Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2019,6
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

On Mutual Authorizations: Semantics, Integration Issues, and Performance

Gabriela Suntaxi, Aboubakr Achraf El Ghazi,
Klemens Böhm

2019

Fakultät für **Informatik**

On Mutual Authorizations: Semantics, Integration Issues, and Performance

Gabriela Suntaxi
Karlsruhe Institute of Technology
Karlsruhe, Germany

Aboubakr Achraf El Ghazi
Karlsruhe Institute of Technology
Karlsruhe, Germany

Klemens Böhm
Karlsruhe Institute of Technology
Karlsruhe, Germany

ABSTRACT

Studies in fields like psychology and sociology have revealed that reciprocity is a powerful determinant of human behavior. None of the existing access control models however captures this reciprocity phenomenon. In this paper, we introduce a new kind of grant, which we call *mutual*, to express authorizations that actually do this, i.e., users grant access to their resources only to users who allow them access to theirs. We define the syntax and semantics of mutual authorizations and show how this new grant can be included in the Role-Based Access Control model, i.e., extend RBAC with it. We use location-based services as an example to deploy *mutual* authorizations, and we propose two approaches to integrate them into these services. Next, we prove the soundness and analyze the complexity of both approaches. We also study how the ratio of *mutual* to *allow* and to *deny* authorizations affects the number of persons whose position a given person may read. These ratios may help in predicting whether users are willing to use mutual authorizations instead of deny or allow. Experiments confirm our complexity analysis and shed light on the performance of our approaches.

CCS CONCEPTS

• **Security and privacy → Access control**; **Authorization**.

KEYWORDS

access control models, reciprocity, mutual authorizations

1 INTRODUCTION

1.1 Motivation

Protecting information from unauthorized access is important to guarantee data confidentiality. Access policies state who may access which resource. Such policies are often implemented using authorizations of a specific access control model. Various models have been proposed; a prominent, mature one is Role-based Access Control (RBAC) [20].

Traditional access policies are designed to fulfill the needs of an organization or the individual needs of the users of a system. When, however, users decide who may access their resources, social factors and human behavior are significant. An important trend in psychology, economics, and sociology is reciprocity. Studies [6, 7, 24] have revealed that, although humans are self-interested, they often deviate from this attitude reciprocally. Reciprocity means that, in response to friendly actions, people are more cooperative. Reciprocity comes into play with access control when persons grant access to their resources to users that allow them the same. Think of a bike sharing system: It is more natural for a user to allow usage of his bike to others who allow him using theirs.

Example 1.1. Considering a bike sharing system, think of two users Anne and Bob who own bikes *bikeA* and *bikeB*, respectively. Anne wants to let Bob locate and use *bikeA* only if Bob allows her to locate and use *bikeB*. With conventional access control models, Anne can check whether Bob has allowed her this and then set her access privileges for him accordingly.

Example 1.2. Think of a population consisting of Anne, who again owns *bikeA*, and $10,000$ other individuals, each one owning another bike. Anne now wants to allow any other individual I to locate and use *bikeA* iff I allows her to locate and use his/her bike. With conventional models, Anne would have to repeat the procedure from Example 1.1 $10,000$ times. This also requires privileges to be public, which is unrealistic because they are sensitive information. Next, Anne would have to watch out for changes in the access policies of others continuously, e.g., Anne would have to revoke the access to any I as soon as I revokes her access to his bike. With the reciprocity feature envisioned in turn, all that Anne has to do is to specify one authorization.

Existing access control models do not explicitly support reciprocity. Although there exist various models [16, 20, 25], they all consider only two kinds of grants, *allow* and *deny*. But for reciprocity, a new kind of grant is needed. We call it *mutual* and authorizations that make use of it *mutual authorizations*.

Depending on the domain, access control models can protect different resources. Think of location-based services (LBS). Here, protecting the physical position of the users is crucial. They can be used to infer other personal information such as political affiliations, state of health or personal preferences [1]. Users often are unwilling to share their position with all users in the system. But it is natural to share it with users willing to share theirs.

Example 1.3. Continuing Example 1.2, one would expect Anne to allow using *bikeA* to her father unconditionally. However, this does not hold for all other individuals. Anne is more likely to allow others using her bike if she gets something in return. In the first case, *allow* authorizations are sufficient. In the second case however, *mutual* authorizations are needed.

Such reciprocal sharing also makes sense with health information or web-browsing histories (i.e., "I let you see my health record if I can see yours."), to give further examples.

This paper defines the syntax and semantics of mutual authorizations. One can then add these new authorizations to any existing access control model. Because of the importance of RBAC, we select this model to study and illustrate how this addition can look like conceptually. We use LBSs as a running example to deploy mutual authorizations. Specifically, we consider two types of location-dependent queries, k-nearest neighbor queries, and range queries.

1.2 Challenges

Defining the syntax and semantics of mutual authorizations is subject to several challenges. First, the entire semantic of authorization has to be redefined to incorporate a new grant to support reciprocity; however, especially for mutual authorizations, the semantic is not trivial. While implementations of mutuality exist, we are not aware of any integration in an access-control context.

Example 1.4. Consider a social network (SN). To befriend Bob, Anne has to send him a *friend request*. If Bob accepts, they are now friends. Such requests represent some concepts of mutuality. However, controlling access to resources is only done by means of *allow* and *deny*. For instance, Anne can only allow or deny her friends to see her photos.

For a general extension, one needs a general model that subsumes the existing ones. Since such a general model does not exist, we propose one. A second challenge is the complexity of the computations required to give access to a resource. In models with only *allow* and *deny*, to determine who can access the resources of a user u, it is enough to know all authorizations that u has specified. With mutual authorizations however, this is not the case. Namely, one also must know all authorizations assigned to u.

Regardless of the grants, the third challenge is the integration of services (LBS in our example) with access policies. This includes that (1) existing implementations are kept, and (2) solutions are efficient. The result of a given service in the presence of authorizations is not obvious a priori. Namely, it is necessary to verify the authorizations and to determine whether a given user should get access to the resources required for the service. This integration is needed because in order to guarantee data confidentiality.

Example 1.5. Think of a LBS provider (LBSP). A user has issued a location-dependent query that the LBSP now executes, knowing the position of the user. How can the LBSP take in authorizations to answer location-dependent queries? The result of that query is the set of persons who fulfill the query constraint, and whose positions the querying user may see. Alternative designs of this integration are conceivable. In particular, one may (1) execute the query first and then filter the result based on the authorizations, or (2) first compute the set of positions that the querying user may see, called *view of the user*, and then execute the query on this view. Deciding which alternative is better is not trivial.

One must also guarantee the soundness of such an integration, i.e., correctness and completeness. Intuitively, the integration is complete if all persons who satisfy the query constraints and whose information the querying user may see are part of the result. The integration is correct if the users in the result satisfy the query constraints, and the querying user is allowed to see their information. We formalize these properties in Section 2.6. Since there is more than one way to integrate *mutual* authorizations into a given service, the processing of a given query is unclear as well.

Lastly, it is not clear how the ratio of *mutual* to *allow* and *deny* authorizations affect the size of the views of the users. This ratio is important to predict whether users may be willing to use mutual authorizations instead of *deny* or *allow*.

Example 1.6. Consider three users, Anne, Bob, and Carol who have assigned *deny* authorizations to each other, i.e., the size of the view of each user is zero. Now suppose that (1) Anne replaces her *deny* authorization assigned to Bob with a *mutual* one, and (2) so does Carol with Anne. Although the share of mutual authorizations has increased, the size of the view of each user is still zero.

Example 1.7. Suppose that, given percentages of *allow* and *deny* authorizations of the entire population, the probability P_s that a user s, chosen at random, can see the positions of 10 percent of the population is 0.4. Assume further that, if 10 percent of *deny* authorizations are replaced by *mutual*, P_s increases to 0.8. Then it is likely that users start considering using *mutual* authorizations instead of *deny* because only 10 percent of *deny* authorizations need to be replaced by *mutual* ones to double P_s. However, the opposite might happen if 90 percent were needed.

Though we expect the number of persons whose information one may read to increase if *deny* authorizations are replaced by *mutual* etc., the extent is unclear. It is challenging to compute it, based on the share of *mutual* to *deny* and *allow* authorizations.

1.3 Contributions

We start by introducing a conceptual structure of authorizations, which we use to describe existing access control models. We then present a new kind of grant, called *mutual*. It allows to model reciprocal behavior. We define the syntax and semantics of *mutual* authorizations and show how to include them into RBAC conceptually, i.e., extend RBAC with it. As a use case for the deployment, we use LBSs, i.e., the two types of queries mentioned earlier. Their semantics is well-defined, in contrast to, say, similarity queries for health records. In a general setting, where resources have a different degree of sensitivity, deciding whether an exchange of information is fair may be difficult.

Example 1.8. Anne may not find it fair to open her health record if she has a stigmatizing disease in exchange for looking at the record of Bob who is in perfect health.

Covering the specifics of such other use cases is beyond the scope of this current article. However, Section 7 does feature a checklist of steps that one would have to carry out to this end.

To integrate *mutual* authorizations into LBSs, we propose two approaches, called *filtering-querying* and *querying-filtering*. We prove that both approaches are correct and complete. We also conduct complexity analyses of them, to determine under which conditions each approach performs better. The analysis shows that there is no clear winner because the outcome depends on the query constraints, the size of the data set and the view size of the querying user. But if one knows these parameters, our model can say which approach is better. Next, we analyze how the difference in the ratio of *mutual* to *deny* and *allow* authorizations affects the the view size of a user. Finally, we conduct experiments to validate our complexity analysis and to evaluate the performance of our approaches. Next to other insights, our complexity analysis is a good estimation of the behavior of our algorithms.

2 OUR AUTHORIZATION MODEL

Access control models have been studied widely, and several models have been proposed [16, 20]. Each model has its own syntax and semantics. The authorizations supported by these models allow assessing their expressiveness [2]. The following is a conceptual structure of authorizations which subsumes existing work.

2.1 A Conceptual Structure of Authorizations

We first introduce the elements of our structure.

- **Person.** Persons are individuals, together with attributes, e.g., name, role. $Attr_{Person}$ is the set of these attributes. Attributes are not atomic, especially, their value is a non-empty set of atomic values. From the point of view of an authorization, we see two types of persons: users and subjects.
- **User.** A *user* u is a person who assigns an authorization. U denotes the set of all users.
- **Subject.** A *subject* s is a person who receives an authorization. A single authorization can have multiple subjects. So-called person constraints specify the subjects of an authorization. A person constraint $Cons_{Person}$ has the syntax: $Cons_{Person} = Cons_{Person} \land Cons_{Person} \,|\, \mathcal{A}tt_p = value \,|\, \mathcal{A}tt_p \geq value \,|\, \mathcal{A}tt_p \leq value$, where $\mathcal{A}tt_p$ refers to an attribute in $Attr_{Person}$, and *value* refers to an atomic value. Since $\mathcal{A}tt_p$ is not atomic, for a given person s, the expression $\mathcal{A}tt_p$ $(=|\geq|\leq)$ *value* resolves to $\exists x \in s.\mathcal{A}tt_p, x \ (=|\geq|\leq) \ value$, where $s.\mathcal{A}tt_p$ denotes the set of atomic values of the attribute $\mathcal{A}tt_p$ for s. S is the set of all subjects. Given a person constraint $Cons_{Person}$, the induced set of subjects contains all persons $s \in S$ that fulfill $Cons_{Person}$. For instance, the set of subjects induced by $age < 20 \land income < 20k$ contains all persons $s \in S$ whose age is smaller than 20 and their income is less than 20k.
- **Resource.** A *resource res* is a physical or informational unit, together with attributes, e.g., type, owned by a user u for which u controls access. $Attr_{Resource}$ stands for the attributes of the resources. Attributes are not atomic, especially, their value is a non-empty set of atomic values. So-called resource constraints specify the resources of an authorization. A resource constraint $Cons_{Resource}$ has the syntax: $Cons_{Resource} = Cons_{Resource} \land Cons_{Resource} \mid \mathcal{A}tt_r = value \mid \mathcal{A}tt_r \geq value \mid \mathcal{A}tt_r \leq value$, where $\mathcal{A}tt_r$ is an attribute in $Attr_{Resource}$, and and *value* refers to an atomic value. Since $\mathcal{A}tt_r$ is not atomic, for a given resource *res*, the expression $\mathcal{A}tt_r$ $(=|\geq|\leq)$ *value* resolves to $\exists x \in res.\mathcal{A}tt_r, x \ (=|\geq|\leq) \ value$, where $res.\mathcal{A}tt_r$ denotes the set of atomic values of the attribute $\mathcal{A}tt_r$ for res. — In general, resources have to be classified by sensitivity levels. However, if not stated differently, this article assumes the same sensitivity level of all resources. Next, to ease presentation, we assume that all resources are of the same type. Dealing with resources of a different type, e.g., entire browsing history vs. history of today in that other example, is future work. On the other side, one can formulate sophisticated policies, which, say, discern between my position during the workday and during evenings, within our conceptual structure, using resource attributes.
- **Operation.** An *operation op* is an action that one can invoke on a resource, e.g., *read* or *write*. *Op* denotes the set of all operations.
- **Grant.** A *grant gr* is a right to execute an operation. *Gr* denotes the set of all grants.
- **Time.** A *time* t is an interval of time $[t_i, t_f]$ during which an authorization is valid, where t_i and t_f are the initial and final time, respectively. If an authorization is valid from the time it is entered in the system until the time it is deleted from the system, we write $[0, \infty]$.

Definition 2.1 (Authorization). Let a user $u \in U$, a person constraint $Cons_{Person}$, a resource constraint $Cons_{Resource}$, an operation $op \in Op$, a grant $gr \in Gr$ and a time t be given. An **authorization** A is a 6-element tuple $\langle\ u, Cons_{Person}, Cons_{Resource}, op, gr, [t_i, t_f]\ \rangle$. The authorization A indicates that user u assigns the grant gr to the subjects specified by $Cons_{Person}$ to invoke the operation op on the resources specified by $Cons_{Resource}$. A is valid during the interval of time $[t_i, t_f]$. — We call the set of all authorizations $\mathcal{A}$. Given an authorization A, we say that $user(A)$ assigns A to $subjects(A)$, and $subjects(A)$ receive authorization A from $user(A)$.

We assume that resources always have an attribute *owner*, i.e., $owner \in Res_{Att}$, and only the owners can write authorizations to control access to their resources. However, for brevity, (1) we omit the attribute *owner* from the set of resource attributes, and (2) given an authorization A assigned by a user u, we do not explicitly write the resource constraint *owner=u*, but we assume it to be present.

Example 2.2 illustrates how to express an authorization, Definition 2.1.

Example 2.2. Let us consider a user Ana who wants to *allow read* access to her file *File1* to all persons with the role of *Cashier*. Let us call this authorization A_{Ana}. A_{Ana} can be expressed using our conceptual structure as follows: $A_{Ana} = \langle\ Ana,\ role = Cashier,\ name = File1, read, allow, [0, \infty]\ \rangle$, where *role* is a person attribute in $Person_{Att}$, *name* is a resource attribute in $Attr_{Resource}$, *read* is an operation in *Op*, *allow* is a grant in *Gr*, and $[0, \infty]$ indicates that A_{Ana} is valid from the time it is entered in the system until it is deleted from the system.

We now introduce further notation: $subjects(A)$, $res(A)$, $op(A)$ and $grant(A)$ denote, respectively, the subjects induced by $Cons_{Person}$ of A, the resources induced by $Cons_{Resource}$ of A, the operation op of A, and the grant gr of A.

2.2 Existing access control models

We now use our conceptual structure of authorization, Definition 2.1, to describe existing access control models. The Role-Based Access Control Model, RBAC, is one of the most prominent models in the area [5, 9]. In RBAC, roles and authorizations regulate access to resources [20]. Each role is mapped to a set of authorizations and each subject in the system is assigned to a set of roles. Another popular access control model is discretionary access control. In this model the owner of a resource decides who can access the resource [17]. RBAC can be configured to support discretionary access control authorizations, [17]. In RBAC with discretionary access control, using our conceptual structure of authorization, the set of person attributes is $Attr_{Person} = \{role, name\}$. Depending on the organizations needs, it is possible to consider different resource attributes, $Attr_{Resource}$, and set of operations, Op. For instance, $Attr_{Resource} = \{type\}$, where *type* specifies the type of resource, e.g., printer or file, and $Op = \{read, write\}$. The set of grants in RBAC is $Gr = \{allow, deny\}$. RBAC does not specify an interval of time during which an authorization is valid, so the valid interval of time of an authorization is $[0, \infty]$. Example 2.2 illustrates an authorization in RBAC. A Task Role-Based Access Control Model, TRBAC, was proposed in [16]. The authors consider a task as a fundamental unit of a business activity and they emphasize that tasks and roles are different concepts. Although a role contains a set of tasks, a role can have mutually exclusive tasks which require access to different resources. In TRBAC, each role contains a set of tasks. Roles are assigned to subjects. Subjects can be assigned any task that belongs to one of their roles. Authorizations are modeled based on the tasks and roles of the subjects. With our conceptual structure of authorization, the set of person attributes in TRBAC is $Attr_{Person} = \{role, task\}$. Similar to RBAC, depending on the organizations needs, it is possible to consider different resource attributes, $Attr_{Resource}$, and set of operations, *Op*. The set of grants is $Gr = \{allow, deny\}$ and the valid interval of time of an authorization is $[0, \infty]$. Example 2.3 illustrates an authorization in TRBAC.

Example 2.3. Consider the authorization from Example 2.2. Assume that the role of *Cashier* has two tasks: *approve customer order* and *review customer order statistics*, denoted by *t1* and *t2*, respectively. Anne wants to *allow write* access to file *File1* to the persons responsible for *t1* and *allow read* access to those responsible for *t2.* These authorizations can be expressed using our conceptual structure as follows: $A_{Ana1} = \langle$*Ana, role=Cashier* $\wedge$ *task=t1, name=File1, write, allow,* $[0, \infty]\rangle$ and $A_{Ana2} = \langle$*Ana, role= Cashier* $\wedge\ task = t2,\ name = File1,\ read,\ allow, [\ 0, \infty]\ \rangle$, where *role* and *task* are person attributes in $Attr_{Person}$.

An Attribute-Based Access Control Model, ABAC, was introduced in [29]. ABAC considers that subjects and resources have a set of attributes which specify their characteristics. The authorizations in ABAC are defined based on these attributes and not only on the roles and tasks that persons perform in an organization like RBAC or TRBAC. Then the subjects and resources involve in an authorization are defined exactly like in our definition, Definition 2.1. However, ABAC does not formalize the operations and grants that can be supported by an authorization. The authors assume that an authorization is created with the purpose of allow access. Furthermore, an authorization in ABAC does not take into account the user who has assigned the authorization. This is because the authors have considered a single administrator entity who is on charge of assigning authorizations. Then in ABAC, using our conceptual structure of authorization, the set of *users* is a singleton set, $U = \{Administrator\}$. The sets of person attributes and resource attributes depend on the organization needs. For instance, $Attr_{Person} = \{name, age\}$ and $Attr_{Resource} = \{type\}$. The sets of operations and grants are $Op = \{access\}$ and $Gr = \{allow\}$, respectively. The interval of time during which an authorization is valid is $[0, \infty]$. Example 2.4 illustrates an authorization in ABAC.

Example 2.4. Let us consider a *system administrator* who wants to *allow access* to the movie called *ABC* to everyone *older than 16 years of age*. We call this authorization A_{admin}. A_{admin} is expressed using our conceptual structure as: $A_{admin} = \langle$ *Administrator, age* $\geq 16, name = ABC, access, allow, [0, \infty]\ \rangle$, where *age* is a person attribute in $Attr_{Person}$, *name* is a resource attribute in $Attr_{Resource}$, *access* is an operation in *Op*, *allow* is a grant in *Gr*, and $[0, \infty]$ is the valid time period of the authorization.

A Relation-Based Access Control Model, RelBAC, was presented in [8]. RelBAC was designed to cover the access control needs in social networks. In RelBAC, authorizations are modeled based on the relationships between users, e.g., friend or colleague. It is, in RelBAC, users are classified in groups, e.g., group of friends or colleagues, and the *allow* or *deny* authorizations are assigned to these groups. With respect to our conceptual structure of authorization, the set of person attributes in RelBAC is $Attr_{Person} = \{relationship\}$, the set of resource attributes is $Attr_{Resource} = \{type\}$, the set of operations is $Op = \{read, tag, publish, share, comment\}$ and the set of grants is $Gr = \{allow, deny\}$. Example 2.5 illustrates an authorization in RelBAC.

Example 2.5. Let us consider a user Anne who wants to *allow read* access to her *photos* to all her friends. This authorization, can be expressed using our conceptual structure as follows: $A_{Ana} = \langle$ *Ana, relationship=Ana's friend, type = photos, read, allow,* $[0, \infty]\ \rangle$, where *relationship* is a person attribute in $Attr_{Person}$, *type* is a resource attribute in $Attr_{Resource}$, *read* is an operation in *Op*, *allow* is a grant in *Gr*, and $[0, \infty]$ is the valid time period of the authorization.

A Fine-Grained Access Control Model for relational databases, FGAC, was proposed in [22]. The authorizations in FGAC allow specifying whether subjects can access to attributes of a table in a relational database or not. Authorizations are assigned to subjects based on their names or roles. In FGAC, using our conceptual structure of authorization, the sets of person attributes, resource attributes, operations and grants are $Attr_{Person} = \{name, role\}$, $Attr_{Resource} = \{Table\text{'s attributes}\}$, $Op = \{read, write, delete\}$ and $Gr = \{allow, deny\}$, respectively. The interval of time during which an authorization is valid is $[0, \infty]$. A Tuple based access control model, TBAC, was proposed in [25]. TBAC regulates the access to tuples in relational databases. In TBAC, the access to a tuple is granted to subjects based on their attributes, which is similar to the person attributes $Attr_{Person}$ defined in our conceptual structure of authorization. The set of resource attributes is $Attr_{Resource} = \{\text{Table's tuples}\}$, the set of operations is $Op = \{read, write, delete\}$, the set of grants is $Gr = \{allow, deny\}$, and the interval of time is $[0, \infty]$.

For the sake of simplicity, in the remaining of this paper, we omit from the notation of an authorization, the element time.

In the next section, Section 2.3, we introduce a new type of authorization, called *mutual authorization.*

2.3 Mutual Authorization – Syntax and Semantics

Existing access control models are based on the grants $Gr = \{deny, allow\}$. An *allow* authorization *A* uses *allow* and states that $user(A)$ authorizes $subjects(A)$ to invoke $op(A)$ on $res(A)$. A *deny* authorization uses *deny* and states that $user(A)$ forbids $subjects(A)$ to invoke $op(A)$ on $res(A)$.

We extend *Gr* with a new kind of grant, which we call *mutual.* Mutual grants capture the reciprocity phenomenon by means of *mutual authorizations.* Here, given an authorization *A*, the decision whether a person *s* is allowed to invoke $op(A)$ on $res(A)$ does not only depend on the authorization that s has received but also on the ones that s has assigned. Given two authorizations A and B, we use $res(A) = res(B)$ to indicate that the resources in both authorizations are of the same type.

Definition 2.6 (Mutual authorization). Given an authorization A, A is **mutual** if $grant(A) = mutual$. A mutual authorization A states that $user(A)$ allows invoking $op(A)$ on $res(A)$ to the subjects in $subjects(A)$ who have issued an authorization B to $user(A)$ to invoke $op(B)$ on $res(B)$ where the following expression evaluates to true: $(res(B) = res(A)) \wedge (grant(B) = allow \vee grant(B) = mutual) \wedge (op(B) = op(A))$.

This new authorization can be added to any model in line with our conceptual structure. We use the RBAC model to study/showcase how this addition can be done.

We assume that authorizations remain stable for some time. This rules out that users relax their authorizations for a moment, merely to spy out all others, i.e., changing a *deny* authorization to a *mutual* one and right after accessing the resource returning to *deny*. This aspect also affects existing access control model such as the Relation-based access control model. Doing away with it is future work.

For simplicity and to ease the presentation, in the remaining of this paper, we will consider RBAC and a setting with positions as the only resources type. Therefore, we restrict the elements of an authorization, as follows: (1) The set of attribute for persons is $Attr_{Person} = \{role, name\}$. Given a subject s, $r(s)$ is the set of roles of s. The set of subjects that receive a given authorization A is $subjects(A) = \{s \in S \mid name{=}s \vee (\exists r : role{=}r \wedge r \in r(s))\}$. (2) The resources are the positions of the users. We assume that each user $u \in U$ has one physical position, p_u. (3) The set of operations is $Op = \{read\}$. (4) The set of grants is $Gr = \{allow, mutual, deny\}$.

2.4 Conflict Resolution

Definition 2.7 (Authorization conflict). Given a set of authorizations $A_C \subseteq \mathcal{A}$, a subject $s \in S$ and a user $u \in U$, an **authorization conflict** exists with respect to u and s if s has received more than one authorization on the same resource with different grants assigned by u. An authorization conflict exists with respect to u and s if $\exists A, B \in A_C : (user(A) = user(B) = u) \wedge (s \in subjects(A) \cap subjects(B)) \wedge (res(A) = res(B)) \wedge (grant(A) \neq grant(B))$.

Example 2.8. Consider a person s with roles $r(s) = \{r1, r2\}$ and the authorizations $A = \langle u, role = r1, p_u, read, mutual\rangle$ and $B = \langle u, role = r2, p_u, read, deny\rangle$. Authorizations A and B are in conflict with respect to u and s. Namely, A assigns a *mutual* grant to s while B assigns a *deny* grant to s for reading the same resource p_u.

To solve authorization conflicts, i.e., decide which authorization prevails over the others when in conflict, several conflict resolution strategies have been proposed [11], such as *recency-overrides* where authorizations specified later take precedence over earlier ones. We in turn resort to a *deny-mutual* precedence strategy, where authorizations are assigned precedence based on their grants.

Definition 2.9 (Deny-mutual precedence strategy). A *deny-mutual precedence strategy* is a prioritization of the grants in Gr which states that a *deny* authorization precedes a *mutual* one and a *mutual* one precedes an *allow* one. We write $deny \gg mutual \gg allow$.

We select the precedence $deny \gg mutual \gg allow$ because assigning a higher precedence to *deny* eliminates the risk of possible leakage [11]. Next, we interpret an operation not granted explicitly as denied. — Intuitively, the process which solves conflicts is as follows. Given a set of authorizations $\mathcal{B}$ and a subject s, we take all authorizations $A \in \mathcal{B}$ where $s \in subjects(A)$ and group them by the user who has assigned them. We call the function that does this grouping *authorization-grouping function*. Each group contains authorizations where the user who has assigned them is the same, and there are no two or more sets containing authorizations assigned by the same user. Then, for each group of authorizations, we select the authorization with the highest precedence based on $deny \gg mutual \gg allow$. Given two authorizations A and B, we write $A \gg B$ to denote that $grant(A) \gg grant(B)$. We now formalize the notion of conflict resolution.

Definition 2.10 (Authorization-Grouping Function). An —**authorization-grouping function** $group : \mathcal{P}(\mathcal{A}) \times S \to \mathcal{P}(\mathcal{P}(\mathcal{A}))$ takes as input a set of authorizations $\mathcal{B} \subseteq \mathcal{A}$ and a subject $s \in S$ and outputs a set $\mathcal{C}$ of sets of authorizations such that:

(1) $\bigcup_{\mathcal{D} \in \mathcal{C}} \mathcal{D} = \{A \in \mathcal{B} \mid s \in subjects(A)\}$.
(2) $\forall \mathcal{D}_1, \mathcal{D}_2 \in \mathcal{C} : \mathcal{D}_1 \neq \mathcal{D}_2 \Rightarrow \mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$.
(3) $\forall \mathcal{D} \in \mathcal{C}, \forall A, B \in \mathcal{D} : user(A) = user(B) \wedge s \in subjects(A) \cap subjects(B)$.
(4) $\forall \mathcal{D}_1, \mathcal{D}_2 \in \mathcal{C}, \forall A \in \mathcal{D}_1, \forall B \in \mathcal{D}_2 : \mathcal{D}_1 \neq \mathcal{D}_2 \Rightarrow user(A) \neq user(B)$.

Example 2.11. Consider the authorizations A, B and the subject s from Example 2.8 and the authorizations $\mathcal{B} = \{A, B, C, D, E\}$, where $C{=}\langle v, name{=}s, p_v, read, allow\rangle$, $D{=}\langle u, name{=}v, p_u, read, mutual\rangle$ and $E{=}\langle v, role = r1, p_v, read, mutual\rangle$. The output of the authorization-grouping function $group(\mathcal{B}, s)$ is the set $\mathcal{C}{=}\{\{A, B\}, \{C, E\}\}$, where $\mathcal{C}$ contains all authorizations with subject s, and each set in $\mathcal{C}$ has authorizations assigned by the same user.

A *user-grant* tuple is a tuple $\langle t_{user}, t_{grant}\rangle$ where t_{user} is a user in U and t_{grant} is a grant in Gr. Our resolve-conflicts function does not consider the resources involved in the authorizations in conflict because we restrict our study to a specific resource, the physical positions of users.

Definition 2.12 (Resolve-Conflicts Function). A **resolve-conflicts function** $resC : \mathcal{P}(\mathcal{A}) \times S \to \mathcal{P}(U \times Gr)$ is a function that takes as input a set of authorizations $\mathcal{B} \subseteq \mathcal{A}$ and a subject $s \in S$ and outputs a set $\mathcal{C}$ of *user-grant* tuples. For each set of authorizations $\mathcal{B}_1 \subseteq \mathcal{B}$ with respect to subject s that are in conflict, $\mathcal{C}$ contains a user-grant tuple $\langle t_{user}, t_{grant}\rangle$, where t_{user} is the user who has assigned the authorizations in $\mathcal{B}_1$, and t_{grant} is the grant with the highest precedence in $\mathcal{B}_1$ that t_{user} has given to s. Given a tuple $t \in resC(\mathcal{B}, s)$, we use t_{user} and t_{grant} to refer to the first and second element of t, respectively. Formally,

$$resC(\mathcal{B}, s) = \begin{cases} \{\langle user(C), grant(C)\rangle \mid \\ C \in \mathcal{B}, \forall A \in \mathcal{B} : C \gg A\} & \text{if } \forall A, B \in \mathcal{B} : user(A) \\ & = user(B) \wedge s \in subjects(A) \cap subjects(B) \\ \bigcup_{\mathcal{B}_1 \in group(\mathcal{B}, s)} resC(\mathcal{B}_1, s) & \text{otherwise} \end{cases}$$

Example 2.13. Consider the authorizations A, B from Example 2.8. To resolve conflicts, we invoke the function $resC(\{A, B\}, s)$. Since

$user(A) = user(B) \wedge s \in subjects(A) \cap subjects(B)$ and $deny \gg mutual$, the resolve-conflicts function outputs the set $resC(\{A, B\}, s) = \{\langle u, deny \rangle\}$.

2.5 Authorized Access Request

The semantics of all authorization can be reduced to questions of the form: "Can s read the position of u?". We call this an *access request.* To answer it in the context of mutual authorizations, one must consider the authorizations that u has assigned to s and the ones that u has received from s.

Definition 2.14 (Access request). An **access request** $Req = \langle s, read, p_u \rangle$ is a tuple consisting of a person s, the operation *read* and a position of a person u, p_u. An access request indicates that s requests to *read* the physical position of u.

An access request $\langle s, read, p_u \rangle$ is authorized if, after resolving conflicts with respect to s, there exists (1) a tuple with the grant *allow* or (2) a tuple with the grant *mutual*, and after resolving conflicts with respect to u there is a tuple either with the grant *allow* or *mutual.* The following definition formalizes this notion.

Definition 2.15 (Authorized access request). Given the set of authorizations $\mathcal{A}$, an **access request** $\langle s, read, p_u \rangle$ is **authorized** if one of the following conditions is met:

(1) $\exists t \in resC(\mathcal{A}, s) : t_{user} = u \wedge t_{grant} = allow$
(2) $\exists t \in resC(\mathcal{A}, s), \exists e \in resC(\mathcal{A}, u) : t_{user} = u \wedge t_{grant} = mutual \wedge e_{user} = s \wedge (e_{grant} = allow \vee e_{grant} = mutual)$.

So far we have introduced our conceptual structure of authorizations and we have described existing access control models using this structure. We have also discussed how authorization conflicts are solved and the syntax and semantics of mutual authorizations. In addition, it is important to define the soundness principle that an algorithm in the context of LBSs and *mutual* authorizations should fulfill. In the next section, Section 2.6, we define this principle.

2.6 Soundness Criteria

An algorithm is sound if it is both *correct* and *complete* [23]. We now introduce two constraints, location and authorization constraints, which will be used to define the soundness and completeness of an algorithm in the context of LBSs and mutual authorizations.

Definition 2.16 (Location constraint). A **location constraint**, *LCons*, is a predicate on physical positions.

Let $dist(p_x, p_s)$ denote the distance between the physical positions of users x and s.

Example 2.17. Consider a distance d and the physical positions of two persons u and s, p_u and p_s, respectively, $dist(p_u, p_s) \leq d$ is a location constraint.

Definition 2.18 (Authorization constraint). Given two persons, u and s, an **authorization constraint** *ACons* is a predicate on a set of authorizations M that involve persons u and s.

Example 2.19. Let a set of authorizations M and two persons u and s be given. The access request $\langle s, read, p_u \rangle$ is an authorization constraint. If $\langle s, read, p_u \rangle$ is authorized, the predicate evaluates to *true*; otherwise it evaluates to *false.*

Definition 2.20 (Query). Given a set of persons $\mathcal{P}$, a **query** $Q(\mathcal{C})$ is a set of location and authorization constraints $\mathcal{C}$. Its output is the elements of $\mathcal{P}$ that fulfill $\mathcal{C}$. $Ans_{\mathcal{P}}(Q(\mathcal{C}))$ is the output of $Q(\mathcal{C})$.

A user algorithm is an algorithm which outputs a set of users who fulfill a set of constraints given as algorithm input.

Definition 2.21 (User algorithm). A **user algorithm** $\Pi : \mathscr{P}(U) \times \mathscr{P}(Cons) \rightarrow \mathscr{P}(U)$ is an algorithm that has as input a set of users $U_1 \in \mathscr{P}(U)$ and a set of constraints $\mathcal{C}$ and outputs a set of users U_2.

In the context of LBSs and mutual authorizations, a user algorithm Π computes a location and an authorization constraint on the physical positions of a given set of persons P. Based on these two constraints, we define the correctness and completeness of Π.

Definition 2.22 (Correctness in the context of LBSs and mutual authorizations). Let a user algorithm Π, a set $U_1 \in \mathscr{P}(U)$ and a set of constraints $\mathcal{C} = \{LCons, ACons\}$ be given. Π is **correct** with respect to U_1 and $\mathcal{C}$ if for all $u \in \Pi(U_1, \{LCons, ACons\})$, $u \in Ans_{U_1}(Q(LCons)) \wedge u \in Ans_{Ans_{U_1}(Q(LCons))}(Q(ACons))$.

In other words, correctness is given if for all users u in the output of Π, (1) u is a person in $U1$ that fulfills *LCons* and (2) u is a person in $Ans_{U_1}(Q(LCons))$ that fulfills *ACons*.

Definition 2.23 (Completeness in the context of LBSs and mutual authorizations). Let a user algorithm Π, a set $U_1 \in \mathscr{P}(U)$ and constraints $\mathcal{C} = \{LCons, ACons\}$ be given. Π is **complete** with respect to U_1 and $\mathcal{C}$ if for all persons u with $u \in Ans_{U_1}(Q(LCons)) \wedge u \in Ans_{Ans_{U_1}(Q(LCons))}(Q(ACons))$, $u \in \Pi(U_1, \{LCons, ACons\})$.

To illustrate completeness, think of a user algorithm Π that always outputs an empty set. Then Π fulfills the correctness principle. However, Π is not useful.

Definition 2.24 (Soundness in the context of LBSs and mutual authorizations). Let a user algorithm Π, a set $U_1 \in \mathscr{P}(U)$ and constraints $\mathcal{C} = \{LCons, ACons\}$ be given. Π is **sound** with respect to U_1 and $\mathcal{C}$ if (1) Π is correct with respect to U_1 and $\mathcal{C}$ and (2) Π is complete with respect to U_1 and $\mathcal{C}$.

3 INTEGRATING LBS WITH MUTUAL AUTHORIZATIONS

Before proceeding to describe our set of primitives and algorithms for integration LBSs with *mutual* authorizations, we present our algorithm for resolving conflicts.

3.1 Resolve Conflicts Algorithm

Authorization conflicts can be solved at design time, i.e., during the insertion of authorizations in a system, or at query time, i.e., when an access to a resource is required. Solving authorization conflicts at design time could require to modify the structure of an organization, see Example 3.1. For this reason, we consider authorization conflicts have to be solved at query time. In the next, we present our algorithm to solve authorization conflicts, see Algorithm 1. We use the left arrow "←" to indicate that the value on the right hand side is assigned to the term on the left hand side.

Example 3.1. Let us consider Example 2.8. Assume now that the authorization A has been inserted first in the system and now user u wants to insert authorization B. Based on the *deny-mutual* precedence strategy, authorization B has precedence over A. Then, during the insertion process, with respect to subject s, authorization B should be inserted and A should be deleted. However, this change will affect all users who have either $role1$ or $role2$. In this case it may be necessary to modify the organizational structure of the business in such a way that this authorization update will not affect to other users.

Given a set of authorizations $\mathcal{B} \subseteq \mathcal{A}$ a set of users $\mathcal{U}$ and a set of subjects $\mathcal{S}$, the *resolveConflicts* algorithm, Algorithm 1, resolves the existing authorization conflicts in the set of authorizations $\mathcal{B}$ with respect to each subject $s \in \mathcal{S}$ and the users in the set $\mathcal{U}$. Algorithm 1 starts by initializing an empty map *autMap* which will store pairs of keys and values. The key of the map is a pair consisting of a user and a subject, and the value of the map corresponds to the grant of an authorization. For each authorization A in $\mathcal{B}$, the algorithm verifies if (1) $user(A)$ is inside the set of users $\mathcal{U}$ and (2) the intersection between $subjects(A)$ and $\mathcal{S}$ is not an empty set. If the previous is true, for each $s \in subjects(A) \cap \mathcal{S}$, the algorithm checks if there is an entry in the map *autMap* with key $(user(A), s)$. If there is such an entry and the grant stored in this entry has lower precedence than the grant of the authorization A, the algorithm updates the value of the entry to $grant(A)$. Otherwise, the algorithm adds the entry with key $(user(A), s)$ and value $grant(A)$ to the map *autMap*.

Algorithm 1: resolveConflicts

```
Input   : Authorization Set $\mathcal{B}$, user Set $\mathcal{U}$, subject Set $\mathcal{S}$
Output: Map autMap
Initialize: $autMap\langle (user, subject), grant\rangle \longleftarrow empty\ map$;
foreach authorization $A$ in $\mathcal{B}$ do
    if $user(A) \in \mathcal{U} \wedge subjects(A) \cap \mathcal{S} \neq \emptyset$ then
        foreach $s \in subjects(A) \cap \mathcal{S}$ do
            if $autMap.containsKey((user(A), s))$ then
                if $autMap.get((user(A), s)) \ll grant(A)$ then
                    $autMap.put((user(A), s), grant(A))$;
            else
                $autMap.put((user(A), s), grant(A))$;
return autMap;
```

In the next, given as input to Algorithm 1 the sets $\mathcal{B}$, $\mathcal{U}$, $\mathcal{S}$, we prove that Algorithm 1 solve all authorization conflicts in the set $\mathcal{B}$ with respect to each subject $s \in \mathcal{S}$ and the set of users $\mathcal{U}$.

Lemma 3.2. *Given an authorization set $\mathcal{B} \subseteq \mathcal{A}$, a set of users $\mathcal{U}$, a set of subjects $\mathcal{S}$ and the subset of authorizations $N = \{A \in \mathcal{B} \mid user(A) \in \mathcal{U}, subjects(A) \cap \mathcal{S} \neq \emptyset\}$, for each $s \in \mathcal{S}$ and for each tuple $\langle t_{user}, t_{grant}\rangle \in resC(N, s)$ exists an entry $e = ((t_{user}, s), t_{grant})$ in resolveConflicts$(\mathcal{B}, \mathcal{U}, \mathcal{S})$.*

Proof. First, we will show that each entry in the map *autMap*, output by Algorithm 1, corresponds to one authorization in the set $N = \{A \in \mathcal{B} \mid user(A) \in \mathcal{U}, subjects(A) \cap \mathcal{S} \neq \emptyset\}$. In Line 2, Algorithm 1 considers all authorizations $A \in \mathcal{B}$, and in Line 3 the *if* condition evaluates if A satisfies the constraint $user(A) \in \mathcal{U}, subjects(A) \subseteq \mathcal{S}$. Entries are added in the map, Lines 7 and 9, using only authorizations that fulfill the *if* condition. Second, we will show that for each entry $e = ((u, s), grant)$ in *autMap*, where $u \in \mathcal{U}$ and $s \in \mathcal{S}$, the grant $authMap.get(u, s)$ is the one with the highest precedence with respect to u and s in the set $\mathcal{B}$. For each pair of elements (u, s), Algorithm 1 verifies if there is an entry with key (u, s) in *autMap*, Line 5. If such an entry exists, the grant of the entry is updated only if the grant of the entry has lower precedence than the grant of the authorization that is being evaluated, $grant(A)$, Lines 6-7. If there is not such an entry, Algorithm 1 creates a new entry in the map with key (u, s) and value $grant(A)$, Line 9. Once Algorithm 1 has evaluated all authorizations in $\mathcal{B}$, *autMap* will contain, for each pair of elements u, s, the grant with the highest precedence in the set $\mathcal{B}$ with respect to u and s. Then each entry $e = ((u, s), grant)$ corresponds to a tuple in $\langle t_{user} = u, t_{grant} = grant\rangle$ in $resC(N, s)$. □

3.2 Primitives and Algorithms for Mutual Authorizations

Depending on the services offered by a system, one may need different primitives. A primitive is a basic unit that performs a specific functionality, and that can be combined with other primitives. In the case of LBSs, to answer queries, we need to know which persons a given person has allowed reading his physical position. The two primitives *Primitive-Request* and *Primitive-View* are sufficient to this end, as we will show in Section 3.4.

- *Primitive-Request:* Given two persons u and s, may s read the physical position of u?
- *Primitive-View:* Given a person s, whose physical positions is s allowed to read? We call this set $View_s$, the view of s.

In the following, we present our algorithms, *Pr-Request* algorithm and *Pr-View* algorithm to implement the primitives *Primitive-Request* and *Primitive-View*, respectively. Since authorization conflicts can exist, both algorithms make use of the *resolveConflicts* algorithm, Algorithm 1.

Given two persons u and s, the *Pr-Request* algorithm, Algorithm 2, determines if person s can can read the physical position of person u. The *Pr-Request* algorithm, Algorithm 2, starts by initializing, among others, the set Set_{req} which contains the person who request the access and the set Set_{pp} which contains the person whose physical position is requested, Line 1. Next, the algorithm invokes the *resolveConflicts* algorithm on the sets $\mathcal{A}$, Set_{pp} and Set_{req}. The output of the *resolveConflicts* algorithm is stored in the map $ReceiveAut_s$. Since Set_{pp} and Set_{req} have one element each, $ReceiveAut_s$ contains only one entry e with key (u, s) and the value corresponds to the grant of the authorization with the highest precedence that u has assigned to s. If *e.getValue*() = $allow$, the algorithm returns $true$. If *e.getValue*() = $mutual$, the algorithm invokes the *resolveConflicts* algorithm on the sets $\mathcal{A}$, Set_{req} and Set_{pp}. The output of the *resolveConflicts* algorithm is stored in the map $ReceiveAut_u$, Line 7. $ReceiveAut_u$ contains only one entry t with key (s, u) and the value corresponds to the grant of the authorization with the highest precedence that s has assigned to u. If *t.getValue()* = $allow$ or *t.getValue()* = $mutual$, the algorithm returns $true$, which indicates that s is allowed to read the physical position of u, p_u. Otherwise s is not allowed.

Given a person s, the *Pr-View* algorithm, Algorithm 3, outputs the view of s. Algorithm 3 starts by initializing, among others, the sets $\mathcal{U}$

and $\mathcal{S}$. The algorithm assigns the set of all users U to the set $\mathcal{U}$ and the person s, given as input, to the set $\mathcal{S}$. During the initialization, the algorithm also invokes the *resolveConflicts* algorithm on the sets $\mathcal{A}$, $\mathcal{U}$ and $\mathcal{S}$, and stores the output in the map $ReceiveAut_s$. For each entry e in $ReceiveAut_s$, if *e.getValue*() = *allow*, the user that is part of the key of the entry e is stored in the set $Views_s$, Lines 3-4. Given an entry $e \in ReceiveAut_s$, we use the notation $e.getKey.User()$ to refer to the user that is part of the key of entry e. If *e.getValue*() = *mutual*, $e.getKey.User()$ is added to the set *MutualRA*, Lines 5-6. If the set *MutualRA* is not an empty set, then the algorithm invokes the *resolveConflicts* algorithm on the sets $\mathcal{A}$, $\mathcal{S}$ and *MutualRA*, and stores the output in the map $AuthMaps_s$. For each user $u \in MutualRA$, the algorithm verifies if $AuthMaps_s$ has an entry with key (s, u) with value equal to *allow* or *mutual*. If there is such an entry, the algorithm adds u to the set $Views_s$.

Algorithm 2: Pr-Request

```
Input  : Authorization Set $\mathcal{A}$, Access request $\langle s, read, p_u \rangle$
Output : Boolean resp
1  Initialize: $Set_{pp} \leftarrow \{u\}$, $Set_{req} \leftarrow \{s\}$,
     $ReceiveAut_s\langle(user, subject), grant\rangle \leftarrow$ empty map,
     $ReceiveAut_u\langle(user, subject), grant\rangle \leftarrow$ empty map;
2  $ReceiveAut_s \leftarrow resolveConflicts(\mathcal{A}, Set_{pp}, Set_{req})$;
3  foreach entry e in $ReceiveAut_s$ do
4      if e.getValue() = allow then
5          return true;
6      if e.getValue() = mutual then
7          $ReceiveAut_u \leftarrow resolveConflicts(\mathcal{A}, Set_{req}, Set_{pp})$;
8          foreach entry t in $ReceiveAut_u$ do
9              if t.getValue() = allow ∨ t.getValue() = mutual
                 then
10                 return true;
11 return false ;
```

In the next, we prove that given the authorization set $\mathcal{A}$ and a person s, the output of the *Pr-View* algorithm, Algorithm 3, contains all persons that s is allowed to read their physical positions and not more. We only present the proof of Algorithm 3. The proof of Algorithm 2 can be done in similar way as the proof as the proof of Algorithm 3.

Lemma 3.3. *Given an authorization set $\mathcal{A}$ and a person s,*

(1) For all persons $u \in$ Pr-View($\mathcal{A}, s$), Algorithm 3, s is authorized to read the physical position of u with respect to Definition 2.15.
(2) If $u \notin$ Pr-View($\mathcal{A}, s$), then s is not authorized to read the physical position of u with respect to Definition 2.15.

Proof. We will prove that for each user $u \in$ *Pr-View*($\mathcal{A}, s$), $\langle s, read, p_u \rangle$ is authorized, Definition 2.15, if either condition (1) or (2) is met:

(1) $\exists t \in resC(\mathcal{A}, s) : t_{user} = u \land t_{grant} = allow$.
(2) $\exists t \in resC(\mathcal{A}, s), \exists l \in resC(\mathcal{A}, u) : t_{user} = u \land t_{grant} = mutual \land l_{user} = s \land (l_{grant} = allow \lor l_{grant} = mutual)$

We have proven in Lemma 3.2 that for each tuple $\langle t_{user}, t_{grant} \rangle \in resC(\mathcal{A}, s)$ exists an entry $e = ((t_{user}, s), t_{grant})$ in the output *resolveConflicts*($\mathcal{A}, \{U\}, \{s\}$). Then in conditions (1) and (2), it is possible to replace a tuple $t = \langle t_{user}, t_{grant} \rangle \in resC(\mathcal{A}, s)$ with an entry $e = ((t_{user}, s), t_{grant}) \in$ *resolveConflicts*($\mathcal{A}, \{U\}, \{s\}$). First, a person u is added to *Pr-View*($\mathcal{A}, s$), if $t_{user} = u \land t_{grant} = allow$, Lines 4-5. It is condition (1) was evaluated. Second, if $t_{grant} = mutual$, then u is added to the set *MutualRA*, Lines 6-7. Then for each $u \in MutualRA$, u is added to *Pr-View*($\mathcal{A}, s$) if exists an entry f in *resolveConflicts*($\mathcal{A}, \{s\}, MutualRA$) such that $f = ((s, u), allow)$ or $f = ((s, u), mutual)$, Lines 12-13. It is condition (2) was evaluated. □

Algorithm 3: Pr-View

```
Input  : Authorization Set $\mathcal{A}$, person $s$
Output : Set $Views_s$
1  Initialize: $\mathcal{U} \leftarrow U$, $\mathcal{S} \leftarrow \{s\}$ , $Views_s \leftarrow \emptyset$,
     $ReceiveAut_s\langle(user, subject), grant\rangle \leftarrow$
     $resolveConflicts(\mathcal{A}, \mathcal{U}, \mathcal{S})$,
     $AA_s\langle(user, subject), grant\rangle \leftarrow$ empty map, $MutualRA \leftarrow \emptyset$;
2  foreach entry e in $ReceiveAut_s$ do
3      if e.getValue() = allow then
4          add e.getKey.User() to $Views_s$;
5      if e.getValue() = mutual then
6          add e.getKey.User() to MutualRA;
7  if MutualRA.size() ≠ 0 then
8      $AuthMaps_s \leftarrow resolveConflicts(\mathcal{A}, \mathcal{S}, MutualRA)$;
9      foreach u in MutualRA do
10         if $AuthMaps_s.get((s, u))$ =
             mutual ∨ $AuthMaps_s.get((s, u))$ = allow then
11             add u to $Views_s$;
12 return $Views_s$;
```

3.3 System Architecture

We consider a system architecture which contains a location-based service provider *LBSP*. The *LBSP* has : (1) a user database DB_U, which stores the position of each user u, p_u, and (2) an authorization database DB_A, which stores the authorizations $\mathcal{A}$. See Figure 1. We assume a database management system featuring R-tree indexing for spatial query processing on DB_U and B-tree indexing for authorizations queries on DB_A. In our prototype, we have implemented the LBS ourselves in Java, as well as access control, in the form of the primitives described in Section 3.2. Note that our focus is on the conceptual level; studying design alternatives regarding the architecture is future work.

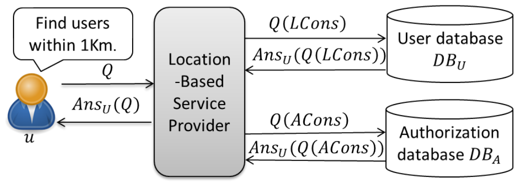

Fig. 1: System Architecture

The LBSP supports location-dependent queries. There are different types of such queries [12], and we focus on two of them here: k-nearest neighbor queries and range queries.

Definition 3.4 (Location-dependent query). Given a set of persons $\mathcal{P}$, a **location-dependent query** $Q(LCons)$ is one that takes a location constraint *LCons* and outputs the persons that fulfill it.

Definition 3.5 (k-Nearest Neighbor query). Given an integer k and a person s, a **k-nearest neighbor (kNN) query** $knn(k, s)$ is a location-dependent query where the location constraint $knn_{k,s}(p_u)$ is: $\forall M \subseteq U, \big(\forall x \in M, dis(p_x, p_s) < dis(p_u, p_s)\big) \Rightarrow |M| \leq k$, where U is the set of all users. In words, if the previous predicate evaluates to true for a physical position p_u, then the corresponding person u is in the result of the $knn(k, s)$ query; otherwise not.

Definition 3.6 (Range query). Given a distance d and a person s, a **range query** $range(d, s)$ is a location-dependent query with the constraint $range_{d,p_s}(p_x)$: $dist(p_x, p_s) \leq d$.

Definition 3.5 is a higher order logic. However, to facilitate proofs that our proposed approaches are sound, Section 3.4.3, we will use a recursive definition, Definition 3.7.

Definition 3.7 (k-Nearest Neighbor query recursive definition). A **k-nearest neighbor query** *knn* is a location- dependent query, where the location constraint consists of two elements (k, s), where k is an integer number and s is the persons who issues the query. The result $Ans(knn)$ of such a query is the set of users $u \in U$ such that $|Ans(knn)| = k \wedge \forall u \in Ans(knn), \forall v \in U \setminus Ans(knn)$: $dist(p_s, p_u) \leq dist(p_s, p_v)$.

Definition 3.8 (Bounded result-size query). Given a location-dependent query $Q(LCons)$, Q is a **bounded result-size query** if the location constraint (*LCons*) contains an explicit restriction on the number of elements of $Ans(Q)$. Otherwise, Q is a unbounded result-size query.

kNN and range queries are examples of bounded result-size and unbounded result-size queries, respectively. If Q is a bounded result-size query, after executing Q and filtering $Ans(Q)$ for users whose position s is authorized to see, the filtered query result may not fulfill the original constraint *LCons* any more.

Definition 3.9 (Authorizations received). Given the set of authorizations $\mathcal{A}$, the **authorizations that** s **has received** are all authorizations $A \in \mathcal{A}$ such that $s \in subjects(A)$.

Example 3.10. Consider a kNN query with constraint $knn = (2, s)$. Suppose that (1) the neighbors of s are u, v and w, and their distances to s are 1, 2 and 3 km, respectively, and (2) s has received two authorizations A, B where $user(A) = v$, $grant(A) = allow$, $user(B) = w$ and $grant(B) = allow$. The LBSP evaluates the kNN query and outputs $Ans(kNN) = \{u, v\}$. After filtering $Ans(kNN)$ based on the authorizations s has received, the result contains only $\{v\}$. This does not meet the constraint $knn = (2, s)$. The parameter should have been set to $k = 3$, to obtain $\{u, w\}$ after the filtering.

Example 3.11. Continuing with Example 3.10, suppose that s wants to find all persons within 2 km, i.e., $range(2km, s)$. The LBSP outputs $Ans(range) = \{u, v\}$. After filtering $Ans(range)$ with respect to the authorizations that s has received, the filtered result is $\{v\}$, similar to Example 3.10. This result fulfills the constraint $range(2km, s)$.

3.4 Integrating LBSs with Mutual Authorizations

To integrate *mutual* authorizations in the system architecture, we see two design alternatives, called *Querying-Filtering (QF)* and *Filtering-Querying (FQ)*. QF has the advantage that it can leverage existing LBS implementations. However, it has some limitations that could affect the performance, like the need to restart querying, as we will discuss. FQ does not have this need to restart.

3.4.1 Querying-Filtering Approach (QF). Given a location-dependent query $Q(LCons)$, the QF approach works as follows: (1) The LBSP executes the location-dependent query Q on the user database DB_U and returns $Ans(Q)$. (2) It filters $Ans(Q)$ for the persons whose position s may read. For the filtering, there are two options:

(a) Verify for each person $u \in Ans(Q)$ if $\langle s, read, p_u\rangle$ is authorized, i.e., execute *Primitive-Request.* If so, then u is added to the final answer.
(b) Compute *Primitive-View*, $View_s$, Algorithm 3. The final answer is the intersection of $View_s$ and $Ans(Q)$.

With Option (a), for each person in $Ans(Q)$, it is necessary to read all authorizations in $\mathcal{A}$ to solve authorization conflicts. With (b), although it is still needed to solve authorization conflicts, the elements of $\mathcal{A}$ will be read at most two times. The first time, the algorithm obtains the authorizations assigned to the querying person. At the second time, it verifies for the *mutual* authorizations whether the access request is authorized. In the following, we will focus on QF only in combination with (b).

Algorithms 4 and 5 show the details for implementing QF for kNN and range queries, resp. Our algorithms assume that the LBSP uses index structures to answer location-dependent queries, like B-tree or R-tree. We call the services used by LBSP to compute kNN and range queries, *computeKNN*(knn) and *computeRange*($range$), respectively, where knn and $range$ are the location constraints of the queries. In what follows, we describe both algorithms. Algorithm 4 receives as input the set of authorizations $\mathcal{A}$, the parameters of the kNN query k and a person s. The variables $View_s$, $temp_{all}$ and $temp_{visible}$ store the view of person s, the set of all persons that satisfy the kNN query, and the intersection of the sets $temp_{all}$ and $View_s$, respectively. Since kNN queries are bounded result-size queries, we need to evaluate if the final result satisfies the parameter k of the query. To do so, Algorithm 4 uses a *while* loop which encapsulates the entire process of querying and filtering. In Line 4, the algorithm invokes a function *estimateK*(k, k_{old}, s) which estimates an integer value, $k_{all} \geq k$ such that after computing the k_{all}-nearest neighbors of s and filtering the result based on $View_s$, the filtered result fulfills the constraint (k, s). Next, using the function *computeKNN*(k_{all}, s), the algorithm computes the k_{all}-nearest neighbors of s ordered by distance in ascending order and stores the result in $temp_{all}$. Then, for all $u \in temp_{all}$, if $u \in View_s$, u is added to $temp_{visible}$, while keeping the order by distance. If $temp_{visible}$ contains at least k elements, the algorithm outputs the first k. Otherwise, it assigns the current value of k_{all} to the parameter k_{old}, and the process of querying and filtering starts again. In each iteration, the function *estimateK* computes a new integer value $k_{all} > k_{old}$.

Algorithm 5 receives as input the set of authorizations $\mathcal{A}$, a distance parameter *dist* and a person s. Similar to Algorithm 4, the algorithm starts by invoking Algorithm 3 to compute the view of s with respect to the set of authorizations $\mathcal{A}$. The output of Algorithm 3 is stored in the set $View_s$. Then the algorithm computes the function *computeRange* with the distance *dist* and the person s. The result of this functions is stored in the set $temp_{all}$. Finally, Algorithm 5 filters the result by intersecting $temp_{all}$ and $View_s$, and outputs the final result set *Ans*.

Algorithm 4: Querying-Filtering kNN

```
Input  : Authorization Set $\mathcal{A}$, int $k$, person $s$
Output : Set $Ans$
1  Initialize: $View_s \leftarrow \emptyset$, $Ans \leftarrow \emptyset$, $temp_{all} \leftarrow [\,]$,
     $temp_{visible} \leftarrow [\,]$, $notEnough \leftarrow true$, $k_{all} \leftarrow 0$, $k_{old} \leftarrow 0$;
2  $View_s \leftarrow Pr\text{-}View(\mathcal{A}, s)$;
3  while $notEnough$ do
4      $k_{all} \leftarrow estimateK(k, k_{old}, s)$;
5      $temp_{all} \leftarrow computeKNN(k_{all}, s)$;
6      $temp_{visible} \leftarrow filter(temp_{all}, View_s)$;
7      if $temp_{visible}.size() \geq k$ then
8          $Ans \leftarrow$ select $topK(k, temp_{visible})$;
9          $notEnough \leftarrow false$;
10     else
11         $k_{old} \leftarrow k_{all}$;
12 return $Ans$;
```

Algorithm 5: Querying-Filtering Range

```
Input  : Authorization Set $\mathcal{A}$, double $dist$, person $s$
Output : Set $Ans$
1  Initialize: $View_s \leftarrow \emptyset$, $Ans \leftarrow \emptyset$, $temp_{all} \leftarrow \emptyset$;
2  $View_s \leftarrow Pr\text{-}View(\mathcal{A}, s)$;
3  $temp_{all} \leftarrow computeRange(dist, s)$;
4  $Ans \leftarrow temp_{all} \cap View_s$;
5  return $Ans$;
```

3.4.2 *Filtering-Querying Approach (FQ).*

Given a location-dependent query $Q(LCons)$, the FQ approach works as follows: (1) The LBSP invokes the *Pr-View* algorithm to determine the persons whose positions s is allowed to see, i.e., *Primitive-View.* (2) The LBSP executes Q over these persons and outputs a final result. Contrary to QF, since the evaluation of the location-dependent queries must take place on the filtered result, the LBSP cannot use the pre-computed materializations, i.e., *computeKNN*(*knn*) and *computeRange*(*range*). Then the LBSP needs new primitives to execute the supported queries. We have identified two primitives: (1) $computeD(p_s, p_u)$, which compute the distance between the physical positions of two given persons s and u, and (2) *sort by distance* $sortByD(M)$, where M is a list of tuples each of which consists of a person u and a distance d. To implement the first primitive, we use the well-known Haversine distance [19], and for the second one, we use the merge sort algorithm. See Algorithm 6 and Algorithm 7 for kNN and range queries, respectively.

Algorithm 6 receives as input the authorization set $\mathcal{A}$, the parameter of the kNN query k and a person s. To compute the view of s, the algorithm invokes Algorithm 3 on the set of authorizations $\mathcal{A}$ and the person s. The output of Algorithm 3 is stored in the set $View_s$. For each person $u \in View_s$, the algorithm computes the distance between u and s and adds the tuple $\langle u, d \rangle$ to the set *Dist*, Lines 4-5. Next, Algorithm 6 sorts by distance the set *Dist* and for each tuple $\langle u, d \rangle$ it stores u in the list $View_{order}$. $View_{order}$ stores all users $u \in View_s$ ordered by distance in ascending order. Finally, the algorithm selects the first k elements of the list $View_{order}$, Line 8.

Algorithm 7 receives as input the set of authorizations $\mathcal{A}$, a distance parameter *dist* and a person s. The steps 2-6 are the same as the ones of Algorithm 6. Then Algorithm 7 analyzes each tuple $t \in Dist$ and verifies if the distance stored in t, $t_{distance}$, is smaller or equal than the distance *dist*. If the previous is true, the person stored in tuple t, t_{pers}, is added to the final answer, Lines 7-10.

Algorithm 6: Filtering-Querying kNN

```
Input  : Authorization Set $\mathcal{A}$, int $k$, person $s$
Output : Set $Ans$
1  Initialize: $View_s \leftarrow \emptyset$, $Dist \leftarrow \emptyset$, $View_{order} \leftarrow [\,]$, $Ans \leftarrow \emptyset$;
2  $View_s \leftarrow Pr\text{-}View(\mathcal{A}, s)$;
3  foreach $person\ u\ in\ View_s$ do
4      double $d \leftarrow computeD(p_s, p_u)$;
5      add tuple $\langle u, d \rangle$ to $Dist$
6  $View_{order} \leftarrow sortByD(Dist)$;
7  $Ans \leftarrow$ select $topK(k, View_{order})$;
8  return $Ans$
```

Algorithm 7: Filtering-Querying Range

```
Input  : Authorization Set $\mathcal{A}$, double $dist$, person $s$
Output : Set $Ans$
1  Initialize: $View_s \leftarrow \emptyset$, $Dist \leftarrow \emptyset$, $neighbors$, $Ans \leftarrow \emptyset$;
2  $View_s \leftarrow Pr\text{-}View(\mathcal{A}, s)$;
3  foreach $person\ u\ in\ View_s$ do
4      double $d \leftarrow computeD(p_s, p_u)$;
5      add tuple $\langle u, d \rangle$ to $Dist$
6  foreach $tuple\ t\ in\ Dis$ do
7      if $t_{distance} \leq dist$ then
8          $Ans \leftarrow t_{pers}$;
9  return $Ans$
```

Advantages and Disadvantages of QF and FQ: With QF, the LBSP can make use of the available index structures in the user database. However, in the case of bounded result-size queries, the LBSP may need to restart the query if the filtered result does not satisfy the initial constraints, cf. Example 3.10. With FQ, bounded result-size queries do not require restarts, Example 3.11. However, the evaluation of location-dependent queries must take place on the filtered result. The LBSP cannot use the indexes structures of the user database to execute queries efficiently. Finally, with both approaches, the costs of updates, i.e., positions of persons and authorizations updates, only depend on the scalability and costs of updating the index structures used. The analysis of the impact of updates is beyond the scope of this paper. It also can be found elsewhere [15].

3.4.3 QF and FQ Are Sound. In this paper, we assume that the algorithms used to evaluate a given location-dependent query are correct and complete with respect to the location constraint *LCons*. This means that the integration of these algorithms into the context of *mutual* authorizations is correct and complete. The proofs that our integration of the algorithms to answer range queries into the context of *mutual* authorizations, Algorithms 5 and 7, are sound can be done in the same manner following the proofs of Lemmas 3.12 and 3.13. Therefore, we only present the proofs for the algorithms that support kNN queries.

LEMMA 3.12. *Let a set of authorizations $\mathcal{A}$ and a location constraint (k, s) of a kNN query be given, where k is an integer, and s is the query issuer. Algorithm 4, QF for kNN queries, is sound.*

PROOF. An algorithm is sound Definition 2.24, if it is correct and complete. Let *Ans* be the result output by Algorithm 4. We first prove that Algorithm 4 is correct with respect to Definition 2.22. We assume that the service used by Algorithm 4 to compute a given kNN query, *computeKNN*(k_{all}, s), where $k_{all} \geq k$, Line 5, is correct with respect to the location constraint (k_{all}, s). If a person u is in *Ans*, u is in $temp_{visible}$, Line 8. If u is in $temp_{visible}$, then u is $temp_{all}$ and u is in $Views$, Line 6. $Views$ is the output of *Pr-View*($\mathcal{A}, s$), so s is authorized to read the physical position of u, Lemma 3.3. Then, $u \in rst(ACons_{\mathcal{A},s}, U)$, where $ACons_{\mathcal{A},s}$ is an authorization constraint. Since u is in $temp_{all}$, u is in *computeKNN*(k_{all}, s). Since *computeKNN*(k_{all}, s) is correct, then u satisfies the location constraint (k_{all}, s) with respect to U. Furthermore, the size of $temp_{visible}$ is greater or equal than k, and *topK* selects the k first elements from $temp_{visible}$. Then $u \in rst((k, s), rst(ACons_{\mathcal{A},s}, U))$. Consequently, Algorithm 4 is correct. Now we prove that Algorithm 4 is complete with respect to Definition 2.23. Consider a person u with $u \in rst(ACons_{\mathcal{A},s}, U) \wedge u \in rst((k, s), rst(ACons_{\mathcal{A},s}, U))$. Because u satisfies the authorization constraint with respect to U, u is in $View_s$. Since $View_s = rst(ACons_{\mathcal{A},s}, U)$, Definition 2.18, u is in $rest((k, s), View_s)$ and $rest((k, s), View_s) \subseteq rest((k_{all}, s), U)$, then u is in $rest((k_{all}, s), U)$. We know that *computeKNN*(k_{all}, s) is complete. Then u is in $temp_{all}$ and u is in $temp_{visible}$. Because $u \in rst((k, s), View_s)$, then u is in *topK* and u is *Ans*. Hence, Algorithm 4 is complete; consequently, it is sound. □

LEMMA 3.13. *Let a set of authorizations $\mathcal{A}$ and a location constraint (k, s) of a kNN query be given, where k is an integer, and s is the query issuer. Algorithm 6, FQ for kNN queries, is sound.*

PROOF. An algorithm is sound, Definition 2.24, if it is correct and complete. Let *Ans* be the result output by Algorithm 6. We first prove that Algorithm 6 is correct with respect to Definition 2.22. If a person u is in *Ans*, u is in the list $View_{order}$, Line 8. Then there is a 2-element tuple $\langle u, d\rangle$ in $Dist$, which means u is in $View_s$. $View_s$ is the output of *Pr-View*($\mathcal{A}, s$). So s is authorized to read the position of u, Lemma 3.3. Then $u \in rst(ACons_{\mathcal{A},s}, U)$, where $ACons_{\mathcal{A},s}$ is an authorization constraint. Algorithm 6 uses the primitives *computeD*, *sortByD* and *topK* to compute the k-nearest neighbors of a given person s. We assume that the combination of these primitives to compute a given kNN query is correct with respect to the location constraint (k, s). Since these primitives compute the result using as input the set $View_s$, Line 3, $u \in rst((k, s), View_s)$, and $View_s = rst(ACons_{\mathcal{A},s}, U)$. Then Algorithm 6 is correct. We now prove that Algorithm 6 is complete with respect to Definition 2.23. Consider a person u with $u \in rst(ACons_{\mathcal{A},s}, U) \wedge u \in rst((k, s), rst(ACons_{\mathcal{A},s}, U))$. Because u satisfies the authorization constraint with respect to U, $u \in View_s$. Since $u \in View_s$, there is a 2-element tuple $\langle u, d\rangle$ in $Dist$. Then u is in $View_{order}$, Line 7. Because u satisfies the location constraint (k, s) with respect to the set $View_s$, u is in *topK*($k, View_{order}$). Then u is in *Ans*. Hence, Algorithm 6 is complete. Therefore, Algorithm 4 is sound. □

4 TIME COMPLEXITY ANALYSIS

A complexity analysis is helpful (1) to predict the behavior of FQ and QF, and (2) to facilitate meaningful comparisons. An average complexity analysis depends on the internal behavior of the database, which is (1) specific to the product, and (2) is not openly available. Furthermore, if there are changes in the system settings, the average analysis is void. So our complexity analysis targets at the worst case, which offers stronger guarantees.

4.1 Time Complexity Analysis of QF and FQ

To fulfill a given location constraint (k, s) of a kNN query, Algorithm 4 uses an estimation function *estimateK*, which estimates a value $k_{all} \geq k$ for a given k, such that after computing the k_{all}-nearest neighbors of s and filtering the result based on $View_s$, the filtered result satisfies the original constraint (k, s). Let k_{real} be the value of k_{all} Algorithm 4 uses to compute the final output, i.e., k_{real} is equal to the value k_{all} of the last run of Algorithm 4. Let further be $\delta = k_{real} - k$.

For the analysis of Algorithm 4, we assume that *estimateK* computes the value k_{real} in the first run, i.e., no restarts are needed. We discuss this assumption later in Section 4.2.

LEMMA 4.1. *Let the number of persons n, a kNN query knn=(k,s), the view size of the query issuer, s, $|View_s|$, and a set of authorizations $\mathcal{A}$, be given. The time complexity of QF with no restarts is*

$$T_C = \mathcal{O}(n + (k + \delta) \cdot |View_s|) + \mathcal{O}(\mathcal{A}) \quad (1)$$

PROOF. The following steps are required to compute a given kNN query with the *querying-filtering* approach, with no restarts:

- *Step*$_1$ computes the view $View_s$ of the query issuer s. We use $\mathcal{O}(\mathcal{A})$ to denote the complexity of this step.
- *Step*$_2$ searches the $(k + \delta)$-nearest neighbors in the user database. The complexity of a kNN query using R-tree indexes is $\mathcal{O}(n)$ [14]. We validated through initial experiments that this complexity applies to the praxis.
- *Step*$_3$ filters the result by checking for each person returned in *Step*$_2$ if the person is in the view $View_s$. The complexity of this step is $\mathcal{O}((k + \delta) \cdot |View_s|)$.

Consequently, the time complexity of executing a kNN query with the *querying-filtering* approach is $T_C = \mathcal{O}(n+(k+\delta)\cdot|View_s|)+\mathcal{O}(\mathcal{A})$. □

LEMMA 4.2. *Let the number of persons n, a kNN query knn = (k, s), the size of the view of the query issuer s, $|View_s|$, and the set of authorizations $\mathcal{A}$ be given. The time complexity of FQ is*

$$T_C = \mathcal{O}\big(|View_s| \cdot \log(n) + |View_s| + |View_s| \cdot \log(|View_s|) + k\big) + \mathcal{O}(\mathcal{A}) \quad (2)$$

PROOF. The following steps are required to compute a given kNN query with the *filtering-querying* approach:

Step$_1$ computes the view, $View_s$ of the query issuer s. We use $\mathcal{O}(\mathcal{A})$ to denote the complexity of this step.

Step$_2$ looks up in the user database to obtain the physical position of each person in the view $View_s$. This has a complexity of $\mathcal{O}(|View_s| \cdot \log(n))$.

Step$_3$ computes the distance between the querying user and each of the persons in the view $View_s$. The complexity of this step is $\mathcal{O}(|View_s|)$.

Step$_4$ orders the persons in the view $View_s$ by distance to the querying user s in ascending order. The order is done using the merge sort algorithm. The complexity of this step is $\mathcal{O}(|View_s| \cdot \log(|View_s|))$.

Step$_5$ selects the k first persons. This has a complexity of $\mathcal{O}(k)$.

Consequently, the time complexity of executing a kNN query with the *filtering-querying* approach is T_C=$\mathcal{O}(|View_s| \cdot \log(n) + |View_s| + |View_s| \cdot \log(|View_s|) + k) + \mathcal{O}(\mathcal{A})$. □

We note that, since $\forall x > 0, n > 0 : x > x \cdot log(n)$, Equation (2) can be further simplified to $T_C = \mathcal{O}\big(|View_s| + k\big) + \mathcal{O}(\mathcal{A})$. However, to allow a more accurate comparison of both approaches in the next section, Section 4.2, we do not simplify it.

4.2 Comparison of the QF and FQ Approaches

To decide which approach is better to answer a given query, one needs to compare the complexity of both approaches, QF and FQ, and find their intersection points:

$$\mathcal{O}(n + (k + \delta) \cdot |View_s|) + \mathcal{O}(\mathcal{A}) = \mathcal{O}\big(|View_s| \cdot \log(n) + |View_s| + |View_s| \cdot \log(|View_s|) + k\big) + \mathcal{O}(\mathcal{A}) \quad (3)$$

Solving (3) for $|View_s|$ yields (4). For given values of n, k and δ, (4) is the size of the view so that the time complexity in the worst case is equal. We refer to this size of the view as $View_{equal}$. $\mathcal{W}$ in (4) is the Lambert-W function [4].

$$View_{equal}(n, k, \delta) = \frac{n \cdot \ln(2) - k \cdot \ln(2)}{\mathcal{W}(2^{1-k-\delta} \cdot n \cdot (n-k) \cdot \ln(2))} \quad (4)$$

We now analyze Eq. (4) with the best case scenario for QF, which is the one where the nearest neighbors of s are the persons whose positions s is allowed to read, i.e., δ=0. Equation (4) depends on the parameters: n, k and δ. To further simplify it, similarly to other approaches [10, 28], we set the parameters k of the kNN query to 20, and δ=0. Then $View_{equal}$ only depends on the number of persons n.

$$View_{equal}(n, 20, 0) = \frac{n \cdot \ln(2) - k \cdot \ln(2)}{\mathcal{W}(2^{-19} \cdot n \cdot (n-20) \cdot \ln(2))} \quad (5)$$

Figure 2 plots the QF and FQ approaches, for $k = 20$ and $\delta = 0$. The x-axis is the number of persons n, the y-axis the size of the view $|View_s|$ of the query issuer s and the z-axis the time complexity T_C. Figure 2 shows the intersection points of both approaches. Given an intersection point and its corresponding number of persons n, Equation (5) yields the size of its view. We conclude that, for a given n, if $|View_s| < View_{equal}$, the time complexity of FQ is smaller than that of QF, and vice versa. In Table 1, using Equation (5), we list the intersection points of QF and FQ, for different numbers of persons n. For instance, if $n = 2000$, $View_{equal} \approx 1014.31$. Then, for $n = 2000$, if the size of the view of the query issuer is smaller than approximately 1014.31, FQ performs better than QF.

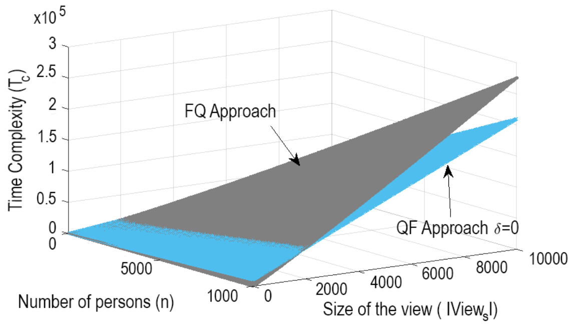

Fig. 2: Complexity of the QF and FQ Approaches – knn Query ($k = 20, \delta = 0$)

Table 1: Values of n and m for which the T_C of *QF* and *FQ* are the same ($k = 20, \delta = 0$)

n	$View_{equal} \approx$
2000	1014.3096
4000	1231.4594
10000	1920.9585
20000	2935.2272
40000	4709.5727
100000	9267.9800
317080	23032.341
3000000	152046.4307

So far, the plot in Figure 2 and the values in Table 1 correspond to the best case scenario for QF, i.e., $\delta = 0$. We now explain why a focus on this case is sufficient.

Let us consider real scenarios such as online social networks like Orkut and LiveJournal. The number of connections that a person s has in these networks is the number of persons that have declared to have a relationship with s, e.g., friend, colleague. This translates to our authorization model as the size of the view of s. In [13], the authors found that considering about 3 million nodes, the average number of connections of a person in Orkut and LiveJournal is 223.99 and 520.04, respectively. In DBLP with 317080 nodes, the average number of connections is 64.98 [13]. This suggests that the size of the view of a given person increases monotonically with the number of persons. Analogously, Table 1 reveals that $View_{equal}$

grows monotonically with the number of persons n. We can also observe that, if n=2000, $View_{equal}$ is already greater than the average number of connections for 3 million persons in real scenarios. For n equal to 3 million, $View_{equal}$=152046. This indicates that the size of the view of a given person in real scenarios is smaller than $View_{equal}$ for a given n. This implies that FQ performs better than QF even in the best case scenario of QF. So we do not dwell into the behavior of QF with restarts.

The analysis of the approaches for range queries can be done in the same way. Range queries are unbounded result-size queries. QF for range queries does not need any restart. In the analysis of QF for kNN queries, we did not consider restarts because we assumed *estimateK* to compute k_{real} in the first run. Therefore, the skeleton structure of the complexity analysis is the same for both type of queries. For these reasons, we omit this part.

5 THE SIZE OF THE VIEW OF A PERSON

In this section, we study the impact of *mutual* authorizations on the number of persons whose position a given person s is allowed to read, i.e., $View_s$. This is important not only from the point of view of the LBSP but also from the user perspective. A user may want to know how the use of *mutual* authorizations compared to the use of *deny* or *allow* in the population affects the number of persons whose position he is allowed to read.

To study the impact of *mutual* authorizations on the view of a person, we derive the probability $P(|View_s|=N)$ that a person s chosen at random can see the physical position of a specific number of persons N, given the share of *mutual* to *deny* and *allow* authorizations in the entire population. To do so, we look at the authorizations after solving all authorization conflicts. This allows us to represent the authorizations and persons as a so-called authorization graph G. Since every person of a pair assigns an authorization to the other one, at least implicitly, G is a complete digraph.

Definition 5.1 (Authorization Graph). Given a set of authorizations $\mathcal{A}$, an **authorization graph** $G = (V, E)$ is a complete digraph with labeled directed edges, as follows: The vertices represent the persons. A directed edge with label $grant$ between u and v indicates that there exists a user-grant tuple $\langle u, grant\rangle \in resC(\mathcal{A}, v)$, where $grant \in Gr$.

The number of incoming edges of any node is $|E_{in}| = |V| - 1$.

To compute $P(|View_s| = N)$, where $N \leq |E_{in}|$, we need a concrete distribution of *allow*, *deny* and *mutual* authorizations. The arguments behind our analysis hold for any distribution. For the sake of simplicity, we now assume a uniform distribution of *allow*, *deny* and *mutual* authorizations, where the numbers of these authorizations are parameters of the distribution, i.e., the probability that a random chosen authorization has an *allow* grant is given by the number of *allow* authorizations over the total number of authorizations, for *mutual* and *deny* accordingly. To not restrict ourselves to a specific scenario, we assume that the only information available is: ($i1$) the number of nodes in the authorization graph G, $|V|$, ($i2$) the number of *deny* edges in G, $|d|$, ($i3$) the number of *mutual* edges in G, $|m|$, and ($i4$) the number of *allow* edges in G, $|a|$, such that $|E| = |a| + |m| + |d|$.

Example 5.2 illustrates how one can compute the probability that the size of the view of a random person s is one, i.e., $P(|View_s| = 1)$.

Example 5.2. Think of a node s with two incoming and two outgoing edges. s has a view of size 1 if s has either C_1, C_2, C_3 or C_4, where:

(C_1) One *allow* and one *deny* incoming edge.
(C_2) One *allow* incoming edge, one *mutual* incoming edge and one *deny* outgoing edge pointing to the node the *mutual* incoming edge originates from.
(C_3) One *mutual* incoming edge, one *deny* incoming edge, and one *mutual* or *allow* outgoing edge pointing to the node the *mutual* incoming edge originates from.
(C_4) Two *mutual* incoming edges, one *mutual* or *allow* outgoing edge, and one *deny* outgoing edge such that the outgoing edges point to the nodes the *mutual* incoming edges originate from.

Then, to compute $P(|View_s| = 1)$, it suffices to sum up the individual probabilities of all the above cases.

Example 5.2 shows that the distinction between *allow* and *mutual* outgoing edges is not relevant in cases C_2, C_3 and C_4. Then, to simplify the computation of $P(|View_s| = N)$ in these cases, we treat *allow* and *mutual* outgoing edges as belonging to one group. Lemma 5.3 proves the correctness of this simplification.

Lemma 5.3. *Let (1) a value $r \in \mathbb{N}$, (2) a multiset $X = X_1 \cup X_2 \cup X_3$, and (3) a multiset $Y = Y_1 \cup X_3$, where X_1, X_2 and X_3, and Y_1 and X_3 are pairwise disjoint multisets, their corresponding underlying set is a unit set, and $|Y_1| = |X_1 \cup |X_2|$, be given. Let A_r be the event of choosing r elements from the multiset X such that the chosen elements belong either to the submultiset X_1 or to X_2, and let B_r be the event of choosing r elements from the multiset Y such that the chosen elements belong to the submultiset Y_1. Then*

$$P(A_r) = P(B_r) = \frac{\binom{|Y_1|}{r}}{\binom{|Y|}{r}} \tag{6}$$

Proof. By induction on r. We first prove that Equation (6) is true for $r = 1$. Then we assume that Equation (6) holds for r and prove that it also holds for $r + 1$. Let us start by proving that Equation (6) is true for $r = 1$.

Let a be the chosen element. The probability of choosing 1 element, from a multiset of $|X|$ elements such that the chosen element belongs either to the multisets X_1 or X_2 is $P(a \in X_1)+P(a \in X_2)$. There are $\binom{|X|}{1}$ possible ways to choose 1 element from a multiset of $|X|$ elements. So:

$$P(A_1) = P(B_1)$$
$$P(a \in X_1) + P(a \in X_2) = P(a \in Y_1)$$
$$\frac{\binom{|X_1|}{1}}{\binom{|X|}{1}} + \frac{\binom{|X_2|}{1}}{\binom{|X|}{1}} = \frac{\binom{|Y_1|}{1}}{\binom{|Y|}{1}}$$
$$\frac{\binom{|X_1|}{1}}{\binom{|X|}{1}} + \frac{\binom{|X_2|}{1}}{\binom{|X|}{1}} = \frac{\binom{|X_1|+|X_2|}{1}}{\binom{|X|}{1}}$$
$$\frac{\frac{|X_1|!}{(|X_1|-1)!}}{\cancel{\frac{|X|!}{(|X|-1)!}}} + \frac{\frac{|X_2|!}{(|X_2|-1)!}}{\cancel{\frac{|X|!}{(|X|-1)!}}} = \frac{\frac{(|X_1|+|X_2|)!}{(|X_1|+|X_2|-1)!}}{\cancel{\frac{|X|!}{(|X|-1)!}}}$$
$$\frac{|X_1| \cdot \cancel{(|X_1| - 1)!}}{\cancel{(|X_1| - 1)!}}$$
$$+ \frac{|X_2| \cdot \cancel{(|X_2| - 1)!}}{\cancel{(|X_2| - 1)!}} = \frac{(|X_1| + |X_2|) \cdot \cancel{(|X_1| + |X_2| - 1)!}}{\cancel{(|X_1| + |X_2| - 1)!}}$$
$$|X_1| + |X_2| = |X_1| + |X_2|$$

Then the claim is valid for $r = 1$. We still need to prove that Equation (6) is true for $r + 1$ assuming it is true for r.

Let C be the event of choosing 1 element that belongs either to the submultiset X_1 or X_2 given event A_r. The probability of choosing $r + 1$ elements from $|X|$ elements such that the chosen elements belong either to the submultiset X_1 or X_2 is equal to the probability that events A_r and C occur, $P(A_r \cap C) = P(A_r) \cdot P(C|A_r)$.Because of the hypothesis, we have:

$$P(A_r) = \frac{\binom{|Y_1|}{1}}{\binom{|Y|}{r}}$$

Let us discuss now the probability $P(C|A_r)$. In event A_r, r elements have been selected already from the total number of elements $|X|$. The remaining number of elements is $|X| - r$. Let $x1$ and $x2$ be the number of elements selected from the submultisets X_1 and X_2, respectively, where $r = x1 + x2$. The probability that event C happens given event A_r is $P(C|A_r) = P(b \in X_1|A_r) + P(b \in X_2|A_r)$, where b is the chosen element. Given event A_r, the number of possible ways of choosing one element from the total remaining elements $|X| - r$ is $\binom{|X|-r}{1}$. The number of possible ways of choosing one element from the $|X_1| - x1$ remaining elements of the submultiset X_1, is $\binom{|X_1|-x1}{1}$. The number of possible ways of choosing one element from the $|X_2| - x2$ remaining elements of the submultiset X_2 is $\binom{|X_2|-x2}{1}$. So:

$$P(A_{r+1}) = P(B_{r+1})$$
$$P(A_r) \cdot P(C|A_r) = P(B_{r+1})$$
$$\frac{\binom{|Y_1|}{1}}{\binom{|Y|}{r}} \cdot \left(\frac{\binom{|X_1|-x1}{1}}{\binom{|X|-r}{1}} + \frac{\binom{|X_2|-x2}{1}}{\binom{|X|-r}{1}} \right) = \frac{\binom{|Y_1|}{r+1}}{\binom{|Y|}{r+1}}$$
$$\frac{\binom{|X_1|+|X_2|}{r}}{\binom{|X|}{r}} \cdot \left(\frac{\binom{|X_1|-x1}{1}}{\binom{|X|-r}{1}} + \frac{\binom{|X_2|-x2}{1}}{\binom{|X|-r}{1}} \right) = \frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}}$$
$$\frac{\frac{(|X_1|+|X_2|)!}{(|X_1|+|X_2|-r)! \cdot \cancel{r}!}}{\frac{|X|!}{(|X|-r)! \cdot \cancel{r}!}}$$
$$\cdot \left(\frac{\frac{(|X_1|-x1)!}{(|X_1|-x1-1)!}}{\frac{(|X|-r)!}{(|X|-r-1)!}} + \frac{\frac{(|X_2|-x2)!}{(|X_2|-x2-1)!}}{\frac{(|X|-r)!}{(|X|-r-1)!}} \right) = \frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}}$$
$$\frac{\frac{(|X_1|+|X_2|)!}{(|X_1|+|X_2|-r)!}}{\frac{|X|!}{(|X|-r)! \cdot}}$$
$$\cdot \left(\frac{(|X_1| - x1)}{(|X| - r)} + \frac{(|X_2| - x2)}{(|X| - r)} \right) = \frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}}$$
$$\frac{\frac{(|X_1|+|X_2|)!}{\cancel{(|X_1|+|X_2|-r)} \cdot (|X_1|+|X_2|-r-1)!}}{\frac{|X|!}{\cancel{(|X|-r)} \cdot (|X|-r-1)!}}$$
$$\cdot \frac{\cancel{(|X_1| + |X_2| - r)}}{\cancel{(|X| - r)}} = \frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}}$$
$$\frac{\frac{(|X_1|+|X_2|)!}{(|X_1|+|X_2|-r-1)!}}{\frac{|X|!}{(|X|-r)!}} \cdot \frac{(r+1)!}{(r+1)!} = \frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}}$$
$$\frac{\frac{(|X_1|+|X_2|)!}{(|X_1|+|X_2|-(r+1))! \cdot (r+1)!}}{\frac{|X|!}{(|X|-(r+1))! \cdot (r+1)!}} = \frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}}$$
$$\frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}} = \frac{\binom{|X_1|+|X_2|}{r+1}}{\binom{|X|}{r+1}}$$

□

We now compute the total number of possible graphs (possible outcomes) $|\mathcal{G}|$ that can be built with $|a|$ *allow*, $|m|$ *mutual* and $|d|$ *deny* edges. $|\mathcal{G}|$ is required to compute $P(|Views_s| = N)$.

LEMMA 5.4. *Given (1) an authorization graph $G = (V, E)$, (2) $|d|$ deny edges, (3) $|m|$ mutual edges and (4) $|a|$ allow edges, the number of graphs $|\mathcal{G}|$ that one can build is:*

$$|\mathcal{G}| = \binom{|E|}{|a|} \cdot \binom{|E| - |a|}{|m|} \tag{7}$$

PROOF. Consider an authorization graph G with $|V|$ nodes and $|E|$ edges. Now we want to assign labels to the $|E|$ edges in G. There are $\binom{|E|}{|a|}$ possible ways to assign $|a|$ *allow* labels to the total number of edges $|E|$. Next, there are $\binom{|E|-|a|}{|m|}$ possible ways to assign $|m|$ *mutual* labels to the remaining edges, $|E| - |a|$. Finally, there are $\binom{|E|-|a|-|m|}{T_d}$ possible ways to assign $|d|$ *deny* labels to the remaining edges $|E| - |a| - |m|$. Consequently, $|\mathcal{G}| = \binom{|E|}{|a|} \cdot \binom{|E|-|a|}{|m|} \cdot \binom{|E|-|a|-|m|}{|d|}$. Using the facts that $|E| = |a| + |m| + |d|$, and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, we have $|\mathcal{G}| = \binom{|E|}{|a|} \cdot \binom{|E|-|a|}{|m|}$. □

To compute $P(|Views_s| = N)$, we now generalize Example 5.2. To do so, we establish some notions.

Definition 5.5 (Corresponding outgoing edge). Given an incoming edge of node s coming from u, the **corresponding outgoing edge** is the edge of s pointing to u.

Definition 5.6 (Corresponding incoming edge). Given an outgoing edge of node s pointing to u, the **corresponding incoming edge** is the edge of s coming from u.

Notation: Given a node s, (1) In_a is the number of *allow* incoming edges of s, (2) In_m its number of *mutual* incoming edges, (3) In_d its number of *deny* incoming edges, (4) In_{m1} its number of *mutual* incoming edges with a corresponding *allow* or *mutual* outgoing edge, (5) In_{m2} its number of *mutual* incoming edges with a corresponding *deny* outgoing edge, (6) Out_{am} its number of *allow* or *mutual* outgoing edges with a corresponding *mutual* incoming edge, (7) Out_d its number of *deny* outgoing edges with a corresponding *mutual* incoming edge.

Lemma 5.7. *Let $|E_{in}|, In_a, In_m, In_d, In_{m1}, In_{m2}, Out_{am}$ be given.*

1. *The number of incoming edges is: $|E_{in}| = In_a + In_m + In_d$.*
2. *The number of mutual incoming edges is: $In_m = In_{m1} + In_{m2}$.*
3. *The number of mutual or allow outgoing edges with corresponding mutual incoming edges is $Out_{am} = In_{m1}$.*
4. *The number of deny outgoing edges with corresponding mutual incoming edges is $Out_d = In_{m2}$.*

Proof. First, by Definition 5.1, an authorization graph has only *allow*, *mutual* or *deny* edges. Then, $|E_{in}| = In_a + In_m + In_d$. Second, since In_{m1} and In_{m2} are *mutual* incoming edges with corresponding *allow* or *mutual* and *deny* outgoing edges, it follows that $In_m = In_{m1} + In_{m2}$. Third, Out_{am} are *allow* or *mutual* outgoing edges with corresponding *mutual* incoming edge. Then, $Out_{am} = In_{m1}$. Fifth, Out_d are *deny* outgoing edges with corresponding *mutual* incoming edges. Then, $Out_d = In_{m2}$. □

Example 5.8. Consider case C_3 of Example 5.2. Using the above notation, C_3 is represented as: $In_a = 0, In_{m1} = 1, In_{m2} = 0$ $In_d = 1$, $Out_{am} = 1$, $Out_d = 0$, and $In_m = In_{m1} + In_{m2}$.

Definition 5.9 (Events). Let a node s, an integer $N \leq |E_{in}|$, In_a, In_m, In_d, In_{m1}, In_{m2}, Out_{am}, and Out_d, such that $N = In_a + In_{m1}$, be given.

- s_{In_a} is the event of s having In_a edges.
- s_{In_m} is the event of s having In_m edges.
- s_{In_d} is the event of s having In_d edges.
- $s_{Out_{am}}$ is the event of s having Out_{am} edges.
- s_{Out_d} is the event of s having Out_d edges.

Events $s_{In_a}, s_{In_m}, s_{In_d}, s_{Out_{am}}$ and s_{Out_d} are what we call *dependent events.*

We now compute the individual probabilities of each case in Example 5.2 in the general case. The general case is the probability $P(s_E)$ that a node s chosen at random has In_a, In_m, In_d, Out_{am} and Out_d edges. Lemma 5.10 computes $P(s_E)$, i.e., the joint probability that the dependent events $s_{In_a}, s_{In_m}, s_{In_d}, s_{Out_{am}}, s_{Out_d}$ happen.

Lemma 5.10. *Let (1) an authorization graph $G = (V, E)$ with $|d|$ deny edges, $|m|$ mutual edges, $|a|$ allow edges, (2) an integer $N \leq |E_{in}|$, and the values (3) In_a, In_m, In_d, Out_{am}, and Out_d such that $N = In_a + In_{m1}$ be given. The probability $P(s_E)$ that a random node s has In_a, In_m, In_d, Out_{am} and Out_d edges is:*

$$P(s_E) = \frac{\binom{|E_{in}|}{N-In_{m1}} \cdot \binom{|E|-N+In_m}{|a|-N+In_{m1}}}{\binom{|E|}{|a|}} \cdot \frac{\binom{|E_{in}|-N+In_{m1}}{In_{m1}+In_{m2}} \cdot \binom{|E|-N-In_{m2}}{|m|-In_{m1}-In_{m2}}}{\binom{|E|-N+In_{m1}}{|m|}} \cdot \frac{\binom{|E|-|E_{in}|}{|d|-|E_{in}|+N+In_{m2}}}{\binom{|E|-N-In_{m2}}{|d|}} \cdot \frac{\binom{In_{m1}+In_{m2}}{In_{m1}} \cdot \binom{|E|-|E_{in}|-In_{m1}}{|a|+|m|-N-In_{m1}-In_{m2}}}{\binom{|E|-|E_{in}|}{|a|+|m|-N-In_{m2}}} \cdot \frac{\binom{|E|-|E_{in}|-In_{m1}-In_{m2}}{|d|-|E_{in}|+N}}{\binom{|E|-|E_{in}|-In_{m1}}{|d|-|E_{in}|+In_{m2}+N}} \tag{8}$$

Proof. The joint probability of s_{In_a}, s_{In_m}, s_{In_d}, $s_{Out_{am}}$, and s_{Out_d} is $P(s_E) = P(s_{In_a}) \cdot P(s_{In_m} | s_{In_a}) \cdot P(s_{In_d} | s_{In_a} \cap s_{In_m}) \cdot P(s_{Out_{am}} | s_{In_a} \cap s_{In_m} \cap s_{In_d}) \cdot P(s_{Out_d} | s_{In_a} \cap s_{In_m} \cap s_{In_d} \cap s_{Out_{am}})$. We divide the proof in five parts. In the first part we compute the probability of event s_{In_a}. In the second part we compute the probability of event s_{In_m} given event s_{In_a}. In the third part we compute the probability of event s_{In_d} given events s_{In_a} and s_{In_m}. In the fourth part we compute the probability of event $s_{Out_{am}}$ given events s_{In_a}, s_{In_m} and s_{In_d}. In the fifth part we compute the probability of event s_{Out_d} given events s_{In_a}, s_{In_m}, s_{In_d}, and $s_{Out_{am}}$. Let us start by computing the probability of event s_{In_a}. Fist, from the total number of incoming edges $|E_{in}|$, we fix In_a *allow* edges. We can choose In_a *allow* edges from the $|E_{in}|$ incoming edges in $\binom{|E_{in}|}{In_a}$ possible ways. Second, the number of possible ways for assigning the remaining *allow* labels $|a| - In_a$ to the total remaining edges $|E| - In_a$ is $\binom{E-In_a}{|a|-In_a}$. After this step, T_a *allow* edges have been assigned. Third, we assign the $|m|$ *mutual* labels. There are $\binom{|E|-|a|}{|m|}$ possible ways to assign $|m|$ *mutual* labels to the remaining edges $|E| - |a|$. Finally, there are $\binom{|E|-|a|-|m|}{|d|}$ possible ways to assign $|d|$ *deny* labels to the remaining edges $|E| - |a| - |d|$. Using the facts that $|E| = |a| + |m| + |d|$, and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, $\binom{|E|-|a|-|m|}{|d|} = 1$. To this point, we have finished the analysis of the probability that event s_{In_a} happens. Then the probability of event s_{In_a} is given by the number of different graphs where s_{In_a} happens divided by the total number of possible graphs, Equation (7). So:

$$P(s_{In_a}) = \frac{\binom{|E_{in}|}{In_a} \cdot \binom{|E|-In_a}{|a|-In_a} \cdot \cancel{\binom{|E|-|a|}{|m|}} \cdot \overset{\nearrow 1}{\cancel{\binom{|E|-|a|-|m|}{|d|}}}}{\binom{|E|}{|a|} \cdot \cancel{\binom{|E|-|a|}{|m|}}}$$

Because of Lemma 5.7 and using the fact that $N = In_a + In_{m1}$:

$$P(s_{In_a}) = \frac{\binom{|E_{in}|}{N-In_{m1}} \cdot \binom{|E|-N+In_{m1}}{|a|-N+In_{m1}}}{\binom{|E|}{|a|}} \tag{9}$$

Let us move now to the computation of the probability that event s_{In_m} happens given event s_{In_a}. First, we have already assign In_a *allow* incoming edges. Then from the total number of remaining incoming edges $|E_{in}|-In_a$, we fix In_m *mutual* edges. We can choose In_m *mutual* edges from the remaining incoming edges $|E_{in}|-In_a$ in $\binom{|E_{in}|-In_a}{In_m}$ possible ways. Second, the number of possible ways for assigning the remaining *mutual* labels $|m| - In_m$ to the remaining, not yet assigned edges $E-In_a-In_m$ is $\binom{|E|-In_a-In_m}{|m|-In_m}$. After this step, T_m *mutual* edges and In_a *allow* edges have been assigned. Third, we assign the remaining $|a| - In_a$ *allow* labels. There are $\binom{|E|-In_a-|m|}{|a|-In_a}$ possible ways to assign $|a| - In_a$ *allow* labels to the remaining, not yet assigned edges $|E| - In_a - |m|$. Fourth, there are $\binom{|E|-|a|-|m|}{|d|}$ possible ways to assign $|d|$ *deny* labels to the remaining edges $|E| - |a| - |m|$. Finally, the total number of possible graphs that can be built with $|E|-In_a$ edges is $\binom{|E|-In_a}{|m|}\cdot\binom{|E|-In_a-|m|}{Ta-In_a}\cdot\binom{|E|-|a|-|m|}{|d|}$. Using the facts that $|E| = |a| + |m| + |d|$, and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, the probability that event s_{In_m} happens given event s_{In_a} is:

$$P(s_{In_m}|s_{In_a}) = \frac{\binom{|E_{in}|-In_a}{In_m} \cdot \binom{|E|-In_a-In_m}{|m|-In_m} \cdot \cancel{\binom{|E|-In_a-|m|}{|a|-In_a}} \cdot \cancel{\binom{|E|-|a|-|m|}{|d|}}}{\binom{|E|-In_a}{|m|} \cdot \cancel{\binom{|E|-In_a-|m|}{Ta-In_a}} \cdot \cancel{\binom{|E|-|a|-|m|}{|d|}}}$$

Because of Lemma 5.7 and using the fact that $N = In_a + In_{m1}$:

$$P(s_{In_m}|s_{In_a}) = \frac{\binom{|E_{in}|-N+In_{m1}}{In_{m1}+In_{m2}} \cdot \binom{|E|-N-In_{m2}}{|m|-In_{m1}-In_{m2}}}{\binom{|E|-N+In_{m1}}{|m|}} \tag{10}$$

Let us move now to the computation of the probability that event s_{In_d} happens given events s_{In_a} and s_{In_m}. First, we have already assign $In_a + In_m$ *allow* and *mutual* incoming edges. Then from the total number of remaining incoming edges $|E_{in}| - In_a - In_m$, we fix In_d *deny* edges. We can choose In_d *deny* edges from the remaining incoming edges $|E_{in}| - In_a - In_m$ in $\binom{|E_{in}|-In_a-In_m}{In_d}$ possible ways. Using the fact that $|E_{in}| = In_a + In_m + In_d$ and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, $\binom{|E_{in}|-In_a-In_m}{In_d} = 1$. Second, the number of possible ways for assigning the remaining *deny* labels $|d| - In_d$ to the remaining, not yet assigned edges $E - In_a - In_m - In_d$ is $\binom{|E|-In_a-In_m-In_d}{|d|-In_d}$. After this step, T_d *deny* edges and $In_a + In_m$ *allow* and *mutual* edges have been assigned. Third, we assign the $|a| - In_a$ *allow* labels. There are $\binom{|E|-In_a-In_m-|d|}{|a|-In_a}$ possible ways to assign $|a|-In_a$ *allow* labels to the remaining, not yet assigned edges $|E| - In_a - In_m - |d|$. Fourth, there are $\binom{|E|-|a|-|d|}{|m|-In_m}$ possible ways to assign the remaining *mutual* labels $|m| - In_m$ to the remaining, not yet assigned edges $|E| - |a| - |d| - In_m$. Finally, the total number of possible graphs that can be built with $|E| - In_a - In_m$ edges is $\binom{|E|-In_a-In_m}{|d|}\cdot\binom{|E|-In_a-In_m-|d|}{Ta-In_a}\cdot\binom{|E|-In_m-|a|-|d|}{|m|-In_m}$. Using the fact that $|E| = |a| + |m| + |d|$, and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, the probability that event s_{In_d} happens given events s_{In_a} and s_{In_m} is:

$$P(s_{In_d}|s_{In_a} \cap s_{In_m}) =$$

$$\frac{\cancelto{1}{\binom{|E_{in}|-In_a-In_m}{In_d}} \cdot \binom{|E|-|E_{in}|}{|d|-In_d} \cdot \cancel{\binom{|E|-In_a-In_m-|d|}{|a|-In_a}} \cdot \cancel{\binom{|E|-In_m-|d|-|a|}{|m|-In_m}}}{\binom{|E|-In_a-In_m}{|d|}|) \cdot \cancel{\binom{|E|-In_a-In_m-|d|}{|a|-In_a}} \cdot \cancel{\binom{|E|-In_m-|a|-|d|}{|m|-In_m}}}$$

Because of Lemma 5.7 and using the fact that $N = In_a + In_{m1}$:

$$P(s_{In_d}|s_{In_a} \cap s_{In_m}) = \frac{\binom{|E|-|E_{in}|}{|d|-|E_{in}|+N+In_{m2}}}{\binom{|E|-N-In_{m2}}{|d|}} \tag{11}$$

Let us move now to the computation of the probability that event $s_{Out_{am}}$ happens given events s_{In_a}, s_{In_m} and s_{In_d}. First, we have already assigned $|E_{in}|$ incoming edges and we still have $|E| - |E_{in}|$ edges, which are not yet assigned. Since, the Out_{am} outgoing edges can have either *allow* or *mutual* labels and because of Lemma 5.3, we can consider the *allow* and *mutual* labels as one group for this part of the proof. From the total number of outgoing edges E_{out}, we fix Out_{am} edges labeled either with *allow* or *mutual*. The Out_{am} outgoing edges should be assigned exactly to the nodes that have an outgoing edge labeled with *mutual* pointing to the randomly chosen node. Then the Out_{am} outgoing edges cannot be any outgoing edge from the $|E_{out}|$ edges; they have to be exactly the In_m edges that are pointing to the nodes that have a *mutual* outgoing edge pointing to the randomly chosen node. Therefore, we can choose Out_{am} edges from In_m outgoing edges in $\binom{In_m}{Out_{am}}$ possible ways. Second, the number of possible ways for assigning the remaining *allow* and *mutual* labels $|a| + |m| - In_a - In_m - Out_{am}$ to the remaining, not yet assigned edges $|E| - |E_{in}| - Out_{am}$ is $\binom{|E|-|E_{in}|-Out_{am}}{|a|+|m|-In_a-In_m-Out_{am}}$. Third, there are $\binom{|E|-|a|-|m|-In_d}{|d|}$ possible ways to assign $|d| - In_d$ *deny* labels to the remaining edges $|E| - |a| - |m|$. Finally, the total number of possible graphs that can be built with $|E| - |E_{in}|$ edges is $\binom{|E|-|E_{in}|}{|a|+|m|-In_a-In_m}\cdot\binom{|E|-|a|-|m|-In_d}{|d|-In_d}$. Using the facts that $|E| = |a| + |m| + |d|$, and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, the probability that event $s_{Out_{am}}$ happens given events s_{In_a}, s_{In_m} and s_{In_d} is:

$$P(s_{Out_{am}}|s_{In_a} \cap s_{In_m} \cap s_{In_d}) =$$

$$\frac{\cancelto{1}{\binom{In_m}{Out_{am}}} \cdot \binom{|E|-|E_{in}|-Out_{am}}{|a|+|m|-In_a-In_m-Out_{am}} \cdot \cancel{\binom{|E|-|a|-|m|-In_d}{|d|-In_d}}}{\binom{|E|-|E_{in}|}{|a|+|m|-In_a-In_m} \cdot \cancel{\binom{|E|-|a|-|m|-In_d}{|d|-In_d}}}$$

Because of Lemma 5.7 and using the fact that $N = In_a + In_{m1}$:

$$P(s_{Out_{am}} \mid s_{In_a} \cap s_{In_m} \cap s_{In_d}) = \frac{\binom{In_{m1}+In_{m2}}{In_{m1}} \cdot \binom{|E|-|E_{in}|-In_{m1}}{|a|+|m|-N-In_{m1}-In_{m2}}}{\binom{|E|-|E_{in}|}{|a|+|m|-N-In_{m2}}} \tag{12}$$

Let us move now to the computation of the probability that event s_{Out_d} happens given events s_{In_a}, s_{In_m}, s_{In_d} and $s_{Out_{am}}$. First, we have already assigned $|E_{in}| - Out_{am}$ edges and we still have $|E|-|E_{in}|-Out_{am}$ edges, which are not yet assigned. From the total number of outgoing edges E_{out}, we fix Out_d *deny* edges. The Out_d outgoing edges should be assigned exactly to the nodes that have an outgoing edge labeled with *mutual* pointing to the randomly chosen node. Then the Out_d outgoing edges cannot be any outgoing edge from the $|E_{out}|$ edges; they have to be exactly the In_m edges that are pointing to the nodes that have a *mutual* outgoing edge pointing to the randomly chosen node. Since from the In_m edges, we have already fixed Out_{am} edges, we can choose Out_d edges from $In_m - Out_{am}$ outgoing edges in $\binom{In_m - Out_{am}}{Out_d}$ possible ways. Using the fact that $Out_d = In_{m2}$, $In_{m2} = In_m - Out_{am}$ and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, $\binom{In_m - Out_{am}}{Out_d} = 1$. Second, the number of possible ways for assigning the remaining *deny* labels $|d| - In_d - Out_d$ to the remaining, not yet assigned edges $|E| - |E_{in}| - Out_{am} - Out_d$ is $\binom{|E|-|E_{in}|-Out_{am}-Out_d}{|d|-In_d-Out_d}$. Third, there are $\binom{|E|-In_a-In_m-Out_{am}-|d|}{|a|+|m|-In_a-In_m-Out_{am}}$ possible ways to assign $|a|+|m|-In_a-In_m-Out_{am}$ the remaining *allow* and *mutual* labels to the remaining edges $|E| - In_a - In_m - Out_{am} - |d|$. Finally, the total number of possible graphs that can be built with $|E| - |E_{in}| - Out_{am}$ edges is $\binom{|E|-In_a-In_m-In_d-Out_{am}}{|d|-In_d} \cdot \binom{|E|-In_a-In_m-Out_{am}-|d|}{|a|+|m|-In_a-In_m-Out_{am}}$. Using the facts that $|E| = |a| + |m| + |d|$, and $\forall n \in \mathbb{N}, \binom{n}{n} = 1$, the probability that event s_{Out_d} happens given events s_{In_a}, s_{In_m}, s_{In_d} and $s_{Out_{am}}$ is:

$$P(s_{Out_d} | s_{In_a} \cap s_{In_m} \cap s_{In_d} \cap s_{Out_{am}}) = \frac{\cancelto{1}{\binom{In_m-Out_{am}}{Out_d}} \cdot \binom{|E|-|E_{in}|-Out_{am}-Out_d}{|d|-In_d-Out_d} \cdot \cancel{\binom{|E|-In_a-In_m-Out_{am}-|d|}{|a|+|m|-In_a-In_m-Out_{am}}}}{\binom{|E|-In_a-In_m-In_d-Out_{am}}{|d|-In_d} \cdot \cancel{\binom{|E|-In_a-In_m-Out_{am}-|d|}{|a|+|m|-In_a-In_m-Out_{am}}}}$$

Because of Lemma 5.7 and using the fact that $N = In_a + In_{m1}$:

$$P(s_{Out_d} | s_{In_a} \cap s_{In_m} \cap s_{In_d} \cap s_{Out_{am}}) = \frac{\binom{|E|-|E_{in}|-In_{m1}-In_{m2}}{|d|-|E_{in}|+N}}{\binom{|E|-|E_{in}|-In_{m1}}{|d|-|E_{in}|+In_{m2}+N}} \tag{13}$$

Finally, using Equations (9) to (13), the probability that a randomly chosen node has (1) In_a *allow* incoming edges, (2) In_m *mutual* incoming edges, (3) In_d *deny* incoming edges, (4) Out_{am} outgoing edges labeled either with *allow* or *mutual*, and (5) Out_d *deny* outgoing edges such that each of the $Out_{am} + Out_d$ outgoing edges are pointing to one of the nodes that have a *mutual* edge pointing to the node s, where $In_a + In_m + In_d = |E_{in}|$ and $Out_{am} = In_{m1}$, $Out_d = In_{m2}$, and $In_m = In_{m1} + In_{m2}$, is:

$$P(s_E) = \frac{\binom{|E_{in}|}{N-In_{m1}} \cdot \binom{|E|-N+In_m}{|a|-N+In_{m1}}}{\binom{|E|}{|a|}} \cdot \frac{\binom{|E_{in}|-N+In_{m1}}{In_{m1}+In_{m2}} \cdot \binom{|E|-N-In_{m2}}{|m|-In_{m1}-In_{m2}}}{\binom{|E|-N+In_{m1}}{|m|}} \cdot \frac{\binom{|E|-|E_{in}|}{|d|-|E_{in}|+N+In_{m2}}}{\binom{|E|-N-In_{m2}}{|d|}} \cdot \frac{\binom{In_{m1}+In_{m2}}{In_{m1}} \cdot \binom{|E|-|E_{in}|-In_{m1}}{|a|+|m|-N-In_{m1}-In_{m2}}}{\binom{|E|-|E_{in}|}{|a|+|m|-N-In_{m2}}} \cdot \frac{\binom{|E|-|E_{in}|-In_{m1}-In_{m2}}{|d|-|E_{in}|+N}}{\binom{|E|-|E_{in}|-In_{m1}}{|d|-|E_{in}|+In_{m2}+N}}$$

□

Given an integer $N \leq |E_{in}|$, Theorem 5.11 yields the probability $P(|Views_s| = N)$. To obtain it, we sum up all the individual cases of Example 5.2 in the general scenario.

Theorem 5.11. *Let (1) an authorization graph $G = (V, E)$, (2) an integer $N \leq |E_{in}|$, (3) $|d|$ deny edges, (4) $|m|$ mutual edges and (5) $|a|$ allow edges be given. The probability $P(|Views_s| = N)$ that a node s chosen at random has a view of size N is:*

$$P(|Views_s| = N) = \sum_{In_{m1}=0}^{min\{N,|m|\}} \sum_{In_{m2}=0}^{min\{|E_{in}|-N,|m|\}} P(s_E) \tag{14}$$

Proof. Given a node s, there are different cases in which the size of the view of s, $|Views_s|$, can be equal to exactly N. One case is that node s has N incoming edges edges labeled with *allow* and $E_{in} - N$ incoming edges labeled with *deny*. Another case is that node s has N incoming edges labeled with *allow*, In_{m2} incoming edges labeled with *mutual*, $|E_{in}| - N - In_{m2}$ incoming edges labeled with *deny* and In_{m2} outgoing edges labeled with *deny* pointing to the nodes that assigned the In_{m2} *mutual* incoming edges, where $In_{m2} \leq |E_{in}| - N$. Another case is that node s has $N - 1$ incoming edges labeled with *allow*, one labeled with *mutual* coming from a node u, $Ein - N$ incoming edges labeled with *deny*, and for that *mutual* incoming edge, there is an *allow* or *mutual* outgoing edge from s to u. Every time that we decrease one *allow* incoming edge, we need to increase (1) one *mutual* incoming edges and (2) one *allow* or *mutual* outgoing edge, and the remaining incoming edges $|E_{in}| - N$ have to labeled with *deny*. Then, to compute the probability, $P(|Views_s| = N)$, we need to sum up the probabilities of all different cases. We start from the probability of having N incoming edges labeled with *allow* and $E_{in} - N$ incoming edges labeled with *deny* i.e., there are no incoming edges labeled with *mutual*.Then maintaining the N *allow* incoming edges, (1) we increase by one the *mutual* incoming edges such that for each *mutual* incoming edge coming from a node u there is one *deny* outgoing edge from node s to node u, and (2) we set the remaining incoming edges $N - In_{m2}$ to be *deny*. We repeat this process until we reach to $N - In_a$ *mutual* incoming edges and $N - In_a$ *deny* outgoing edges, while maintaining N *allow* incoming edges. Next, from the N incoming edges, we increase by one the number of *mutual* incoming edges such that for each *mutual* incoming edge coming from a node u there is an *allow* or *mutual* outgoing edges. We finish with the probability of having $|E_{in}|$ incoming edges labeled with *mutual* such that there are In_{m1} outgoing edges labeled with *allow* or *mutual* and In_{m2} outgoing edges labeled with *deny*, where $In_{m1} = N$, $In_{m2} = |E_{in}| - In_{m1}$ and $In_m = |Em1| + In_{m2}$.

Then using Lemma 5.10, we obtain:

$$P(|View_s| = N) = \sum_{In_{m1}=0}^{min\{N,|m|\}} \sum_{In_{m2}=0}^{min\{|E_{in}|-N,|m|\}} P(s_E)$$

□

Using Theorem 5.11 one can compute the probability that a node s chosen at random has a view size equal to a given $N \leq |E_{in}|$ depending on the share of *mutual* to *deny* and *allow* authorizations of the population. This will allow us to study in Section 6.1 how the changes in the authorizations, i.e., replacements of *allow* or *deny* authorizations with *mutual* ones, affect the size of the view of s.

6 EXPERIMENTS

This section presents an experimental analysis of the impact of *mutual* authorizations on the size of the view of a person, and an experimental validation of the complexity analysis of QF and FQ.

6.1 Experimental Analysis – Size of the Views

This subsection features two sets of experiments, named *Mutual/Deny* and *Mutual/Allow*. The goal of them is to analyze how the ratios of *mutual* to *deny* and of *mutual* to *allow* authorizations, respectively, affect the probability that a random person can read the position of a given number of persons $N \leq |E_{in}|$, i.e., $P(|View_s| = N)$.

6.1.1 Experiment Setup. We create 100 authorization graphs, 50 graphs for each set of experiments. All graphs have $|V| = 100$ nodes and $|E| = 9900$ edges. We construct the graphs of both experiments by starting with a graph that has 50 percent *allow* edges, 50 percent *deny* edges and 0 percent *mutual* edges. All the graphs fulfill a uniform distribution regarding the number of *allow*, *mutual* and *deny* edge.

For the *Mutual/Deny* experiments, we then modify the percentage of *mutual* and *deny* labeled edges by increasing the former one in steps of 1 while decreasing the later at the same rate. The percentage of *allow*-edges remains unchanged. For each setting, we compute $P(|View_s| = N)$, the probability that a random person can read the position of exactly N persons. We consider three values for N, namely $N = 60$, $N = 80$, $N = |E_{in}|$.

For *Mutual/Deny* experiments, instead of decreasing the share of *deny*-edges, we decrease the one of *allow*-edges.

6.1.2 Results. Figure 3(a) for the *Mutual/Deny* experiments and Figure 4(a) for the *Mutual/Allow* experiments show the probability that a node s chosen at random can read the position of N persons, $P(|View_s| = N)$, contingent on the percentage of *mutual* and *deny* authorizations. Figure 3(b) and Figure 4(b) are semi-log plots corresponding to Figure 3(a) and Figure 4(a), respectively, which give more emphasis to smaller probabilities.

Discussion: Depending on the size of the view, the impact of changing *deny* to *mutual* is different. We can observe in Figure 3(b) that, for larger views, the impact of replacing *deny* authorizations with *mutual* ones is higher as well. For instance, if 5 percent of *deny* authorizations are replaced by *mutual* ones, the probability $P(|View_s| = |E_{in}|)$, increases by a factor of 1000. For probability $P(|View_s| = 60)$ in turn increase is by a factor of 1.3. However, if one is interested in a high probability of having a view of a specific size, say a 80% chance, the results depend on the target size of the view. If the target size of the view is smaller, it is needed to replace a smaller percentage of *deny* authorizations by *mutual* ones in order to reach this probability. For instance, the probability $P(|View_s| = 60)$, given 50 percent of *allow* and 50 percent of *deny* authorizations, is ≈ 0.021. If only 20 percent of *deny* authorizations are replaced by *mutual* ones, this probability increases to 0.8. Consider now the probability $P(|View_s| = |E_{in}|$. Then, if we want this probability to increase to 0.8, it is necessary to replace 49.5 percent of *deny* authorizations by *mutual* ones, i.e., almost all *deny* authorizations should be replaced. As the percentage of *mutual*-edges increases and the percentage of *allow*-edges decreases in the same proportion, we can observe in Figure 4(b) that, for larger views, the impact of replacing *deny* authorizations with *allow* ones is higher. For instance, if 5 percent of *deny* authorizations are replaced by *mutual* ones, the probability $P(|View_s| = |E_{in}|)$ decreases by a factor of 100, whereas the probability $P(|View_s| = 60)$ decreases by a factor of 3.6. Then the decrease of these probabilities depends on the size of the view that one is interested in.

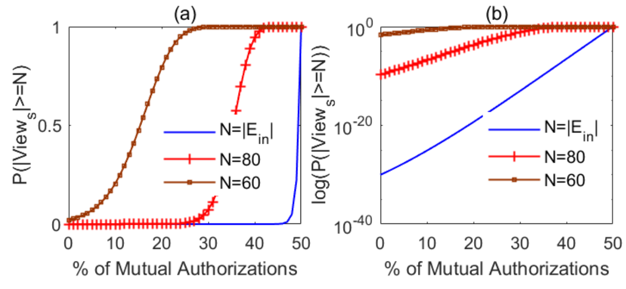

Fig. 3: $P(|View_s| = N)$ - ***Mutual/Deny* experiments**

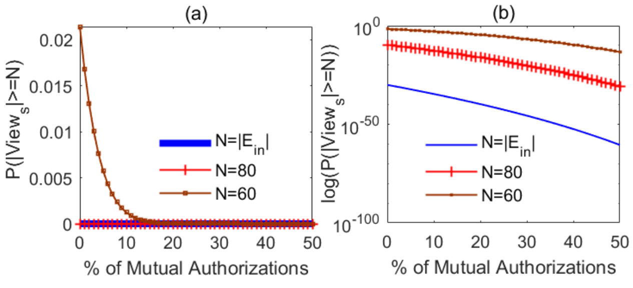

Fig. 4: $P(|View_s| = N)$ - ***Mutual/Allow* experiments**

6.2 Experimental Validation of the Complexity Analysis of QF and FQ

Our complexity analysis of QF and FQ in Section 4 already allows us to compare both approaches. However, since that analysis covers the worst case, experimental results are needed (1) as validation and (2) to determine how far the worst case deviates from the concrete behavior of individual queries. To implement the QF approach, we use the R-tree index structure from Oracle, and the remaining implementation was done in Java.

6.2.1 Experiment Setup. In Section 4, we have found, based on the complexity analysis, that the parameters that affect the performance of FQ and QF are (1) the number of persons n, (2) the size of the view $|View_s|$ of a given person s, (3) the parameter k of the kNN query and (4) the value $\delta = k_{real} - k$. Similarly to the complexity analysis, for simplicity, we set δ to 0 and k to 20. We set the remaining parameters, n and $|View_s|$, as follows:

Number of persons n: We create a dataset with 317080 persons. This number is the size of the DBLP dataset. To assign a position to each person, we choose a random physical position from the Tokyo dataset [27], which contains 573703 real check-ins, i.e., positions.

Size of the view of a person s, $|View_s|$, and query sample: We chose 1500 persons at random from the 317080 persons and assigned the authorizations so that we have 15 classes of different sized views (from 50 to 40000). For each class, we have 100 persons with the respective view size, i.e., 1500 queries in 15 different classes in total.

6.2.2 Experiment Results. Figure 5 shows a comparison of the query-processing times for kNN queries with QF and FQ. We have grouped the persons of our query sample by the size of their view and have plotted the query-processing time. We exclude the database-connection time and network-communication time from the run time reported. The dotted line is the average size of the view in DBLP with 317080 nodes, i.e., 64.98, [13]. The dashed line in Figure 5(c) is the size of the view for which the performance of both approaches is equal, i.e., $View_{equal} \approx 23032, 3$.

Discussion. For real scenarios, i.e., the size of the view is equal to 64.98, FQ performs better than QF for all queries. These results are in line with our complexity analysis, and one may interpret them as an indication that our analysis also holds for the average case. Next, these findings remain correct for a size of the view of up to 800, which is higher than the highest average size of the view in real scenarios, i.e., 520 (Section 4). However, as Figure 5(b) shows, with a view size between 1000 up to 20000, the processing times of most of the queries with QF are lower than that of the ones with FQ, in contrast to our complexity analysis. This can be expected since our analysis has focused on the worst case. In Figure 5(c), we observe that the processing times of most of the queries with QF, or all the queries in the case of the last two groups, i.e., 35000 and 40000, are lower than that of the ones with FQ. These results indicate that for a size of the view greater than the value $View_{equal}$ (dashed line) our complexity analysis holds even for the average case.

7 MUTUAL AUTHORIZATIONS FOR OTHER RESOURCES

In this paper, we have restricted our model to one type of resource, physical positions. We now describe the steps necessary to facilitate mutual authorizations for other domains, e.g., health records.

(1) One must define a scale for the sensitivity of different resources, so that exchanges can be fair.
(2) One must specify the degree of sensitivity of each resource and/or an entity responsible for assigning such a degree to each resource., e.g., its owner or a global authority. This may be intricate, cf. Example 1.8. Next, it must be the case that participants agree with these specifications. For instance, if I find my health record much more precious than yours and vice versa, our approach does not cover this.
(3) One has to define (1) the services that the system will support and (2) the integration of access control with these services. For instance, think of the information need that I want to see the 10 health records most similar to mine. In this case, it is needed a notion of distance, together with an implementation.
(4) It remains just to adjust Definition 2.6 of mutual authorization. One has to specify that resources with the same degree of sensitivity can be exchanged.

8 RELATED WORK

Several access control models have been proposed, like RBAC [20], Task RBAC [16] and Attribute-based Access Control [29]. The difference to ours is that they only consider the grants *deny* and *allow*. These two grants are not enough to capture *mutual* authorizations.

Besides access control models, encryption techniques have also been studied to achieve data confidentiality [3, 26]. The main idea is to encrypt the resources and to enforce access control with the decryption keys assigned to the users. The approach in [26] encrypts the data with different keys, depending on the authorizations to be enforced. After encryption, the decryption keys are given to users based on their access privileges. In [3], the data is encrypted together with an access structure, which represents a set of attributes together with values which users must have in order to access the data. The decryption key given to the users is generated based on their attribute values. Users can decrypt a ciphertext c if their decryption key matches the attribute values of the access structure associated with c. This work does not consider mutual access to resources. This is because the decryption keys are generated and distributed without considering reciprocity.

There is other work that focuses on formalizing and verifying the authorization constraints in RBAC and its extensions. These proposals use Colored Petri-Nets [21] or the Unified Modeling Language UML [18]. They focus on (1) introducing formal techniques to verify the design and consistency of authorizations on RBAC models, i.e., model checking, and on (2) providing a graphical representation of the authorizations, as visualizations. This work is confined to *allow* and *deny* authorizations as well. This is because it is based on access control models existing at that time. Next, our authorization graph can be seen as a graphical representation of authorizations in our model.

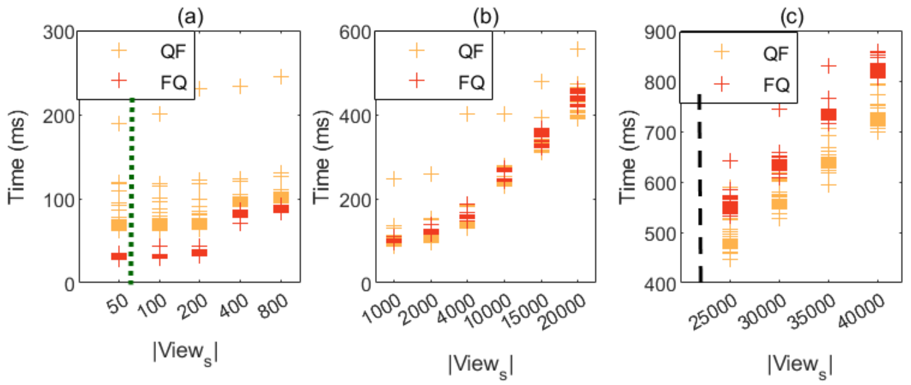

Fig. 5: Comparison of the QF and FQ approaches for kNN queries

9 CONCLUSIONS

Reciprocity is a powerful determinant of human behavior. However, none of the existing access control models explicitly supports it. In this paper, we have proposed a new type of authorization, called *mutual*. It allows users to grant access to their resources to users that allow them the same. We have extended RBAC to incorporate mutual authorizations and have formally defined their syntax and semantics. Since the result of a given service in the presence of mutual authorizations is not obvious, we have studied this as well. To do so, we have selected LBSs as a use case for the deployment of mutual authorizations, and we have proposed two approaches. A complexity analysis tells us when each approach is better. We have validated the results of our analysis experimentally. Further, we have studied how the difference in the ratio of *mutual* to *deny* and *allow* authorizations affects the size of the view of a given user.

In the future, it will be interesting to study how to apply encryption techniques to achieve data secrecy and confidentiality, and generate and distribute the encryption-decryption keys in a reciprocal manner. Another direction is to explore how to incorporate mutual authorizations into existing model-checking techniques like [18, 21]. Yet another direction is to extend mutual authorizations by considering different resources with more than one controller, i.e., users who can regulate access to the resource, and with different levels of sensitivity.

ACKNOWLEDGMENTS

The first author thanks “Escuela Politécnica Nacional, Ecuador - Departamento de Informática y Ciencias de la Computación" for its support. This work was partially funded by the German Research Foundation (DFG) as part of the research Datenschutzkonforme Verwaltung relationaler Datenbestände.

REFERENCES

[1] Vijayalakshmi Atluri, Heechang Shin, and Jaideep Vaidya. 2008. Efficient security policy enforcement for the mobile environment. *Journal of Computer Security* 16, 4 (2008).
[2] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. 2003. A logical framework for reasoning about access control models. *ACM TISSEC* 6, 1 (2003).
[3] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-policy attribute-based encryption. In *IEEE Security and Privacy, 2007.*
[4] Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. 1996. On the Lambert W function. *Advances in Computational mathematics* 5, 1 (1996).
[5] MAC Dekker, Jason Crampton, and Sandro Etalle. 2008. RBAC administration in distributed systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM.
[6] Armin Falk and Urs Fischbacher. 2006. A theory of reciprocity. *Games and economic behavior* 54, 2 (2006).
[7] Ernst Fehr, Urs Fischbacher, and Simon Gächter. 2002. Strong reciprocity, human cooperation, and the enforcement of social norms. *Human Nature* 13, 1 (2002).
[8] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. 2008. Relbac: Relation based access control. In *Semantics, Knowledge and Grid, 2008. SKG'08. Fourth International Conference on*. IEEE.
[9] Koji Hasebe, Mitsuhiro Mabuchi, and Akira Matsushita. 2010. Capability-based delegation model in RBAC. In *Proceedings of the 15th ACM symposium on Access control models and technologies*. ACM.
[10] Alexander Hinneburg, Charu Aggarwal, and Daniel A Keim. 2000. What is the nearest neighbor in high dimensional spaces?. In *Proceeding of the 26th VLDB*.
[11] Hongxin Hu and Gail-Joon Ahn. 2011. Multiparty authorization framework for data sharing in online social networks. In *IFIP DBSec*. Springer.
[12] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. 2010. Location-dependent query processing: Where we are and where we are heading. *ACM Computing Surveys (CSUR)* 42, 3 (2010).
[13] Silvio Lattanzi and Yaron Singer. 2015. The power of random neighbors in social networks. In *Proceedings of the 8th ACM WSDM Conference*.
[14] Rajendra Prasad Mahapatra and Partha Sarathi Chakraborty. 2015. Comparative analysis of nearest neighbor query processing techniques. *Procedia Computer Science* 57 (2015).
[15] Dinesh P Mehta and Sartaj Sahni. 2018. *Handbook of data structures and applications* (2nd edition ed.). Chapman and Hall/CRC.
[16] Sejong Oh and Seog Park. 2003. Task–role-based access control model. *Information systems* 28, 6 (2003).
[17] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)* 3, 2 (2000).
[18] Indrakshi Ray, Na Li, Robert France, and Dae-Kyoo Kim. 2004. Using UML to visualize role-based access control constraints. In *Proceedings of the ninth ACM symposium on Access control models and technologies*. ACM.
[19] C Carl Robusto. 1957. The cosine-haversine formula. *The American Mathematical Monthly* 64, 1 (1957).
[20] Ravi S Sandhu, Edward J Coynek, Hal L Feinsteink, and Charles E Youmank. 1996. Role-based access control models. *IEEE computer* 29, 2 (1996).
[21] Basit Shafiq, Ammar Masood, James Joshi, and Arif Ghafoor. 2005. A role-based access control policy verification framework for real-time systems. In *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005*. IEEE.
[22] Jie Shi and Hong Zhu. 2010. A fine-grained access control model for relational databases. *Journal of Zhejiang University-Science C* 11, 8 (2010).
[23] Jie Shi, Hong Zhu, Ge Fu, and Tao Jiang. 2009. On the soundness property for sql queries of fine-grained access control in dbmss. In *2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*. IEEE.
[24] Luca Stanca, Luigino Bruni, and Luca Corazzini. 2009. Testing theories of reciprocity: Do motivations matter? *Journal of economic behavior & organization* 71, 2 (2009).
[25] Romuald Thion, François Lesueur, and Meriam Talbi. 2015. Tuple-based access control: a provenance-based information flow control for relational data. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM.

[26] Sabrina Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. 2010. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems (TODS)* 35, 2 (2010).
[27] Dingqi Yang, Daqing Zhang, Vincent W Zheng, and Zhiyong Yu. 2015. Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 1 (2015).
[28] Xun Yi, Russell Paulet, Elisa Bertino, and Vijay Varadharajan. 2014. Practical k nearest neighbor queries with location privacy. In *2014 IEEE 30th ICDE*. IEEE.
[29] Eric Yuan and Jin Tong. 2005. Attributed based access control (ABAC) for web services. In *Web Services, 2005. ICWS 2005*. IEEE.