metric.*

3. *Given a cost model for the graph representation of USERS, PRMS, ROLES and their relationships, find cost of individual roles with respect to the whole graph.*

3.5 Overview of Cost Models

The purpose of mapping role engineering to k-partite graphs is to facilitate a more comprehensive analysis of RBAC configurations. We define the problem and formalise our goals in order to create a clear objective for graph based role engineering. To evaluate our graphs, we propose two cost models in the next sections. These cost models measure different components of interest in real RBAC configurations.

The Role Edge Graph Cost Model focuses on the costs associated with roles and their role relationships. This cost model uses a static instance of the enterprise and focuses on minimising the administration required to manage roles. This cost model would be used if the enterprise's main concern is to minimise the management of roles and their relationships, and there is limited change in users and permissions

The Administration Graph Cost Model focuses on the cost of modifying the graph after RBAC has been implemented. An enterprise would choose this model to evaluate their RBAC configuration if their main concern is how to update user permissions when users are added, removed or modified; and role permissions when resources are added, removed or modified.

Both of the proposed cost models evaluate the cost of the entire graph as well as individual roles.

4 Role Edge Graph Cost Model

We first propose a graph model that evaluates RBAC based on the number of roles and role relationships in the configuration. Each role and role related assignment is a relationship

that has to be administered and managed, as a result, each role and role related assignment should be measured. Users and permissions remain constant during the role engineering process, as a result, they do not directly need to be evaluated in the cost model. We evaluate users and permissions through their relationships with roles. This is reflected in the Role Edge Graph Cost Model.

4.1 Cost of a Graph

The first graph cost model we analyse is as follows:

$$\text{cost}(G) = c_1|V_r| + c_2|E| \tag{2}$$

Where c_1 is the role cost constant, c_2 is the user, permission or role assignment cost constant, $|V_r|$ is the number of vertices that represent roles in the graph and $|E|$ is the number of edges in the graph, each representing a user, permission or role assignment. In this model, the vertices that represent USERS and PRMS are not included. Their presence remains constant at any given instance in time. We discuss the modification of users and permissions in Section 5.

Constants c_1 and c_2 represent the static cost of role administration and assignment administration respectively. The model does not differentiate between role-to-role, user-to-role or user-to-permission assignments; they are each evaluated at a cost of c_2. These parameters are tunable based on external analysis, depending on how costly role management and permission management is perceived. The impact of the magnitude of the c_1 and c_2 parameters is also discussed in our experimentation (Section 7) and analysis (Section 7.5).

Every role graph with $k \geq 2$ levels can be evaluated using this model. Direct user-to-permission graphs ($k = 2$), flat role graphs ($k = 3$) and hierarchical graphs ($k > 3$) can be evaluated. When $k = 2$, $|V_r| = 0$ and the administration cost becomes the cost of assigning permissions directly to users in the absence of roles $c_2|E|$. When $k > 2$, the cost of roles $c_1|V_r|$ and the cost of permission assignments in the presence of the roles $c_2|E|$ are evaluated. When $k = 3$, E includes user-to-role and role-to-permission assignments. When $k > 3$, E includes user-to-role, role-to-role and role-to-permission assignments.

The balance between roles and how permissions are assigned can also be measured by this model. When moving from $k = 2$ to $k > 2$, the number of edges should decrease while the number of roles increases. An ideal number of roles is identified when adding more roles does not produce a reduction in user-role, role-role or user-permission administration requirements. At this point, removing roles will increase the number of user-role, role-role or user-permission administration requirements.

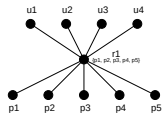
4.2 Cost of a Role

When moving from one role graph solution to another, it is often more efficient to compute the expected change in cost. In role engineering, there are two fundamental operations that can be performed: removing a role and creating a role. The change in cost resultant from both operations requires the evaluation of the role cost.

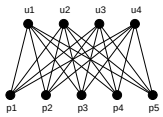
When using the Role Edge Graph Cost, the cost of a role should be computed as follows:

$$\text{cost}(r) = c_1 + c_2\text{d}(r) - c_2[\text{d}^-(r) \times \text{d}^+(r)] + c_2|\text{inherited}| \tag{3}$$

Where $\text{d}^-(r)$ is the number of users or roles that r can be assigned to, $\text{d}^+(r)$ is the number of roles and permissions that can be assigned to r and $\text{d}(r) = \text{d}^-(r) + \text{d}^+(r)$. $|\text{inherited}|$

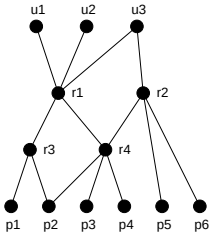


(a) Graph cost = 10.

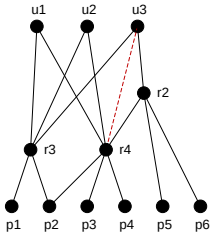


(b) Graph cost = 20.

Figure 3: cost$(r1) = 1 + 9 - 20 + 0 =$ -10.



(a) Graph cost = 18.



(b) Graph cost = 17.

Figure 4: cost$(r1) = 1 + 5 - 6 + 1 = 1$

is the number of connected tail vertices that each connected head vertex has already been assigned through another role vertex.

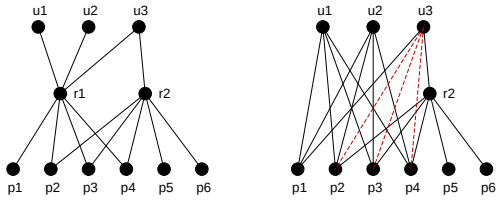
In this evaluation, the cost of a role can be broken down into three parts: the additional cost required to create the role ($c_1 + c_2$d(r)), the anticipated reduction in cost resultant from the presence of the role ($c_2[\text{d}^-(r) \times \text{d}^+(r)]$) and the difference between actual reduction in cost and anticipated reduction in cost ($c_2|$inherited$|$).

The cost required to create the role is the cost of the role itself c_1 and the number of users, roles or permissions that can be assigned to this role c_2d(r). The cost reduction is the number of user-to-permission, user-to-role or role-to-role assignments it saves. This reduction is the anticipated number of assignment reductions ($c_2[\text{d}^-(r) \times \text{d}^+(r)]$) less the number of assignments that have been made through other roles ($c_2|$inherited$|$.)

Examples of graph costs and role cost using Equation 2 with $c_1 = 1$ and $c_2 = 1$ are shown in Figures 3, 4 and 5.

Figure 3 demonstrates how the first two parts of the role cost can be evaluated. In Figure 3(a), $\text{d}^-(r1) = 4$ and $\text{d}^+(r1) = 5$. Without role $r1$, Figure 3(b) shows each of $r1$'s head connected vertices requiring an edge to each of $r1$'s tail connected vertices. This is the $\text{d}^-(r1) \times \text{d}^+(r1)$ cost saving. As a result, the cost of the role is cost$(r1) = 1 + 9 - 4 \times 5 =$ -10. Removing role $r1$ from the graph in figure Figure 3(a) with cost = 10 produces graph Figure 3(b) with a cost = 20.

Computation of the additional cost required to create the role ($c_1 + c_2$d(r)) and the antic-



(a) Graph cost = 15. (b) Graph cost = 16.

Figure 5: cost$(r1) = 1 + 7 - 12 + 3 =$ -1.

ipated reduction produced by the role ($c_2[\mathrm{d}^-(r) \times \mathrm{d}^+(r)]$) is sufficient when there are no overlapping roles. That is, when permissions belong to at most one role. When permissions can belong to more than one role, $c_2|$inherited$|$ needs to be considered.

In Figure 4, $\mathrm{d}^-(r1) = 3$ and $\mathrm{d}^+(r1) = 2$. However, in the absence of role $r2$, it can be seen in Figure 4(b) that the permissions of role $r4$ are already inherited by user $u3$ through role $r2$. The edge from user $u3$ to role $r4$ ($c_2|$inherited$|$) is not required. As a result, the actual cost of $r1 = 1$.

This situation also holds for flat RBAC when $k = 3$ as shown in Figure 5.

5 Administration Graph Cost Model

The Role Edge Graph Cost presented in the Section 4 evaluates the number of roles and edges. While this can be an effective metric, all edges are evaluated equally. The cost model also only evaluates the graph at that instance, without consideration of what administration is required when updating the model. To allow for edge cost flexibility with the consideration of ongoing maintenance costs after deployment, this section will propose a cost model that separates the relationship costs of users, permissions and roles based on the average update cost associated users and permissions.

5.1 Cost of a Graph

The two major components subject to modification after deployment are users and permissions with the number of roles minimal. The following represents the Administration Graph Cost:

$$\mathrm{cost}(G) = c_1 \mathrm{m}(U) + c_2|V_r| + c_3 \mathrm{m}(P) \tag{4}$$

Where c_1 is a user cost constant, c_2 weights the role cost, c_3 is a permission cost constant, $\mathrm{m}(U)$ is the administration cost of a user, $\mathrm{m}(P)$ is the administration cost of a permission and V_r is the number or role vertices.

Upon the assignment or removal of a user's access rights, it is expected that the average number of permission rights need to be created or deleted. The expected administration

cost of a user can be defined as the average update cost of a user.

$$\mathrm{m}(U) = \frac{\sum_{i=1}^{|U|} \mathrm{d}(u_i)}{|U|}$$

Where $|U|$ is the number users and $\mathrm{d}(u_i)$ the number of role or permission nodes connected to user i.

The same applies for permissions. When a resource or a particular permission needs to be removed, the average number of permission edges can be used as an indication of the permission update cost.

$$\mathrm{m}(P) = \frac{\sum_{i=1}^{|P|} \mathrm{d}(p_i)}{|P|}$$

Where $|P|$ is the number of permissions and $\mathrm{d}(p_i)$ the number of user or role nodes connected to permission i.

Each of the three components measures a desirable quality of the RBAC graph. The user component ensures that each role is as descriptive of users as possible. The permission component ensures roles are not overly repetitive and as distinct as possible. The role component ensures that the number of descriptive roles chosen is minimal.

5.1.1 Alternative Cost of a Graph

In Equation 4, c_2 is the fraction of users (or permissions) which need to have their complexity reduced by one edge on average when introducing a new beneficial role. For example, if there are 100 users and $c_2 = 1$ (and $c_1 = c_2 = c_3$), at least 100 user edges need to be removed for a new role to be introduced.

An alternative form of Equation 4 is to scale the second term by the number of users and permissions. i.e. divide c_2 by the mean of $|U|$ and $|P|$, i.e. $(|U| + |P|)/2$, or $\sqrt{(}|U| \times |P|)$.

An alternative form of the equation is now as follows:

$$cost = c_1\mathrm{m}(U) + c_2\frac{|V|_r}{\sqrt{(}|U| \times |P|)} + c_3\mathrm{m}(P) \tag{5}$$

c_2 is now the absolute number of user (or permission) edges which need to be removed for the role to be considered beneficial. e.g. if $c_2 = 10$, then adding a role needs to remove 10 user edges (e.g. be beneficial for 10 users), independent of the organisation size.

Approximately speaking, in Equation 4, c_2 is the fraction of users (or permissions) which need to be simplified for a role to be added, whereas in Equation 5, c_2 represents the number of users (or permissions) which need to be simplified to add a role, i.e. the size of a group for which it becomes administratively beneficial to create a role.

Equation 4 and 5 differ only in their role cost components. Since $|U|$ and $|P|$ are constants for any instance of a given G (the number of users and permissions does not change), $\sqrt{(}|U| \times |P|)$ is also a constant, making Equation 4 a more generic form of Equation 5. For the rest of the paper, we will use Equation 4.

5.2 Cost of a Role

As with the Role Edge Graph Cost, it is often more efficient to compute the change in cost of adding or removing a role.

$$\begin{aligned} \text{cost}(r) &= c_1 \frac{\text{d}^u(r) - \text{d}^u(r) \times \text{d}^+(r)}{|U|} + c_2 + \\ & \quad c_3 \frac{\text{d}^p(r) - \text{d}^-(r) \times \text{d}^p(r)}{|P|} + \\ & \quad c_1 \frac{|\text{inherited}^u|}{|U|} + c_3 \frac{|\text{inherited}^p|}{|P|} \end{aligned} \tag{6}$$

Where $\text{d}^-(r)$ is the number of users or roles that r can be assigned to, $\text{d}^+(r)$ is the number of roles or permissions that can be assigned to r, $\text{d}^u(r)$ is the number of users that can be assigned to r, $\text{d}^p(r)$ is the number of permissions that can be assigned to r and $|\text{inherited}|$ is the number of connected tail vertices that connected head vertex has already been assigned through another role vertex.

In this evaluation, $c_1 \frac{\text{d}^u(r) - \text{d}^u(r) \times \text{d}^+(r)}{|U|}$ represents change in cost associated with the average user update cost. $\text{d}^u(r)$ is the number of new user-to-role edges that need to be created and $\text{d}^u(r) \times \text{d}^+(r)$ is the number of user related edges that is saved. c_2 is the increment to the role cost and $c_3 \frac{\text{d}^p(r) - \text{d}^-(r) \times \text{d}^p(r)}{|PRMS|}$ represents the change to the permission update cost component. $\text{d}^p(r)$ is the number of new role-to-permission edges that need to be created and $\text{d}^-(r) \times \text{d}^p(r)$ is the number of permission related edges that are saved. To ensure permissions that have been assigned through other roles are not included in the cost, $\frac{|\text{inherited}^u|}{|U|}$ modifies the user maintenance cost and $c_3 \frac{|\text{inherited}^p|}{|P|}$ modifies the permission maintenance cost.

6 Algorithms

We propose *RoleAnnealing*, a framework for efficiently identifying administratively beneficial RBAC configurations while ensuring data privacy. *RoleAnnealing* is an algorithm in the spirit of simulated annealing that uses both exact role costs and role cost bounds to reduce computation. To improve performance, we also propose incremental computation, guided search, use of role bounds and separation of create role and delete role operations. As a result, *RoleAnnealing* scalable algorithm that produces high quality RBAC configurations for real enterprise environments.

Simulated Annealing is a probabilistic algorithm for finding the global optimum [8]. Given infinite search time, the approach is guaranteed to reach the optimal solution. Given a short search time, good results are still recoverable. The process starts with an arbitrary instance of the problem and explores solution spaces close to the current instance. Unlike greedy algorithms, the next inferior solution is not always rejected. Based on annealing principles, we use a cooling function where during earlier stages of the simulation when the temperature is high, nearby solutions that increase the cost are accepted with high probability. As the temperature decreases, jumps out from a gradual cost decline are made less likely.

In relation to role engineering, direct application of simulated annealing would be straight forward, slow and potentially infeasible for large scale problems. Algorithm 1 represents

Algorithm 1: *SimulatedAnnealing* - Direct application to Role Engineering

Require:
G - Current graph configuration
α - Inferior solution acceptance constant
Result:
G^* - Best graph configuration identified by *SimulatedAnnealing*

```
1  begin
2      i = 0, G* = G
3      while more beneficial operations do
4          G^m = modify(G)
5          if cost(G^m) < cost(G) then
6              G = G^m
7          end
8          else
9              Δ = cost(G^m) − cost(G)
10             p = e^(−αΔi) // probability of accepting inferior model
11             r = randomNumber() // Generate random number between 0 and 1
12             if r < p then
13                 if cost(G) < cost(G*) then
14                     G* = G
15                 end
16                 G = G^m
17             end
18         end
19         increment i
20     end
21     if cost(G) < cost(G*) then
22         G* = G
23     end
24 end
```

the direct application of Simulated Annealing to the Role Engineering Problem. To improve performance, additional optimisations and search space techniques can be used. We propose incremental computation, bounds for role costs, guided solution search heuristics and separation of create and delete operations; Algorithm 2 represents *RoleAnnealing*, an improved and optimised approach for role discovery.

Both algorithms require G, the initial start state of the role graph; α, a constant that determines acceptance probability of inferior solutions and a cost function for evaluating the graph. The cost function can be the Role Edge Graph Cost, Administration Graph Cost or any other well defined cost metric for role engineering. The result is G^*, the optimal or near optimal role engineering graph solution. Depending on the allowable operations, G^* can represent either flat RBAC or hierarchical RBAC.

In Algorithm 1 for SimulatedAnnealing, Line 4 modifies the graph G to produce graph G^m. Depending on the cost of G^m, it is either accepted as an improvement (Line 6), probabilistically accepted if it is an inferior solution (Line 8 – 16) or rejected. This SimulatedAnnealing application is computationally expensive. The most expensive components relate

Algorithm 2: *RoleAnnealing* - Optimised with incremental computation and guided search space heuristics

Require:
G - Current graph configuration
α - Inferior solution acceptance constant
Result:
G^* - Best graph configuration identified by *RoleAnnealing*

```
1  begin
2      i = 0, G* = G
3      while more has beneficial operations do
           // Phase 1 - Role Creation
4          role = guidedSearchForRoleToCreate() // create roles that have more
               than 4 edges and permission sets that are intersects of
               existing roles
5          Δ = costLowerBound(role)
6          if Δ ≤ 0 then
7              Δ = costExact(op.role)
8          end
9          if Δ < 0 then
10             G = createRole(G, role)
11         end
12         else
13             p = e^(−αΔi) // probability of accepting inferior model
14             r = randomNumber() // Generate random number between 0 and 1
15             if r < p then
16                 if cost(G) < cost(G*) then
17                     G* = G
18                 end
19                 G = modify(G, op)
20             end
21         end
22         increment i
23     end
24     if cost(G) < cost(G*) then
25         G* = G
26     end
       // Phase 2 - Role Deletion
27     for every role in G* do
28         if costExact(role) > 0 then
29             deleteRole(G*, role)
30         end
31     end
32 end
```

to the modification of the graph. Randomly choosing a role node to delete or insert can also be expensive.

To improve on Algorithm 1, we propose *RoleAnnealing* which separates create and delete role operations and uses incremental computation, role cost bounds and guided solution space search techniques for fast identification of RBAC configurations (Algorithm 2). *RoleAnnealing* is performed in two phases; creation of roles in Phase 1 and deletion of roles in Phase 2.

In Phase 1, instead of randomly modifying the graph at every iteration, Line 4 of Algorithm 2 uses guided search techniques to identify a role creation; the role should not already exist in the graph and should have size of at least 2 and is assigned to at least two users or roles. The permission sets of the roles are selected to be intersects of existing roles or a particular user's permission set. Duplicate roles and roles containing single permissions will never be beneficial and roles with randomly generated permission sets are very unlikely to be of benefit. In this phase, only create operations are allowed; separating create and delete operations significantly speeds up the annealing process.

The next step is to calculate the cost of the operation; this can be done using Equation 3 or 6. Both of these computations require the identification of the permissions that have been assigned through other roles. This operation may require traversal of $k-3$ depth where $k-3$ is the number of levels of the role graph less one level for users, one level for permissions and one for the role to create. This computation can be reduced with negligible effect on the *RoleAnnealing* process through the analysis of the |inherited| component of Equations 3 and 6.

When using the Role Edge Graph Cost Model (Equation 2), $|\text{inherited}| \geq 0$ so $c_1 + c_2\text{d}(r) - c_2[\text{d}^-(r) \times \text{d}^+(r)] \leq \text{cost}(r)$, giving us a lower bound on role cost. In the worst case scenario, $|\text{inherited}| = \text{d}^-(r) \times \text{d}^+(r)$ when every connected tail vertex has already been assigned to every connected head vertex through another role and $\text{exact}(r) = c_1 + c_2\text{d}(r)$. This gives the upper bound on exact role cost.

Let $\text{lower}(r) = c_1 + c_2\text{d}(r) - c_2[\text{d}^-(r) \times \text{d}^+(r)]$. The upper and lower bounds on role cost for the Role Edge Graph Cost Model are as follows:

$$\text{lower}(r) \leq \text{cost}(r) \leq c_1 + c_2\text{d}(r) \tag{7}$$

A beneficial role produces a cost reduction; when $\text{cost}(r) < 0$. When evaluating a role, if $\text{lower}(r) > 0$, $\text{cost}(r) > 0$ so cost of the role does not need to be computed to see that the role is not beneficial. Only when $\text{lower}(r) < 0$ does $\text{cost}(r)$ needs to be computed as it is still possible that $\text{cost}(r) > 0$.

Similar concepts can be applied to the Administration Graph Cost Model. Since $c_1\frac{|\text{inherited}^u|}{|U|} + c3\frac{|\text{inherited}^p|}{|P|} \geq 0$, $c_1\frac{\text{d}^u(r) - \text{d}^u(r) \times \text{d}^+(r)}{|U|} + c_2 + c_3\frac{\text{d}^p(r) - \text{d}^p(r) \times \text{d}^-(r)}{|P|} \leq \text{cost}(r)$, giving a lower bound for role cost. In the worst case situation, $c_1\frac{|\text{inherited}^u|}{|U|} + c3\frac{|\text{inherited}^p|}{|P|} = c_1\frac{\text{d}^u(r) \times \text{d}^+(r)}{|U|} + c_3\frac{\text{d}^p(r) - \text{d}^p(r) \times \text{d}^-(r)}{|P|}$ and $cost(r) = c_1\frac{\text{d}^u(r)}{|U|} + c_2 + c_3\frac{\text{d}^p(r)}{|P|}$. This gives us an upper bound on role cost.

Let $\text{lower}(r) = c_1\frac{\text{d}^u(r) - \text{d}^u(r) \times \text{d}^+(r)}{|U|} + c_2 + c_3\frac{\text{d}^p(r) - \text{d}^p(r) \times \text{d}^-(r)}{|P|}$. The upper and lower bounds on role cost for the Administration Graph Cost Model are as follows:

$$\text{lower}(r) \leq \text{cost}(r) \leq c_1\frac{\text{d}^u(r)}{|U|} + c_2 + c_3\frac{\text{d}^p(r)}{|P|} \tag{8}$$

According this role cost, the administrative user cost component is reduced if $\text{d}^u(r) + |\text{inherited}^u| < \text{d}^u(r) \times \text{d}^+(r)$ and the administrative permission cost component is reduced if $\text{d}^p(r) + |\text{inherited}^p| < \text{d}^p(r) \times \text{d}^-(r)$.

In general:

- When creating roles, cost(r) needs to be computed if lower$(r) < 0$ to avoid creating roles that are not beneficial.
- When deleting a role, cost(r) need to be computed if lower$(r) < 0$ to avoid leaving roles that should be deleted.

Existing methods that have used only the lower bound on role cost as the actual role cost over estimate the benefit of roles and do not prune roles that are actually costly to the infrastructure [1]. However, using a combination of lower bound and actual role cost improves performance.

In Algorithm 2, Line 5 computes the lower bounds of the create role operation. Only if $\Delta < 0$ does the actual cost of the role need to be computed (Line 7). When the operation is accepted, the modification the graph occurs either in Line 10 or probabilistically in Line 19.

When there are no more beneficial roles that can be created, a sequential check to see if any roles that have been created can then be deleted is performed in Line 27–29.

7 Experimental Results

7.1 Setup

The *RoleAnnealing* approach for optimising a given cost model was evaluated on access control on real data from undisclosed enterprise domains, real data from The Department of Computer Science at The University of Melbourne and synthetic data where an hierarchical RBAC structure is known. All tests were run using a single core on a 2.38GHz Dell Zeon E5440 Server.

The two real data sets that originate from an undisclosed domain (real data *r1* and *r2*) are obfuscated and contains user-to-role and role-to-permission assignments. The smaller of the two data sets has 117 users, 25 roles and 23 permissions (*r1*). The larger data set has 327 users, 237 roles and 10378 permissions (*r2*). Using each data set, the original RBAC graph G^o consists of users, roles and permissions as nodes and user-to-role and role-to-permission assignments as edges.

In the data from the Department of Computer Science and Software Engineering (real data *r3*), there is no formal access control administration set up; users request permissions to certain groups when required. All systems refer to a Unix groups access control system where each group is considered an assignable permission to a user. There are 2039 users, 309 permissions and 1630 permission assignments. The RBAC graph G^o contains the original users and permissions as nodes with their user-to-permission assignments as edges. To create the initial starting graph G using data *r3* for *RoleAnnealing*, each user and permission is added as a node and each user-to-permission assignment is added as an edge. A role with each user's permission set is then added to G.

To further validate our approach, synthetic data *s1*, *s2* and *s3* was created using the data generator for role engineering testing by Zhang et al. [19]. Given the number of users, roles, permissions, average and variance of user-to-role assignments and average and variance of user-to-permission assignments, the original graph G^o with the desired qualities can be generated.

Graph G^o from the real data sets *r1* and *r2* as well as the synthetically generated data *s1*, *s2* and *s3* contain the original roles. To create the initial starting graph for *RoleAnnealing*,

we use the underlying user permission assignments (*UP*) and add a role with each user's permission set to create G.

RoleAnnealing starts with G as the initial configuration and performs operations that permit role hierarchy RH relationships during each iteration. Roles identified by the final G^* are then compared with the roles from the original G^o.

Since the synthetic data is generated in a random manner, it is possible that G^o does not necessarily represent the most optimal infrastructure of the given data. It offers a good representation but it may not be the absolute best representation. If two roles from G' are always assigned together, it would be more beneficial to merge these roles and assign them as a single role. If a set of permissions is common between multiple roles, creation of a role based on the overlapping permission set would be beneficial to the role hierarchy RH architecture. When these situations occur, *RoleAnnealing* can infact identify these roles. As a result it is possible for the G^* produced by *RoleAnnealing* to have a lower cost than the original G^o.

7.2 Effectiveness

Table 1: *RoleAnnealing* results using Role Edge Graph Cost and Administration Graph Cost on synthetically generated data and real data with $c_1 = c_2 = c_3 = 1$ and $\alpha = 0.01$

Setup			Role Edge Graph Cost		Administration Graph Cost	
			Graph Cost	Details	**Graph Cost**	Details
Synthetic Data	s1	100u 10r 100p	cost(G^o) = 654 cost(G^*) = 578	$\|ROLES_{G^o}\| = 10$ $\|ROLES_{G^*}\| = 52$ 26.14 seconds	cost(G^o) = 17.04 cost(G^*) = 15.01	$\|ROLES_{G^o}\| = 10$ $\|ROLES_{G^*}\| = 11$ 3.94 seconds
	s2	500u 50r 500p	cost(G^o) = 2982 cost(G^*) = 2816	$\|ROLES_{G^o}\| = 50$ $\|ROLES_{G^*}\| = 134$ 3.03 minutes	cost(G^o) = 56.55 cost(G^*) = 52.67	$\|ROLES_{G^o}\| = 50$ $\|ROLES_{G^*}\| = 30$ 2.26 minutes
	s3	1000u 100r 1000p	cost(G^o) = 5767 cost(G^*) = 5576	$\|ROLES_{G^0}\| = 100$ $\|ROLES_{G^*}\| = 245$ 5.58 minutes	cost(G^o) = 103.33 cost(G^*) = 97.99	$\|ROLES_{G^o}\| = 100$ $\|ROLES_{G^*}\| = 24$ 5.69 minutes
Real Data	r1	117u 25r 23p	cost(G^o) = 1103 cost(G^*) = 272	$\|ROLES_{G^o}\| = 25$ $\|ROLES_{G^*}\| = 18$ 3.43 seconds	cost(G^o) = 34.86 cost(G^*) = 23.16	$\|ROLES_{G^o}\| = 25$ $\|ROLES_{G^*}\| = 13$ 11.94 seconds
	r2	3729u 237r 10378p	cost(G^o) = 29112 cost(G^*) = 28281	$\|ROLES_{G^0}\| = 237$ $\|ROLES_{G^*}\| = 598$ 7 hours 51.52 minutes	cost(G^o) = 207.37 cost(G^*) = 97.26	$\|ROLES_{G^0}\| = 237$ $\|ROLES_{G^*}\| = 23$ 168 minutes
	r3	2039u no roles 1630p	cost(G^o) = 1630 cost(G^*) = 1474	$\|ROLES_{G^o}\| = 0$ $\|ROLES_{G^*}\| = 35$ 1.09 minutes	cost(G^o) = 8.05 cost(G^*) = 8.05	$\|ROLES_{G^o}\| = 0$ $\|ROLES_{G^*}\| = 0$ 3.58 seconds

Given the initial starting G of each of the data sets, *RoleAnnealing* (Algorithm 2) was performed using both the Role Edge Cost (Equation 2) and the Administration Graph Cost (Equation 4). We set $c_1 = c_2 = c_3 = 1$ and $\alpha = 0.01$. Modifying these parameters are discussed in Section 7.2.1 and 7.2.2. The process was terminated after 1000 operations were consecutively rejected. Each experiment was performed 10 times and Table 1 shows the best result from each data set.

Using Role Edge Graph Cost (Equation 2), *RoleAnnealing* was able to identify RBAC configurations of lower cost than the original configurations in all six data sets. The G^* identified by *RoleAnnealing* often had a larger number of roles (*s1, s2, s3, r2, r2*). According to the

Role Edge Graph Cost, the original configurations would benefit if roles are created using intersects or supersets of the permissions in the original roles.

Creating roles using the overlapping intersects of two or more roles reduces the administration on role-to-permission relationships; permissions would only have to be directly assigned to the intersect role, the intersect role is assigned to the original roles and the permissions are inherited. Creating roles that are a combination of existing roles can reduce administration on user-to-permission relationships. Instead of assigning multiple roles together, only one role that contains these roles need to be assigned. *RoleAnnealing* using Equation 2 can effectively identify these roles and creates them to reduce overall cost.

Using the Administration Graph Role Cost (Equation 4), *RoleAnnealing* can identify configurations of lower cost and lower role quantity (*s2, s3, r1,* and *r2*). The Administration Graph Role Cost focuses on the benefit to the administrators after the RBAC configuration as been implemented. The subset of the roles that are identified using this cost are the roles produce the most benefit. It is interesting that in *r3*, the configuration of most administrative benefit has no roles. There are no roles in the original data; *r3* originates from an educational domain where the RBAC model is not suitable.

The Role Edge Graph Cost identifies a slightly larger set of roles for administrators to choose from. The Administration Graph Role Cost identifies a smaller set of roles; the top roles that would be most beneficial to the configuration. Given any automated approach is will be inspected by human administrators, these results offer good recommendations for what roles should be implemented.

For most of the data (*s1, s2, s3, r1* and *r3*), *RoleAnnealing* requires less than 30 minutes to process, often significantly less. The largest data set *r2* requires less than 8 hours with cost function Equation 2 and less than 3 hours with cost function Equation 4. In some of the author's experiences, these are acceptable wait periods for role recommendations as such computations are required infrequently. Manual processing to derive the the same results would take significantly longer than our automated approach. *r2* is a particularly large data set; the original solution contains almost 30 000 user-to-role and role-to-permission assignments that represents almost 600 000 user-to-permission assignments.

7.2.1 Effect of Acceptance Variable α

We also test *RoleAnnealing* with different values of inferior acceptance variable α. We use the setup as described in Section 7.1 with Role Edge Cost Model (Equation 2) on data set *s3* and measure the cost of the graph in relation to *RoleAnnealing* processing time given different α values. The results of our experimentation are shown in Figure 6, where the values for α were 0.0001, 0.01 and 100.

Modifying α effects the probability of accepting an inferior solution, as described in Line 13 of Algorithm 2.

$$p = e^{-\alpha \Delta i}$$

Where e is Euler's constant, Δ is the change in the graph and i is the iteration number. If Δ is large, p is smaller than what it would be if Δ is small. As i increases during each iteration, p decreases. When $\alpha = 0.0001$, the probability of accepting an inferior solution is high and many inferior solutions are accepted at the start of the algorithm. When $\alpha = 100$, the probability of accepting an inferior solution is so small, it can be considered as negligible. Only beneficial changes to the solution are accepted. As a result, the rate at which *RoleAnnealing* approaches the synthetic optimum is faster when α is larger. This can be seen in Figure 6. While in this particular data set the final results are similar, it is possible

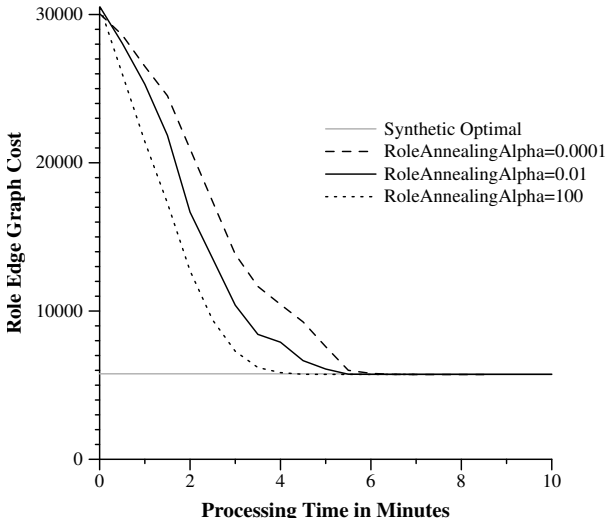


Figure 6: Processing times for RoleAnnealing, varying the value of α. As α increases, probability of accepting inferior solutions decreases.

Table 2: Average RoleAnnealing results varying $c1$ and $c2$ parameters for cost$(G) = c_1|V_r| + c_2|E|$ using data s3

$c1$	$c2$	Result
1	15	$\|ROLES_{G^*}\| = 217.7$
1	10	$\|ROLES_{G^*}\| = 222.8$
1	5	$\|ROLES_{G^*}\| = 218.1$
1	1	$\|ROLES_{G^*}\| = 190.9$
5	1	$\|ROLES_{G^*}\| = 94.1$
10	1	$\|ROLES_{G^*}\| = 90.9$
15	1	$\|ROLES_{G^*}\| = 91$

that a higher α follows a suboptimal path to a local minimum. This is shown in Figure 7 of Section 7.3 when no inferior solutions are accepted for data set *r2*.

In *RoleAnnealing*, decreasing the value of α will increase the probability of finding a better optimal solution but will require more iterations to converge. Increasing the value of α allows identification of good solutions sooner, although these solutions at higher α may not be close to the optimal solution. An argument for acceptance of near optimal solutions is in consideration of the constant change in enterprise environments. Instead of finding the exact optimal solution for any given moment in time, find a near optimal solution that will be effective and useful for larger periods of time.

7.2.2 Modifying Cost Constants

The *RoleAnnealing* framework is also capable of accepting, testing and comparing different cost models. We use a similiar setup as described in Section 7.1 and Section 7.2 with data set *s3* and modify c_1 and c_2 using Role Edge Cost (Equation 2) and c_1, c_2 and c_3 using Administration Graph Cost (Equation 4). In these tests, we show the average result from 10 runs. The experimental results using Equation 2 are shown in Table 2 and the experimental

Table 3: Varying $c1$, $c2$ and $c3$ parameters for cost$(G) = c_1\text{m}(U) + c_2|V_r| + c_3\text{m}(P)$ using data s3

$c1$	$c2$	$c3$	Result
1	1	1	$\|ROLES_{G^*}\| = 23.5$
10	1	1	$\|ROLES_{G^*}\| = 23.8$
1	10	1	$\|ROLES_{G^*}\| = 0$
1	1	10	$\|ROLES_{G^*}\| = 23.7$
1	10	10	$\|ROLES_{G^*}\| = 0$
10	1	10	$\|ROLES_{G^*}\| = 23.3$
10	10	1	$\|ROLES_{G^*}\| = 0$

results using Equation 4 are shown in Table 3

From the Role Edge Graph Cost results in Table 2, it can be seen that decreasing role relationship constant c_2 in relation to the role cost constant c_1 and increasing the role cost constant c_1 in relation to the role relationship constant c_2 can help reduce the number of roles, favouring the roles that are most cost effective.

In the Administration Graph Cost results as shown in Table 3, it can be seen that modifying the constants to weight different administration components can significantly effect the results. When the role cost (c_2) is too high, the cost of having a role is never justified. As a result, no roles are identified in the final result. Modifying c_1 and c_3 shifts the weighting between user cost and permission cost. While different roles can be found, a similar number of roles is identified.

In all automated role engineering techniques, human analysis is expected to be performed before implementation and deployment of any given infrastructure. Using *RoleAnnealing* to produce a graph solution that offers a selection of roles gives administrators choice from the best set of roles. Modifying cost parameters determines the number and quality of the roles that are returned for inspection.

7.3 Efficiency

Straight forward *SimulatedAnnealing* requires several orders of magnitude more time than *RoleAnnealing*. Instead of a direct comparison with *SimulatedAnnealing*, we compare *RoleAnnealing* with and without the following optimisations: **incremental computation**, **guided search**, use of **role bounds**, **probabilistic acceptance of inferior solutions** and **separation of create role and delete role operations**.

Each of these modified algorithms were tested using the set up described in Section 7.1 with the Role Edge Cost Model of Equation 2 on real data *r2*. Processing times for are in Figure 7.

The most obvious improvement is that of **incremental computation**. Instead of calculating graph cost and operating on the graph during each iteration, *RoleAnnealing* updates the cost components incrementally as well as calculates the change in cost at each iteration. Only when the change in cost is beneficial or if an inferior solution is to be probabilistically accepted, modification of the graph occurs.

At first, **guided search** does not significantly improve the quality of the result. However when the configuration stabilises closer to the original data cost, better, guided choices for roles are made and a final configuration with more desirable roles are identified faster.

As with guided search, using **role bounds** becomes more critical towards the later stages of optimisation, when there are similar beneficial roles exist in the configuration. Exact

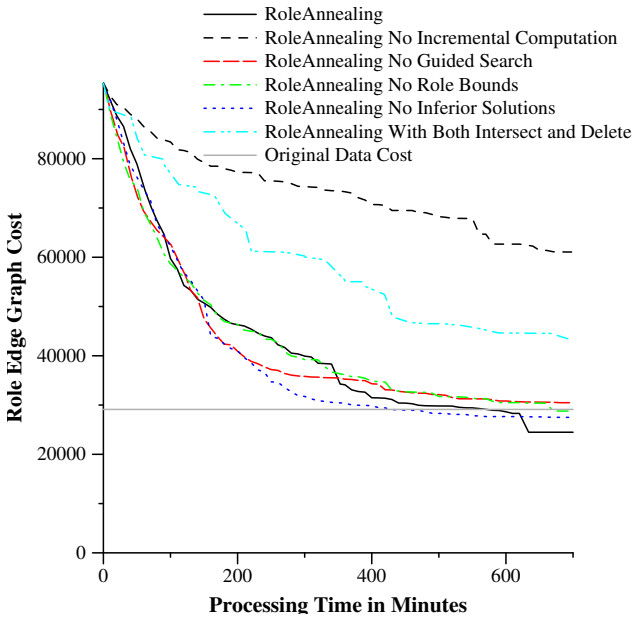


Figure 7: Processing times for RoleAnnealing, with and without optimisations using data $r2$

cost calculations of roles that are more connected with other roles become more costly, as a result, the benefit of using role bounds increases.

Accepting inferior solutions causes *RoleAnnealing* to initially have solutions of higher cost. However, doing so allows the chance of avoiding of potential local minima and final solutions of lower cost can be identified.

Finally, we **separated create role and delete role operations** in *RoleAnnealing*. In SimulatedAnnealing, an operation or modification of a graph involves either deleting or creating a role. However, analysis showed deleting a role during the iteration process was rarely beneficial. As a result, only create role operations are performed during solution search iterations in *RoleAnnealing* (Algorithm 2 lines 3–22) with delete operations to remove roles that were useful during the role creation process but were not beneficial to the overall RBAC configuration to later in the algorithm, after all beneficial roles have been created (Algorithm 2 lines 27–29). This created a significant speed up in *RoleAnnealing*.

RoleAnnealing finds better solutions faster with incremental computation, guided search heuristics, use of role bounds and probabilistic acceptance of inferior solutions.

7.4 Comparison with an Existing Approach

Using the Role Edge Cost Model, the results of *RoleAnnealing* can be compared with Iterative Optimisation for graph based role engineering by Zhang et al. [16]. The algorithm was run on the same data, using the same setup described in Section 7.1 and 7.2. Results of the Iterative Optimisation are in Table 4. While Iterative Optimisation can produce graph infrastructures of less administration cost than the same infrastructure without roles, it can be seen that *RoleAnnealing* is superior. The test on data *r2* was terminated after 29 hours

Table 4: Iterative Optimisation Results using Role Edge Graph Cost on synthetically generated data and real data with $c_1 = c_2 = c_3 = 1$.

Data	Graph Cost	Details
s1	cost(G^o) = 654 cost(G^*) = 1675	$\|ROLES_{G^*}\| = 384$ 74 minutes 19.13 seconds
s2	cost(G^o) = 2982 cost(G^*) = 6273	$\|ROLES_{G^*}\| = 1215$ 31 hours 44.38 minutes
s3	cost(G^o) = 5767 cost(G^*) = 8890	$\|ROLES_{G^*}\| = 1496$ 56 hours 40.39 minutes
r1	cost(G^o) = 1103 cost(G^*) = 325	$\|ROLES_{G^*}\| = 57$ 9.63 seconds
r2	cost(G^o) = 29112 cost(G^*) = 471754	$\|ROLES_{G^*}\| = 3739$ Results as at 29 hours
r3	cost(G^o) = 1630 cost(G^*) = 2179	$\|ROLES_{G^*}\| = 421$ 14 minutes 27.28 seconds

due to insigificant rate of improvement. In general, the iterative approach requires more processing time and produces suboptimal results. Reasons for some of these limitations are as follows.

The iterative approach does not compute change of cost, performing each merge then measuring the cost improvement. If there is no cost improvement, the graph is reverted. *RoleAnnealing* uses change in cost as well as role cost bounds to reduce the required processing time. Operations are only performed if they are accepted; less computation is required.

Iterative Optimisation merge operations are also exhaustive. Pairs of roles are continuously checked for possible merge operations until no more operations are possible. For example, *RoleAnnealing* required 1809 iterations to identify a graph with no further improvements for *r3*, using guided search heuristics to create roles that are most likely of administrative benefit. On the same data, Iterative Optimisation required 7360 iterations and trapped itself into a local minimum.

In Iterative Optimisation, a role cannot be deleted after it is created. The roles are merely rearranged. The creation of a role may also decrease the benefit of another role. While the operations are commutative, the effect on the cost model is not. This also results in termination at a local minimum. Delete operations are included in *RoleAnnealing* to ensure only the most beneficial roles remain.

Finally, *RoleAnnealing* also has pre-processing operations performed, merging roles and users if they are identical before the optimisation begins. This reduces the size of the graph and allows for more efficient modification during *RoleAnnealing*. Hierarchical relationships are maintained automatically in the graph if the role hierarchy RH is permitted. If RH is not, *RoleAnnealing* can also be constrained to produce non-hierarchical RBAC through constraining the set of allowable of operations.

7.5 Discussion and Analysis

This research analyses the cost components of the RBAC infrastructure and proposes effective cost models with exact role costs with upper and lower bounds on role cost using k-partite graph representation. In an ideal RBAC structure, there should be a minimal number of high quality roles. Where a high quality or an ideal role contains many permissions, can be used by many users and is as distinct as possible. A role with more permissions and more user assignments can save more direct user permission assignments. A distinct role reduces redundancy and potentially reflects a business entity. Existing studies do not fully

analyse these role properties.

Our cost models attempt to evaluate this, with the Role Edge Graph Cost focusing on costs of each RBAC component and the Administration Graph Cost focusing on maintenance cost. This newly proposed Administration Graph cost is important after deployment, when roles become relatively more stable and users and permissions within the structure are modified with increased frequency.

Using the cost models, exact costs and bounds of individual role costs can be computed. Using only the role bounds degrades role engineering results and using only exact role calculations effects performance. We show when bounds can be used in conjunction with costs through the introduction of upper and lower bounds on exact role cost. This improves performance while causing no degradation to results.

To test the cost models, we propose *RoleAnnealing*, a comprehensive cost model framework that is capable of evading local minima and producing a more suitable role set. It performs faster and produces higher quality results than an existing greedy graph based role engineering algorithm. While *RoleAnnealing* is an algorithm in the spirit of simulated annealing, it is significantly different. Direct application of simulated annealing would not scale for large role engineering problems. To address this, incremental computation as well as guided solution search techniques are used. To improve efficiency further, role bounds are used instead of exact role costs are used when they cause no degradation of results. We show through experimentation how each of our optimisations benefits *RoleAnnealing*.

8 Conclusions and Future Work

In this paper, we formalise the role engineering problem as a k-partite graph problem and propose two cost models that consider the structural components specific to the role engineering domain. We compute costs of individual roles, and their upper and lower bounds, and show when the bounds can be used for role evaluation to reduce computation.

The cost models are used as the basis for graph optimisation using *RoleAnnealing*, a statistical framework for optimising properties in large and complex systems based. Through experimentation, we show *RoleAnnealing* identifies RBAC configurations of high quality while maintaining data privacy. Testing on both real and synthetic data sets show *RoleAnnealing* to exhibit superior efficiency and effectiveness in comparison to straight forward simulated annealing and other existing approaches.

While the proposed cost models coupled with the *RoleAnnealing* framework offer an effective method for identifying the optimal graph configuration for role engineering, we are working on several different techniques to improve performance, such as reducing the complexity of graph representation and speeding up how operations can be performed. Future avenues for research include additional graph reductions before *RoleAnnealing* and testing on skewed data for the possibility of data partitioning. Ways to partition the data before *RoleAnnealing* is also an area for future research.

References

[1] A. Colantonio, R. D. Pietro, and A. Ocello. A cost-driven approach to role engineering. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2129–2136, New York, NY, USA, 2008. ACM.

[2] E. J. Coyne. Role engineering. In *RBAC '95: Proceedings of the first ACM Workshop on Role-based access control*, pages 4–5, New York, NY, USA, 1996. ACM Press.

[3] A. Ene, W. Horne, M. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *SACMAT'08: Proceedings of the thirteenth ACM symposium on Access control models and technologies*, Estes Park, Colorado, June 2008.

[4] E. B. Fernandez and J. C. Hawkins. Determining role rights from use cases. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 121–125, New York, NY, USA, 1997. ACM Press.

[5] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, Inc., 2003.

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

[7] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett. Observations on the role life-cycle in the context of enterprise security management. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 43–51, New York, NY, USA, 2002. ACM Press.

[8] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[9] M. Kuhlmann, D. Shohat, and G. Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 179–186, New York, NY, USA, 2003. ACM Press.

[10] H. Lu, J. Vaidya, and V. Atluri. Optimal boolean matrix decomposition: Application to role engineering. In *IEEE 24th International Conference on Data Engineering*, Cancun, Mexico, April 2008.

[11] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional RBAC roles. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 33–42, New York, NY, USA, 2002. ACM Press.

[12] H. Roeckle, G. Schimpf, and R. Weidinger. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, pages 103–110, New York, NY, USA, 2000. ACM Press.

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

[14] J. Vaidya, V. Alturi, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In *SACMAT '07: Proceedings of the twelth ACM symposium on Access control models and technologies*, Sophia Antipolis, France, 2007. ACM Press.

[15] S. Vanamali. Role Engineering: The Cornerstone of Role-Based Access Control. White Paper, July 2008.

[16] D. Zhang, K. Ramamohanarao, and T. Ebringer. Role engineering using graph optimisation. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 139–144, New York, NY, USA, 2007. ACM.

[17] D. Zhang, K. Ramamohanarao, T. Ebringer, and T. Yann. Permission set mining: Discovering practical and useful roles. In *ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference*, pages 247–256, Washington, DC, USA, 2008. IEEE Computer Society.

[18] D. Zhang, K. Ramamohanarao, S. Versteeg, and R. Zhang. Graph based strategies to role engineering. In *CSIIRW '10: Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research*, New York, NY, USA, 2010. ACM.

[19] D. Zhang, K. Ramamohanarao, and R. Zhang. Synthetic data generation for study of role engineering. http://www.cs.mu.oz.au/~zhangd/roledata, 2008.