2018

Ontology-Guided Pre-Release Inference Disruption

Mark Stephen Daniels
University of South Carolina

Ontology-Guided Pre-Release Inference Disruption

by

Mark Stephen Daniels

Bachelor of Science
College of Charleston 1984

Master of Science
Johns Hopkins University 1992

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2018

Accepted by:

Csilla Farkas, Major Professor

John Rose, Committee Member

Gabriel Terejanu, Committee Member

Lannan Luo, Committee Member

Benjamin Schooley, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

Dedication

I am dedicating this work to my wife and best friend Debbie, who has been a constant source of support and encouragement during the challenges of school and life. I am truly thankful for having you in my life. Your love is what has kept me going.

This work is also dedicated to our children, Matthew and Kelly. You will never know how much help you have provided me on this journey.

I could never have accomplished this without my family's unwavering love, inspiration, and support.

Acknowledgments

I am forever grateful to my advisor Csilla Farkas, whose guidance and advice have made this research possible. Taking on a student that works full time and lives 100 miles from campus has to be a challenge. I will never forget Professor Farkas giving many hours of her time for early morning phone calls and endless trips to campus on weekends. I am not sure I would have made it to the end with any other advisor.

I am especially grateful to my committee, Professor John Rose, Professor Gabriel Terejanu, Professor Lannan Lou, and Professor Benjamin Schooley. Your input and advice helped me become a better scholar.

To Mom and Dad - thanks for helping me achieve so much in my life. Your love, support, and encouragement have helped me get where I am today. Thanks to my little sister Bridget, for being there when I need you. Bill, Carol - thanks for everything you do. Billy and Ellen - thanks for just being there, it really meant a lot.

Thanks to Debbie for graphics, proofreading, coffee, pizza, and patience. Thanks to Matthew for all the late-night discussions and math tutoring and Kelly for making me think, laugh, and relax. Thanks to Dan Furlong and Melissa Forinash for your friendship and encouragement.

A special thanks to all my friends and colleagues at MUSC. Your assistance and advice was invaluable. Lastly, thanks to Mike Caputo for your support, always asking how I am doing, and just spending time chatting.

Abstract

We investigate privacy violations occurring when non-confidential patient data is combined with medical domain ontologies to disclose a patient's protected health information (PHI). We propose a framework that detects privacy violations and eliminates undesired inferences. Our inference channel removal process is based on controlling the release of the data items that lead to undesired inferences. These data items are either blocked from release or generalized to eliminate the disclosure of the PHI. We show that our method is sound and complete. Soundness means the only inference paths generated logically follow from released data and corresponding domain knowledge. Completeness means we detect all inference channels leading to undesired data disclosures. Our approach maximizes data availability by minimizing the number of data items to be generalized or removed.

In Phase 1 of our research, we construct an optimal solution which disrupts all privacy violations. We have developed a cost model based on the number of data items that are removed or generalized. We calculate the cost for each solution and select the solution with the lowest cost as the optimal solution.

Phase 2 of our research introduces heuristic-based improvements into our approach. We have developed a method to construct a solution, called an inference disruption cover. We use the entropy of the concepts within domain ontology to guide selection of the best facts in a disruption cover for generalization.

In Phase 3, we extend our privacy model to incorporate personal privacy preferences and safety. We provide mechanisms to specify a patient's personal privacy restrictions as well as a clinician's safety criteria. We introduce privacy and safety

labeling of data items. We develop conflict resolution strategies when privacy and safety labels are contradictory. Our conflict resolution strategy favors safety over personal privacy.

Lastly, in Phase 4, we propose a graphical tool to support patients' understanding of the privacy settings. Our tool provides brief tutorials about health data types, regulations, and typical healthcare data sharing. We also allow patients to view their medical data and data inferred using domain knowledge. This will help the patient understand the impact of releasing their data through a Health Information Exchange or for secondary use. Our graphical interface allows patients to request that specific data items be blocked from being released. An important aspect of our approach is that it sets the foundation for creation of patient-specific privacy policies.

In summary, the primary contribution of this work is a sound and complete framework capable of efficiently detecting and disrupting healthcare-focused inference violations. We extend the privacy model to incorporate patient-specific privacy and safety preferences. Finally, our proof-of-concept prototype implementation supports privacy preserving data release and real-time policy composition.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACL ..... Access Control List

API ..... Application Programming Interface

DAG ..... Directed Acyclic Graph

EMR ..... Electronic Medical Record

HIE ..... Health Information Exchange

GUI ..... Graphical User Interface

HIPAA ..... Health Insurance Portability and Accountability Act

FHIR ..... Fast Healthcare Interoperability Resources

KB ..... Knowledge Base

LUB ..... Least-Upper-Bound

MAC ..... Mandatory Access Control

PFC ..... Participating Fact Combination

PHI ..... Protected Health Information

PIA ..... Pre-Release Inference Analyzer

RBAC ..... Role-Base Access Control

RDF ..... Resource Description Framework

TRDBAC ..... Temporal Reflexive Database Access Control

Chapter 1

Introduction

Protected Health Information (PHI) is defined as health information collected on or about a patient's past, current, and future condition and identifiers that link data back to the individual patient. PHI is protected by the Privacy Rule of the Federal Health Insurance Portability and Accountability Act (HIPAA) of 1996 [14]. While a healthcare entity may act as the custodian of a patient's data, the data is traditionally considered to be "owned" by the patient. Without the patient's consent, access to his/her health data is restricted to those with a need to know. In addition, data may be released outside of a health care entity for reasons such as public health, quality improvement and research studies; in these cases, the data is usually de-identified and not linked to a specific patient. Federal, state and local law determines what data can and cannot be accessed or released and for what purposes. The intent of these laws is to protect a patient's privacy.

On the black market, a stolen medical identity often sells for multiple times more than that of a stolen credit card number, making it a prime target for attackers [25]. Large-scale data collection and development of health care ontologies allow malicious users to automate the inference of medical facts with a high level of confidence. Current legislation and due diligence in data release protocols is not sufficient to protect against post-release inference capabilities. Health Information Exchanges (HIE) are now commonplace and allow the exchange of identified patient data between disparate entities. While regulated, the HIE data exchange process has the potential to expose a patient's private data [10]. With a heightened awareness of threats to

their medical data, patients are becoming concerned and need a proactive capability to set limits on release of their personal information.

There has been significant work on understanding the database inference problem. Farkas, et al., [12, 4] look at inference channels in statistical and relational databases as well as inference issues with data mining which incorporate disparate data sets and metadata. This work assumes that data resides with the data custodian, allowing controls on data design, access, and query construction to be locally enforced. Jain, et al., [18, 16, 19] have looked at the use of Resource Description Framework (RDF) [15] metadata and ontologies to control access to sensitive data sets. Their work also assumes data containing privacy-breaching inference channels has not be released outside the organization. Iwaya [17] uses ontologies to assess and alter data sets prior to release, but the goal of his work is to ensure that the data set is anonymized / de-identified, which is not always the desired outcome. Lastly, Ellick [6] has worked on schemes to segment private medical data and many commercial medical software vendors have developed tools to identify HIPAA-designated patient identifiers as well as constructed data release masking or obfuscating filters, but none of this work addresses the post release inference problem.

We present an integrated privacy framework, called the *Pre-Release Inference Analyzer* (PIA), which guarantees that an attacker cannot access unauthorized data from a released data set even if the attacker can access domain knowledge and inference tools. The goal of the framework is to identify privacy violations that can be inferred by leveraging domain ontologies. Our solution will block inference paths that lead to disclosure of sensitive data. We propose a data modification approach based on ontology-guided data generalization.

We propose the PIA framework which is composed of three functional modules: inference path generation (Reason), evaluation (Detect Violations), and solution determination (Build Solution). Once the "Initial Data" is received, the framework will

iterate over all possible inference disruption solutions. The optimal solution is then selected and applied to the Initial Data. This modified data set is then returned as an authorized data set. Note that a solution will always be found although the cost may prohibit the solution from returning a useful data set.

Our current work is focused on the medical / healthcare domain, but we feel that the approach is applicable to many other domains. We are planning to extend our model to support patient-specific policies in the future. To the best of our knowledge, our work is the first that introduces an integrated framework designed to pre-evaluate data releases by identifying privacy violating inferences introduced by domain ontology-based reasoning.

Our theoretical results show that our solution is sound and complete. Intuitively, soundness means that the only inference paths generated logically follow from the release data set and the corresponding domain knowledge. Completeness means that we detect all inference channels leading to undesired data disclosures. We also show that our approach only modifies data items contributing to an inference channel disruption and preserves data availability by minimizing the number of data items to be modified.

Our work is the first, to the best of our knowledge that introduces sound, complete, and minimal algorithms to leverage domain knowledge to suggest database instance modifications to defeat the privacy violating inference paths. Soundness ensures that the framework does not introduce any data items that should not exist in the final data set. Completeness ensures that all data items that should exist in the final data set are present. Minimality ensures that we maximize data availability by modifying the minimal data required to reach the goal.

We use the Apache Jena framework for reasoning over our data sets and domain ontologies, both represented as RDF data sets. The PIA framework identifies all inference-introduced privacy violations and proposes a solution that defeats the in-

ference paths while maximizing data availability. In addition to theoretical results, empirical results from our prototypes indicate that with integration of solution limiting heuristics, implementation of the approach is practical. We need to further evaluate using large-scale data sets, but test results to this point are positive.

We develop sound, complete, and minimal algorithms for the identification of violations as well as the nomination and evaluation of solution sets. More specifically, our algorithms evaluate all database instance items that contribute to privacy violations. We evaluate the items as a collection and evaluate their impact to violating paths, both directly and indirectly, as well as their involvement in non-violating paths. We develop a cost function that considers each data item individually and collectively to determine the best modification solution set to return.

1.1 Motivation

Medical data is considered sensitive and private, in fact it may be the most private piece of information that a person can posses. While there are many reasons to release a patient's medical data, including payment, treatment, quality, research, and syndromic surveillance, current due diligence in release protocols is not sufficient against post-release inference capabilities. With advances in healthcare ontologies, data contained in a patient's record has much more semantic depth. This depth is beneficial, as it allows healthcare professionals to gain better insight into a patient's condition, treatment, and outcome. Data scientists can use predictive models and reasoning engines to expose "unseen" facts, providing caregivers new information to assist in the assessment and treatment of a patient. Unfortunately, with the good comes bad – the rich semantic depth of a patient record also gives a criminal actor the ability to expose private data that was never intended to be shared. On the black market, an individual's medical identity is worth $10-20$ times more than their credit identity, making healthcare data a prime target for attackers. Private medical

data can be used for blackmail, pharmacy purchases, insurance fraud and many other illegal activities. With expanded data collection and advances in healthcare ontologies, an actor does not require extensive medical knowledge to infer missing or intentionally excluded medical facts. Technology advances have given cyber attackers new weapons to automate the inference of medical facts with a high level of confidence. Attackers can 'regenerate' unreleased sensitive data based on non-sensitive released data, complex ontologies and reasoning tools. There are United States laws that protect a patient's privacy; unfortunately, many cyber attacks are launched from foreign soil. With a heightened awareness of threats to a their medical data, patients are becoming concerned and need a proactive capability to set limits on release of personal information.

1.2 Running Example

In our running example, we address the privacy problem in data from the healthcare sector. The running example database contains our medical ontology with concepts and relations about patients, physicians, medications, and diseases. A graphical illustration of the ontology and ontological instances is shown in Figure 1.1.

For privacy labels, we use privacy values: Public, Low, Medium, and High. The label values range from *Public* indicating no privacy to *High* indicating the highest privacy level available.

We define a set of patterns which are used for the assignment of privacy labels. Patterns are generic and expressed as RDF triples. Each pattern may contain a "wildcard" designator in any part of the triple. Our database and the ontology are also stored in the RDF triple format allowing for straightforward pattern matching. A pattern is associated with a single privacy label (see Table 1.1). We map patterns to instance facts to determine appropriate privacy labels for each fact.

For our running example, we introduce two individuals (see Figure 1.1), a patient

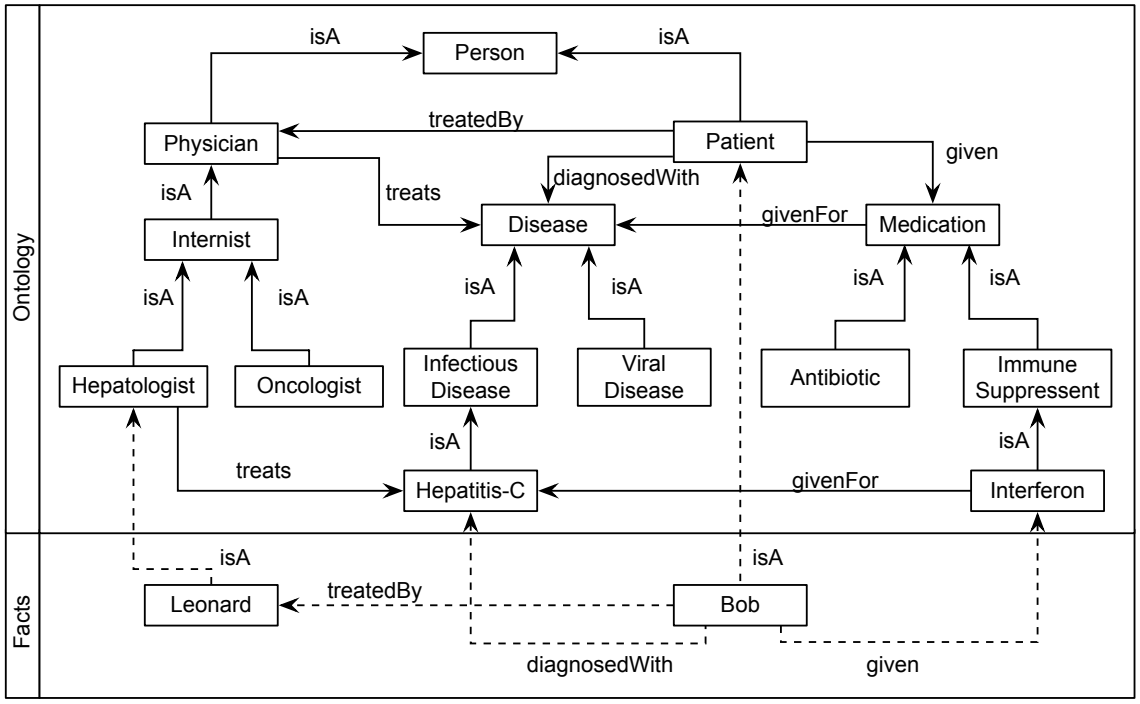


Figure 1.1 Running Example Medical Ontology - includes people, medical specialties, medications, and diseases. Several instance data items (facts) are also shown.

Table 1.1 Example pattern templates and associated privacy labels.

Pattern	Privacy Label
$(*, *, Patient)$	Not Private (public)
$(*, *, Physician)$	Not Private (public)
$(*, ISA, Hepatologist)$	Low Privacy
$(*, diagnosedWith, *)$	Low Privacy
$(*, given, Interferon)$	Medium Privacy
$(*, diagnosedWith, Hepatitis - C)$	High Privacy

named Bob and a physician named Leonard. Leonard is Bob's physician of record (Bob, hasPhysician, Leonard) and Leonard is a hepatologist (Leonard, ISA, Hepatologist) who treats liver associated diseases such as Hepatitis C (Hepatologist, treats, Hepatitis-C). Bob also has relationships to the disease Hepatitis-C (Bob, diagnosedWith, Hepatitis-C) and the medication Interferon (Bob, given, Interferon). Note that medication Interferon may be used in the treatment of the disease Hepatitis C (Inter-

feron, givenFor, Hepatitis-C). Mapping these instance facts to our privacy mapping patterns gives us the privacy assignments shown in Table 1.2.

Table 1.2 Example instance facts with privacy labels.

Instance	Privacy Label
(Bob,ISA,Patient)	Not Private (public)
(Leonard,ISA,Physician)	Not Private (public)
(Leonard,ISA,Hepatologist)	Low Privacy
(Bob,given,Interferon)	Medium Privacy
(Bob,diagnosedWith,Hepatitis-C)	High Privacy

Example 1.2.1 (Inference Violation)**.** Consider the medical ontology and its instances in Fig 1.1. Assume the following domain ontology rule exists: (Patient, treatedBy, Hematologist), (Patient, Given, Interferon)→(Patient, likelyHas, Hepatitis-C).

To protect privacy, we are not releasing that Bob has Hepatitis-C. We are willing to release all other instance data. Once released, an observer noticing that the patient's diagnosis is not revealed could reason over the released data and domain ontologies. Reasoning could infer what disease the patient has based on physician relationships and the use of specific medications. For Bob, we observe that his physician is a hematologist, who is a liver specialist and he takes a medication used to treat Hepatitis-C. Given these facts and rule, it is probable that Bob is being treated for Hepatitis C, so this new fact may be inferred. The new fact increases our available information, but also violates our privacy restriction by exposing unauthorized data that was more private than we intended to release (see Table 1.2).

To eliminate the undesired inference, we investigate the rules and patient data that contributed to that inference. When analyzing the rule, we determine that there are two facts, (Bob, given, Interferon) and (Bob, treatedBy, Hepatologist), that contribute to the inference. The second fact, (Bob, treatedBy, Hepatologist), is itself inferred from relations (Bob, treatedBy, Leonard) and (Leonard, ISA, Hepatologist). Taking all four facts into consideration, we could choose to remove the fact (Bob,

treatedBy, Hepatologist) from the data set. This appears to break the inference, but unfortunately this fact would get re-created in a subsequent inference based on facts (Bob, treatedBy, Leonard) and (Leonard, ISA, Hepatologist). If we choose to remove the fact (Bob, given, Interferon) instead, the inference would be broken and the removed fact is not re-created in a subsequent fix point, based on the data available.

1.3 Research Tasks

This dissertation presents our research findings addressing the following areas:

1. Privacy Analysis – Determine if reasoning over domain knowledge and non-sensitive may generate privacy violations. Develop methods to disrupt undesired inferences by removal or generalization of select data items.

   Focus areas:

   - Privacy model
   - Data inferencing and privacy
   - Cost model

   Publications:

   - M. Daniels and C. Farkas, "Health Data Privacy: A Case of Undesired Inferences", Proceedings of the 2018 IEEE Conference on Biomedical and Health Informatics, pp. 291 - 294
   - M. Daniels and C. Farkas, "Undesired Inferences: Ensuring Privacy in Health Data", submitted to Information Systems journal, under review
   - M. Daniels and C. Farkas, "Medical Privacy in the 21st Century", in progress, to be submitted to JAMIA - Journal of the American Medical Informatics Association

2. Efficient Disruption – Develop heuristics that increase our methods computational efficiency while still guaranteeing removal of privacy violations.

   Focus areas:

   - Exhaustive results analysis
   - Heuristic methods
   - Enhanced cost model

   Publications:

   - M. Daniels, J. Rose and C. Farkas, "Protecting Patients' Data: An Efficient Method for Health Data Privacy", submitted to the 13th International Conference on Availability, Reliability and Security (ARES 2018), under review

3. Privacy & Safety –Extend privacy model to include safety and patient privacy preferences.

   Focus areas:

   - Enhanced Privacy model
   - Enhanced cost model

   Publications:

   - Enabling Preferences in the Healthcare Data Privacy Model, to be submitted

4. Graphical User Interface – Design a graphical user interface to provide tutorials and privacy information. Also allow patients to view their data and any new data generated by reasoning over domain knowledge.

   Focus areas:

- Patient privacy tutorials and information
- Presentation of reasoning on patient data
- Setting of patient preferences

Publications:

- Educating Patients: Understanding and Participating in the Release of Their Health Data, to be submitted

1.4 Dissertation Outline

The remainder of the dissertation is organized as follows:

Chapter 2 discusses related work.

Chapter 3 describes the architecture and approach of our framework.

Chapter 4 describes our optimal exhaustive approach.

Chapter 5 describes efficient disruption using heuristics.

Chapter 6 extends our privacy model.

Chapter 7 describes our patient-focused user interface tool.

Chapter 8 provides conclusions and future research directions.

Chapter 2

Related Work

The primary focus of our work is defeating privacy violating inferences in data that is approved for release using traditional privacy controls. There has been increased attention given to the protection of healthcare data over the last few years. As we increase the volume and detail of data being stored about a patient, the potential for a privacy breach increases. Advances in ontologies and semantic processing enrich this data further increasing its value to the clinician and patient as well as the actor seeking to leverage it for for profit or unethical motives.

2.1 Medical Data Privacy

Research in the area of medical data privacy / protection has focused on restricting access of specific data items. Traditional methods look at data items as discrete objects and identify confidentiality levels specific to those objects. These methods then allow object access only to appropriate users. If objects are classified as a group (or document), the group classification is typically the highest classification of all member objects or a classification based on the static aggregate of data objects. These traditional controls work on an object level and do not address information that can be inferred either between the objects within the collection or when combined with domain knowledge or other publicly accessible data collections. If a data item is classified in advance and viewed in isolation, it can be protected using one of many traditional data access methods. Pfleeger [22] describes these methods including directories or containers, Access Control List (ACL), and Role-Based Access Control

(RBAC). These methods are widely used and common in the majority of commercial operating systems, databases, and applications.

Extensions to the traditional controls have been proposed to either deal with unique data sets or access patterns. Rashid et al. [24], proposed the Temporal Reflexive Database Access Control (TRDBAC) to address confidentiality with regards to data with temporal sensitivity. Rashid's concern was that a data item in isolation may not be sensitive, but if it has a discoverable temporal relationship to one or more other data items, it may then be sensitive.

Ellick [6] worked on schemes to segment private medical data and many commercial medical software vendors have developed tools to identify HIPAA-designated patient identifiers as well as constructed data release masking or obfuscating filters, but none of this work addresses the post release inference problem.

Most of the techniques used to ensure the data in a release will not disclose an individual's identity will also alter its effectiveness in data mining tasks. One of the most common tasks seen in data mining is the classification problem. To address classification problems, the data miner will usually build a model based on a training data set. The model is based on the assumption that the training data set is 'realistic' and carries distribution characteristics similar to the full (unclassified) data set. Trying to build a model on a data set that consists of perturbed data would not provide realistic results since data values are removed, generalized, or altered, all altering the distribution of the attributes. Agrawal [1] describes a mechanism for applying pre-release processing on the original data set that alters it in a predicted manner. He claims that the alterations can be made in a way that preserves the distribution of the data. Once released, the researcher can apply algorithms that will reconstruct nearly accurate distributions for the altered data, but will not reveal the original values of the data, therefore preserving privacy.

Afrawal's pre-release processing of the sensitive attributes creates new values using

one of two data modification approaches. The first is alteration based on value-class membership where attributes are 'partitioned into a set of disjoint, mutually-exclusive classes' – discretization would be an example of this approach. The second approach is based on value distortion where values are altered based on a random value from a known distribution. Agrawal's research is focused on reconstructing attribute distributions for the building of a decision tree classifier model and considers both Uniform and Gaussian distributions in his results. Once data was released, Agrawal described three distribution reconstruction algorithms (Global, ByClass, Local), all based on an iterative approach to estimating the original distribution. Each of the 3 algorithms applies the reconstruction at differing levels:

- Global applies it to each attribute while looking at the entire training set
- ByClass first splits the training set up by class and then applies the reconstruction to each class
- The Local approach is similar to ByClass, but performs reconstruction at each node; this is the point where decisions are made regarding how to construct the node branch decision (usually based on some information gain strategy).

Using a derived privacy metric, based on how closely a modified attribute can be estimated, Agrawal was able to show very good performance on decision trees built using the ByClass and Local methods when compared to decision trees created using the original data. They were also able to show that as the privacy level of the released data increased, the accuracy of each of the approaches degraded.

2.2 Inference Problem

There has been significant work on understanding the database inference problem, but the majority of this work assumes data remains with the data custodian, allowing controls on design, access, and query construction to be locally enforced. The goal of

recent work is to disallow an unauthorized user to see inferred data items that exceed their access permissions. Various methods have been proposed in the literature to disrupt or hide the conclusion of an inference channel making it unavailable to an unauthorized user.

Farkas, et al., [12, 4] look at sensitive data disclosures created by indirect access to data via unchecked inference channels. They describe the inference problem as it relates to both statistical data and relational (general) databases. In addition to single data sources, they also investigate the inference problem as it relates to data mining where the data instance incorporates numerous disparate data sets and metadata.

Brodsky, et al., [4] look at data dependent and independent disclosures caused by unauthorized inferences. They provide a framework that evaluates incoming queries based on Mandatory Access Control (MAC) and an audit of the historical data previously provided to the requestor. Brodsky's approach evaluates possible inferences based on what the user already has (previous queries) and what they ask for (current query). The results of these inferences are then evaluated by MAC for appropriateness to release to the requestor based on the user's authorization level. This scheme requires the data custodian to maintain control of the data sent and release data to requestors only through carefully controlled, evaluated, and auditable queries. Brodsky does not consider cases where externally released data is combined with ontologies or other publicly accessible data to generated privacy violating inferences.

2.3 Medical Ontologies and Inference Engines

The use of ontologies to enhance the semantic value of healthcare data is increasing at a rapid rate. The combination of medical ontologies and inference engines are helping clinicians better understand patient data as well as generate knowledge to help treat patients and enhance medicine in general. Early ontologies were focused on enhanc-

ing operational efficiency and billing, but their use has expanded into areas touching all aspects of clinical care. Noor and Cheng [21, 7] are using these technologies to better understand patient data properties and predict drug-drug interactions, while Fernández-Breis [13] is helping to determine phenotypes and identify patient cohorts for vital medical research. While medical ontologies are often focused on specific domains, work like BRIDG [3] is looking across domains and across information system databases. This work seeks to provide semantic interoperability that significantly increases the value of data and its ability to generate new knowledge.

A key factor in clinical decision support is a patient's problem list; the completeness of the list may determine the quality of the decision support process. Devarakonda [9] says that if a patient problem list is complete and accurate it will help clinicians provide better patient care, but the list is often inaccurate, duplicative, and out of date due to the manual effort required to validate and update information. Devarakonda proposes an automated method based on the IBM Watson inference technology. Wright [26] is using semantic processing over patient clinical and billing data to automate the generation of a more complete problem list.

Healthcare is a data-rich environment and the more enhanced the ontologies and inference processing get, the better we can leverage data for the good of the patient and healthcare operations. While ontologies are widely used, there are needs to extend and enhance their value. Quesada-Martinez [23] claims that many medical ontologies provide rich domain-based knowledge, but are closer to controlled vocabularies or taxonomies than the ontologies needed for medical semantics. To address this, Quesada-Martinez proposes a method to enhance ontologies by looking at the structure of labels across ontologies and analyzing any patterns found. These patterns are then investigated to determine if they reference existing ontological entities.

The healthcare domain has spawned numerous ontologies in recent times. Many of these started as proprietary vocabularies and taxonomies and started to see widespread

adoption as the need for tighter systems integration and health information exchanges grew. There are still many issues that impede full adoption and use. One such issue is ontology translation. Most ontologies were developed using a specific language (i.e., English, French, etc.) and then translated to other languages. Healthcare information exchange is now being conducted globally by medical facilities in many countries speaking many languages. The reliability of these translations remains an open problem still being studied [8].

With all the sensitive patient data being stored and generated, access control is paramount. Ontologies can also be used to help with access control. Jain, et al., [18, 16, 19] have looked at the use of RDF metadata and ontologies to control access to sensitive data sets. Their work also assumes data containing privacy-breaching inference channels have not been released outside the organization. Iwaya [17] uses ontologies to assess and alter data sets prior to release, but the goal of his work is to ensure that the data set is anonymized / de-identified, which is not always the desired outcome.

Chapter 3

Pre-Release Inference Framework

3.1 Architecture

We have developed an integrated privacy framework, "Pre-Release Inference Analyzer" (PIA). PIA ensures that no unauthorized data can be generated from a set of authorized data, even if the attacker has access to domain knowledge and inference tools. Our solution disrupts inference paths that lead to disclosure of sensitive data. The disruption is accomplished by modification of the initial authorized data set. We propose two data modification techniques: data removal and data generalization. The research challenge is to minimize the need for data alteration, thus maximizing data availability. These approaches are discussed in Chapter 4.

To facilitate privacy violation detection, we decorate inferred data with privacy labels. We evaluate inferred data to determine appropriate privacy labels. Data items are represented using RDF triples, (subject, predicate, object), there may be differing privacy criteria for each element of the triple. To resolve the potential privacy to a single label, we choose to assign privacy labels based on patterns (Definition 3.3.4). Each pattern will have an assigned privacy label. Patterns may include variables as the exact instance values in a data set are not known. Once the inference process is complete and paths examined, all patterns can be reduced to ground patterns (Example 3.3.1). In the mapping of patterns to privacy labels, a more specific variation of a pattern is at least as privacy restrictive, if not more, than a more general variation of that pattern. For efficiency and without loss of generality, we exclude evaluating

data against patterns that are "contained" in other patterns (Definition 3.3.5).

3.2 Approach

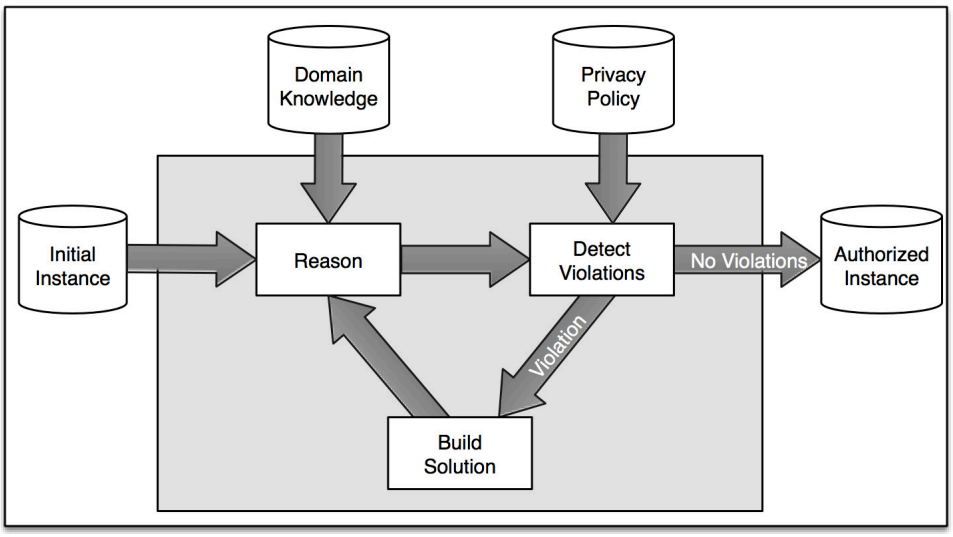


Figure 3.1 Pre-Release Inference Analyzer (PIA) Architecture

As depicted in Figure 3.1, our logical process can be viewed as having three primary processes, **reasoning**, **violation detection**, and **solution construction**. The three parts of the overall process are managed by a controlling function that forms a processing loop within the overall framework. In aggregate, our approach can be categorized into **Pre-Processing**, **Assessment**, **PIA Process**, and **Post-Processing** as follows:

Pre-Processing

- The initial database is minimized by removing redundant data items. Efficiency is improved since redundant data items, which would be regenerated through reasoning, unnecessarily increase the number of solutions that must be evaluated.

Assessment

- Derive all data items that can be inferred from the minimized database and domain knowledge.
- Assign privacy labels to inferred data items. Privacy labels are assigned according to patterns. The most restrictive privacy label is assigned to each newly generated fact.
- Evaluate privacy policy violations. Inferred data item labels are compared with a privacy threshold provided for the data release. Ground facts that participate in inference of privacy violations are identified.
- Solutions and associated costs are generated based on alteration (removal and generalization) of identified facts.

PIA Process Each solution is evaluated as follows:

- Copy initial database altering data items per solution.
- Derive all data items that can be inferred from the altered database and domain knowledge.
- Assign privacy labels to inferred data items.
- Evaluate privacy policy violations. Inferred data item labels are compared with privacy threshold provided for this data release
- Solutions are marked valid if no privacy violations found, else marked invalid.

Post-Processing

- Initial database is returned as authorized release if no violations found in initial assessment.
- Initial database altered by a valid solution with lowest cost is returned as authorized release if violations were found in initial assessment.

Our approach to unauthorized inference removal leverages domain knowledge. It alters inference participating facts to disrupt associated inference paths. We show that our approach terminates and is sound, complete, and minimal.

- By sound, we mean that every generated (inferred) data item logically follows from the initial data instance and domain knowledge.
- By complete, we mean that every data item that logically follows from the initial data instance and domain knowledge is generated.
- By minimal, we mean that there is no other valid solution that costs less (based on our cost methodology).

Detailed descriptions of modules and their properties are presented in Section 3.4.1.

3.3 Preliminaries

In this section, we introduce the formal definitions and models used by the PIA framework.

3.3.1 Data Model

We model external data (input and output) in a standard and consistent manner. The primary source of domain knowledge for our framework is a collection of one or more ontologies (Definition 3.3.1). Within our framework, we use a consistent ontology model to represent all referenced ontologies and instance data (database). We model domain ontology metadata as a RDF Knowledge Base (KB) [15]. This approach encapsulates ontological concepts, relations and domain information (i.e. domain rules) in a common model.

A 'domain ontology' is an ontology whose semantic terms and definitions are specific to a given area or topic (e.g. legal, geographic, medical). In our work, we are focusing on data and ontologies in the healthcare / medical domain.

Definition 3.3.1 (Ontology)**.** An ontology $O = (C, R, Dom, \nu)$ is a 4-tuple where:

- $C = \{c_1, \ldots, c_n\}$ represents the set of all concepts in O.
- $Rn = \{rn_1, \ldots, rn_m\}$ is the set of relations among the ontological concepts. A mapping function, denoted ρ, maps relations such that $\rho : rn_i \rightarrow (c_l, c_k)$, $i = 1, \ldots, m$, $(l, k) = 1, \ldots, n$, $c_l, c_k \in C$, and $rn_i \in Rn$. For convenience and without loss of generality, we represent the relation mapping $\rho : rn_j \rightarrow (c_i, c_k)$ as the triple (c_i, rn_j, c_k).
- Dom is a set of the domains of the concepts $\{c_1, \ldots, c_n\}$ of O, such that $Dom = \{dom(c_1), \ldots, dom(c_n)\}$, where $dom(c_i)$ is the domain of c_i.
- ν is a mapping from each concept to its domain $\nu : c_i \rightarrow dom(c_i)$.

The data that an entity desires to release to one or more other entities is referred to as the 'release data set'; for simplicity, we will just use the term 'data set'. We will use the terms 'initial data set' and 'final data set' to describe the data being released prior to processing by our framework (initial) and after processing (final).

Within the constructs of our framework, we refer to data sets as databases. Formally, each database is an ontological instance (Definition 3.3.2). Ontological instances are knowledge bases which includes information from referenced ontologies as well as an instance data set. The knowledge base describes both the instance values and concepts as well as the relations between them. Inheritance of concept properties and relations (Definition 3.3.3) is based on the ontological ISA hierarchy.

Definition 3.3.2 (Ontological Instance)**.** An instance of the ontology O is defined as $OI = (I, DB)$, where:

- $I = \{inst_1, \ldots, inst_z\}$ is a set of instance values, such that:
  - $(inst_i, \text{ISA}, c_j) \in DB$
  - $inst_i \in \nu(c_j)$
- $DB = \{db_1, \ldots, db_x\}$ represent the database of relations between instances and ontological concepts. Each relation is a triple of either the form (c_i, rn_j, c_k) or $(inst_i, ISA, c_k)$ or $(inst_i, rn_j, inst_k)$.

Definition 3.3.3 (Inheritance)**.** Relationships among concepts and instances are inherited based on the concept hierarchy of the ontology. $\forall c_i, c_j, rn, o_i, o_j$; if $\exists(c_i, rn, c_j)$ and (o_i, ISA, c_i) and (o_j, ISA, c_j), where o_i is either a concept or an instance, then (o_i, rn, o_j) must also exist.

Intuitively, a relation describing a class also describes that class's family of subclasses. Inheritance is transitive for the special relation 'ISA' such that, if (a,ISA,b) and (b,ISA,c) then (a,ISA,c). The special relation 'ISA' is also reflexive, (a,ISA,a).

Data items within a database are represented using RDF triples, (subject, predicate, object). While we assign privacy labels to a data item triple, there may be differing privacy criteria for each element of that triple. To resolve potential privacy inconsistencies of triple elements to a single label, we assign privacy labels based on patterns (Definition 3.3.4) to determine a triple's final privacy label. Each pattern will have an assigned privacy label. Patterns may include variables as the exact instance values in a data set are not known until processed. Once the reasoning process is complete and inference paths evaluated, each pattern can be reduced to a ground pattern (Example 3.3.1).

Definition 3.3.4 (Patterns)**.** A pattern is defined as $p = (c_i, rn, c_j)$, where $c_{(i,j)}$ are constants, concept names, or variable names and rn is a relation name or variable

name. A ground pattern $gp = (c_i, rn, c_j)$, is a pattern where c_1 and c_2 are constants or concept names and rn is a relation name.

Note: We use the wild-card symbol '*' in any position of a pattern to represent an unspecified variable name.

Example 3.3.1 (Patterns)**.** Examples of patterns are: (x, isPatientOf, Smith), (x, takesMedication ,y), (y, isMedicationType, Controlled), where x and y are free variables.

Examples of ground patterns: (John, isPatientOf, Smith), (John, takesMedication, Morphine), (Morphine, medicationType, Controlled); where all free variables have been resolved to constants.

Definition 3.3.5 (Pattern Containment)**.** Let $p_1 = (c_1, rn_1, c_2)$ and $p_2 = (c_3, rn_2, c_4)$ be patterns. We say that p_2 is contained in p_1, denoted as $p_2 \subseteq_{pc} p_1$, iff the following hold:

- (c_3, ISA, c_1), or both c_3 and c_1 are variables, or c_1 is a variable
- (rn_2, ISA, rn_1), or both rn_2 and rn_1 are variables, or rn_1 is a variable
- (c_4, ISA, c_2), or both c_4 and c_2 are variables, or c_2 is a variable

In the mapping of patterns to privacy labels, a more specific variation of a pattern is at least as privacy restrictive, if not more, than a more general variation of that pattern. For efficiency and without loss of generality, we exclude evaluating data against patterns that are 'contained' in other patterns (Definition 3.3.5).

Example 3.3.2 (Pattern Containment)**.** Let p_1 = (Male, Consumes, Fruit), p_2 = (Boy, Eats, Orange). If (Boy, ISA, Male), (Eats, ISA, Consumes) and (Orange, ISA, Fruit) exists, then $p_2 \subseteq_{pc} p_1$.

Let p_3 = (Male, Consumes, x), p_4 = (Boy, Eats, Orange). If (Boy, ISA, Male) and (Eats, ISA, Consumes) exist and x is a variable, then $p_4 \subseteq_{pc} p_3$.

To determine if a rule is satisfied by a database instance (and therefore generates a new data item), we must determine if the patterns, and symbols in those patterns, all map to constants in the database. We address this determination in steps, looking first at symbols, then patterns, and finally the inference rule.

Definition 3.3.6 (Symbol Mapping)**.** Let $S = \{s_1, \ldots, s_n\}$ be a set of symbols, including constants (values, concept names, and relation names) and variables. Let $\mathcal{C}$ be the set of constants (values, concept names, and relation names). γ is a symbol mapping, $\gamma : S \rightarrow C$, such that the mapping preserves constants (i.e., if s_1 is a constant and $\gamma : s_1 \rightarrow c_2$ then $s_1 = c_2$) and equalities.

Definition 3.3.7 (Pattern Mapping)**.** Let γ be a symbol mapping and (c_i, rn, c_j) a pattern. Γ is a pattern mapping using γ, such that $\Gamma\ (c_i, rn, c_j) = (\gamma(c_i),\ \gamma(rn),\ \gamma(c_j))$, and Γ preserves constants and equalities.

Example 3.3.3 (Pattern Mapping)**.** Let S = {x,enrolledIn,Mark, y, CSCE-899, USC} be a set of symbols and C a set of constants, C = {enrolledIn, Mark, CSCE-899, University, USC}. Given a pattern (Student, enrolledIn, x), we can find a symbol mapping γ, such that $\gamma(Student) = Student$, $\gamma(enrolledIn) = enrolledIn$, $\gamma(x) = CSCE - 899$. Using γ, we have a pattern mapping Γ (Student, enrolledIn, x) = ($\gamma(Student)$, $\gamma(enrolledIn)$, $\gamma(x)$) = (Student, enrolledIn, CSCE-899).

The mapping of symbols and patterns allows us to determine if the body of an inference rule (Definition 3.3.8) is satisfied. Satisfaction of a rule (Definition 3.3.9) will lead to the inference or generation of a data item. If this data is not already present in the ontological instance (Definition 3.3.2), it will be added.

Definition 3.3.8 (Inference Rule)**.** An inference rule is a Horn Clause expression of the form $\forall x_1, \ldots, x_k(p_1 \wedge \cdots \wedge p_n) \rightarrow q$ where $x_1, \ldots, x_k$ are all the free variables that appear in patterns $p_1, \ldots, p_n$ and q is a pattern, such that q does not contain any variables that do not also appear in $p_1, \ldots, p_n$.

Example 3.3.4. Using the database sample shown in Table 3.1, an example rule ((x, ISA, Patient) $\wedge$ (x, takesMedication, Aspirin) $\wedge$ (x, complainsOf, StomachPain)) $\rightarrow$ (x, likelyHas, StomachUlcer) would conclude that 'John likelyHas StomachUlcer' as all parts of the premise are satisfied. It would not conclude that 'Mary likelyHas StomachUlcer' since one part of the premise 'x, complainsOf, StomachPain' is not satisfied when x is 'Mary'.

Table 3.1 Sample triples from instance database showing facts for patients "Bob" and "Mary".

Subject	Predicate	Object
Bob	ISA	Patient
Mary	ISA	Patient
Bob	complainsOf	StomachPain
Bob	complainsOf	Headaches
Mary	complainsOf	KneePain
Mary	complainsOf	Headaches
Bob	takesMedicine	Aspirin
Mary	takesMedicine	Insulin
Mary	takesMedicine	Aspirin

Definition 3.3.9 (Rule Satisfaction)**.** Single Rule: let $r = p_1 \wedge \cdots \wedge p_k \rightarrow q$ be a rule and DB a database. We say DB satisfies r if there exists a mapping Γ from $p_1, \ldots, p_k$ to the DB, i.e., given $\Gamma(p_1), \ldots, \Gamma(p_k) \subseteq DB$ then $\Gamma(q)$ must also be in the database. Rule Set: let $R = \{r_1, \ldots, r_n\}$ be a set of rules and DB a database. We say DB satisfies R if there is a Γ such that DB satisfies Γ for all r_i, $i = 1, \ldots, n$, and Γ preserves constants and equalities.

Example 3.3.5 (Rule Satisfaction)**.** Let there be a rule, r: (x, ISA, Male) $\wedge$ (x, ISA, CEO) $\rightarrow$ (x, earns, \$500K) Assume the database DB contains the following facts: (John, ISA, Male), (Mary, ISA, Female), (John, ISA, CEO), (John, earns, \$500K). We can find mapping γ from the symbols in the body of r to DB as: γ(earns) = earns, γ(\$500K) = \$500K, γ(Male) = Male, γ(CEO) = CEO, γ(ISA) = ISA. We

can find mapping Γ from the body of r to DB as: Γ(x,ISA,Male) = (John,ISA,Male), Γ(x,ISA,CEO) = (John,ISA,CEO), but then Γ(x,earns,\$500K) = (John,earns,\$500K) must be in DB.

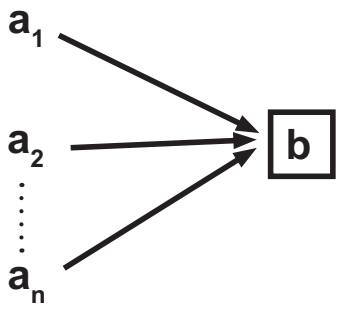


Figure 3.2 Single Inference Path - This figure shows a graphical representation of the inference path for rule $a_1 \wedge a_2 \wedge \cdots \wedge a_n \rightarrow b$, where b is an authorized inference.

Conceptually, our framework's internal data model is based on an undirected multi-graph. In this model, nodes represent semantic facts. Each node in the graph is a triple, defined as (subject, predicate, object). Graph edges connect the nodes that participate in an inference path with the inference conclusion (Figure 3.2). Nodes may participate in numerous inference paths and conclusions from one path may contribute to the satisfaction of another path (Figure 3.3). Edges are colored to indicate specific inference paths. Only one node in any colored path will be designated as the conclusion. The conclusion will have a privacy label assigned and may be designated as a privacy violation.

The internal model allows nodes in the rules making up an inference path to be decorated with attributes as they are discovered and collectively analyzed. This allows attributes, such as privacy labels, to be easily set and accessed. The model (as shown in Figure 3.3) allows us to show which nodes participate in which inference paths as well as distinguish between asserted and inferred concepts. This also allows

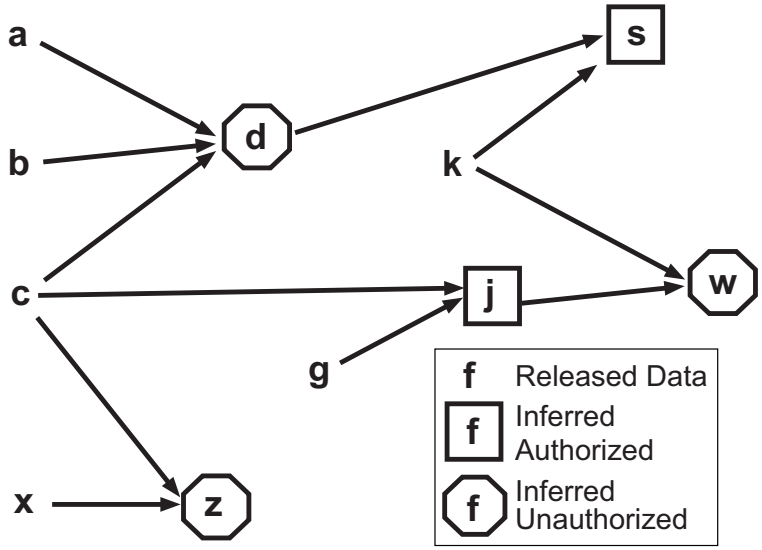


Figure 3.3 Inference Paths - Multiple intersecting paths. Conclusions are represented by squares (authorized) and octagons (unauthorized) to visually differentiate them. Note that the conclusion of one path may be a participant in another path.

tracking of privacy violating nodes (facts). Through examination of the interaction between nodes on the various inference paths, we can establish metrics on inference participation by a given fact. These metrics allow us to nominate candidate concepts for inclusion in our inference disruption process.

3.3.2 Privacy Model

Our privacy model defines a partial order set of privacy levels and associated labels. Dominance between labels is defined to indicate which privacy labels are more sensitive or restrictive than others. The model provides a mapping function to determine the privacy level of a specific data item and provide a label. The privacy levels and associated labels are based on domain-specific criteria. We are concerned with the

privacy of a patient's medical data and define multiple levels of data privacy. In our approach, we use patterns as the basis for privacy label assignment. A pattern may be broad in scope, using very general concepts in its definition or narrow in scope, using specific concepts that are more granular and impart more detailed knowledge. A pattern may contain 'wildcard' characters to indicate that only part of the pattern needs to be matched for its application to be applicable. A given data item may match numerous patterns and patterns from different domains. Through inheritance, a specific data value would also match more general variations (ancestors) of the same concept. Additional patterns may be matched if any patterns contained a 'wildcard'. Regardless of the number of patterns matched, the privacy label used for the data item must represent the highest level of privacy found for all matched patterns. Our partial-order privacy labels are monotonically increasing in value and correlate to increasing privacy levels. Using an access lattice construct, we are able to evaluate labels and determine the appropriate privacy level and label even if multiples are matched. The final level will always be the least upper bound of all labels found using pattern matching.

Traditionally, an authorized release is a data set that does not contain information exceeding a specified privacy threshold. This threshold may vary depending on the intended recipient(s) of the data. Threshold determination is often made based on the presence or absence of discrete data values known to violate a given privacy level.

Definition 3.3.10 (Privacy Mapping)**.** Let $L = (\{l_1, \ldots, l_n\}, \leq)$ denote the set of privacy labels and the partial order among the labels; and let $P = (\{p_1, \ldots, p_m\}, \subseteq_{pc})$ denote all patterns and the containment among the patterns. The privacy mapping λ, denoted $\lambda : P \rightarrow L$, assigns a privacy label to each pattern and satisfies the following: if $p_2 \subseteq_{pc} p_1$ then $\lambda(p_2) \geq \lambda(p_1)$.

Intuitively, the privacy label of the more specific pattern must dominate the privacy pattern of the more general pattern.

Table 3.2 Patterns, potentially with wildcards, are matched to an RDF triple allowing for mapping of a privacy label to the triple.

Pattern	Example
$(*, *, *)$	(*, *, *)
$(c_i, *, *)$	(Ebola, *, *)
$(*, r_j, *)$	(*,infectedWith,*)
$(*, *, c_k)$	(*, *, Blood)
$(c_i, r_j, *)$	(Blood,infectedWith,*)
$(*, r_j, c_k)$	(*,infectedWith,Ebola)
$(c_i, *, c_k)$	(Blood,*,Ebola)
(c_i, r_j, c_k)	(Blood,infectedWith,Ebola)

Definition 3.3.11 (Privacy Label Multiple Inheritance)**.** Let p be a pattern such that $p \subseteq_{pc} p_i, i = 1, \ldots, k$. Let $\lambda(p_1), \ldots, \lambda(p_k)$ represent the privacy labels of $p_1, \ldots, p_k$ respectively. The privacy label of p, denoted as $\lambda(p)$, must satisfy $\lambda(p) \geq LUB(\lambda(p_1), \ldots, \lambda(p_k))$.

Intuitively, this requirement aids the assignment of privacy labels to patterns without privacy labels (i.e., newly generated triples).

Example 3.3.6 (Privacy Labels)**.** Let p, p_1, and p_2 be patterns such that $p \subseteq_{pc} p_1$ and $p \subseteq_{pc} p_2$. Let $l_1 = (TopSecret, \{A, B\})$ and $l_2 = (Secret, \{C\})$ be the privacy labels of p_1 and p_2 respectively. The privacy label of $p \geq LUB(l_1, l_2) = (TopSecret, \{A, B, C\})$.

Definition 3.3.12 (Authorized Data Instance)**.** Given a user u with privacy classification $\lambda(u)$, we say that data instances $I = \{i_1, \ldots, i_l\}$ are authorized to read for user u only if $\lambda(u) \geq \lambda(i_j), j = 1, \ldots, l$, where $\lambda(i_j)$ is the privacy label of instance i_j.

Note, in this paper we address confidentiality (privacy) of patients' data. While our model is similar to the Bell-La Padula model [11] used for Mandatory Access

Control (MAC), the write operations are different in medical databases. Our ongoing work addresses audit and preservation of each write operation of the authorized user.

In order to set a threshold on the highest acceptable privacy label for data within an authorized data instance, we define a single label to serve as the violation threshold. No data item in the authorized data instance should have a label greater than that of the violation threshold.

Definition 3.3.13 (Violation Threshold Label)**.** The violation threshold label, denoted v, is the label which must not be dominated by any other label in the database.

Using Definition 3.3.10, we say that if v is a violation threshold label, then $\nexists\ l$; $l \in L$ and $l > v$.

When viewed in isolation, one may assume an initial data instance is an authorized data instance. This assumption is reasonable given traditional data access models; if all distinct data items in the instance are authorized to be read by user u, then the collection of data items (instance) must also be authorized to be read by user u. Unfortunately, with the introduction of domain knowledge and reasoning tools, new information may be generated that is not authorized to read by user u. This new information must be addressed if we want the released instance to truly be an authorized data instance.

3.3.3 Inference Processing

In order to eliminate the generation of new privacy violating data items, we disrupt the inference path leading to their conclusion. An optimal set of disruptions must be derived to guarantee that the data being released is an authorized data instance. Before discussing our approach in detail, we must review several aspects of the inference process and approaches used to measure solution efficiency.

Each generated data item comes into existence as the conclusion of some set of inference rules satisfied over DB, the initial data instance (Definitions 3.3.8 and 3.3.9).

Definition 3.3.14 (Minimal Inference Graph)**.** A minimal inference graph is a partial order directed acyclic graph (DAG) of the rules $r_1, \ldots, r_n$ such that the body of rule r_j contains q_i, the conclusion of rule r_i, that proceeds r_j in the DAG, (i,j = 1,. . . ,n).

Example 3.3.7 (Minimal Inference Graph)**.** Let DB_{orig} be a database instance and r a rule:

- DB_{orig} = {(Joe, hasGender, Male),(Male, Eats, Fruit)}
- r = (x, hasGender, Male) $\wedge$ (Male, Eats, Fruit) $\rightarrow$ (x, Eats, Fruit)
- Constants in DB_{orig} are denoted C, where C = {Joe, Male, Fruit, hasGender, Eats}.
- Symbols in r are denoted S, where S = {x, Joe, Male, Fruit, hasGender, Eats}.
- Symbols map as follows: γ(x) = Joe, γ(Male) = Male, γ(Fruit) = Fruit. γ(hasGender) = hasGender, γ(Eats) = Eats.
- Patterns in r map as follows: Γ(x, hasGender, Male) = (γ(x), γ(hasGender), γ(Male)) = (Joe, hasGender, Male); Γ(Male, Eats, Fruit) = (γ(Male), γ(Eats), γ(Fruit)) = (Male, Eats, Fruit); Γ(x, Eats, Fruit) = (γ(x), γ(Eats), γ(Fruit)) = (Joe, Eats, Fruit)

After application of rule r on DB_{orig}, the conclusion of r, (Joe, Eats, Fruit), is added to the database; DB_r is the resulting database:

- DB_r = (Joe, hasGender, Male),(Male, Eats, Fruit),(Joe, Eats, Fruit)

Definition 3.3.15 (Rule Containment)**.** Given 2 inference rules with the same conclusion:

- $r_1 : p_1 \wedge \cdots \wedge p_k \rightarrow q$ (more restrictive)
- $r_2 : \bar{p}_1 \wedge \cdots \wedge \bar{p}_n \rightarrow q$ (more general).

We say that r_2 is contained in r_1 iff $\forall$ pattern $p_i, (i = 1, \ldots, k)$, in the body of rule r_1, $\exists$ pattern $\bar{p}_j, (j = 1, \ldots, n)$, in the body of rule r_2, such that $\bar{p}_j \subseteq_{pc} p_i$.

Intuitively, this means that if a database instance satisfies the more specific rule, is also satisfies the more general rule.

Example 3.3.8 (Rule Containment)**.** Let r_1 and r_2 be rules such that

- r_1 = (Joe, takes, antibiotic) $\wedge$ (antibiotic, treats, infection) $\rightarrow$ (Joe, has, infection)
- r_2 = (Joe, takes, amoxicillin) $\wedge$ (amoxicillin, treats, BacterialInfection) $\rightarrow$ (Joe, has, infection)

Given (amoxicillin, ISA, antibiotic) and (BacterialInfection, ISA, infection) hold, then $r_2 \subseteq_{pc} r_1$.

A fact participates in the inference process if that fact is found in the body of any rule that is executed during reasoning. Since rule variables are resolved during reasoning, all facts are grounded upon completion of the reasoning process. A rule may be executed multiple times, but with different conclusions. Since a conclusion of one rule may be used to satisfy a variable in the body of another rule, not all facts found in rule bodies exist in DB.

Definition 3.3.16 (Inference Participant)**.** Let s be a data item found in instance database DB and r a rule satisfied by DB. We say that s is an inference participant over r if at least one pattern in r maps to s.

A fact may participate in the inference of numerous conclusions. The degree of direct participation of a fact is the number of times f is found in bodies of executed

rules. Since, as previously stated, a conclusion of one rule may be used to satisfy a variable in the body of another rule, we define degree of indirect participation to also include conclusions that a fact indirectly participates in (initial conclusion is used to infer additional conclusions). Note that removal of a fact from DB would cause any rule dependent on that fact to not be satisfied and therefore not reach a conclusion. Likewise, if a rule was dependent on a conclusion which was not generated (due to removal of a fact), then that rule would also not be satisfied and therefore also not reach a conclusion.

We define database inference removal to be the process of disrupting an inference path which concludes in the generation of a privacy violating data item. Our approach to database inference removal leverages domain knowledge and alters facts on the inference path to disrupt the generation of privacy violating data items. We show that our approach terminates and is sound, complete, and minimal.

- By sound, we mean that every generated (inferred) data item logically follows from the initial data instance and domain knowledge.
- By complete, we mean that every data item that logically follows from the initial data instance and domain knowledge is generated.
- By minimal, we mean that there is no other valid solution that costs less (based on our cost methodology).

3.4 Pre-Release Inference Disruption

In this section, we describe the initial approach used in our framework to detect and disrupt privacy violating inferences. The approach is exhaustive and seeks to determine an optimal solution. We describe the goal of the approach followed by selected algorithms. We then provide proof of desired properties along with current implementation and results.

3.4.1 Algorithms

In this section, we present our algorithms to construct an authorized data instance, DB_F, from an initial data instance, DB_0, and domain knowledge, O. We assume availability of a sound RDF reasoner.

Disrupt Violations (Main)

The approach for defeating privacy violating inferences is shown in Algorithm 1. For convenience, we reference a function "Reason" to call a RDF reasoner. We first create a minimized database, DB_m, to remove all redundant data. Reasoning and privacy mapping are then performed to allow detection of privacy violations; the set of identified violations are stored in SV. If violations are found, a solution set is built and evaluated (Algorithm 4). The selected (optimal) solution, S, is lastly applied to the initial database yielding an authorized data instance, DB_F. Note the selected solution, S, will be null if no violations are detected. In this case, the initial data instance will be released with no alterations.

ALGORITHM 1: DisruptViolations

Input: DB_0 - initial instance database
Input: O - ontology
Input: v - violation threshold label
Input: λ - privacy mapping function
Output: DB_F - authorized instance database

```
1  DB_m = MinimizeDB(DB_0, O)                    // minimized database
2  DB_mf = Reason(DB_m,O)                        // fix point database
3  C = set of inferred facts from DB_mf          // inferred fact list
4  SV = PrivacyMapAndDetect(C, v, λ)             // privacy violation list
5  if SV ≠ ∅ then
6  |  S = ExhaustiveDisruption(DB_0,DB_m,O,v,λ,DB_mf,SV)
7  else
8  |  S = ∅
9  end
10 DB_F = AlterData(DB_0, S)                     // authorized release
11 return DB_F
```

Minimize Database

Removing redundant data items early in the disruption process improves algorithm efficiency. Any solutions based on redundant data items would not prove useful since altered data items would be regenerated in the reasoner step. To avoid redundant data items, we reduce the initial data instance to a minimized data instance (Algorithm 2) before processing. A minimized data instance does not contain any data items that follow from remaining data items. As can be seen by the steps of Algorithm 2, we evaluate each fact, f, in DB for redundancy by reasoning over a copy of DB with f removed ($DB - f$) and testing for regeneration of f. The output data instance, DB_m, has all redundant data items removed.

ALGORITHM 2: MinimizeDB

Input: DB - instance database
Input: O - ontology
Output: DB_m - minimized database

```
1 DB_m = DB
2 forall f ∈ DB do
3 |   DB_w = DB_m − f
4 |   DB_wf = Reason(DB_w, O)
5 |   if f ∈ DB_wf then
6 |   |   DB_m = DB_m − f
7 |   end
8 end
9 return DB_m
```

Claim 3.4.1 (Database Minimizing)**.** *Algorithm 2 removes all redundant data items and only redundant data items.*

We prove Claim 3.4.1 in two parts. We first address removal of "only" redundant data items followed by addressing removal of "all" redundant data items.

Proof of Database Minimizing - remove only redundant data. Assume, by contradiction, that our algorithm removed a data item f, $f \in DB$, that was not redundant. Then f must have been generated in the reasoning step (line 4) of Algorithm

2, because it must have appeared in DB_{wf} (condition at line 5). So, if in the final returned minimized database f is not redundant, it can only occur if another fact, f_1, that contributed to generating f, was removed incorrectly, i.e., f_1 was removed although it was not redundant. Then f_1 must have been generated in the reasoning step, because it must also have appeared in DB_{wf}. So, if in the final returned minimized database f_1 is not redundant, it can only occur if another fact, f_2 that contributed to generating f_1, was removed incorrectly, i.e., f_2 was removed although it was not redundant. But then there must exist an inference path, $f_1, \ldots, f_k$, over data item $f_i \in DB$ $(i = 1, \ldots, k)$ for each redundant data item that contributed to generating f. Since each data item $f_i \in DB$ $(i = 1, \ldots, k)$ was removed from DB_m only if it could be generated from the remaining data instances and we have finite number of facts in DB, therefore, we eventually reach the empty set. This contradicts our original assumption. □

Proof of Database Minimizing - removes all redundant data. Assume, by contradiction, that our algorithm did not remove f, a redundant data item. In step 3 of Algorithm 2, we investigate all data items of the original database. So, since f is redundant, it must be regenerated by the reasoning step (line 4). In this case, by the condition at line 5, f would be removed from DB_m. This contradicts our original assumption that f is redundant, but was not removed. □

Privacy

In this section, we present an algorithm to detect privacy violations. We assume that all generally released data items are permitted to the users. Here, we investigate access control violations of all newly inferred facts. We use the privacy mapping function λ and violation threshold label v, such that an inferred fact will be added to the privacy violation list if the relation $\lambda(\text{inferred fact}) > v$ holds.

ALGORITHM 3: PrivacyMapAndDetect

Input: C - Set of inferred fact in DB
Input: v - Violation threshold label
Input: λ - Privacy Mapping Function
Output: SV - Set of violation facts

```
1 SV = ∅        // Initialize privacy violation (SV) set to null
2 forall c ∈ C do
3 |  λ(c) is the privacy label for the triple held in c
4 |  if ¬(λ(c) ≤ v)                          // c violates privacy
5 |   then
6 |  |  add c to SV                        // Add violation to set
7 |  end
8 end
9 return SV
```

Exhaustive Disruption

Exhaustive Disruption (Algorithm 4) initially creates a superset of all ground facts participating in the inference path of an identified privacy violation. Each member of the superset is evaluated for its ability to eliminate all undesired inferences. For evaluation, we remove all facts of the potential solution set from the initial database. We reason over the altered database, and check whether all violations have been avoided. The final solution is one that avoids all privacy violations and has the lowest cost.

A valid solution, determined by the absence of detected violations (line 25) is marked accordingly and its cost noted. While iterating though all solutions, a valid solution with the lowest cost seen is tracked (lines 12 and 13). Once all solutions are evaluated, the valid solution with lowest cost is returned as optimal (line 31). Note that a worst case solution (i.e., all facts in the data instance altered to the ontology root) is possible.

Definition 3.4.1 (Valid Data Alteration)**.** Given a data item $d = (c_1, r_1, c_2)$ and ontology O, a valid data alteration of d is $d' = (c_3, r_2, c_4)$, such that:

ALGORITHM 4: ExhaustiveDisruption

Input: DB_0 - initial database
Input: DB_m - minimal initial database
Input: O - Ontology
Input: v - violation threshold label
Input: λ - privacy mapping function
Input: DB_{mf} - fix point of DB_m over O
Input: SV - privacy violating concepts in DB_{mf}
Output: $\bar{S}$ - selected solution

```
1  begin
2      SS = SolutionSet(DB_m, O, DB_mf, SV)      // Comment Candidate
        Powerset
3      minCost = +∞
4      forall S ∈ SS do
5          DB_w = AlterData(DB_0, S)
6          DB_wf = Reason(DB_w, O)
7          C = set of inferred facts from DB_wf
8          SV' = PrivacyMapAndDetect(C, v, λ)
9          if SV' = ∅ then
10             set solution state of S to "valid"
11             if solution cost of S < minCost then
12                 minCost = solution cost of S    // track minimum cost
13                 S̄ = S                          // track best solution
14             end
15         else
16             set solution state of S to "invalid"
17         end
18     end
19     return S̄
20 end
```

- $c_3 = c_1$, c_3 is the immediate parent or grandparent concept of c_1 in O, or c_3 is the root concept of O; i.e., (c_1,ISA,c_3) in O.
- $r_2 = r_1$, r_2 is the immediate parent or grandparent relation of r_1 in O; i.e., (r_1,ISA,r_3) in O.
- $c_4 = c_2$, c_4 is the immediate parent or grandparent concept of c_2 in O, or c_4 is the root concept of O; i.e., (c_2,ISA,c_4) in O.

By "immediate" parent and grandparent, we mean those parent and grandparent relations that exist after the transitive "ISA" edges are removed from ontology O.

In our initial work, we focus only on the data item alteration of the object component of a triple (i.e., c_2 in (c_1, r, c_2)). We also limit our alterations to component removal and generalization. For our purposes, removal means we replace the component with the ontology's root component – we denote removal of component a as $\not{a}$. Generalization means we replace the component with its parent or grandparent based on ISA relations in the given ontology – we denote generalization of a to its parent as a$'$ and generalization to is grandparent as a$''$.

Algorithm 5, solution set, returns the power set of all facts that contributed to the inference that violates the privacy policy. We define a solution set as a set of one or more solutions. A solution is defined as a 2-tuple comprised of an alteration set and a solution status. An alteration set is one or more alterations, each a 3-tuple comprised of original RDF triple, object of altered triple, and alteration cost. Solution status is a 2-tuple comprised of solution state and solution cost. Values for solution state are "unknown", "valid", and "invalid". Alteration and solution costs are real numbers.

Example 3.4.1. Let SS be a solution set with one solution S such that S = ((((Dave, hasDisease, CHF), CV, 0.5), ((Dave, takesMed, Med-3.1-4), Med-3, 0.75)), (valid, 1.25)). The solution S is interpreted as follows: the first alteration alters (Dave, hasDisease, CHF) to (Dave, hasDisease, CV) at a cost of 0.5, the second

alteration alters (Dave, takesMed, Med-3.1-4) to (Dave, takesMed, Med-3) at a cost of 0.75. The solution state is "valid" with a total solution cost of 1.25.

To build the solution set SS, Algorithm 5 first evaluates each rule which concludes a detected privacy violation. The rule evaluation is performed by Algorithm 6 which evaluates the lineage of the fact by investigating each satisfied fact in the left side (body) of the rule. If the fact is grounded, it is added to the candidate list. If the fact is inferred, then Algorithm 6 is called recursively on the rule which generated the inferred fact. At completion, Algorithm 5 returns a candidate list containing all candidate facts contributing either directly or indirectly to generation of a detected privacy violations.

Once the candidate list is built, the solution set is constructed as a quaternary powerset of candidate list members.

The solution set is built to support an exhaustive approach and considers all possible alteration options for each candidate. A candidate list of size n will produce $2^{2n} - 1$ solutions (see Example 3.4.2).

ALGORITHM 5: Solution Set

Input: DB_m - minimal initial database
Input: O - Ontology
Input: DB_{mf} - fix point of DB_m over O
Input: SV - privacy violating concepts in DB_{mf}
Output: SS - powerset of candidate facts

```
1 P = ∅                                  // P - remember paths followed
2 C = ∅                                  // C - candidate list
3 forall v ∈ SV do
4 |   r = rule satisfied by O and DB_m which generates v
5 |   GetCandidates(DB_m,O,DB_mf,r,P,C)
6 end
7 SS=QuaternaryPowerset(C)
8 return SS
```

Example 3.4.2. Let v be a data item identified as a privacy violation. Let $r1$ be a rule that generated v, $r1 : A \wedge B \rightarrow v$. The candidate list would consist of

$\{A, B\}$. Each element in the candidate list would have valid 3 alteration options (i.e., for A, options would be A', A'', A'''). The solution set would consist of the following solutions: $\{\{A \to A'\}, \{A \to A''\}, \{A \to A'''\}, \{B \to B'\}, \{B \to B''\}, \{B \to B'''\}, \{A \to A', B \to B'\}, \{A \to A', B \to B''\}, \{A \to A', B \to B'''\}, \{A \to A'', B \to B'\}, \{A \to A'', B \to B''\}, \{A \to A'', B \to B'''\}, \{A \to A''', B \to B'\}, \{A \to A''', B \to B''\}, \{A \to A''', B \to B'''\}\}$. See Figure 3.4.

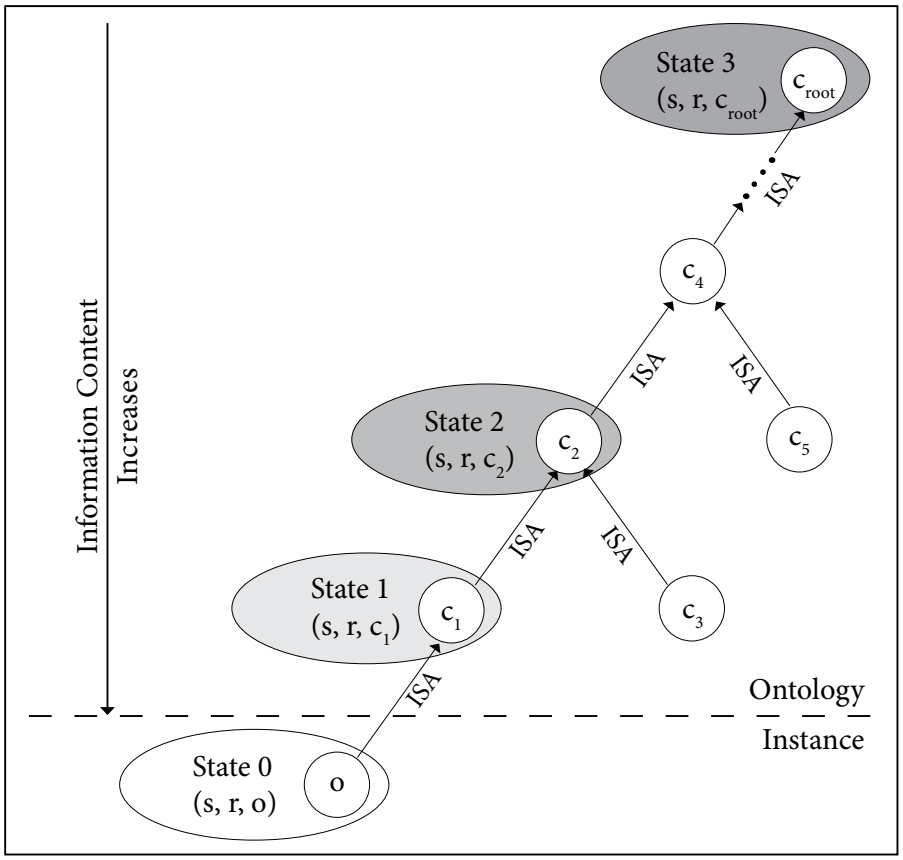


Figure 3.4 Solution States used in generalization. Replacement of the RDF triple object's concept is based on position in the ISA hierarchy of the ontology. Information detail is lost as we generalize up the hierarchy.

Alteration options are defined which perform either full removal or either one or two levels of generalization. For convenience, we define a mapping function θ defined

as $\theta : (O, c) \rightarrow \hat{c}$ to assist in building generalizations. θ maps an ontology, O, and a concept, c, to $\hat{c}$, the parent of concept c in the ontology hierarchy (i.e., c,ISA,$\hat{c}$). If no parent is defined for c in O, θ maps to the root of the ontology. θ only considers "immediate" ISA relationships and ignores transitive ISA relationships.

ALGORITHM 6: GetCandidates

Input: DB_m - minimal initial database
Input: O - Ontology
Input: DB_{mf} - fix point of DB_m over O
Input: r - rule to follow
Input: P - rules processed
Input: C - Candidates
Output: C - Updated Candidates

```
1  F = facts on left side (body) of rule r
2  forall f ∈ F do
3  |  if f ∈ DB_m then
4  |  |  f is ground fact
5  |  |  if f ∉ C then
6  |  |  |  add f to C                          // add to candidate list
7  |  |  end
8  |  else
9  |  |  f is inferred fact
10 |  |  r' = rule satisfied by O and DB_m which generates f
11 |  |  if r' ∉ P then
12 |  |  |  add r' to P                          // remember path
13 |  |  |  GetCandidates(DB_m,O,DB_mf,r',P,C)   // recursive call
14 |  |  end
15 |  end
16 end
17 return C                                      // updated candidate list
```

Claim 3.4.2 (Candidate List Complete)**.** *The candidate list, C, includes all facts that contribute to a privacy violation.*

Proof of Candidate List Complete. Assume, by contradiction, that data item f contributes to the generation of a privacy violation v, but $f \notin C$. Let r be the rule that generates v. There are two ways which f can contribute to the generation of v, direct or indirect. If direct, then f is a ground fact which satisfies rule r. In this case,

f would be identified as contributing to v in Algorithm 6 lines 2 (component of rule r) and 3 (ground fact) and added to C (line 9). Since $f \notin C$, we know that f does not directly contribute to v, therefore, f must indirectly contribute to the generation of v. In this case, f is a ground fact satisfying a rule in v's inference path. So, f must satisfy some rule r_i such that the head of rule r_i is in the body of rule r_{i+1}, $i = 1, \ldots, n$, and the head of rule r_n is in the body of rule r. The inference path is followed by the recursive calling of Algorithm 6 on line 19. When the recursive call reaches a rule where f is a ground fact satisfying that rule, f would be identified as contributing in lines 2 and 3 and added to C (line 9). This is a contradiction to our assumption that f contributed to the generation of v but $v \notin C$.

□

Claim 3.4.3 (Candidate List Sound)**.** *The set of solutions computed by Algorithm 5 contains only data items that contribute to a privacy violation.*

Candidate List Sound. Trivially follows. □

Inference disruption solutions are built using combinations of data from the candidate list, denoted C, built in Algorithm 6. The solution set, SS, is constructed by Algorithm 7. The solution set is exhaustive and incorporates all alteration options for each candidate concept. Our approach is to build a modified powerset based on a quaternary counter. Each bit of the quaternary value represents a concept in C and the state of the bit represents an alteration option. Note that counting is performed with bits reversed (i.e., 10, 20, 30, 01, 11, 21, etc.). The solution set will contain $t^q - 1$ solutions where t is the number of alteration options and q is the number of concepts in the candidate list. When counting, the quaternary value with all 0's is ignored as it creates an empty solution.

Example 3.4.3 (Quaternary Powerset)**.** Let C be a candidate list with two concepts, a and b. Alteration options for a would be a', a'', and $\not{a}$. Alteration options for b are

constructed similarly. The quaternary counter increments from 00 to 33 as shown in Table 3.3. Counting produces 15 solutions ($2^4 - 1$) which consider all concepts and alteration options.

Table 3.3 Solution generation (quaternary) example values showing the count and corresponding solution.

Count	Solution		Count	Solution		Count	Solution
10	a'		21	$a''\ b'$		32	$\not{a}\ b''$
20	a''		31	$\not{a}\ b'$		03	$\not{b}$
30	$\not{a}$		02	b''		13	$a'\ \not{b}$
01	b'		12	$a'\ b''$		23	$a''\ \not{b}$
11	$a'\ b'$		22	$a''\ b''$		33	$\not{a}\ \not{b}$

Claim 3.4.4 (Violation Detection and Removal)**.** *All generated data items identified as privacy violations are removed.*

Proof of Violation Removal. Assume there is a generated data item s that is a privacy violation, but s is present in the authorized data instance. For s to be present in the authorized data instance, it must either have existed in the initial data set or been added by the reasoning process. It is assumed that all data items in the initial data set are authorized, therefore s was added by the reasoning process. The authorized data instance is created in Algorithm 1 Step 10 by altering data items based on the selected solution. The data items needed to interrupt the inference of s must therefore not be in the selected solution. The selected solution is taken from the set of all solutions in Algorithm 4 Step 11. This selection is based on the solution flag being set to true. The solution containing s must have its flag set to true. But the solution flag is only set to true if no violations are found for the solution. Therefore s was not a violation. This is a contradiction. □

ALGORITHM 7: Quaternary Powerset

Input: C - Candidates
Output: SS - Solution Set

```
1  t = 4                                          // do quaternary counting
2  q = |C|                                        // candidate list size
3  b[0...q − 1] = 0                               // initialize bit mask
4  p=4^q                                          // number of solutions
5  for i = 0 to p do
6      add 1 to b[0]                                          // add base 4
7      for j = 0 to t do
8          if b[j] < t then
9              exit inner loop                          // no carry digit
10         else
11             b[j] = 0
12             add one to b[j+1]                          // carry digit
13         end
14     end
15     new solution set
16     solutionCost = 0
17     for k = 0 to q do
18         if b[k] > 0 then include fact k
19             new alteration
20             alterationCost = 0
21             alteration source RDF tripple = C[k]
22             case b[k] do
23                 case 1 : do
24                     altered target object = θ(O, C[k])
25                     add 0.5 to alterationCost
26                 end
27                 case 2 : do
28                     altered target object = θ(O, θ(O, C[k]))
29                     add 0.75 to alterationCost
30                 end
31                 case 3 : do
32                     altered target object = root concept from ontology O
33                     add 1.0 to alterationCost
34                 end
35             end
36             add alteration to solution set
37             add alterationCost to solutionCost
38         end
39     end
40     add solution to SS
41 end
42 return SS
```

ALGORITHM 8: AlterData

Input: DB - instance database
Input: S - solution set
Output: DB_F - updated instance database

```
1 DB_F = DB                                   // create copy of DB
2 forall ∈ S do
3 |  process all alterations in the solution
4 |  a' = alteration RDF triple
5 |  DB_F = DB_F − a'    // remove original RDF Triple from DB_F
6 |  update object of triple a' to alteration object
7 |  DB_F = DB_F ⋃ a'    // insert altered RDF Triple into DB_F
8 end
9 return DB_F                                 // return altered DB
```

Table 3.4 Cost Values for generalization actions.

Action	Cost
remove c	+1.0
add root ($\not\in$) - removal	-0.0
add 1 level generalized (c')	-0.5
add 2 levels generalized (c'')	-0.25

Solution Cost

To select a minimal participating fact combination (PFC) which provides unauthorized inference disruption, Algorithm 4 uses an exhaustive approach and evaluates all alteration options for every PFC. The selected minimal alteration combination is based on the cost of PFCs which are successful in disrupting all unauthorized inferences. Our cost model is implemented in two parts: 1.) determine individual concept alteration costs, and 2.) determine total PFC alteration cost.

Definition 3.4.2 (Concept Alteration Cost)**.** Concept Alteration Cost, denoted as $\boldsymbol{Cost}_f$, is defined as follows. Let $f = (c_1, p_1, c_2)$ be a concept that participates in a violation inference and therefore a candidate to defeat the violation inference. We say that the concept alteration cost is a value from 0.5 to 1.0 that indicates the cost of altering f. We use the values in table 3.4 to indicate the cost based on the specific

alteration being performed on f (One level of alteration = 0.5, two levels of alteration = 0.75, removal = 1.0).

The value of $\boldsymbol{Cost}_f$ is calculated in Algorithm 7 by the **CASE** statement enclosing lines 22 through 35. Within this group of statements, the appropriate value from Table 3.4 is added to variable "alterationCost" depending on the specific alteration being performed. This variable holds the value of $\boldsymbol{Cost}_f$ for the current data item (f) being processed.

Definition 3.4.3 (Combination Alteration Cost)**.** Concept Alteration Cost, denoted as $\boldsymbol{Cost}_S$, is defined as follows. Let $S = \{f_1, \ldots, f_n\}$ be a set of concepts that are altered to disrupt a group of violation inferences and $\boldsymbol{Cost}_{f_i} (i = 1, \ldots, n)$ the Concept Alteration Cost for f_i. The Combination Alteration Cost for S is

$$\boldsymbol{Cost}_S = \sum_{i=1}^{n} \boldsymbol{Cost}_{f_i}.$$

Intuitively, we say that the combination alteration cost for a set of concepts is the sum of the concept alteration costs for all concepts in the set.

The value of $\boldsymbol{Cost}_S$ is calculated in Algorithm 7 by statement 37 which is part of the loop over all data items in the PFC. This statement accumulates the $\boldsymbol{Cost}_f$ values in the variable "solutionCost" for all data items in the PFC. At termination of the loop, this variable holds the value of $\boldsymbol{Cost}_S$ for the set of data items processed.

In our model, the total cost of a PFC alteration is based on the number of facts in the combination and the cost of specific alterations performed on those facts (level of generalization or removal). Example 3.4.4 shows the PFC alteration cost of a two fact combination. This example assumes both concepts must be altered to disrupt the violations. The fact alteration cost values in this example range from 1.0 (both concepts are generalized 1 level) to 2.0 (both concepts are removed).

Example 3.4.4 (Alteration Cost)**.** Let a and b be concepts in PFC s. The ontology root for a and b are denoted $\not\phi$ and $\not\phi$ respectfully. First and second level generalization

of a and b are denoted a' a'', b' and b'' respectfully. Assume that both a and b must be altered to disrupt an identified violation. Using the values in Table 3.4 and the definitions for $\boldsymbol{Cost}_f$ and $\boldsymbol{Cost}_S$, the cost for each combination of alterations for s is shown in Table 3.5.

The most costly combination $\not{a}$, $\not{b}$ removes both concepts (replaces with root concept). The least costly solution a', b' replaces both concepts with their first level generalizations.

Table 3.5 Cost Alteration Example - Assumes both concepts must be generalized to disrupt inference.

Replace a, b with:	$\boldsymbol{Cost}_S = \boldsymbol{Cost}_f(a) + \boldsymbol{Cost}_f(b)$
$\not{a}$, $\not{b}$	2.0
$\not{a}$, b'	1.5
$\not{a}$, b''	1.75
a', $\not{b}$	1.5
a', b'	1.0
a', b''	1.25
a'', $\not{b}$	1.75
a'', b'	1.25
a'', b''	1.5

Once all costs are calculated, each PFC is evaluated to determine if it successfully disrupts all the identified unauthorized inferences. As PFCs are being evaluated by Algorithm 4, the two **If** statements at steps 25 and 11 allow us to track the lowest cost solution that was successful in disrupting all unauthorized inferences.

3.4.2 Properties & Proofs

Theorem 3.4.1 (Disrupt Violations)**.** *Algorithm 1 computes an authorized data instance, denoted DB_s for user u with privacy clearance $\lambda(u)$ from an initial data instance, denoted DB_i. The algorithm terminates and DB_s is sound, complete, and minimal.*

Proof. First, Algorithm 1 must terminate since the input data set is finite and all solution candidates are based on members of the initial data set's powerset. Since the initial data set is finite, the powerset is also finite. Therefore, there is a finite number of solutions that could be evaluated, leading to termination. We still need to show that the computed data set DB_f is

1. sound,
2. complete, and
3. minimal.

□

Proof of soundness. Assume that s is a data item that does not logically follow from DB_i and domain knowledge, but $s \in DB_s$. If $s \in DB_i$, it must logically follow since it is already present in DB_i. If $s \notin DB_i$, then s must be generated by the inference process. However, in the inference process every new data item generated must come from a rule of the form $p_1 \wedge \cdots \wedge p_k \rightarrow q$ and all components of the rule body must either be in DB_i or there must be an inference path that generates from DB_i. Recursively, we can show that there must be an inference path that generates each component of the rule body not in DB_i. If every component in the body of the rule generating s can be generated from DB_i, then s naturally follows from DB_i and domain knowledge. This contradicts our initial statement that s does not follow from DB_i and domain knowledge. □

Proof of completeness. Assume that s is a data item that logically follows from DB_i and domain knowledge, but $s \notin DB_s$. If $s \notin DB_s$, the following must hold: $s \notin DB_i$ and no inference path exists that concludes with s. However, if s logically follows from DB_i and domain knowledge, then either $s \in DB_i$ or the inference process generates a path that concludes with s. If $s \in DB_i$, then it naturally follows that

$s \in DB_s$. If $s \notin DB_i$, but an inference path exists that concludes in s, then it follows that $s \in DB_s$. In either case, $s \in DB_s$. This contradicts our initial statement that $s \notin DB_s$. □

Proof of Minimality. Assume there is a valid solution q that is not selected as the final solution, but has the lowest cost. Algorithm 4 determines the final solution. Since q is a valid solution, it returns no violations and is identified as valid in Step 25. The solution cost for q is then compared to the lowest cost of all solutions processed to that point in Step 11. Since, as stated, q has the lowest cost, no other lower cost solution will be identified by Step 11. Therefore, at the end of the evaluation loop, q will be selected as the final solution. But this contradicts our initial statement that q has lowest cost and is not selected. □

Chapter 4

Exhaustive Disruption

4.1 Implementation & Empirical Results

We developed a prototype implementation of our inference disruption framework written in the Java programming language. The prototype uses the Apache Jena semantic framework [2] for RDF parsing and reasoning. The implementation logic closely follows the algorithms described in section 3.4.1. As described in section 3.3.1, external data is structured using the RDF standard [15].

The execution steps of a sample prototype execution are shown in Fig. 4.1.

In this execution, the initial database consisted of 3780 data items. The inference process generated an additional 3944 data items, giving a total of 7724 triples in the database. There are two data items in the database identified as privacy violations. We found six ground triples that participate in the violation inference paths and a total of $4^6 - 1$ or 4095 solution sets were constructed and evaluated. Of the successful solutions, we found that solution number 4 had the lowest cost (100) and would be recommended as optimal to disrupt the identified violations.

To evaluate computational efficiency, we developed a series of test database input files as described in Table 4.1. We arranged these files into three groups based on the number of privacy violations known to exist (0, 1, 2, or 4). Within each group, we provide five files based on patient counts in the file (100, 500, 1000, 2500, and 5000). The total number of facts, including asserted and inferred, for each file is shown in the table. In our test scenarios, each violation was supported by tree ground facts

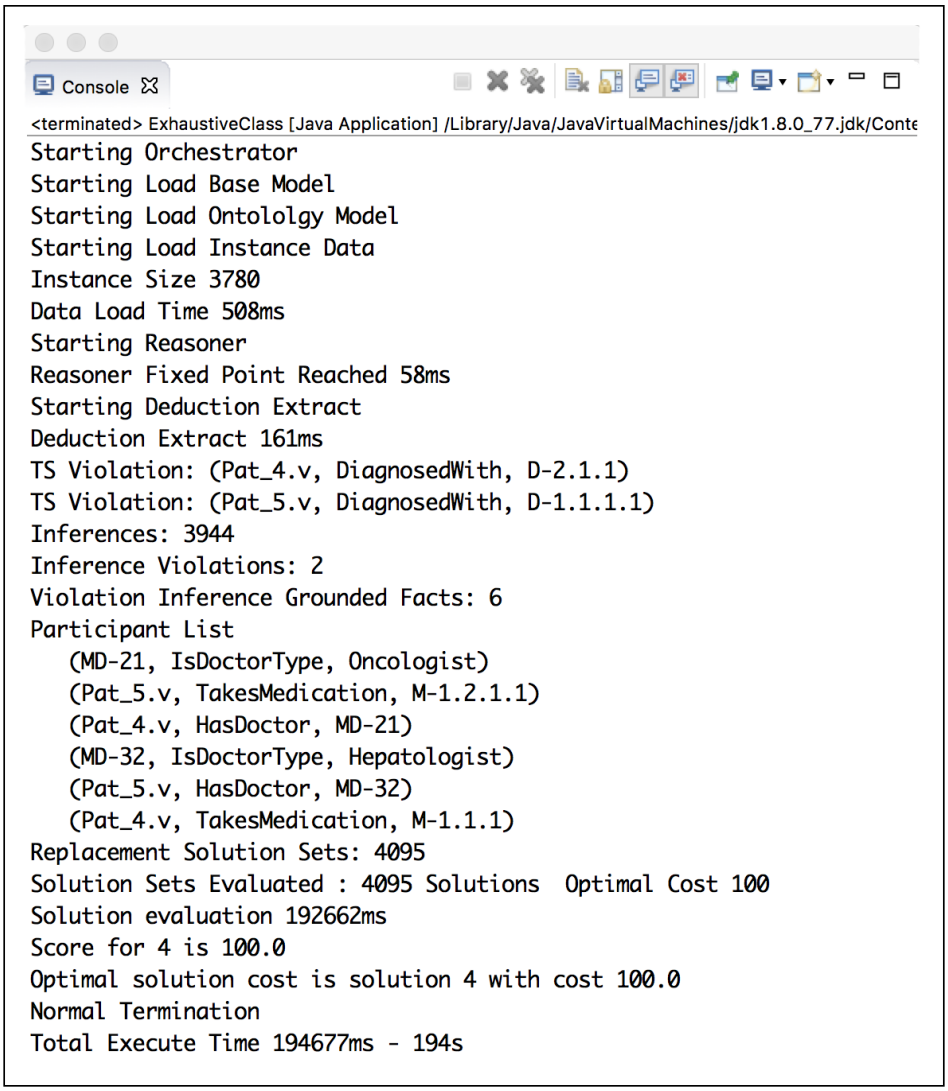


Figure 4.1 Prototype Execution. Screen shot of prototype execution. This run found two privacy violations based on 6 ground facts.

(participants) satisfying multiple rules. As described in the discussion of Algorithm 5, the number of solution sets is $(4^p - 1)$ where p is the number of ground triples in the inference paths of all violation data items. As shown in Table 4.1, our test data has solution set sizes of: $4^0 - 1 = 0$, $4^3 - 1 = 63$, and $4^6 - 1 = 4095$. In each test execution, the expected violations were identified and proper privacy labels applied. A valid solution with minimal cost was found and reported for each test execution.

Table 4.1 Prototype execution summary timing collected for 100, 500, 1000, 2500, and 5000 instance databases grouped by violation count.

Patients	Asserted	Inferred	Facts	Violations	Participants	Solutions	Load Time	Reason Time	Discover Time	Evaluate Time	Total Time
100	369	376	745	0	0	0	149	36	26	0	745
500	1741	1806	3547	0	0	0	246	31	74	0	926
1000	3456	3592	7048	0	0	0	307	35	98	0	1019
2500	8596	8947	17543	0	0	0	481	28	218	0	1387
5000	17168	17875	35043	0	0	0	781	27	459	0	1952
100	403	414	817	1	3	63	155	31	29	930	1744
500	1904	1983	3887	1	3	63	281	47	86	2190	3193
1000	3779	3941	7720	1	3	63	341	29	113	3506	4617
2500	9400	9811	19211	1	3	63	471	28	203	8334	9707
5000	18777	19606	38383	1	3	63	746	48	528	16103	18189
100	404	417	821	2	6	4095	173	44	28	21403	22593
500	1905	1986	3891	2	6	4095	267	30	57	80630	81837
1000	3780	3944	7724	2	6	4095	319	30	88	163234	164497
2500	9401	9814	19215	2	6	4095	493	26	209	443200	444847
5000	18778	19609	38387	2	6	4095	747	28	393	927606	929728
100	403	414	817	4	10	1048575	158	36	20	3657618	3691252
500	1904	1983	3887	4	10	1048575	238	35	70	18322095	18357303
1000	3779	3941	7720	4	10	1048575	264	37	110	33965367	33996151
2500	9400	9811	19211	4	10	1048575	446	37	236	74842091	74869826
5000	18777	19606	38383	4	10	1048575	721	36	479	148937788	148965619

As discussed in Section 9, cost is determined by removal and addition of information. For the file group with zero violations, the cost is obviously 0. For the one violation file group, the cost for all solutions ranged from 0.5 to 3.0, with the optimal cost of a successful solution being 0.5. For the two violation file group, the cost for all solutions ranged from 0.5 to 6.0, with the optimal cost of a successful solution being 1.0. A histogram of the one and two violation costs is shown in Figures 4.2 and 4.3 respectively.

Execution times were captured during each test execution and are shown in Table 4.1. The timing values are reported in milliseconds and labeled as follows:

- *Load* - The time to load the external ontology and instance (RDF) files and

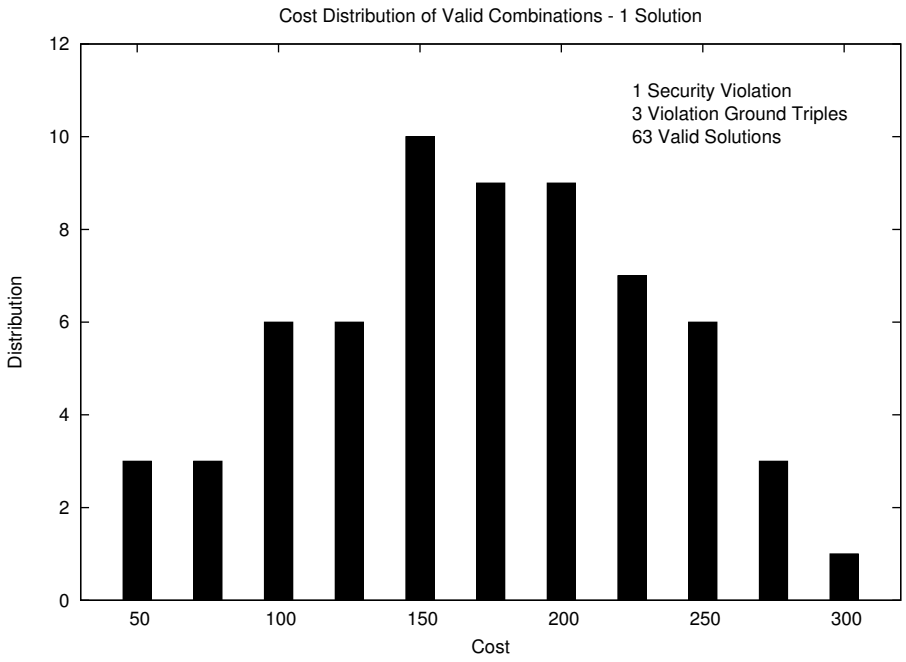


Figure 4.2 Solution Cost Distribution 1. Graph for one privacy violation with three ground facts in its inference path.

build the internal Jena data model.

- *Reason* - The time to execute the Jena RDF rule reasoner over the internal data model.
- *Discover* - The time for Jena to identify and return all generated data items along with their inference paths, associated rules, and ground triples.
- *Evaluate* - The cumulative time to evaluate each potential solution. This time includes iterative executions of the reason and discover logic.
- *Total* - This is the total time from execution start to finish.

We observe that data load time increases with external data file size, which is expected. Execution time for the Jena RDF rule reasoner is similar for all executions regardless of model size or solution set size. We notice the discover time increases as

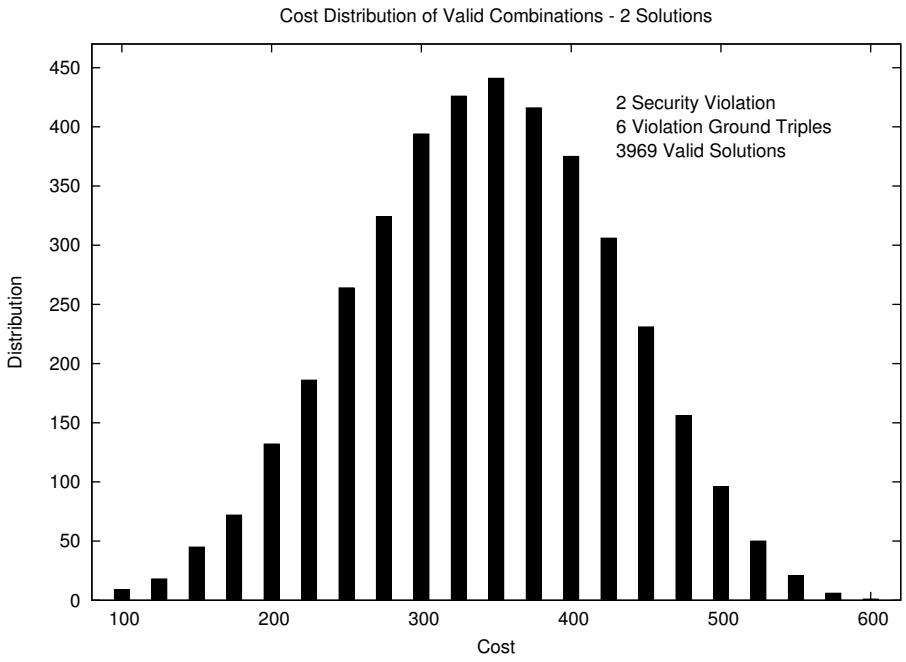


Figure 4.3 Solution Cost Distribution 2. Graph for two privacy violation with six ground facts in their inference paths.

the data model size increases. There are two primary tasks which contribute to the evaluate time; these tasks are reason and discover, each are executed once for each solution in the solution set. As can be seen in the Table 4.1, increasing the violation count from zero to one to two also increases the solution set size from 0 to 63 to 4095 respectively. This increase in solution set size causes the total run time for to increase from 2 seconds to 18 seconds to approximately 15 minutes. While the reason time is fairly constant, the discovery time does increase with data model size, this increase however appears to have linear growth. The solution set size, on the other hand, has a large impact on total execution time since it grows exponentially with each new triple found in a violation inference path.

The trends for total execution time can be seen in the graph found in Figure 4.4. In this graph, the X axis shows the size of the data instance (number of patients) for each test file. The Y axis shows $\log_2$ of the total execution time for a given test file.

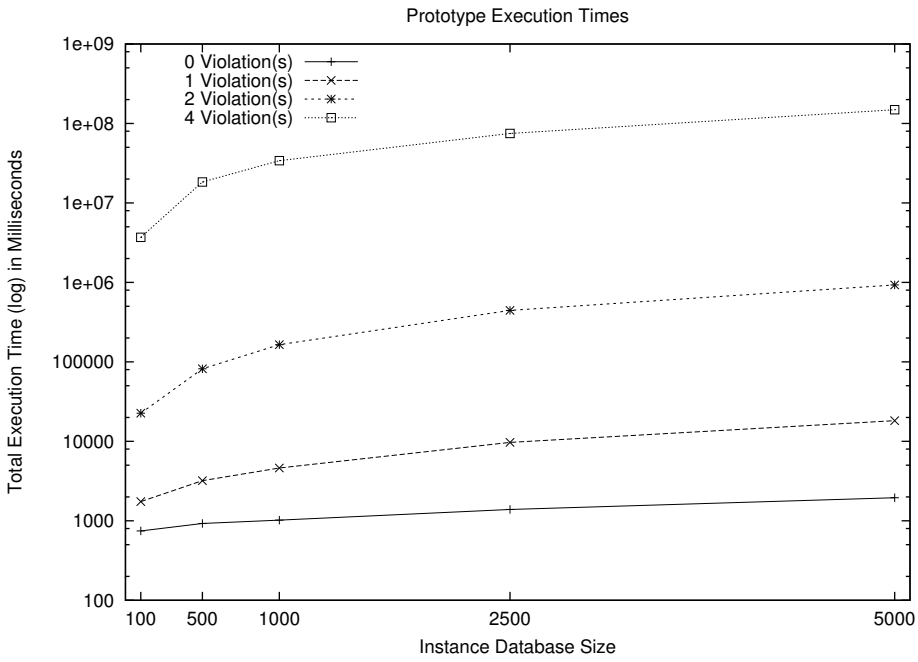


Figure 4.4 Prototype Execution Timing. Trend lines show execution time (log) in milliseconds. Individual line corresponds to number of violations found in data instance.

Testing of our initial prototype implementation used a limited representation of medical data and domain knowledge. Obviously, based on our findings, it is not feasible to run this process on a database containing a large number of facts with violations, but we have shown that we can use domain knowledge to identify and modify data items to disrupt violation inferences. We are currently extending our model to include heuristics to increase efficiency while still providing low cost inference disruption.

4.2 Findings

In our exhaustive approach, the solution set is based on every combination of data items satisfying any rule in the path of a violating inference. The solution set can grow to a computationally prohibitive size if there are many violations or a large

number of data items in the inference rule bodies (n data items will produce a set size of $2^{2n} - 1$). During solution evaluation, we reason over and assess each of these $2^{2n} - 1$ solutions. As can be seen in Table 4.1, database size does increase a single solution's execution time, but solution set size has a larger impact since it determines how many solution evaluations must be performed. In the next section, we investigate heuristics that can help reduce complexity and allow us to find a sufficient solution in a feasible time.

Chapter 5

Efficient Disruption

In this section, we investigate heuristics that our methods can use to reduce overall computational complexity and allow us to find a sufficient solution in reasonable time.

We investigate attributes and characteristics of both generated data and the inference paths that infer them. We look at the interaction between inference paths, both those generating violation data items and those generating authorized data items. We seek to discover information that can be leveraged to increase efficiency in our process. We will incorporate discovered information into our algorithms as appropriate to improve overall process performance.

To improve executional efficiency, we will address three areas that have the potential to reduce total execution time: reduction of PFC set size, optimal PFC evaluation order, and early recognition of a satisfactory solution. We expect improvements based on the following:

- mimimized PFC set size decreases the number of potential solutions that have to be evaluated
- use of additional meta-data to pre-compute a better informed solution cost before PFC evaluation supports heuristic-based decisions
- optimized order of PFC evaluation and the solution selection criteria allowing evaluation of the most likely effective solutions first and quickly recognize a satisfactory solution early in the process.

We feel these enhancements will eliminate the need for a full exhaustive evaluation of all potential solutions.

Initial meta-data to be investigated include:

- Number of authorized inferences a data item participates in
- Number of unauthorized inferences a data item participates in
- Entropy of a data item.

The following sections will discuss reduction in solutions set size, hypergraph disruption covers, an enhanced cost function, order of PFC evaluation, and early solution selection.

5.1 Participating Fact Combination Set Size

To reduce the number of combinations constructed, we first look at eliminating combinations that are not capable of being successful. While we cannot guarantee a given combination will be effective in defeating violation generation, we can verify that certain combinations are not capable of defeating generation of those violations. Early elimination of ineffective combinations will decrease overall processing time compared to the full exhaustive approach. As described in section 3.4, we address violation avoidance by disrupting violation inference paths. The inference path for a given violation is a partially ordered set of one or more satisfied rules; these rules conclude with the generation of a violating data item. Each rule in the path is satisfied by existence of specific data items; if those data items are removed or altered, the rule may not be satisfied and the violation may not be generated. While removal or alteration of an arbitrary data item does not guarantee path disruption, if at least one data item on the path is not removed or altered, the violation will continue to be generated.

Definition 5.1.1 (Inference Disruptor)**.** Let $r : p_1 \wedge \cdots \wedge p_k \rightarrow q$ be an inference rule and Γ a mapping from r to a database DB as defined in Definition 3.3.9. We say that p_i $(i = 1, \ldots, k)$ is an inference disruptor for r if:

- p_i is a ground fact, and
- there is no inference rule $\bar{r}$ such that $\bar{p}_1 \wedge \cdots \wedge \bar{p}_j \rightarrow p_i$.

That is, if $\Gamma(p_i)$ is removed from the database DB, r would not be satisfied by DB and therefore q would not be generated by r.

Definition 5.1.2 (Inference Disruption Cover)**.** Given a database DB and inference rules $r_1, \ldots, r_n$ that generated undesired inferences. We say that the set $P = \{p_1, \ldots, p_l\}$ is a inference disruption cover for DB if:

- For each $r_i (i = 1, \ldots, n)$ there is a $p_j \in P$ such that p_j is an inference disruptor.
- There is no P' such that $P' \subset P$ and for each $r_i (i = 1, \ldots, n)$ there is a $p_j \in P'$ such that p_j is an inference disruptor for r_i.

Definition 5.1.3 (Minimal Inference Disruption Cover)**.** Let P be an inference disruption cover for DB and $\boldsymbol{Cost}_P$ the combination alteration cost of all alterations in P (Definition 3.4.3). We say that P is a minimal inference disruption cover for DB if no P' exists such that P' is also an inference disruption cover for DB and $\boldsymbol{Cost}_{P'} < \boldsymbol{Cost}_P$.

Example 5.1.1 (Inference Disruption Cover)**.** Let there be a set of rules that generate data items as follows:

- r1: $a \wedge b \wedge c \rightarrow x$
- r2: $b \wedge d \wedge e \rightarrow y$
- r3: $c \wedge e \wedge f \rightarrow z$

An inference disruptor for r1 would be any member of the set {a,b,c}; for r2 any member of the set {b,d,e}; and for r3 any member of the set {c,e,f}. Assuming alteration cost of 1 for each fact, the minimal inference disruption cover of the rule set {r1, r2, r3} are {a,e},{b,e}.{c,b}, {c,d}, or {c,e}. This minimal cover requires two elements, thus the minimal cost is 2.

5.2 Hypergraph Cover

In this section, we discuss a method for building a minimal inference disruption cover. We build a minimal spanning tree over a connected set of hypergraphs. We treat the collection of data items in each violation inference path as a hypergraph. We form a minimal spanning tree of these hypergraphs based on common data items. The minimal spanning tree will provide us with a minimal disruption cover.

Since different satisfied rules with the same conclusion may not be based on the same set of ground facts, for our hypergraph cover approach to be complete, we must consider all contributing ground facts from all satisfied rules. We must disrupt all rules with that conclusion to be effective. Therefore, we must consider "same conclusion" rules as linked during evaluation. If an inference disruptor is selected from one of a set of linked rules, then an inference disruptor must be selected from all of the linked rules.

Example 5.2.1 (Inference Disruption)**.** Given the set of satisfied rules $\{r1 : a \wedge b \rightarrow c, r2 : d \wedge e \rightarrow f, r3 : c \wedge f \rightarrow g\}$, the list of participating ground facts for g is $\{a, b, d, e\}$. We can disrupt g, by altering any of the participating ground facts since all must exist to satisfy rules $r1$, $r2$, and $r3$. If we add an additional rule, $r4 : h \wedge i \rightarrow c$, to the set of satisfied rules, the list of participating ground facts for g becmes $\{a, b, d, e, h, i\}$. With the addition of rule $r4$, alteration of participating ground facts e or f will still disrupt generation of g by removing f, but alteration of any one fact in the set $\{a, b, h, i\}$ will not disrupt the generation of g since while

it may prevent one rule from generating c, there is a second rule that can generate c with the remaining facts. To disrupt the generation of c we must disrupt both $r1$ and $r4$ with at least one fact from each of the two sets: $\{a, b\}$ and $\{h, i\}$.

Hypergraph Construction

To illustrate the hypergraph construction, we step through a sample construction. Let there be a set of generated data items {d, g, i, m, p, r} of which three are privacy violations {g, p, r}. Assume the rules that generate these data items, $r_1, \ldots, r_6$, are as follows:

- $r_1 : a \wedge b \wedge c \rightarrow d$
- $r_2 : d \wedge e \wedge f \rightarrow g$
- $r_{3.1} : a \wedge e \wedge h \rightarrow i$
- $r_{3.2} : u \wedge v \rightarrow i$
- $r_4 : j \wedge k \wedge l \rightarrow m$
- $r_5 : i \wedge n \wedge m \wedge o \rightarrow p$
- $r_{6.1} : e \wedge f \wedge q \rightarrow r$
- $r_{6.2} : s \wedge f \wedge t \rightarrow r$

Note that $r_{3.1}$ and $r_{3.2}$ both generate i and $r_{6.1}$ and $r_{6.2}$ both generate r.

Before discussing hypergraph cover construction, we introduce the concept of a disruption container. A disruption container is a grouping of hypergraphs that generate the same conclusion (multiple rules with the same head). Since we must disrupt all hypergraphs associated with a conclusion to truly disrupt the conclusion, we must ensure that the hypergraphs are disrupted as a set. We use the disruption container to enforce the disruption of all hypergraphs within a same conclusion set.

Example 5.2.2 (Disruption Container)**.** Let there be three privacy violations g, p, and r as shown in Figure 5.3. Looking at violation p, we have shown that it can be disrupted by altering any one fact in the set $\{j, k, l, n, o\}$. But the hypergraph for p includes a disruption container. This container holds two hypergraphs that concluded the fact i (see Figure 5.2). If we alter any single fact found in the container ($\{u, v, a, e, h\}$), we would disrupt one rule generating i, but not the other. This would allow the violation p to still be generated. We need to alter a member of both contained hypergraphs {u,v} and {a,e,h} to completely disrupt i and avoid the generation of p.

To construct the hypergraph set, we transform the fired rules as follows:

1. Create a directed acyclic graph (DAG) for each rule, such that edges point from the data items in the body to the data item of the head (see Figure 5.1). Generated violation data items are shown in an octagon, non-violation data items are in a square.

2. Connect rules, such that there is an edge from the head of a rule r_i to the head of a rule r_j iff r_i's conclusion is in the body of r_j. Figure 5.2 shows resolution of rule dependencies on generated data items and graphically links the components of inference paths together. Note that rules that form the same conclusion are grouped in a disruption container. In these cases, we consider the container to be a potential inference disruptor so both rules (AND) must be disrupted for the container to be a valid disruptor.

3. Create hypergraphs from each DAG by creating a hyperedge with same name as the head of the DAG and vertices corresponding to every data item in the DAG that is not the head of a sub-DAG (i.e., does not have an edge pointing to it). Figure 5.3 shows each of the individual graphs as a hypergraph with generated data items removed from the hypergraph and violation hyperedges

named based on the associated privacy violation. Note that hypergraphs in a disruption container remain distinct from each other.

4. Connect hypergraphs by connecting their common vertices. Figure 5.4 shows violation hypergraphs linked with a dashed line connecting their vertices (common ground facts).

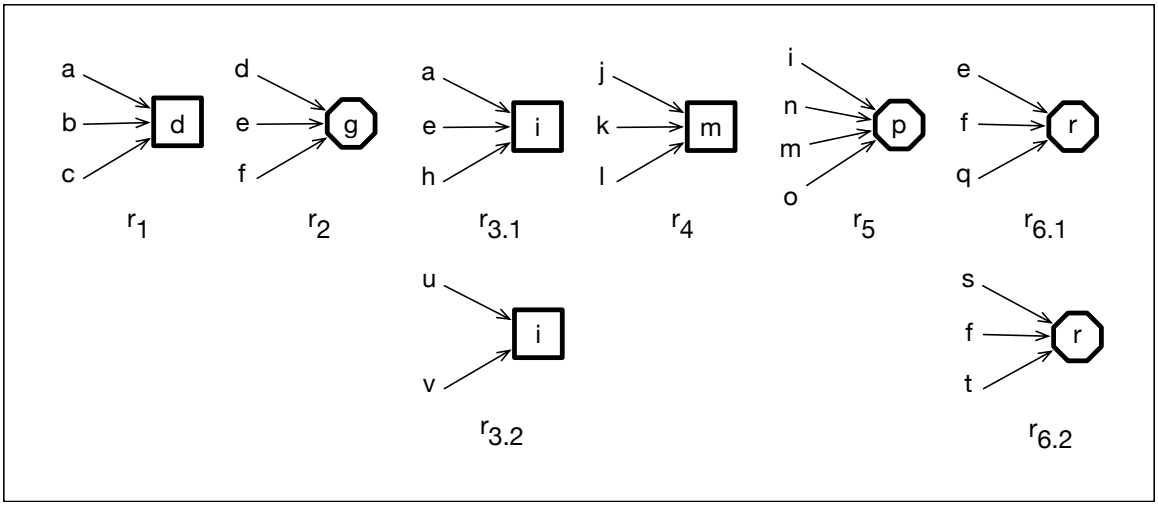


Figure 5.1 Shows all satisfied rules, not just ones generating new data (2 rules generate "i" and "r"). Octagons indicate violations, squares are safe inferences.

Dependent Cover

In this section, we look at violations that do not need to be disrupted and therefore are not included in our hypergraph cover. If we have the case where one violation is dependent upon another (see Figure 5.5), we need only address the violation being depended on. In Figure 5.5, if we disrupt violation i then violation d will still be generated, but if we disrupt d then i will not be generated since it depends on the existence of d. We can see from the hypergraph in Figure 5.6 that $d \subseteq i$.

Claim 5.2.1 (Dependent Disruption)**.** *Disruption of an inference subset will also disrupt the inference superset which contains it.*

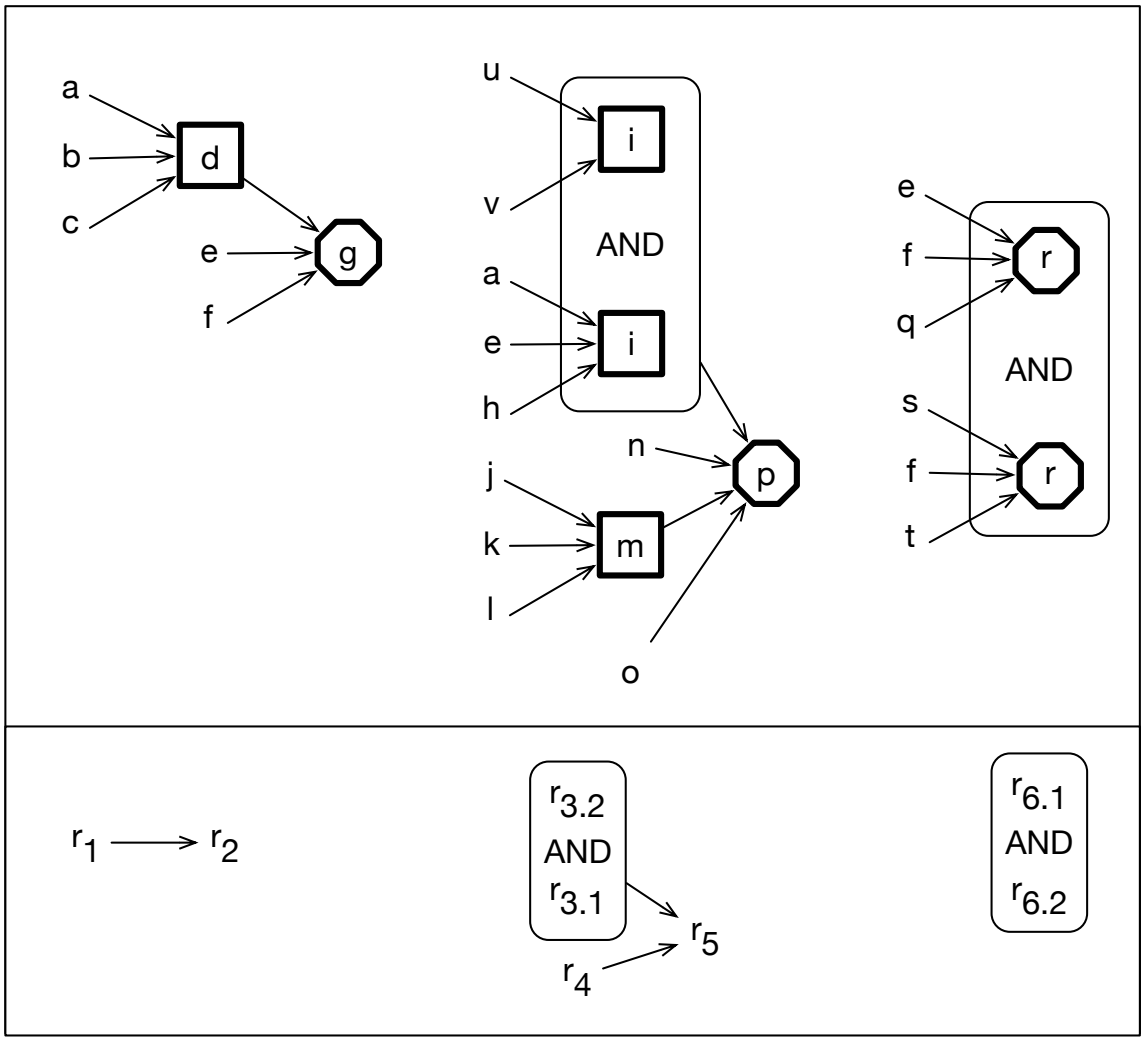


Figure 5.2 Expansion of Figure 5.1 showing rule dependencies and disruption containers.

Proof of Dependent Disruption. Let inference data item i be dependent on inference data item d as shown in Figure 5.5. Assume by contradiction, that data item d is disrupted and no longer generated, but data item i is still generated. Since the generation of data item i is based on the inference rule $(d \wedge h \wedge g \rightarrow i)$, then d, h, and g must all exist to satisfy the rule body and cause the rule head to generate i. But, we have said that the rule generating d was disrupted and d was not generated and therefore does not exist. Without the existence of d, the inference rule body

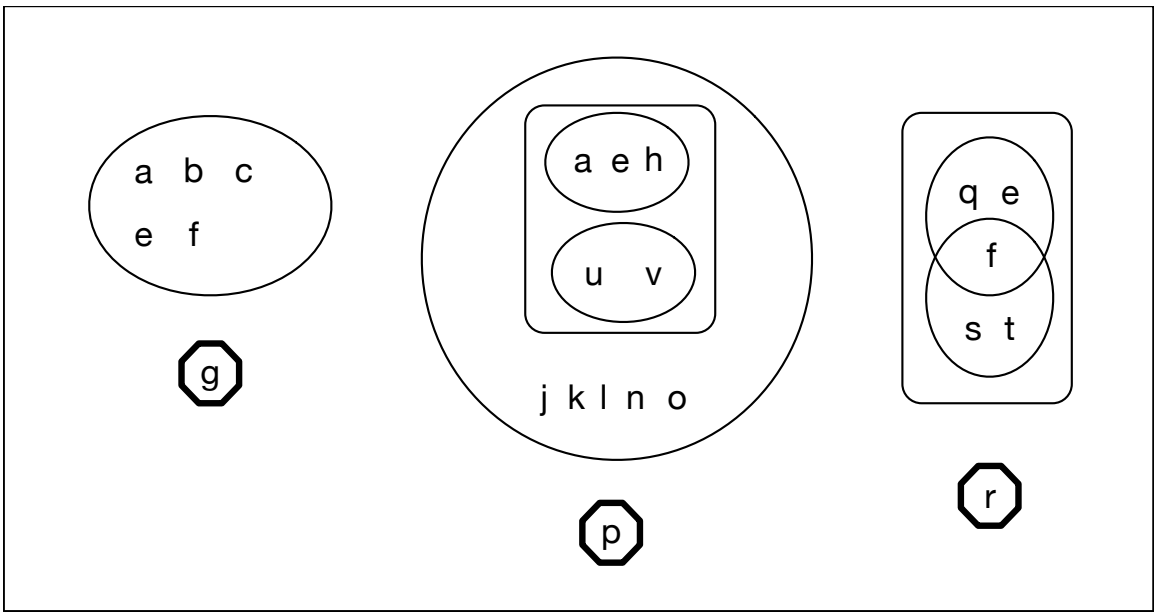


Figure 5.3 Rules from Figure 5.1(b) represented as hypergraphs.

would not be satisfied and i would not be generated. This is a contradiction to our assumption that d was disrupted but i was still generated. □

If there are two violation hypergraphs, A and B, and $A \subseteq B$, then we do not need to disrupt the path for B directly because we are disrupting it indirectly by disrupting the path for A. Given Claim 5.2.1, we can remove any superset hypergraphs from the hypergraph set prior to determining disruption covers. Removal of these superset hypergraphs may reduce the number of participating ground facts that need to be considered in PFC construction, reducing the final PFC size and avoiding redundant and unnecessary evaluations.

Hypergraph Cover Approach

For our hypergraph cover approach, we treat ground facts and disruption containers within hypergraphs as items that are logically connected with the OR operator. In other words, only one item (ground fact or disruption container) needs be altered to disrupt the hypergraph. We treat the contents of a disruption container as a group

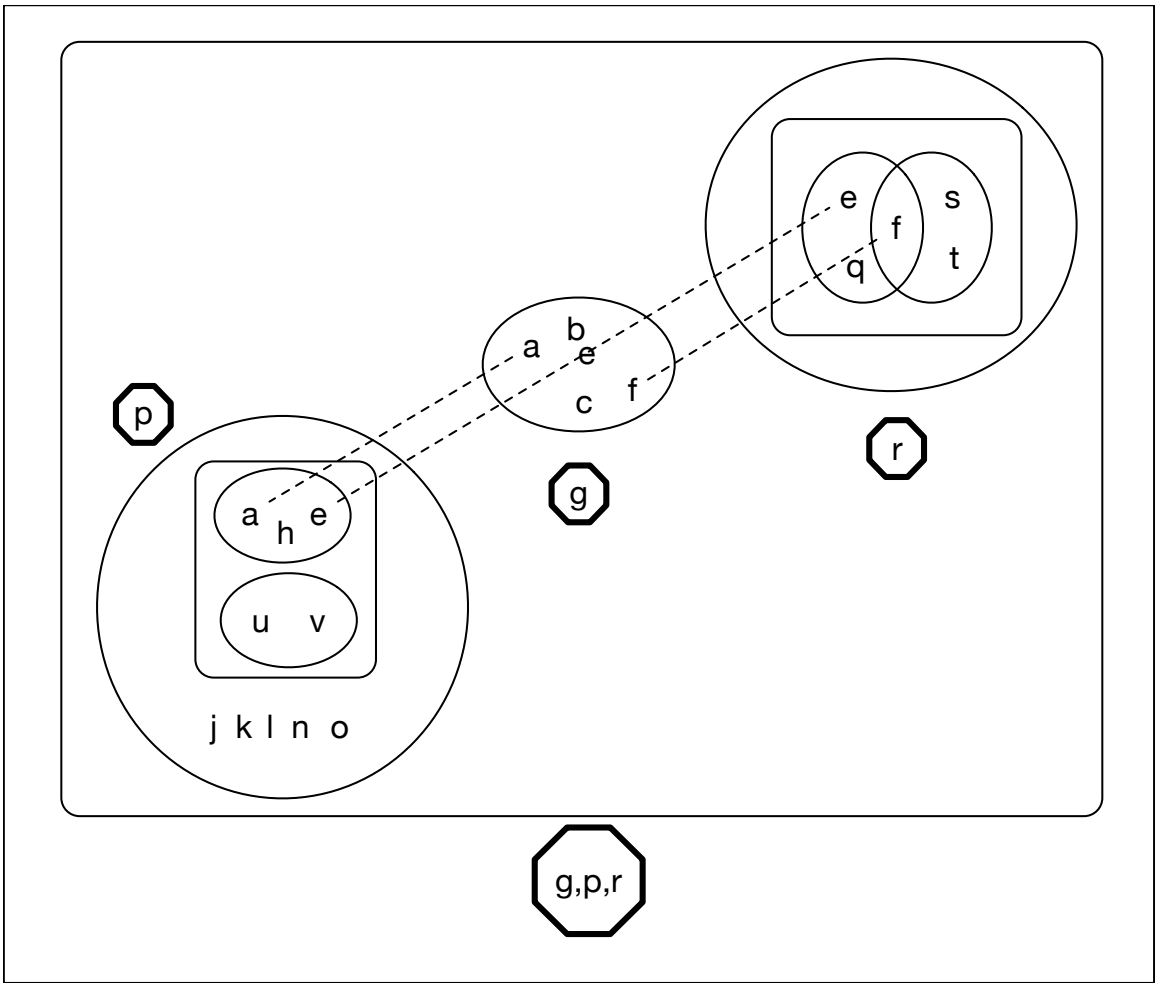


Figure 5.4 Shows Figure 5.3 with like concepts linked.

of hypergraphs that are logically connected with the AND operator. In other words, all hypergraphs within a disruption container **must** be disrupted for the container to be disrupted.

After building the hypergraphs as described earlier, we use their construction to build a series of disruption sentences which form disruption sentence covers.

Definition 5.2.1 (Disruption Sentence)**.** Given a rule $r : p_1 \wedge \cdots \wedge p_n$, a disruption sentence of r, denoted as DS_r, is an **OR** of all $p_i (i = 1, \ldots, n)$, i.e., $p_1 \vee \cdots \vee p_n$.

Example 5.2.3. Let r be a rule, $r : u \wedge v$ and $DS_r = u \vee v$. Disrupting either of

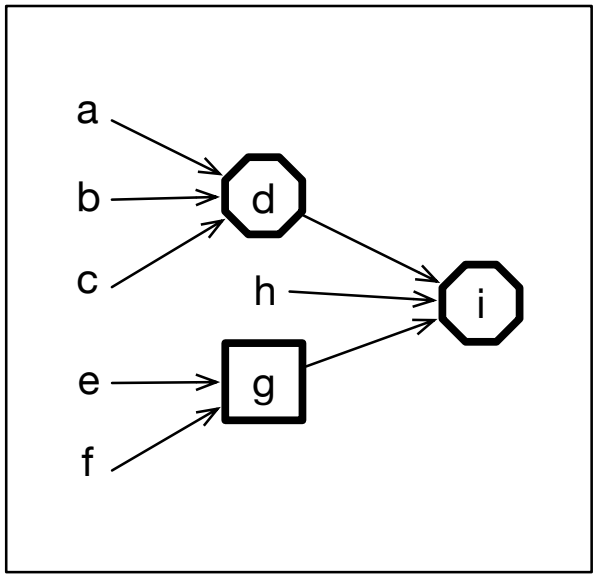


Figure 5.5 Multi-violation connected rules. Connect rules where violation "i" is dependent on another violation "d".

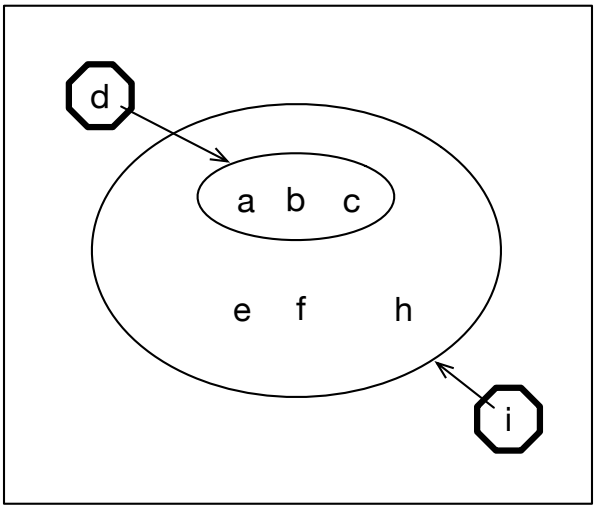


Figure 5.6 Hypergraphs of connected rules with dependent violations.

the items in the rule body, u or v, will cause r to not be satisfied. Therefore, to be a cover, either u and v must be represented by a "1" when variables are resolved in the disruption sentence (See Figure 5.2).

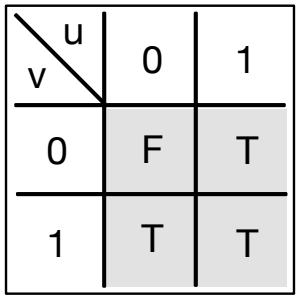


Figure 5.7 Truth table for u, v.

Corollary 5.2.1 (Disruption Sentence Cover)**.** *Given a rule r and its disruption sentence, any truth arrangement such that DS_r is **TRUE** will generate a disruption cover.*

Proof of Corollary 5.2.1. Let r be a rule, $r : a \wedge b \rightarrow c$, DS_r the disruption sentence for r, $DS_r = a \vee b$, and C a set of facts.

Assume that DS_r resolves to **TRUE**, but C is not a cover for r. Since DS_r resolves to **TRUE** for C, either a or b must be in C. If either a or b are in C, then removing C would leave r unsatisfied. However, if r is unsatisfied, then it would not generate c and would be a cover. This contradicts our original statement that P was not a cover.

Now assume that DS_r resolves to **FALSE**, but C is a cover for r. Since DS_r resolves to **FALSE** for C, neither a nor b can be in C. If neither a nor b are in C, then removing C would leave r satisfied. However, if r is satisfied, then it would generate c and would not be a cover. This contradicts our original statement that P was a cover. □

The disruption sentences corresponding to the inference violations in Figure 5.1 are as follows:

- Logic string for $g = (a \vee b \vee c \vee e \vee f)$

- Logic string for $p = (((u \vee v) \wedge (a \vee e \vee h)) \vee n \vee j \vee k \vee l \vee o)$
- Logic string for $r = ((e \vee f \vee q) \wedge (f \vee s \vee t))$

Since all three violations need to be disrupted, we connect disruption sentences with the AND operator, giving a full disruption sentence of $DS_{(g,p,r)}$ as $((a \vee b \vee c \vee e \vee f) \wedge (((u \vee v) \wedge (a \vee e \vee h)) \vee n \vee j \vee k \vee l \vee o) \wedge ((e \vee f \vee q) \wedge (f \vee s \vee t)))$. This full disruption sentence can then be used to determine if a combination of participating ground facts forms a complete inference disruption cover. We first construct a cover logic statement corresponding to the ground facts and logic operators found in the full disruption sentence.

To then determine if a combination of participating facts is a cover, we set the corresponding variables in the cover logic statement to "1". We set all other variables in the cover logic statement to "0". A bitwise evaluation of the cover logic statement is then performed. If the evaluation resolves to "1" then the combination provides a Inference Disruption Cover, otherwise it does not.

Example 5.2.4 (Inference Disruption Cover)**.** Let L be the logic string from Figure 5.4, $L = ((a \vee b \vee c \vee e \vee f) \wedge (((u \vee v) \wedge (a \vee e \vee h)) \vee n \vee j \vee k \vee l \vee o) \wedge ((e \vee f \vee q) \wedge (f \vee s \vee t)))$ and G be the set off all participating ground facts participating in violations g, p, and r, $G = \{a, b, c, e, f, h, j, k.l, n, q, s, t, u, v\}$. The following are example participating ground fact combinations followed by the cover logic string with combination variables set to "1" and their bitwise evaluation.

Combination $\{a, e, f\} : ((1 \vee 0 \vee 0 \vee 1 \vee 1) \wedge (((0 \vee 0) \wedge (1 \vee 1 \vee 0)) \vee 0 \vee 0 \vee 0 \vee 0 \vee 0) \wedge ((1 \vee 1 \vee 0) \wedge (1 \vee 0 \vee 0))) = ((1) \wedge (((0) \wedge (1)) \vee 0) \wedge ((1) \wedge (1))) = ((1) \wedge ((1) \wedge 0) \wedge (1)) = (1 \wedge 0 \wedge 1) = 0.$

Combination $\{f, k\} : ((0 \vee 0 \vee 0 \vee 0 \vee 1) \wedge (((0 \vee 0) \wedge (0 \vee 0 \vee 0)) \vee 0 \vee 1 \vee 0 \vee 0 \vee 0) \wedge ((0 \vee 1 \vee 0) \wedge (1 \vee 0 \vee 0))) ((1) \wedge (((0) \wedge (0)) \vee 1) \wedge ((1) \wedge (1))) = ((1) \wedge ((0) \vee 1) \wedge (1)) = ((1) \wedge (1) \wedge (1)) = 1.$

The combination $\{a, e, f\}$ does not provide a cover since it does not disrupt any of the ground facts in violation g and only disrupts one of the two hypergraphs in the disruption container found in g. The combination $\{f, k\}$ does provide a cover for g, p, and r.

5.3 Cost

Our investigation into a better costing model continues our emphasis on developing a semi-greedy approach. We defined two cost calculations in Chapter 3, the Concept Alteration Cost (Definition 3.4.2) and the Combination Alteration Cost (Definition 3.4.3). These cost calculations assign a cost to each data item in a participating fact combination. The discrete item cost, $\boldsymbol{Cost}_f$, is based on the alterations applied to a data item: one level of generalization, two levels of generalization, or removal (see Table 3.4). The aggregate cost, $\boldsymbol{Cost}_S$, is then calculated for a combination by adding the components of the $\boldsymbol{Cost}_f$ for each data item in the combination. In the exhaustive approach, each data item is assessed locally and any global impact is ignored. There is also no consideration given to one data item having more informational value than another (i.e., its impact on safe inferences, where it is found in the domain knowledge ontology tree, or entropy loss if modified), neither within a combination or across combinations.

In this section, we discuss refinement to our initial cost model and calculations. These refinements will allow our costing model to provide a better gauge on the impact each participating fact combination will have on data availability. To support the enhanced cost function, we collect meta-data on all data items that participate (directly or indirectly) in a privacy violating inference. The meta-data collected is as follows:

- the alteration cost for a data item

- the number of safe (non-violation) inference paths that the data item participates in
- the depth of the data item's concept in the domain knowledge ontology hierarchy
- the entropy of the data item in the domain knowledge ontology.

During participant fact identification and combination construction, meta-data will be collected and stored. The process will also store meta-data aggregate values for alterations (source and target data items) and combinations of alterations. We address organization and storage of the meta-data in the next sections, but first we discuss the four categories of meta-data.

Alteration Cost

We use the same approach to calculate alteration cost for the efficient approach as we did in the exhaustive approach (see Section 9 – Definitions 3.4.2 and 3.4.3). Values used in the calculations are based on Table 3.4.

Inference Participation

We say that a data item participates in the inference of a generated data item if it satisfies any rule in an inference path which concludes in that generated data item. While we want to disrupt non-safe (violation) inferences, we would like to avoid disrupting safe inferences. To encourage preservation of safe inferences, we increase the cost of a combination if it disrupts safe inferences. We recursively look at all rules that were satisfied during the reasoning process and any rules supporting those rules (recursively). From this rule information, we determine the number of safe (non-violation) inferences that a data item participates in. Because it is desirable to disrupt violation inferences, we do not increase cost of a combination because of safe

inferences that are dependent on violation inferences. This is further illustrated in the following example.

Example 5.3.1 (Inference Participation)**.** Figure 5.8 shows an example of nine rules satisfied by reasoning over some data and domain knowledge. These rules are comprised of six safe (non-violation) inferences, (d, i, m, v, w, y), and three violation inferences, (g, p, r). The rules are satisfied by 14 ground facts, (a,b,c,e,f,h,j,k,l,n,o,q,u,x). Since the safe inference y is dependent on the violating inference p (which we plan to disrupt), it is not counted as a safe inference for any of the ground facts in its path. The violation and non-violation inference participation counts can be seen in Table 5.3.1. Note that not all safe inferences are connected to violation inference paths. In our example, inferences u, v, and y (colored grey in Figure 5.8) are safe and not connected to any violation inferences. Since ground facts u and x are only found in the inference paths of v, w, and y and not in any violation inference paths, they are not candidates for inference disruption and we need not collect meta-data on them.

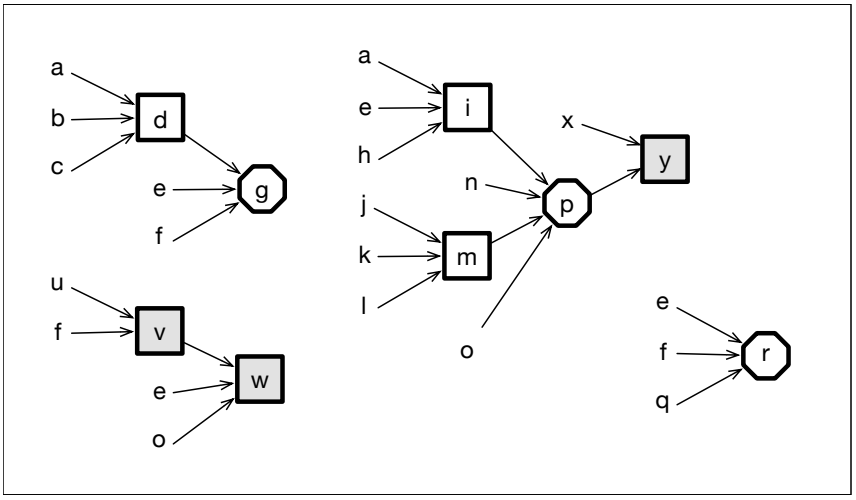


Figure 5.8 All inferences paths. Paths from Figure 5.1 (secondary rules removed for clarity) enhanced to include inference paths not contributing to violations (rule head shown in grey). Rule conclusions v, w, and y, along with ground facts u and x, do not participate in any of the violation inference paths.

Table 5.1 Inference Participation - Figure 5.8 ground facts shown with participation count for both violation and non-violation inferences.

Ground Fact	Violation Count	Non-Violation Count
a	2 {g,p}	2 {d,i}
b	1 {g}	1 {d}
c	1 {g}	1 {d}
e	3 {g,p,r}	2 {i,w,}
f	1 {g}	2 {v,w}
h	1 {p}	1 {i}
j	1 {p}	1 {m}
k	1 {p}	1 {m}
l	1 {p}	1 {m}
n	1 {p}	0 {}
o	1 {p}	1 {w}
q	1 {r}	0 {}
u	0 {}	2 {v,w}
x	0 {}	0 {}

Since we do not want to increase cost more than once if two ground facts in the same cover disrupt the same safe inference, we must maintain inference participation as a list of disrupted safe inferences instead of a discrete count value. Therefore, the inference participation count for a data item is determined by the cardinality of its disrupted safe inference list. The inference participation count for an alteration is the cardinality of the disrupted safe inference list for its unaltered data item. The count for a combination is the cardinality of the union of the disrupted safe inference list for unaltered data items in each alteration.

Depth

By depth, we mean a measure of distance between a data item concept and the root of its associated domain knowledge ontology. When the ontology is viewed as a tree, the depth is the number of ISA relationships between the concept describing a data item and the root node of the ontology. Depth gives a very broad indication of specificity; concepts with a low depth value are close to the root and very general (the root node

as a depth of 0 and maximum generality) while concepts with a larger depth value are farther from the root with numerous concepts along the path (each increasing in specificity); see Figure 5.9.

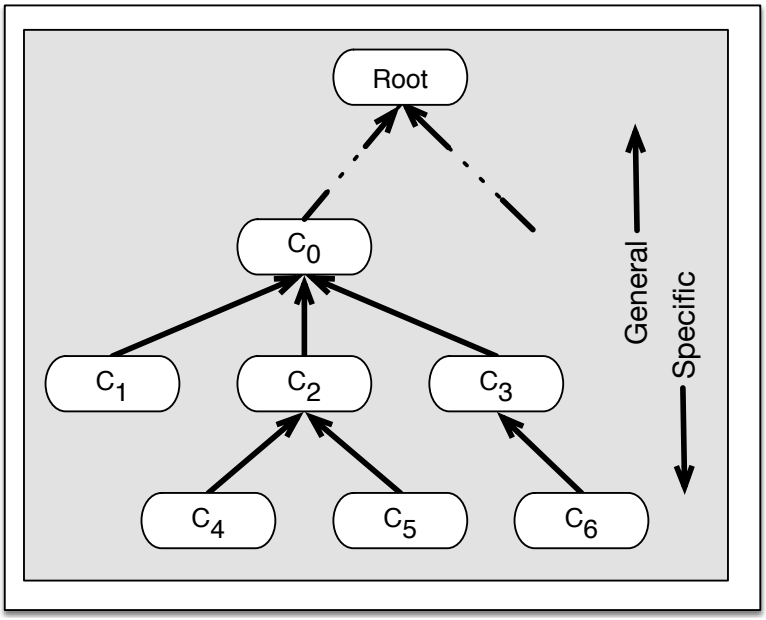


Figure 5.9 Hierarchical Ontology Tree - Concepts at the top of the tree near the root are more general, while concepts at the bottom, farther from the root, are more specific, with C_4, C_5, and C_6 having the most specificity.

Entropy

Entropy is a measure of uncertainty about a distinct piece of information. We apply the idea of entropy to our data items as an additional measure of data availability. We base our approach to entropy measurement on the work of Calmet and Daemi [5]. Calmet and Daemi proposes a a measure of entropy distance based on the Kullback-Leibler distance (relative entropy). They define the degree of a concept within an ontology as the number of sub-concepts under that concept in the ontology. The degree value is then normalized over the size of the ontology (number of unique concepts excluding the root) giving a value between 0 and 1.

The purpose of determining the degree measurement is to show levels of ambiguity. A low degree value (near 0) indicates a concept with little or no sub-concepts; there is minimal ambiguity in the use of this concept in a fact. A high degree value (near 1) indicates that a large number of sub-concepts in the ontology fall under this concept (it is near or at the root of the ontology); there is a high level of ambiguity since a fact using this concept may actually be better identified with one of many of its many sub-concepts.

Using an alphabet Ω consisting of a concept in the ontology, and degree measurement described above as the mass probability distribution, Calmet and Daemi calculate the entropy of concept N, denoted $H(N)$ as:

$$H(N) = -\sum_{\Omega} \frac{deg(N)}{2} log_2(\frac{deg(N)}{2}),$$

resolving to the sum of probability of N and the probability of $\neg N$:

$$H(N) = -\frac{deg(N)}{2} log_2(\frac{deg(N)}{2}) - (1 - \frac{deg(N)}{2}) log_2(1 - \frac{deg(N)}{2}).$$

This calculation assumes the following: $0\ log_2\ 0 = 0$. Using these calculations, a concept with $degree = 0$ (no sub-components) would have an entropy or ambiguity value of 0 (no ambiguity) while a concept with $degree = 1$ (all concepts in ontology are sub-components) would have an entropy or ambiguity value of 1 (maximum ambiguity).

In our enhanced cost model, we use Calmet and Daemi's approach to determine the amount of ambiguity introduced by altering a data item to a concept closer to the ontology tree root. We provide algorithms later in this section.

Example 5.3.2 (Entropy)**.** Let there be an ontology with concepts (root, $c_1, \ldots, c_{18}$) as shown in Figure 5.10. There are 18 unique concepts in the ontology if the root is excluded. Let there be two facts, f_a, and f_b that participate in the paths of some unauthorized inference. Also let f_a, and f_b connect to concepts c_7 and c_{18}

respectively. The degree of alteration options (none, one level generalization, two level generalization, removal) for f_a, and f_b are shown in Table 5.2.

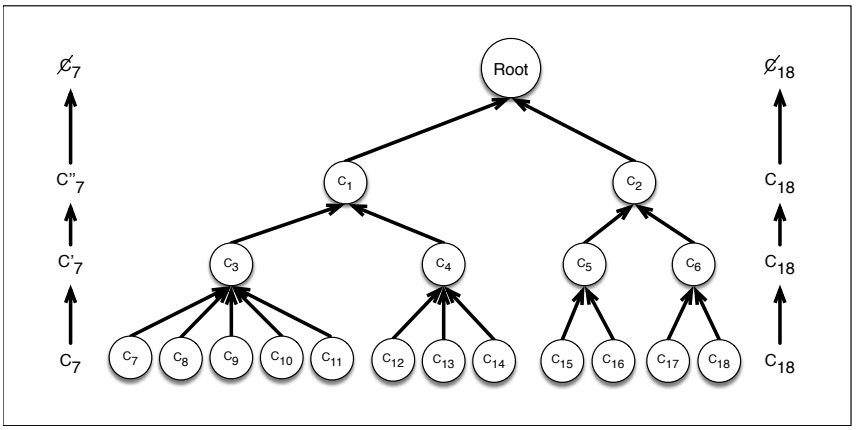


Figure 5.10 Entropy Example - Ontology tree with 18 concepts in the ontology, excluding the ontology root.

Table 5.2 Degree, normalized degree, and entropy values for concepts c_7 and c_{18}.

Concept	Degree	Normalized Degree	Entropy	Concept	Degree	Normalized Degree	Entropy
c_7	0	0.0	0.0	c_{18}	0	0.0	0.0
$c'_7 = c_3$	5	0.2778	0.5814	$c'_{18} = c_6$	2	0.1111	0.3095
$c''_7 = c_1$	10	0.5556	0.8524	$c''_{18} = c_2$	6	0.3333	0.6500
$\not{c}_7 = \text{root}$	18	1.0	1.0	$\not{c}_{18} = \text{root}$	18	1.0	1.0

Note that in Example 5.3.2, the degree increases as we apply alterations bringing the concept closer to the ontology root. In comparing c'_7 and c'_{18}, we see that c'_{18} has a lower degree than c'_7 (2 versus 5) and therefore a lower entropy value (0.3095 versus 0.5814). Intuitively, this indicates that there is less ambiguity introduced by generalizing one level from c_{18} to c'_{18} than by generalizing one level from c_7 to c'_7.

Information Vector

To maintain meta-data in support of the enhanced cost function, we introduce the concept of a meta-data information vector. The generic form of this vector is used to

track the meta-data related to a data item modification at three levels:

1. meta-data is tracked at the individual data item level

2. meta-data is tracked in aggregate for unaltered and altered data items (data item alteration)

3. meta-data is tracked in aggregate for the set of alterations (alteration combination)

While we use the same vector construct to track the three levels of meta-data, the content is different depending on which data level the vector is associated with. If a vector is associated with a data item, its data is specific to that item (Definition 5.3.1), if associated with an alteration, its data is the aggregate of two data items (Definition 5.3.2), and if associated with a participating fact combination, its data is an aggregate of its alterations (Definition 5.3.3).

Definition 5.3.1 (Data Item Information Vector)**.** Let f be a data item. We say the data item information vector for f, $V_f =< a_1, \ldots, a_4 >$, is a set of attributes, such that a_1 is the alteration cost associated with f, a_2 is the list of distinct unsafe inferences that f participates in, a_3 is the ontology hierarchy depth of f, and a_4 is the entropy (uncertainty) of f.

Definition 5.3.2 (Alteration Information Vector)**.** Let m be a data item alteration, with source concept c_1 and target concept c_2. We say the alteration information vector for m, $V_m =< a_1, \ldots, a_4 >$, is a set of attributes, such that a_1 is the sum of alteration cost associated with data items in a, a_2 is a 2-tuple where the first element is a distinct list of safe inferences for c_1 and the second element a similar list for c_2, a_3 is the average ontology hierarchy depths for c_1 and c_2, and a_4 is the difference between the entropy for c_1 and c_2, $(c_2 - c_1)$.

Definition 5.3.3 (Combination Information Vector)**.** Let c be a set of alterations forming a PFC. We say the combination information vector for c, $V_c =< a_1, \ldots, a_4 >$, is a set of attributes, such that a_1 is the sum of alteration cost associated for all alterations in c, a_2 is a 2-tuple where each element is a union of the corresponding element for all alterations in c, a_3 is the average ontology hierarchy depth for all alterations in c, and a_4 is the sum of the entropy values for all alterations in c.

Example 5.3.3 (Data Vector)**.** Let there be an Ontology tree as shown in Figure 5.10 with entropy values shown in Table 5.2. Let there be a participating fact combination C with 2 facts linked to ontology concepts C_6 and C_7 with alterations $A1$ and $A2$ such that $A2$ generalizes C_6 one level to C_2 and $A1$ generalizes C_7 two levels to C_1. Let the safe inference list for C_1, C_2, C_6, and C_7 be $\{b, d, e\}$, $\{b, c, d\}$, $\{a, b\}$, and $\{a, b, c\}$ respectfully. The information vectors for the data items, alterations, and combination are shown in Table 5.3.

Table 5.3 Information Vectors - Table shows the three types of information vectors.

Information Vector	Alteration	Safe Inferences	Depth	Entropy
Data Item C_7	1.0	{a,b,c}	4	0
Data Item C_1	-0.25	{b,d,e}	2	0.8524
Alteration A1: $C_7 \rightarrow C_1(C_7'')$	0.75	({a,b,c},{b,d,e})	3	0.8524
Data Item C_6	1.0	{a,b}	3	0.3095
Data Item C_2	-0.5	{b,c,d}	2	0.6500
Alteration A2: $C_6 \rightarrow C_2(C_6')$	0.5	({a,b},{b,c,d})	2.5	0.3405
Combination C1 = {A1,A2}	1.25	({a,b,c},{b,c,d,e})	2.75	1.1929

Meta-data values are pre-collected on all ground fact data items which are part of a complete minimal inference cover. These data item meta-data values are cached in a table for reference when building various information vectors. In some cases data items meta data may not be available in the table since the data item being generalized to was not in the initial data item set. In this case, the meta-data is collected, used, and added to the table for future reference. As seen in Figure 5.11, a

concept in the ontology hierarchy tree can have multiple identities since in addition to its initial name, it can also be reference as the first level generalization of its children and the second level generalization of its grandchildren.

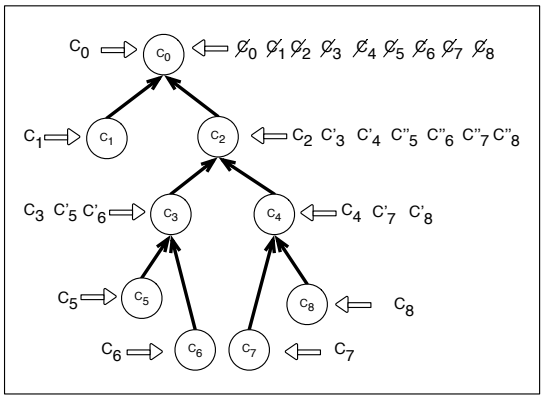


Figure 5.11 Multi-identify nature of concepts when they are potential targets for generalization or removal.

Example 5.3.4 (Multi-Identity)**.** Given Figure 5.11, we can see that node C_2 in the ontology hierarchy tree can be referenced as C_2, C_3', C_4', C_5'', C_6'', C_7'', or C_8''. Likewise, C_0 can also be referenced as the removal of any node in the tree. Note that while not shown in the figure, C_0 is also the first level generalization of both C_1 and C_2 and the second level generalization of both C_3 and C_3.

Dominance

Unlike the exhaustive approach, where discrete cost values could be compared to determine PFC cost dominance, the PFC cost is now based on an information vector. The data item information and alteration vectors are building blocks for the combination information vector. It is the combination information vector that represents the cost of a PFC so a method is needed to compare two combination information vectors and determine which dominates (has the higher cost). We accomplish comparison by

interrogating like meta-data in the two vectors and comparing their alteration impact values, $\boldsymbol{Impact}_S$ (Definition 5.3.4).

Definition 5.3.4 (Alteration Impact Value)**.** The alteration impact value, denoted $\boldsymbol{Impact}_S$, is the alteration cost $\boldsymbol{Cost}_S$ enhanced by the number of safe inferences impacted. Let S be a PFC and U the number of safe inferences disrupted by S. We calculate the alteration impact value of S as follows: $\boldsymbol{Impact}_S = \boldsymbol{Cost}_S + (\boldsymbol{Cost}_S * U)$.

Using this definition of $\boldsymbol{Impact}_S$ increases data availability by avoiding combinations that will disrupt safe (non-violation) inferences when possible. If two combinations have similar alteration cost, the use of $\boldsymbol{Impact}_S$ will allow our methods to chose the combination with lesser data availability impact.

5.4 Order of Evaluation

In Sections 5.1 and 5.2, we introduced a method to reduce the number of PFCs that need to be evaluated by eliminating combinations that do not provide a disruption cover. In Section 5.3, we enhanced meta-data collection and the cost calculation to provide a better measure of data availability and level of data uncertainty. These enhancements allow us to now investigate methods to further minimize combination evaluation and arrive at an acceptable solution sooner.

Our overarching goal is to not just identify a valid solution, but to identify a valid solution with low cost. By low cost, we mean the solution satisfies the property of minimality (see Section 3.2). In our exhaustive approach, minimality was optimal and achieved by searching all valid solutions for one with the lowest cost. To find an optimal solution, all PFCs had to be evaluated prior to minimal solution selection since all valid combinations had to be considered. However, there were inefficiencies inherent in the exhaustive approach – combinations that had no chance of being successful were unnecessarily evaluated; this inefficiency was addressed using covers

(Section 5.2). Solution covers eliminate many combinations from being evaluated, but the exhaustive approach still evaluates all cover combinations. In this section, we will investigate three approaches which strive to reach a solution using less evaluations: low-cost selection semi-exhaustive, low-cost selection heuristic, and low-cost high entropy traversal. The low-cost selection semi-exhaustive approach maintains optimal minimality but may still require more combination evaluations then necessary. The low-cost selection heuristic and low-cost high entropy traversal approaches reduces evaluations further, but relax optimal minimality and instead strives for a "reasonably" low cost solution.

5.4.1 Low-Cost Selection Semi-Exhaustive

The idea behind the low-cost selection semi-exhaustive approach is to consider all PFCs as in the exhaustive approach, but to evaluate them in impact cost order, from lowest cost to highest cost. Once a successful combinations is found, we designate that combination as our solution and require no additional evaluations.

We start by using the $\boldsymbol{Impact}_S$ calculation (Definition 5.3.4) to arrange all cover combinations in Cost Order (Definition 5.4.1) – from lowest cost to highest cost.

Definition 5.4.1 (Cost Order)**.** Let $VC = \{S_1, \ldots, S_n\}$ be the set of all identified PFCs based on minimal disruption covers. We say that VC is in cost order if $\forall i, j (i, j = 1, \ldots, n), (i < j), (\boldsymbol{Impact}_{S_i} \leq \boldsymbol{Impact}_{S_j})$. In other words, the impact cost of S_i is less than or equal to the impact cost of S_j.

Intuitively, the ***Impact*** value for all cover combinations in VC is non-decreasing order from combination S_1 to combination S_n.

Starting with the lowest cost combination, we evaluate each combination using our reasoner and domain knowledge. If the combination removes all violations, we then deem it successful and stop. If the combination does not remove all violations, we step to the next lowest cost combination and evaluate it. We continue this process until we

find a combination that is deemed successful. Based on Claim 5.4.1, we can terminate as soon as the first successful combination has been identified, without needing to evaluate any additional combinations. By evaluating combinations in cost order, we eliminate the exhaustive approach inefficiency of evaluating **all** combinations prior to selecting the one with minimal cost. However, while the sequential approach can terminate early, avoiding the need to evaluate all combinations, it may still have to evaluate a large number before finding one that is successful. Note that if multiple combinations have equal cost, their order of evaluation is irrelevant, since any successful combination with that cost is minimal.

Claim 5.4.1 (Early Termination)**.** *If combinations are evaluated in Cost Order (Defnintion 5.4.1), once a successful combination has been identified, evaluations can stop. Minimality is guaranteed as no other combination can exist which have a lower cost.*

Proof of Early Termination. Assume, by contradiction, that a successful combination S is identified as minimal, but another successful combination S' exists and has a lower cost. Let $VC = \{S_1, \ldots, S_n\}$ be the set of all combinations in cost order (Definition 5.4.1). Since S is a successful combination, it would be some S_k ($1 \leq k \leq n$) in VC. Solutions in SV are evaluated in cost order, from S_1 to S_n, and the first successful combination is selected as minimal. Since we know that S' is also a successful combination, but with a lower cost, it must occur before S in VC (cost order). However, if S' occurs before S then it would be evaluated before S and if S' is a successful combination, it would have been selected as minimal. But S was selected as minimal. This is a contradiction to our assumption that S' exists and has a lower cost than the selected minimal . □

There is no way to predict how many unsuccessful combinations will be evaluated before a successful one is found, but the hope is that a solution is found before reaching

the midpoint of the combination list. If successful, this approach would reduce the number of evaluations by more than 50% over the exhaustive approach.

5.4.2 Low-Cost Selection Heuristic

In this approach, we use heuristics to further reduce the number of evaluations required to get to a satisfactory low cost (but not necessarily minimal) solution.

Similar to the low-cost selection semi-exhaustive approach, we start by using $\boldsymbol{Impact}_S$ cost (Definition 5.3.4) to arrange cover combinations in Cost Order, except in this case we only include full removal combinations. By full removal combinations, we mean those combination covers that only remove concepts and do not perform any generalizations. Once in cost order, we can easily determine the lowest cost of all removal combinations by examining the first combination in the list. The cost of this first combination becomes our baseline minimum cost. We could select this combination as our solution and terminate, but to find a "reasonably" low cost solution, we use heuristics to evaluate a sample of low cost removal combinations with selected generalizations added. The intuition behind low-cost selection heuristic approach is that a successful low cost generalization combination is likely based on a low cost removal combination and by evaluating combination elements based on entropy, we can balance combination success with minimizing alteration cost and maximizing data availability. For this approach, we currently only consider one-level generalizations. There is no reason this approach could not be extended to consider additional levels of generalization in the future. Note that it not necessary to evaluate removal combinations since they are based on minimal inference disruption covers which, by Definition 5.1.3, will disrupt violations by leaving sufficient rules unsatisfied.

Our method will investigate the lowest cost five removal combinations. In each case, we know that the removal combination will be successful since it is a minimal

disruption cover. We also know that generalizing any concept in a removal combination (instead of removing it) will reduce alteration cost. However, the change from removal to generalization may also cause the combination to not be successful in disrupting all violations. Therefore, our goal is to carefully pick which concepts in the removal combination will reduce cost and are most likely to preserve successful violation disruption.

To assist in selecting concepts that have a higher chance of preserving violation disruption, we look at concept entropy and the disruption options. Our goal is to make alterations that leave rules needed by violation inferences unsatisfied. We know that removal of concepts in these rules will make the rules unsatisfied. However, for generalization we want to pick concepts that also leave the rules unsatisfied. Generalizations that are very similar to the original concept are likely to cause fired rules to remain satisfied; so, we select concepts to generalize by looking at how unlike they are from the original concept. We use uncertainty as an indicator of "likeness" between concepts and their generalizations. The more uncertainty that is introduced by a generalization, the more likely it is dissimilar to the original concept and will leave a rule unsatisfied (and disrupt inference violations). We are using a threshold of 0.5 uncertainty preserved (uncertainty of removal (1.0) - uncertainty of generalization) to indicate high dissimilarity and a good choice for generalization.

We first designate the lowest cost removal combination as the current solution and its cost as the current baseline cost. Next, for each of the lowest cost five removal combinations in cost order (low to high), we perform the following steps:

- for each participating fact in the removal combination:
  - compare the uncertainty introduced by removal with the uncertainty introduced by generalizing one level
  - if one level of generalization retains 50% of uncertainty introduced by

removal, switch alteration method to generalization

- re-calculate cost of combination
- if updated cost of combination is less than the current baseline cost
  - apply updated combination alterations to initial database
  - reason over database and domain knowledge
  - if no violations generated, updated cost is new baseline and updated combination is new current solution

When these steps are complete, we will have evaluated up to five variations on the lowest cost removal covers. If any of these variations are successful, we will have captured its cost and marked it as a "reasonably" low cost solution. Since we do not need to reason over the removal combinations, we will at most reason over five combinations in this method, making it a much more computationally feasible approach.

Example 5.4.1 (Heuristic Order)**.** Let there be an Ontology tree as shown in Figure 5.10 with entropy values shown in Table 5.2. Let there be a minimal disruption cover (removal) combination B with 2 facts linked to ontology concepts C_7 and C_{18}. Let B have alterations $A1$ and $A2$ such that $A1$ removes C_7 and $A2$ removes C_{18}. Let the safe inference list for C_6, and C_{18} be $\{b, d, e\}$ and $\{a, b, c\}$ respectfully. The information vectors for the data items, alterations, and combination are shown in Table 5.4.2.

Assume B is the lowest cost combination of the five lowest cost removal combinations. Following the steps described above, the initial baseline ***Impact*** cost of B would be $2 + (2 * 5) = 12.0$ (alteration cost + alteration cost * number of safe inferences). When looking at the first alteration, the uncertainty gain is 1.0. Generalizing C_7 one level (C_7' or C_3) would yield an uncertainty gain is 0.5814, which is above our

threshold of 0.5 (50% reduction), so C_7 is a good choice for generalization instead of removal. Generalizing C_{18} one level (C'_{18} or C_6) would yield an uncertainty gain is 0.3095, which is below our threshold and not a good choice to switch to generalization. The updated information vector for B, denoted B' is shown in 5.4.2. The ***Impact*** cost for the updated combination would be be $1.5 + (1.5 * 5) = 9.0$. Assuming B' removes the violations when applied to the database and reasoned over with domain knowledge, it would be preferable to B since its ***Impact*** cost is lower (9.0 versus 12.0); its uncertainty gain is also lower (1.5814 versus 2.0). The remaining four low cost combinations would be treated likewise and if any were successful with an ***Impact*** cost lower than 9.0, they would be chosen above B'.

Table 5.4 Initial information vectors for minimal disruption cover (removal) combination.

Information Vector	Alteration	Safe Inferences	Depth	Entropy
Data Item C_7	1.0	{a,b,c}	4	0
Data Item C_{ROOT}	-0.0		1	1
Alteration A1: $C_7 \rightarrow C_{ROOT}$	1.0	{a,b,c}	2.5	1.0
Data Item C_{18}	1.0	{b,d,e}	4	0
Data Item C_{ROOT}	-0.0		1	1
Alteration A2: $C_{18} \rightarrow C_{ROOT}$	1.0	{b,d,e}	2.5	1.0
Combination $B = \{A1,A2\}$	2.0	{a,b,c,d,e}	2.5	2.0

Table 5.5 Initial information vectors for minimal disruption cover (removal + generalization) combination.

Information Vector	Alteration	Safe Inferences	Depth	Entropy
Data Item C_7	1.0	{a,b,c}	4	0.0
Data Item C_3	-0.5		3	0.5814
Alteration A1: $C_7 \rightarrow C_3$	0.5	{a,b,c}	3.5	0.5814
Data Item C_{18}	1.0	{b,d,e}	4	0.0
Data Item C_{ROOT}	-0.0		1	1.0
Alteration A2: $C_{18} \rightarrow C_{ROOT}$	1.0	{b,d,e}	2.5	1.0
Combination $B' = \{A1,A2\}$	1.5	{a,b,c,d,e}	3.0	1.5814

5.4.3 Low-Cost High-Entropy Traversal

The last area we investigated is based on heuristic construction of a solution versus selection from a large set of potential solutions. This approach does not require that combinations be built in advance.

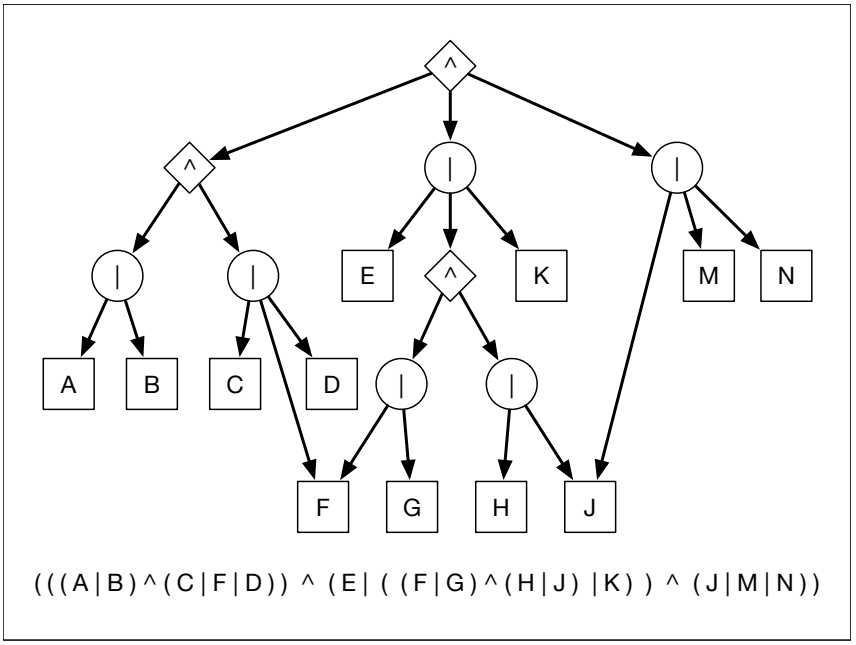


Figure 5.12 Logic graph. This graph is constructed during inference path evaluation and participant discovery and is used to construct the logic equation for cover evaluation.

Before describing this approach, we need to discuss an addition to our data model (Section 3.3.1). A new data structure is added to support the generation of cover logic strings and low-cost high-entropy traversal method. This additional structure is a rooted tree graph (Figure 5.12), which is used to store inference paths and their logical operator connections. The graph has three types of vertices: the collector vertex which is indicated by a diamond, the inference vertex which is indicted by a circle, and the fact vertex which is indicated by a square. We will use this inference graph as a guide to build the cover determination logic statement. In building the statement, the collector vertex simulates a logical "AND" of its children, the inference vertex

simulates a logical "OR" of its children, and fact vertices are ground facts / branch leaves and have no children. We will discuss construction of the rooted tree graph and logic statement when we review the revised Solution Set and GetCandidates algorithms in the next section.

We can now describe the low-cost high-entropy traversal method. We use the violation inference rule representation that is inherent in our rooted tree graph to guide construction of a minimal inference disruption cover with low cost and high entropy. By performing a depth-first traversal of the tree graph, we can select the best node to add into the alteration combination by examining the cost and entropy of nodes below it.

The idea behind this approach is that, while doing a depth-first traversal of the tree graph, we build a cover by selecting a node that fulfills the AND and OR requirements at each logic operator branch. At each OR operator we will select one child item to satisfy the operation. At each AND operator, we must select **all** items to satisfy the operation. Since all nodes are visited using a depth-first traversal, the question of which items to select for each logic operator is based on nodes already visited. We use a combination of cost and entropy to select the "best" item for an operator. The OR operators will pass "up" the selected node and its meta-data, whereas the AND operator will pass up a set of items and an aggregation of their meta-data. At each step the options for selection may include: directly connected facts, facts propagated up the tree by OR operators, or collections of facts assembled and propagated up the tree by AND operators. By traversing the tree in this way, we can construct a list of facts that will provide a minimal disruption cover. This list can be confirmed to be a minimal disruption cover by evaluation using the minimal cover logic statement as previously discussed.

If we only consider removal cost of fact nodes while traversing, we could easily pick the lowest cost node to satisfy a branch. This low cost traversal gives us a cover

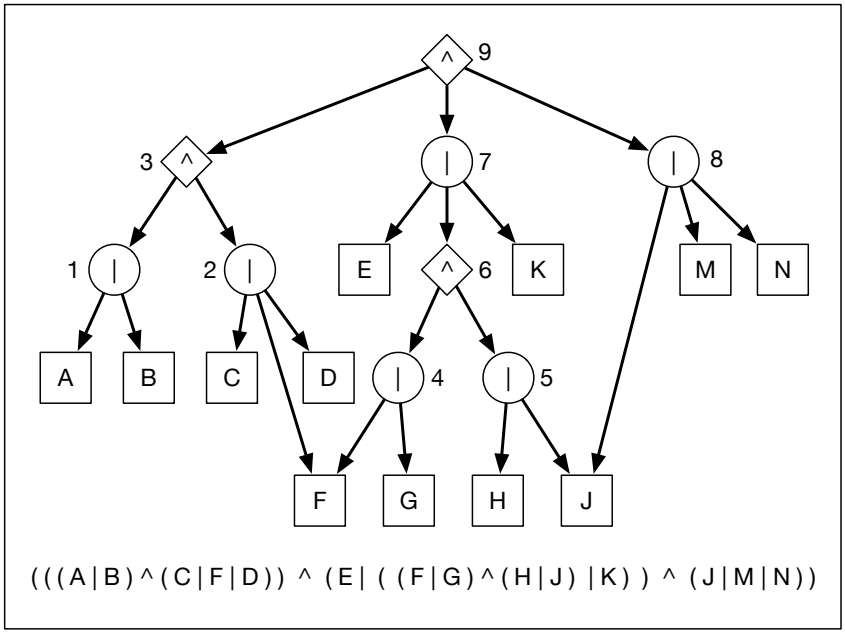


Figure 5.13 This graph is constructed during interrogation of the violations inference paths. Note the numbers which indicate the order of node visiting during depth-first traversal.

with the same (lowest) cost as found when ordering removal covers in the previous section. However, if we looked, not just at removal cost, but factored in a fact node's entropy value, our result would be a high-entropy low-cost disruption cover. This cover would contain facts that, for each OR branch in the graph, are the ones most likely to preserve disruption if generalized. If we selectively switch the highest entropy facts in this cover from removal to generalization, our cost should go down and our probability of preserving disruption should be higher than just selecting the lowest cost removal cover.

Example 5.4.2 (Tree Graph Traversal)**.** Let the tree graph in Figure 5.13 be a representation of the rules and participating facts for a set violations. We assume for this example that all facts in the graph tree have the same impact value. The low-cost high-entropy traversal would process as follows:

1. The node (A or B) with higher entropy is selected

2. The node (C or D or F) with higher entropy is selected

3. The results of steps 1 and 2 are combined

4. The node (F or G) with higher entropy is selected

5. The node (H or J) with higher entropy is selected

6. The results of steps 5 and 6 are combined

7. The node (E or K) or result of step 6 with higher entropy is selected

8. The node (J or M or N) with higher entropy is selected

9. The result of steps 3 or 7 or 8 are combined

Details of these approaches are discussed in the following section.

5.5 Efficient Disruption Approach

In this section, we will discuss details and algorithms for the methods needed to support our efficient approach. Some algorithms in this section are revisions to those found in Subsection 3.4.1 and some are new.

We have presented numerous revisions to our approach for defeating inference violations. In the following sections, we provide additional detail with focus on improving computational feasibility. These modifications align with topics presented earlier in this chapter: reduction of combination set size (Section 5.1), improved cost evaluation (Section 5.3) and the heuristic approach to quickly selecting a solution (Section 5.4), with the goal being development of more computationally efficient processes. Before presenting relevant algorithm changes, we will describe the high level logic flow for each of the three approaches described earlier in this chapter.

First, we describe the flow of Low-Cost Selection Semi-Exhaustive (Section 5.4.1). The basic logic flow for this approach is similar to that of the exhaustive approach and shown in Figure 5.14.

Steps are numbered to allow reference to steps in this figure when specific algorithms are discussed. Steps in this flow are as follows:

1. This step is the same as in the exhaustive approach.
2. This step is the same as in the exhaustive approach.
3. This step is the same as in the exhaustive approach.
    3a. This step is the same as in the exhaustive approach.
4. This step generates all removal covers. These covers are sorted in cost order, low to high.
5. In this step, we iterate over all covers, from lowest cost to highest.
    5a. In this step, we apply the combination alterations to the original database.
    5b. In this step, we reason over the altered database and domain knowledge.
    5c. In this step, we check to see if any violations are still present. If no violations are found, we declare the current combination as the solution and exit the loop.
6. This step is the same as in the exhaustive approach.

Second, we describe the flow of Low-Cost Selection Heuristic (Section 5.4.2). The basic logic flow for this approach is similar to that of the exhaustive approach and shown in Figure 5.15.

The basic logic flow for our efficient approach is similar to that of the exhaustive approach. Steps are numbered to allow reference to steps in this figure when specific algorithms are discussed. Steps in this flow are as follows:

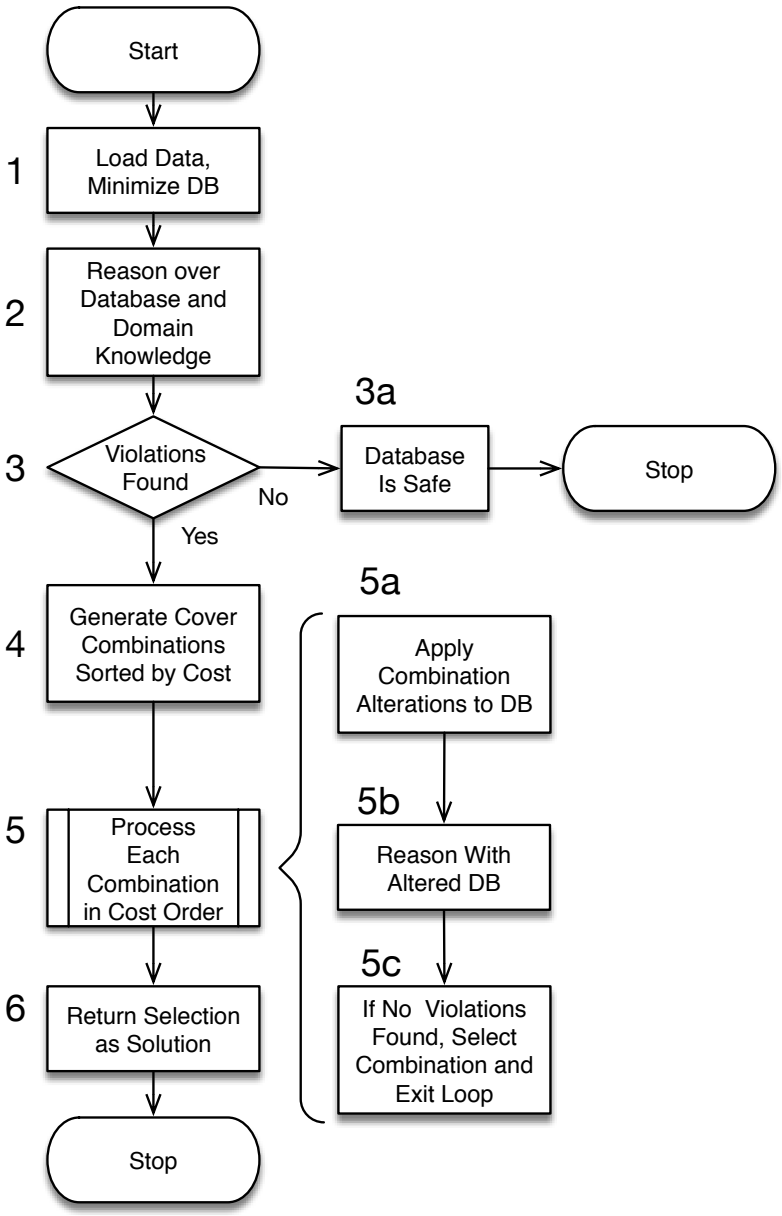


Figure 5.14 High-level logic flow of Low-Cost Selection Semi-Exhaustive.

1. This step is the same as in the exhaustive approach.

2. This step is the same as in the exhaustive approach.

3. This step is the same as in the exhaustive approach.

3a. This step is the same as in the exhaustive approach.

4. This step now only generates full removal covers instead of the full superset of facts using removal and generalization. These covers are also now sorted in cost order.

5. The baseline solution is set to the lowest-cost full-removal cover. This will be our default solution if a lower cost solution is not found.

6. In this step, we iterate over the (up to five) lowest cost removal covers.

   6a. In this step, we iterate over all alterations in the cover being processed.

      6a1. In this step, we look at the levels of uncertainty (entropy) between a removal and a generalization. If we maintain 50% of the uncertainty introduced by removal, we say generalization has a high chance of maintaining disruption.

      6a2. In this step, we switch the alteration from removal to generalization if the uncertainty threshold has been met.

   6b. In this step, we apply the combination alterations to the original database.

   6c. In this step, we reason over the altered database and domain knowledge and check to see if any violations are still present. If no violations are found, set the altered combination as the new baseline.

7. This step is the same as in the exhaustive approach.

Third, we describe the flow of Low-Cost High-Entropy Traversal (Section 5.4.3). The basic logic flow for this approach is similar to that of the exhaustive approach and shown in Figure 5.16.

Steps are numbered to allow reference to steps in this figure when specific algorithms are discussed. Steps in this flow are as follows:

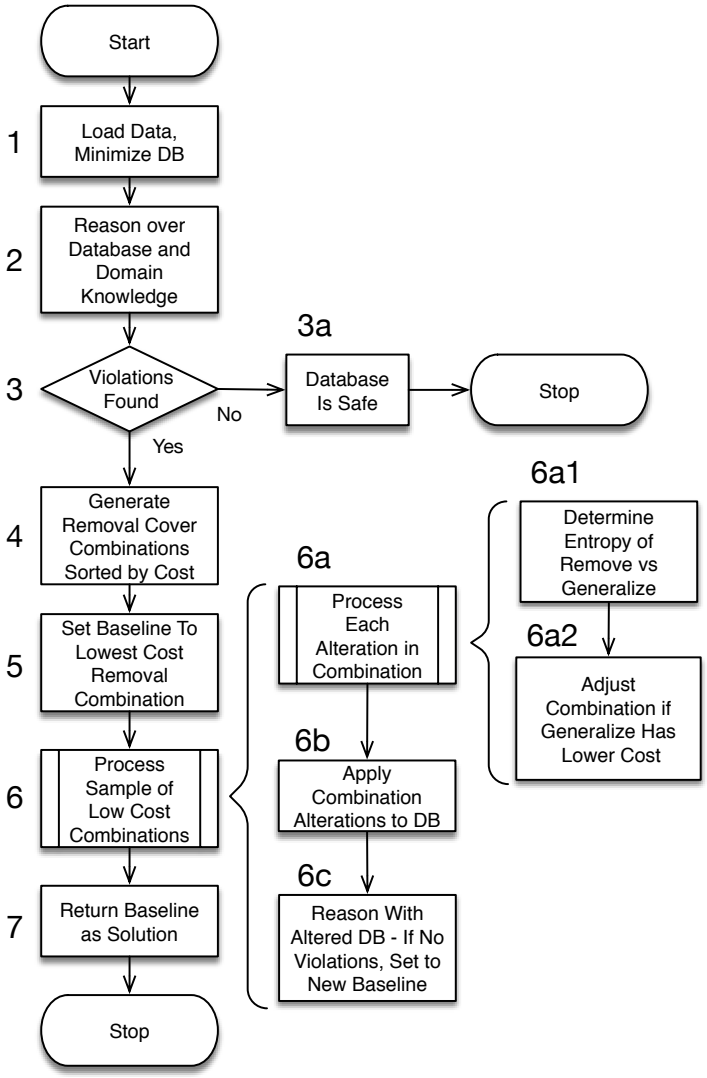


Figure 5.15 Highlevel logic flow of Low-Cost Selection Heuristic.

1. This step is the same as in the exhaustive approach.

2. This step is the same as in the exhaustive approach.

3. This step is the same as in the exhaustive approach.

   3a. This step is the same as in the exhaustive approach.

4. This step generates the minimal cover logic statement and graph tree, but does

not generate any participating fact combinations.

5. In this step, we do a depth-first traversal of the graph tree to create a cover.

   5a. In this step, if an OR operator is encountered, we select the best child node to propagate up.

   5b. In this step, if an AND operator is encountered, we select all child nodes to propagate up.

6. This step looks at high entropy concepts included in the cover. We start with all high entropy concepts and remove the lowest valued one until a solution is reached.

   6a. In this step, we generalize all concepts in the current entropy list.

   6b. In this step, we test (reason over altered database) and select as solution if no violations found.

7. This step is the same as in the exhaustive approach.

We now present our "efficient" modification to the exhaustive approach algorithms and introduce several new algorithms. Algorithm changes for Low-Cost Selection Semi-Exhaustive are trivial and only consist of processing the combinations in cost order. We will first present algorithms to support Low-Cost Selection Heuristic, followed by algorithms to support Low-Cost High-Entropy Traversal.

5.5.1 Low-Cost Selection Heuristic

This discussion will follow the basic flow of Figure 5.15.

The initial algorithm in our framework, Algorithm 1 (DisruptViolations), has a minor modification. This algorithm implements steps 1-3 and 7 in each of the process flows.. In Algorithm 1, line 6 is changed to call Algorithm 9 (EfficientDisruption) instead of Algorithm 4 (ExhaustiveDisruption).

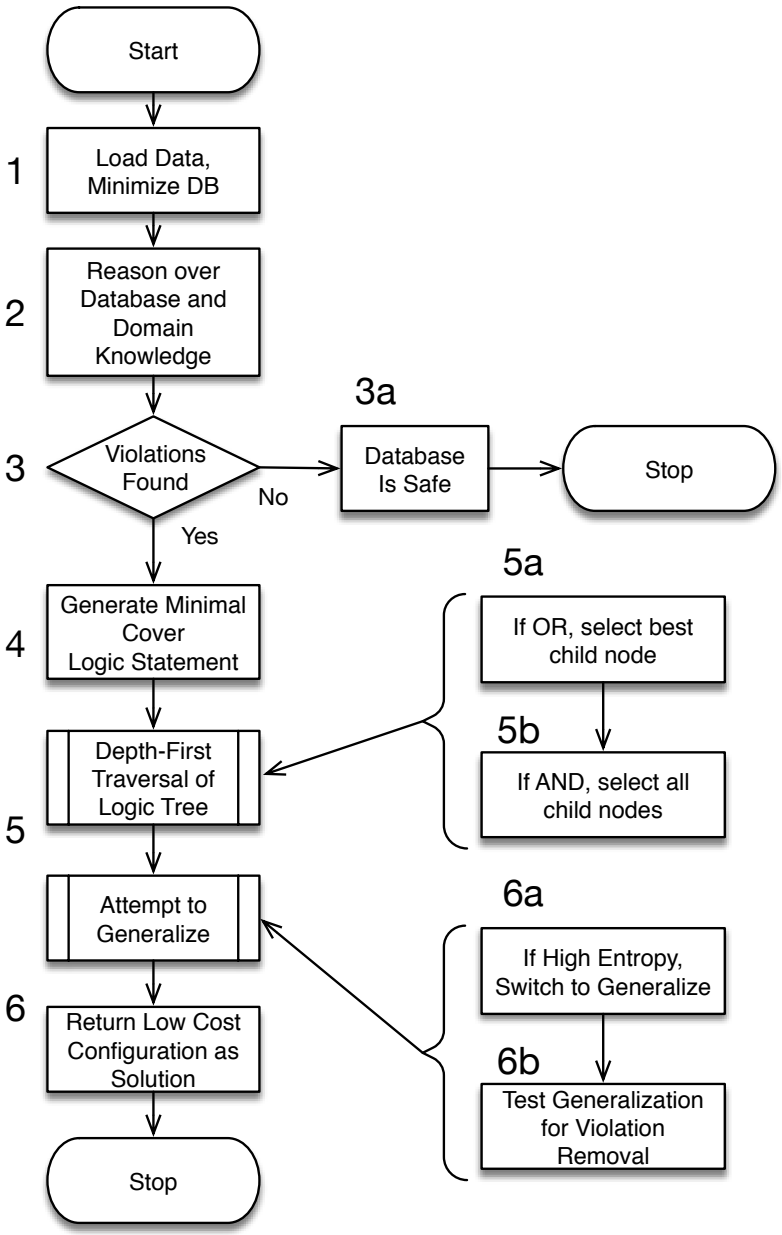


Figure 5.16 High-level logic flow of Low-Cost High-Entropy Traversal.

Next, we address Algorithm 4 (Exhaustive Disruption). This algorithms controls the primary logic used to find an efficient solution to the violation inference problem. This algorithm implements steps 4-6 in Figure 5.15.

The revised method, Algorithm 9 (Efficient Disruption) implements the heuristic

approach described in Section 5.4. Revisions to this algorithm include:

- ContributeToInferenceSet is called to determine the number of inferences each concept participates in
- EfficientCombinations is called instead of SolutionSet which now returns only full removal covers
- the set of combinations in SS is sorted in cost order, lowest to highest
- baseline combination (S) and cost (S_{cost}) are established
- up to the first five combinations (cost order) are examined
- in each alteration, removals are changed to generalizations if at least 50% of removal entropy is retained by generalization
- updated combinations are evaluated if their cost is less than the baseline cost
- if evaluated and violations are removed, updated combination becomes the new baseline

In general, Algorithm 9 will evaluate generalization options for at most the five lowest cost removal combinations. It will return either the lowest cost full removal cover or a generalized version of another low cost removal cover, whichever has lower cost.

The first step in the EfficientDisruption method is to generate a set of full removal inference disruption covers. This is analogous to creating the full set of removal and generalization alteration combinations in the exhaustive approach, but there are two major differences in theEfficientDisruption method. First, only covers are included in the returned combination set; if a combination does not impact every violation inference path at least once, it is not considered a cover. Second, only full removal covers are returned. At this point in the process, we focus on full removals and

ALGORITHM 9: EfficientDisruption

Input: DB_0 - initial database
Input: DB_m - minimal initial database
Input: O - Ontology
Input: v - violation threshold label
Input: λ - privacy mapping function
Input: DB_{mf} - fix point of DB_m over O
Input: SV - privacy violating concepts in DB_{mf}
Output: S - selected solution

```
1  begin
2      ContributeToInferenceSet(DB_m,O,DB_mf,SV)
3      SS = EfficientCombinations(DB_m,O,DB_mf,SV)
4      sort SS based on combination cost, lowest to hightest
5      sampleSize = min(5,|SS|)
6      S = SS[1]
7      S_cost = cost of S
8      for i ← 1 to sampleSize do
9          combination = SS[i]
10         forall alteration ∈ combination do
11             o = unaltered RDF triple in alteration
12             r = removal RDF triple in alteration
13             g = 1 level generalization of o
14             threshold = ((entropy(O,r)−entropy(O,o))/2)
15             if (entropy(O,g) − entropy(O,o)) > threshold then
16                 update alteration replacing r with g
17                 update cost of combination to reflect cost of g
18             end
19         end
20         if updated combination cost < S_cost then
21             DB_w = AlterData(DB_0,combination)
22             DB_wf = Reason(DB_w,O)
23             C = set of inferred facts from DB_wf
24             SV' = PrivacyMapAndDetect(C,v,λ)
25             if SV' = ∅ then
26                 S = alteration
27                 S_cost = cost of alteration
28             end
29         end
30     end
31     return S
32 end
```

are not including any generalization alterations in the combinations. Algorithm 10 (EfficientCombinations) is a revision of the exhaustive approach SolutionSet method and implements details in step 4 in Figure 5.15.

Revisions to this algorithm include:

- addition of call to the AddViolationToGraph method to add violation nodes to the inference graph
- included passing of inference graph (and graph root) to the GetCandidates method so rule body nodes can be added to the inference graph
- addition of call to the ConstructLogicString to build the cover determination logic string
- included passing of logic string to the Powerset method to allow for inference disruption cover testing

The GetCandidates method supports the EfficientCombinations method by recursively interrogating the inference path of a violation's inference path, finding all ground facts on that path. Algorithm 11 is a revision of the GetCandidates method. Revisions to this algorithm include:

- included inference graph (and graph root) as inputs
- addition of call to the AddFactToGraph method to add fact nodes to the inference graph; these are leaf nodes
- addition of call to the AddInfereceToGraph method to add inference nodes to the inference graph; this call will add "OR" nodes to simulate a rule body and "AND" nodes if multiple rules are satisfied and generate the same conclusion
- included passing of inference graph (and graph root) to the recursive GetCandidates method so rule body nodes can be added to the inference graph

ALGORITHM 10: EfficientCombinations

Input: DB_m - minimal initial database
Input: O - Ontology
Input: DB_{mf} - fix point of DB_m over O
Input: SV - privacy violating concepts in DB_{mf}
Output: SS - powerset of candidate facts

```
1  P = ∅                                   // P - remember paths followed
2  C = ∅                                   // C - candidate list
3  S = ""
4  G = new graph
5  root = create node in G with values: fact NULL, type "AND"
6  forall v ∈ SV do
7  |  R = all rules satisfied by O and DB_m which generates v
8  |  c = |R|
9  |  forall r ∈ R do
10 |  |  n = AddViolationToGraph (G,root,r,v,c)
11 |  |  GetCandidates(DB_m,O,DB_mf,r,P,C,G,n)
12 |  end
13 end
14 S = ConstructLogicString(G, root, S)
15 CS=Powerset(C,S)
16 return CS
```

The inference graph data model is constructed in Algorithms 10 and 11 by calls to the new methods AddViolationToGraph (Algorithm 12), AddInferenceToGraph (Algorithm 13), and AddFactToGraph (Algorithm 14). Each of these algorithms adds new paths to the graph such that edges and nodes enable the formation of the cover evaluations logic string.

Algorithm 12 (new) adds a violation inference to the inference graph. Violations are always added as children of the root node. Violation inferences are typically added as an "OR" node, meaning that disruption of any of their children will disrupt the violation. If there are multiple satisfied rules that can generate the violation data item (indicated by the rule set cardinality parameter c), then violation inference "OR" nodes are preceded by an "AND" node. Inclusion of the "AND" node means that all inference rules generating the violation data item must be disrupted for the

ALGORITHM 11: GetCandidates (Efficient)

Input: DB_m - minimal initial database
Input: O - Ontology
Input: DB_{mf} - fix point of DB_m over O
Input: r - rule to follow
Input: P - rules processed
Input: C - Candidates
Input: G - graph
Input: *root* - root of subgraph in G
Output: C - Updated Candidates

```
1  F = facts on left side (body) of rule r
2  forall f ∈ F do
3      if f ∈ DB_m then
4          f is ground fact
5          n = AddFactToGraph (G,root,r,f,0)
6          if f ∉ C then
7              h = entropy(O,f)
8              add h to f
9              add f to C                          // add to candidate list
10         end
11     else
12         f is inferred fact
13         R' = all rules satisfied by O and DB_m which generates f
14         c = |R'|
15         forall r' ∈ R' do
16             n = AddInferenceToGraph (G,root,r',f,c)
17             if r' ∉ P then
18                 add r' to P                          // remember path
19                 GetCandidates(DB_m,O,DB_mf,r',P,C,G,n)
                                                        // recursive
20             end
21         end
22     end
23 end
24 return C                                  // updated candidate list
```

violation inference to be disrupted. The "AND" node is implementing the disruption container described earlier in our hypergraph approach (see Example 5.2.2).

ALGORITHM 12: AddViolationToGraph

Input: G - graph
Input: *root* - root of subgraph in G
Input: r - rule
Input: f - fact
Input: c - count of satisfied rules
Output: b - node for rule r, fact f in G

```
1  if c > 1 then
2  |  a = node ∈ G with values: fact f, type "AND"
3  |  if a = NULL then
4  |  |  a = create node in G with values: fact f, type "AND"
5  |  |  create edge from root to a in G
6  |  end
7  |  b = create node in G with values: rule r, fact f, type "OR"
8  |  create edge from a to b in G
9  else
10 |  b = create node in G with values: rule r, fact f, type "OR"
11 |  create edge from root to b in G
12 end
13 return b
```

Algorithm 13 (new) adds a non-violation inference to the inference graph. Violations are always added as children to a sub-tree root which can exist in any branch of the inference tree. These inferences are also typically added as an "OR" node, but may also be preceded by "AND" nodes if multiple satisfied rules that can generate the inference data item.

Algorithm 14 (new) adds a ground fact to the inference graph. Facts are always added as children to a sub-tree root which can exist in any branch of the inference tree. These nodes are added directly to the parent - there is no need for "OR" or "AND" nodes.

Algorithm 15 (new) constructs the cover determination logic string. The algorithm recursively traverses the inference graph. At each node visited, it either creates a

ALGORITHM 13: AddInferenceToGraph

Input: G - graph
Input: *root* - root of subgraph in G
Input: r - rule
Input: f - fact
Input: c - count of satisfied rules
Output: n - node for rule r, fact f in G

```
if c > 1 then
    a = node ∈ G with values: fact f, type "AND"
    if a = NULL then
        a = create node in G with values: fact f, type "AND"
    end
    create edge in G from root to a
    b = node ∈ G with values: rule r, fact f, type "OR"
    if b is null then
        b = create node in G with values: rule r, fact f, type "OR"
    end
    e = edge ∈ G from a to b
    if e is NULL then
        create edge in G from a to b
    end
else
    b = node ∈ G with values: rule r, fact f, type "OR"
    if b is NULL then
        b = create node in G with values: rule r, fact f, type "OR"
    end
    create edge in G from root to b
end
return b
```

ALGORITHM 14: AddFactToGraph

Input: G - graph
Input: *root* - root of subgraph in G
Input: f - fact

```
b = node ∈ G with values: fact f, type "FACT"
if b is NULL then
    b = create node in G with values: fact f, type "FACT"
end
e = edge ∈ G from root to b
if e is NULL then
    create edge in G from root to b
end
```

binary "AND" expression, a binary "OR" expression or inserts a ground fact label. To maintain appropriate precedence, expressions are contained within parentheses. The logic statement for the inference graph in Figure 5.12 can be seen at the bottom of that figure.

ALGORITHM 15: ConstructLogicString

Input: G - Graph
Input: *root* - root of subgraph in G
Input: *currString* - Logic String
Output: S - Logic String

```
1  S = currString
2  a = node at root
3  case type value for a do
4  |  case "AND" do
5  |  |  S = S + "("
6  |  |  C = list of children for a
7  |  |  forall c ∈ C do
8  |  |  |  S = ConstructLogicString(G,c,S)
9  |  |  |  S = S + " ∧ "
10 |  |  end
11 |  |  S = S + " 1 )"
12 |  end
13 |  case "OR" do
14 |  |  S = S + "("
15 |  |  C = list of children for a
16 |  |  forall c ∈ C do
17 |  |  |  S = ConstructLogicString(G,c,S)
18 |  |  |  S = S + " | "
19 |  |  end
20 |  |  S = S + " 0 )"
21 |  end
22 |  case "FACT" do
23 |  |  S = S + "fact value for a"
24 |  end
25 end
26 return S
```

Algorithm 16 (new) determines if a given participant fact combination is a cover for the identified inference violations. For this method, a cover is defined as the facts in list C that positionally correspond to bits set to 1 in the binary string in variable

b, the binary representation of the combination. The value of the cover participating facts are set to "1" in the logic string, all others are set to "0". A bitwise evaluation is then performed on the logic string which returns a boolean result. If the result is "TRUE" the combination is a cover, if "FALSE", it is not.

ALGORITHM 16: TestForCover

Input: S - Logic String
Input: C - Participating Fact List
Input: b - binary representation of combination to test
Output: *cover* - boolean

```
1 q = |C|
2 for k = 0 to q do
3 |   set all instances of C[k] in S to b[k]
4 end
5 cover = bitwise evaluation of logic string S
6 return cover
```

Algorithm 17 is a revision of the Powerset method (previously called Quaternary-Powerset). The primary modification to this algorithm is to only construct removal combinations that are covers. We are not constructing combinations that contain generalizations. Revisions to this algorithm include:

- change from quaternary (base 4) counting to binary (base 2) counting; we only have two states, do nothing or remove
- test each removal combination to determine if it provides a cover; ignore if it does not
- only build removal combinations based on verified covers; remove code for building generalization combinations

Algorithm 18 (new) is a generic bit adder. The adder uses the value of t as its base number system adding 1 to the binary representation of an integer stored in the elements of the array b[ ]. This algorithm is used by Algorithm 17 to build combinations.

ALGORITHM 17: Powerset (Efficient)

Input: C - Candidates
Input: S - Logic String
Output: CS - Combination Set

```
1  t = 2                                              // do binary counting
2  q = |C|                                            // candidate list size
3  b[0...q-1] = 0                                     // initialize bit mask
4  p=t^q                                              // number of combinations
5  for i = 0 to p do
6  |  b[ ] = BitAdder(b[ ],t)
7  |  cover = TestForCover(S,C,b)
8  |  if cover then
9  |  |  new combination with combinationCost = 0
10 |  |  for k = 0 to q do
11 |  |  |  if b[k] > 0 then include fact k
12 |  |  |  |  new alteration with alterationCost = 0
13 |  |  |  |  alteration source data item = C[k]
14 |  |  |  |  altered target data item = root concept from ontology O
15 |  |  |  |  add 1.0 to alterationCost add alteration to combination
16 |  |  |  |  add alterationCost to combinationCost
17 |  |  |  end
18 |  |  end
19 |  end
20 |  add combination to CS
21 end
22 return CS
```

ALGORITHM 18: BitAdder

Input: $b[\ ]$ - Candidate Bit Indicators
Input: t - Adder Base
Output: $b[\ ]$ - Candidate Bit Indicators + 1

```
1  add 1 to b[0]                                      // add base t
2  for j = 0 to t do
3  |  if b[j] < t then
4  |  |  exit inner loop                              // no carry digit
5  |  else
6  |  |  b[j] = 0
7  |  |  add one to b[j+1]                            // carry digit
8  |  end
9  end
```

The remaining algorithms support enhancements to our cost calculations. These algorithms collect meta-data about concepts, alterations, and combinations. The data is stored and aggregated in information vectors as described in Section 5.3. This meta-data includes alteration cost, inference participation count, and entropy value.

Algorithm 19 determines the number of inferences a concept participates in. Counts are tracked separately for participation in safe violations and violation inference. This method generates the counts by following the inference path for each satisfied inference rule in the database. For each rule generating an inference, Algorithm 20 is called to recursively traverse the rule's inference path. During this traversal, counts are incremented as ground facts in rule bodies on the inference paths are discovered. The algorithms differentiate between counting authorized and unauthorized inferences by declaring a traversal as safe (FALSE) or unsafe/violation (TRUE) in call to the ContributionToInferenceConcept method.

ALGORITHM 19: ContributionToInferenceSet

Input: DB - minimal initial database
Input: O - Ontology
Input: DB_{mf} - fix point of DB_m over O
Input: SV - security violation list

```
1  begin
2      R = rules over O inferring conclusions in DB_mf
3      forall r ∈ R do
4          c = conclusion of r
5          if c ∈ SV then
6              ContributionToInferenceConcept (DB_m, TRUE, r, R)
7          else
8              ContributionToInferenceConcept (DB_m, FALSE, r, R)
9          end
10     end
11 end
```

The last meta-data element needed for our information vector is the concept's uncertainty or entropy value. The entropy calculation is based on the work of Calmet and Daemi [5]; they calculate a concept's entropy based on uncertainty that the

ALGORITHM 20: ContributionToInferenceConcept

Input: DB_m - minimal initial database
Input: v - violation indicator for c
Input: r - rule to process
Input: R - set of rules

```
1  begin
2  |  C = concepts in body of rule r
3  |  forall c ∈ C do
4  |  |  if c ∈ DB then
5  |  |  |  if v is True then
6  |  |  |  |  Increment c.ViolationParticipationCount
7  |  |  |  else
8  |  |  |  |  Increment c.ParticipationCount
9  |  |  |  end
10 |  |  else
11 |  |  |  r' = rule in R that concludes c
12 |  |  |  ContributionToInferenceConcept (DB_m, v, r, R)
13 |  |  end
14 |  end
15 end
```

concept is or is not actually one of its sub-concepts. In our approach, entropy is calculated by Algorithm 21 (ConceptEntropy). This algorithm first determines the degree of a concept (the number of sub-concepts under it in the ontology hierarchy tree) using Algorithm 22. The concept degree, along with the cardinality of the ontology, is then used by Algorithm 21 to calculate h, the entropy of concept c.

ALGORITHM 21: ConceptEntropy

Input: O - ontology
Input: c - concept (constant string)
Output: h - concept entropy of c

```
1  begin
2  |  M = ∅
3  |  g = |O|
4  |  d = ConceptDegree(O, c, M)
5  |  d̂ = (d/g)
6  |  h = −(d̂/2)log(d̂/2) − (1 − d̂/2)log(1 − d̂/2)
7  |  return h
8  end
```

Algorithm 22 interrogates the ontology tree using localized traversal to determine a given concept's degree. We define degree as the number of children under a concept in the ontology's 'ISA' hierarchy tree.

ALGORITHM 22: ConceptDegree

Input: c - concept
Input: O - ontology
Input: M - working list
Output: d - degree

```
1  begin
2      A = facts ∈ O matching (∗,'ISA', c)
3      d = 1
4      forall a ∈ A do
5          ĉ = subject component of fact in a
6          if a ∉ M then
7              add a to M
8              d = d + ConceptDegree (ĉ,O,M)
9          end
10     end
11     return d
12 end
```

5.5.2 Low-Cost High-Entropy Traversal

This discussion will follow the basic flow of Figure 5.16.

We first make a few trivial modifications to existing algorithms. In Algorithm 10, we remove the call to the Powerset method (Algorithm 17). We also alter Algorithms 10 and 15 to pass back the rooted tree graph G instead of the combination list SS.

Next, we modify Algorithm 9 to used the above modified algorithms along with Algorithm 24 (which will be discussed next). We present the modified version of Algorithm 9 as Algorithm 23.

Lastly, we present Algorithm 24, which builds the low-cost high-entropy inference disruption cover discussed in Section 5.4.3. The algorithm is recursive starting with the root node of the tree graph and performing a depth-first traversal of all nodes.

ALGORITHM 23: EfficientDisruptionTraversal

Input: DB_0 - initial database
Input: DB_m - minimal initial database
Input: O - Ontology
Input: v - violation threshold label
Input: λ - privacy mapping function
Input: DB_{mf} - fix point of DB_m over O
Input: SV - privacy violating concepts in DB_{mf}
Output: S - selected solution

```
1  begin
2      e = 0.5                                          // entropyThreshold
3      z = 5                                                 // sampleSize
4      ContributeToInferenceSet(DB_m,O,DB_mf,SV)
5      G = EfficientCombinations(DB_m,O,DB_mf,SV)          // modified
6      (S,E) = TraverseForCover(G)
7      P = highest entropy z concepts in S with entropy > e
8      found = false
9      while (P != ∅) AND (found = false) do
10         M = combination based on remove (P − S) and generalize S
11         DB_w = AlterData(DB_0,M)
12         DB_wf = Reason(DB_w,O)
13         C = set of inferred facts from DB_wf
14         SV' = PrivacyMapAndDetect(C, v, λ)
15         if SV' = ∅ then
16             found = true
17         else
18             Remove lowest entropy concept from P
19         end
20     end
21     if found then
22         S = P                              // at least 1 generalization
23     end
24     return S
25 end
```

ALGORITHM 24: TraverseForCover

Input: n - node
Output: S - cover list
Output: E - cost of S

```
1  begin
2      if n is an AND node then
3          S = ""
4          E = 0
5          forall c, child of n do
6              (s,e) = TraverseForCover(c)
7              S = S + s
8              E = E + e
9          end
10     end
11     if n is an OR node then
12         S = ""
13         E = max integer
14         forall c, child of n do
15             (s,e) = TraverseForCover(c)
16             if e < E then
17                 S = s
18                 E = e
19             end
20         end
21     end
22     if n is an FACT node then
23         S = fact
24         E = cost, combination of entropy and safe inference disruption
25     end
26     return S, E
27 end
```

When visiting each node of the tree, the appropriate action will be taken depending on if the node type is "AND", "OR", or "FACT". Collections of selected facts and aggregated cost information are passed back to the recursive calls for construction of the final low-cost high-entropy cover. This algorithm will always return a minimal removal disruption cover with the highest entropy facts found. The cover returned is also the most likely cover to have facts generalized and still provide disruption. In this context, minimal is based on potential to be generalized and preserve violation disruption (higher entropy).

5.6 Efficient Approach Empirical Results

We initially enhanced our prototype implementation to include the low-cost selection methods introduced in this chapter. Given that the low-cost selection semi-exhaustive approach (Section 5.4.1) still showed potential for requiring a large number of evaluations, we only tested and reported results on the low-cost selection heuristic approach (Section 5.4.2). To evaluate the impact of the low-cost selection methods on computational efficiency, we tested the updated prototype using the same tests and data sets that were used for evaluation of our exhaustive model. The results of our executions are shown in Table 5.6. This table takes the exhaustive results from Chapter 4 and adds data from our efficient executions. Since, for this evaluation, we are only concerned with overall system performance, we remove the Load, Reason, Discover, and Evaluate timing columns. Columns to show the number of disruption covers found and the total execution time for the efficient model were added. Rows are still arranged into three groups based on the number of privacy violations known to exist (0, 1, 2, or 4). Within each group, we provide five files based on patient counts in the file (100, 500, 1000, 2500, and 5000). In each test execution, the expected violations were identified and proper privacy labels applied. A valid solution with minimal cost was found and reported for each test execution.

Table 5.6 Prototype execution summary with efficient methods timing collected for 100, 500, 1000, 2500, and 5000 instance database grouped by violation count.

Patients	Asserted	Inferred	Facts	Violations	Participants	Exhaustive Solutions	Inference Covers	Exhaustive Time (ms)	Efficient Time (ms)	Performance Increase
100	369	376	745	0	0	0	0	745	550	26%
500	1741	1806	3547	0	0	0	0	926	744	20%
1000	3456	3592	7048	0	0	0	0	1019	961	6%
2500	8596	8947	17543	0	0	0	0	1387	1380	1%
5000	17168	17875	35043	0	0	0	0	1952	2026	-4%
100	403	414	817	1	3	63	7	1744	3011	-73%
500	1904	1983	3887	1	3	63	7	3193	3389	-6%
1000	3779	3941	7720	1	3	63	7	4617	3717	19%
2500	9400	9811	19211	1	3	63	7	9707	4836	50%
5000	18777	19606	38383	1	3	63	7	18189	6311	65%
100	404	417	821	2	6	4095	49	22593	3875	83%
500	1905	1986	3891	2	6	4095	49	81837	4431	95%
1000	3780	3944	7724	2	6	4095	49	164497	5197	97%
2500	9401	9814	19215	2	6	4095	49	444847	6051	99%
5000	18778	19609	38387	2	6	4095	49	929728	8697	99%
100	403	414	817	4	10	1048575	637	3691252	7708	>100%
500	1904	1983	3887	4	10	1048575	637	18357303	7564	>100%
1000	3779	3941	7720	4	10	1048575	637	33996151	8423	>100%
2500	9400	9811	19211	4	10	1048575	637	74869826	8741	>100%
5000	18777	19606	38383	4	10	1048575	637	148965619	10485	>100%

As seen in the Table 5.6, execution time was reduced dramatically by using the low-cost selection heuristic methods from this chapter. In our exhaustive approach, the number of fact modifying combinations needing to be tested was based on the list of participating facts modified three ways (removal, one level of generation, two levels of generation). This equated to $(4_p - 1)$ fact modifying combinations, where t is the size of the participating facts size. With our modifications, we initially only look at removal inference covers which do not need to be tested. We then only need test a small set of the lowest cost covers to see if generalization will decrease their cost.

As can be seen in Figure 5.17, timing for all test runs using the modified prototype barely exceeds 10 seconds (10000 ms). When compared to the timing graph for the exhaustive approach (Figure 4.4), the total execution times of the efficient method prototype decreased by several orders of magnitude over the prototype using the exhaustive approach. This is attributable to the drastic drop in the number of times the reasoner had to be executed for the testing of alteration combinations.

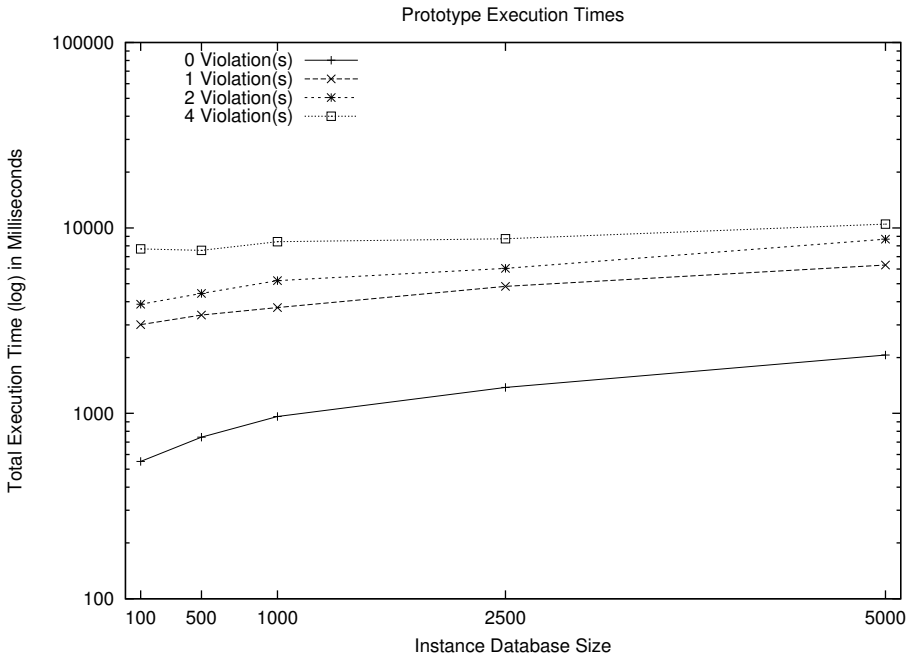


Figure 5.17 Prototype execution timing for execution using efficient methods. Trend lines show execution time (log) in milliseconds. Individual line corresponds to number of violations found in data instance.

Comparing specific violation data set tests for each prototype, we can see that efficiencies are gained as the participating fact count (driven by violation count) increases. For the group of data sets with no violations, the execution times are very similar (Figure 5.18). A small amount of overhead was added when interrogating inference paths to support generation of covers, this overhead quickly diminishes as the data sets size increases.

When a single violation is introduced, the performance improvements start to

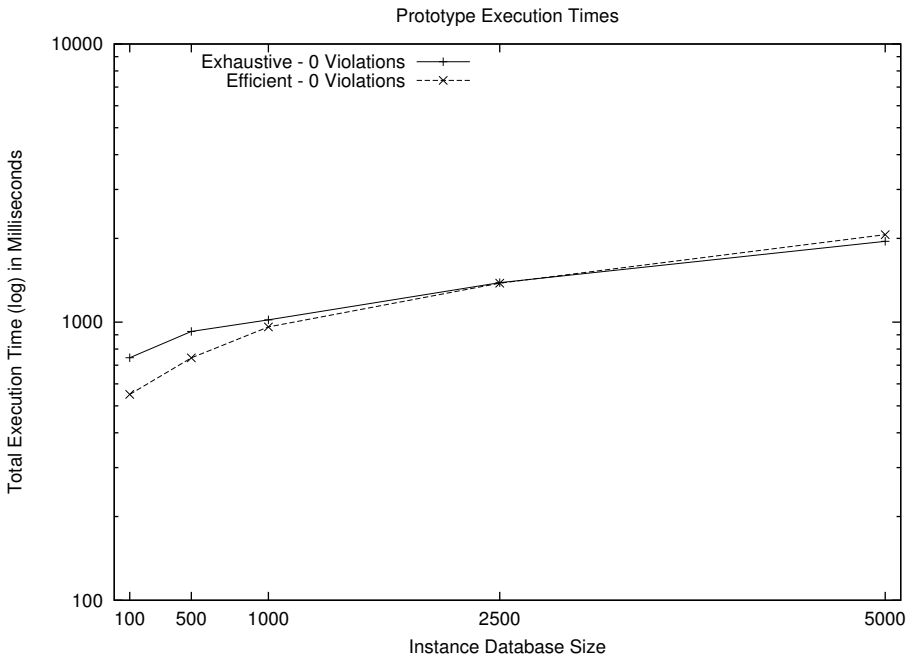


Figure 5.18 Prototype execution timing for 0 violation data sets. Trend lines show execution time (log) in milliseconds. Individual line corresponds to number of violations found in data instance.

become apparent (Figure 5.19). For this group of data sets, the exhaustive approach had to evaluate 63 combinations. The efficient approach identified seven covers and reached a 65% performance increase on the 5000 fact dataset.

The data sets containing two violations show a much larger increase in performance (Figure 5.20). For this group of data sets, the exhaustive approach had to evaluate 4095 combinations. The efficient approach identified 49 covers and reached a 99% performance increase on the 5000 fact dataset.

Lastly, the data sets containing four violations are shown in Figure 5.20. For the 5000 fact data set, the exhaustive approach had to evaluate 1,048,575 alteration combinations and required approximately 40 hours to complete. The efficient approach identified 637 covers and completed in approximately 10 seconds. While we have relaxed our minimality property by not testing every possible combination of generalizations across all alteration combinations, we are now able to quickly find a

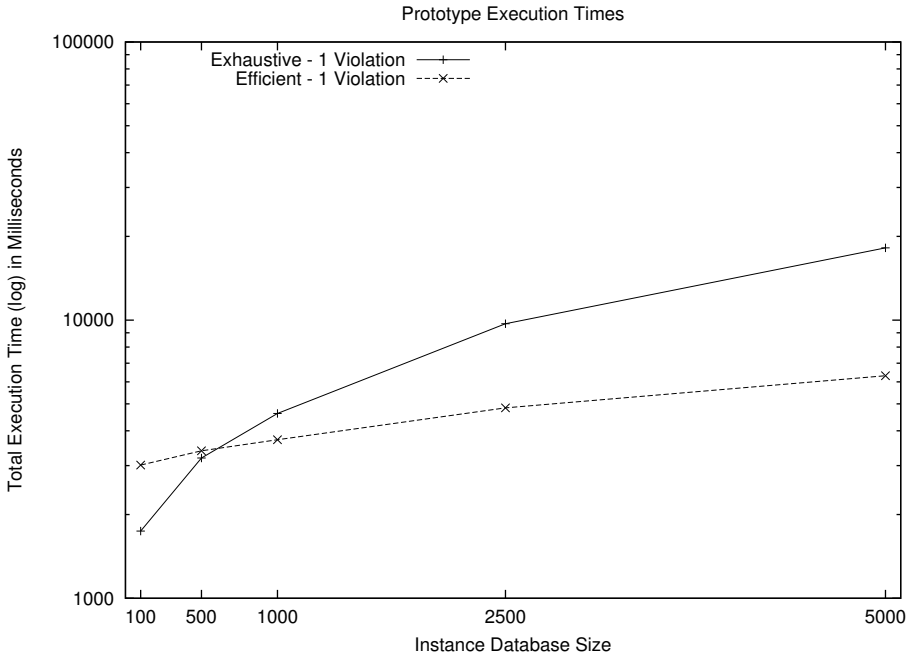


Figure 5.19 Prototype execution timing for 1 violation data sets. Trend lines show execution time (log) in milliseconds. Individual line corresponds to number of violations found in data instance.

solution in a feasible time that has practical cost.

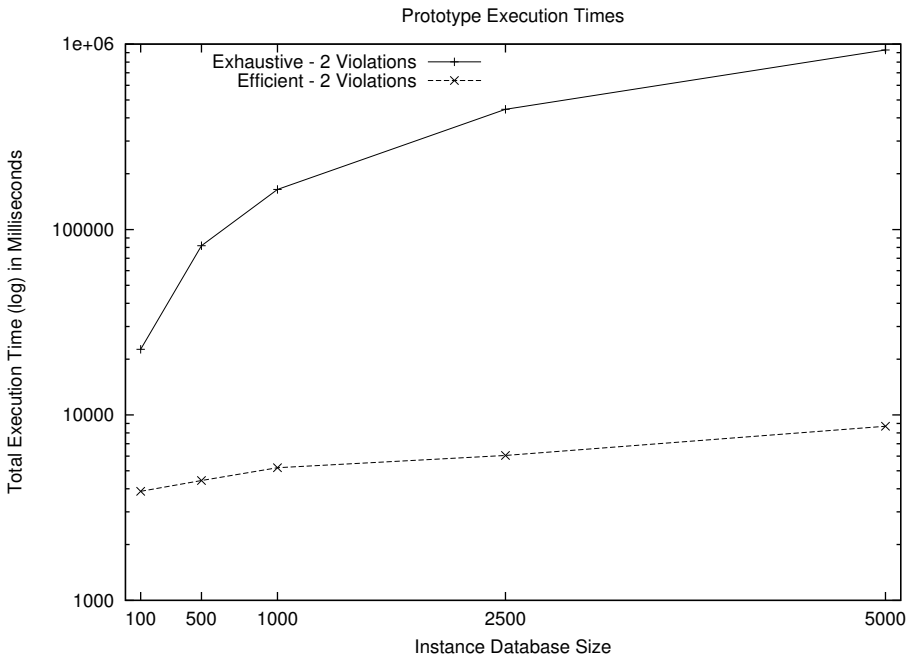


Figure 5.20 Prototype execution timing for 2 violation data sets. Trend lines show execution time (log) in milliseconds. Individual line corresponds to number of violations found in data instance.

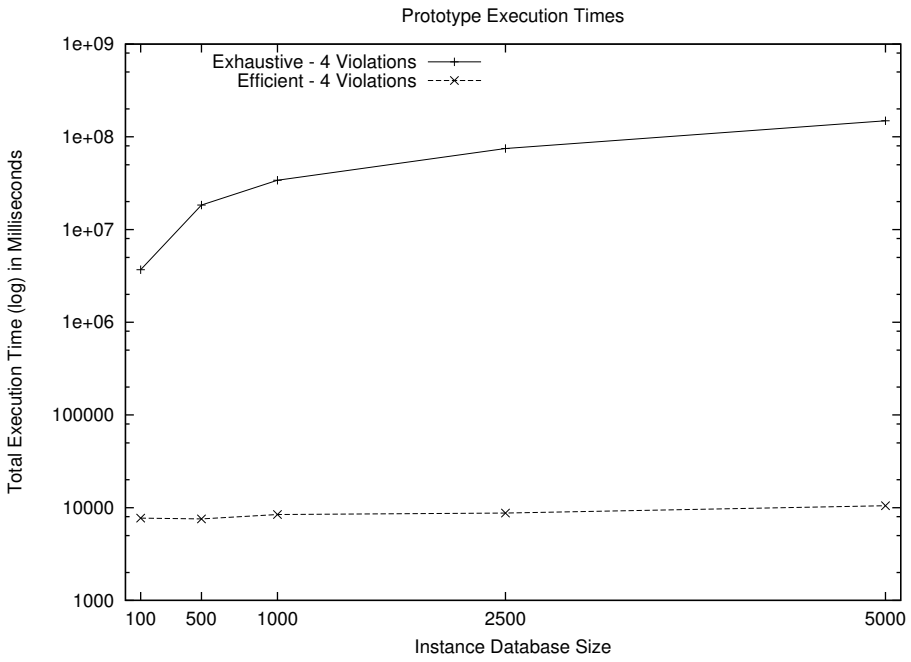


Figure 5.21 Prototype execution timing for 4 violation data sets. Trend lines show execution time (log) in milliseconds. Individual line corresponds to number of violations found in data instance.

Chapter 6

Privacy & Safety

In our initial work, we considered privacy violations to be generated data items that exceed the authorization level of the intended recipient. The privacy labels used to determine a violation are based on the common regulatory privacy rules and protocols found in healthcare (i.e., HIPAA). In this section, we investigate other domains that may influence the privacy authorization and the release of data.

We consider the following:

- Personal Concerns - Patients may have personal reasons for not wanting particular data items to be released. A patient's concerns may apply to both data in the initial data set (intended release) as well as data inferred using domain knowledge (unintended release), but we only address the unintended release of information in this work.

- Safety Concerns - Clinicians, or other subject matter experts, may feel that the suppression (generalization or removal) of certain data items create safety concerns for the patient. Released data may be used to treat a patient, evaluate their condition, recommend a treatment or medication, or determine if they pose a population health risk. Reducing data availability could negatively impact an assessment in each of these areas. In these cases, violations with related safety concerns may need to be left intact even though they generate a violation. Since the violation will persist, the privacy label of the data set should be elevated to the appropriate level to coincide with the permitted violations.

We will develop methods to incorporate personal and safety concerns in our violation evaluation and data release process. The new labels introduced by this work will be captured and stored with existing meta-data and considered when calculating alteration cost (see Section 5.3). There may be cases where the Safety or Privacy label of a data item either requires or prohibits the modification of a data item; in other cases, the label may influence a solution containing that data item by increasing or decreasing its cost. Data to support these scenarios will all be captured in the cost vector discussed in Section 5.3.

6.1 Privacy - Patient Preference

In our heuristic approach to disrupting violation inferences, we either remove a data item or generalize it. We try to minimize the number of data items that are altered, but there may be cases where removal or generalization of a specific data item is desired. We envision several use cases where a patient requests that certain data items not be released without modification. To accommodate these use cases, we would provide the ability for a patient to provide a preference indicator for any data item they are concerned about being released without modification. We propose the patient be presented with a sliding scale of values $(0, \ldots, 3)$ to represent the degree of patient preference to be considered in the inference disruption process (Table 6.1).

Table 6.1 Patient preference value indicates the level of requirement that something should not be released.

Safety Degree	Meaning
0	Fine With Releasing
1	Prefer To Not Be Released
2	Should Not Be Released
3	Must Not Be Released

The patient preference value would be added to the data item information vector (Definition 5.3.1) for each fact in the patient's data. Any data items that do not

have patient preference values specified would default to 0. The value in the data item information vectors would be aggregated in the combination information vector (Definition 5.3.3) for each combination cover. When placing the combinations in cost order, (Definition 5.4.1) we alter the calculation of $\boldsymbol{Impact}_S$ (Definition 5.3.4) to subtract the aggregate patient preference value from the impact value. This subtraction will cause combinations containing data items that the patient has indicated concern over to be given a lower cost, making them more likely to be part of the the chosen solution. There is no guarantee that a patient's preference request will be honored, but since it reduces cost, its selection will be more seen as more desirable by the framework.

6.2 Safety

In the patient preference section, we strive to ensure a data item is altered if possible, but there may also be cases where a data custodian wants to ensure that a data item be released unaltered (or minimally altered). We envision use cases where certain data items need to be released "as-is" with no modification (removal or generalization). These cases could include data requirements in policies, protocols, or regulations. There could also be clinical safety reasons where altering the data may not be in the best medical interest of the patient. Cases may also exist where release of specific data is needed by first responders or clinicians for emergency medicine situations. We refer to these scenarios collectively as safety use cases. To accommodate the safety use cases, we propose an additional sliding scale of values $(0, \ldots, 3)$ to represent the degree of patient safety to be considered in the inference disruption process (Table 6.2)..

The safety value would be added to the data item information vector (Definition 5.3.1) for each fact in the patient's data. Any data items that do not have safety preference values specified would default to 0. During the collection of violation inference

Table 6.2 Safety values indicate the level of requirement that something must be released.

Safety Degree	Meaning
0	Fine With Not Releasing
1	Prefer To Be Released
2	Should Be Released
3	Must Be Released

path participating facts (Algorithm 11), line 6 would be altered to ignore any facts that have a safety value of 5. This ensures that data items which the custodian has indicated "must be released" are not considered for removal or generalization. Facts found on violation inference paths with safety values less than 3 are added to the participating fact. Their inclusion is not guaranteed, but prioritized as follows. The safety values in data item information vectors would be aggregated in the combination information vector (Definition 5.3.3) for each combination cover as with patient preference. When placing the combinations in cost order, (Definition 5.4.1) we alter the calculation of $\boldsymbol{Impact}_S$ calculation (Definition 5.3.4) to add the aggregate safety value to the impact value. This addition will cause combinations containing data items that the custodians have indicated concern over altered release to be given a higher cost, making them more likely to not be the chosen solution. There is no guarantee that a safety request (less than 3) will be honored, but since it increases cost, its selection will be more seen as less desirable by the framework.

6.3 Privacy & Safety Approach

Information Vector

In this section, we provide modification to the information vector described in 5.3. To support safety and patient preference, we introduce new meta-data in the information vector. This information will be used to refine the combination cost calculations. We still maintain the generic form of this vector for modification at three levels:

1. meta-data is tracked at the individual data item level

2. meta-data is tracked in aggregate for unaltered and altered data items (data item alteration)

3. meta-data is tracked in aggregate for the set of alterations (alteration combination)

We re-define Definitions 5.3.1, 5.3.2, and 5.3.3 to accommodate the safety and privacy concepts as follows:

Definition 6.3.1 (Enhanced Data Item Information Vector)**.** Let f be a data item. We say the data item information vector for f, $V_f =< a_1, \ldots, a_6 >$, is a set of attributes, such that a_1 is the alteration cost associated with f, a_2 is the list of distinct unsafe inferences that f participates in, a_3 is the ontology hierarchy depth of f, and a_4 is the entropy (uncertainty) of f. a_5 is the safety level setting of f. a_6 is the patient preference level setting of f.

Definition 6.3.2 (Enhanced Alteration Information Vector)**.** Let m be a data item alteration, with source concept c_1 and target concept c_2. We say the alteration information vector for m, $V_m =< a_1, \ldots, a_6 >$, is a set of attributes, such that a_1 is the sum of alteration cost associated with data items in a, a_2 is a 2-tuple where the first element is a distinct list of safe inferences for c_1 and the second element a similar list for c_2, a_3 is the average ontology hierarchy depths for c_1 and c_2, and a_4 is the difference between the entropy for c_1 and c_2, $(c_2 - c_1)$. a_5 is the safety level setting of c_1. a_6 is the patient preference level setting of c_1.

Definition 6.3.3 (Enhanced Combination Information Vector)**.** Let c be a set of alterations forming a PFC. We say the combination information vector for c, $V_c =< a_1, \ldots, a_6 >$, is a set of attributes, such that a_1 is the sum of alteration cost associated for all alterations in c, a_2 is a 2-tuple where each element is a union of the

corresponding element for all alterations in c, a_3 is the average ontology hierarchy depth for all alterations in c, and a_4 is the sum of the entropy values for all alterations in c. a_5 is the safety level setting of f. a_6 is the patient preference level setting of f. a_5 is the sum of the safety level setting for all alterations in c. a_6 is the sum of the patient preference level setting for all alterations in c.

Alteration Impact

To accommodate patient preference and safety requests, we enhance the calculation of $\boldsymbol{Impact}_S$ the alteration impact of a PFC.

Definition 6.3.4 (Enhanced Alteration Impact Value)**.** The alteration impact value, denoted $\boldsymbol{Impact}_S$, is the alteration cost $\boldsymbol{Cost}_S$ enhanced by the number of safe inferences impacted, patient preference, and safety values. Let S be a PFC, U the number of safe inferences disrupted by S, P the patient preference value, and V the safety value. We calculate the alteration impact value of S as follows: $\boldsymbol{Impact}_S = \boldsymbol{Cost}_S + (\boldsymbol{Cost}_S * U) - P + V$.

Exclusion from Cover

To ensure that data items indicated by the data custodian as "must be released" (safety value of 3), we modifiy Algorithm 11, line 6 from "If $f \notin C$" to "if $f \notin C$ and f safetyValue $\neq 3$". This change will ensure that data items marked with a safety value of 3 are not included in the participant list and not a fact in disruption covers.

Example 6.3.1 (Privacy:Patient Preference & Safety)**.** Let there be an Ontology tree as shown in Figure 5.10 with entropy values shown in Table 5.2. Let Figure 6.1 show a rule that was satisfied by C_4, C_6, C_7 and generated the privacy violation V_1. Let C_4, C_6, C_7 have patient preference and safety values as shown in Table 6.3. When constructing the list of violation participating facts, only C_4 and C_7 would be included; fact C_6 would not be included on the list since its safety value is set to 3.

Lastly, let there be a participating fact combination C, as shown in Table 6.3, with two facts linked to ontology concepts C_6 and C_7 with alterations $A1$ and $A2$ such that $A2$ generalizes C_6 one level to C_2 and $A1$ generalizes C_7 two levels to C_1. The $\boldsymbol{Impact}_S$ value for $C1$ would be $\boldsymbol{Impact}_{C1} = 1.25 + (1.25 * 5) - 2 + 1$, which is 6.5. The value of $\boldsymbol{Impact}_{C1}$ would then be used to determine where $C1$ fell in Cost Order (Definition 5.4.1).

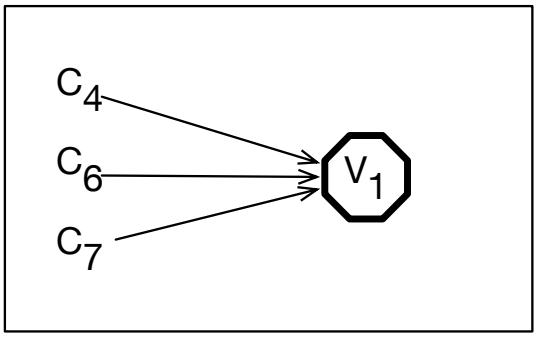


Figure 6.1 Multi-identity nature of concepts when they are potential targets for generalization or removal.

Table 6.3 Information Vectors - table shows data item information vectors.

Information Vector	Safe Inferences	Depth	Entropy	Patient Preference	Safety
Data Item C_7	{a,b,c}	4	0	0	1
Data Item C_4	{b,d,e}	2	0.4138	3	3
Data Item C_6	{a,b}	3	0.3095	2	0

Table 6.4 Information Vectors - table shows data item, alteration, and combination information vectors.

Information Vector	Alteration	Safe Inferences	Depth	Entropy	Privacy	Safety
Data Item C_7	1.0	{a,b,c}	4	0	0	1
Data Item C_1	-0.25	{b,d,e}	2	0.8524		
Alteration A1: $C_7 \rightarrow C_1(C_7'')$	0.75	({a,b,c},{b,d,e})	3	0.8524	0	1
Data Item C_6	1.0	{a,b}	3	0.3095	2	0
Data Item C_2	-0.5	{b,c,d}	2	0.6500		
Alteration A2: $C_6 \rightarrow C_2(C_6')$	0.5	({a,b},{b,c,d})	2.5	0.3405	2	0
Combination C1 = {A1,A2}	1.25	({a,b,c},{b,c,d,e})	2.75	1.1929	2	1

6.4 Privacy & Safety Empirical Results

We ran experiments using the new cost function presented in this chapter. Our empirical results, with respect to privacy, safety and security, support our claim that these requirements can be seamlessly integrated into the PIA framework. The additional cost calculations described in this chapter did not impact the system's performance.

Chapter 7

Graphical User Interface

Up to this point, our privacy evaluation and modification process has been static. Rules are established in advance based on what data custodians or patients "think" they are willing to release. There is no feedback during the process or a way for custodians or patients to see "what if" scenarios in relation to what data might be generated and released.

People trust systems and presume that their information is safe. They assume that data they do not want released, will not be. In most cases, people understand neither what data was collected on them nor what release of that data would imply; much less what could be inferred from this data when reasoned over along with domain knowledge.

We will investigate the design and development of a user-friendly graphical user interface (GUI) prototype that would allow a data custodian or patient to understand in advance the privacy aspects of healthcare data being released and what can be inferred from that data. The GUI prototype leverages services from our PIA framework (see Figure 7.1). This prototype would reason over a patient's data and show items, both existing and generated, along with identified privacy levels. Since patient records are becoming voluminous, we would only show data identified as violations, but allow for "drill down" to access other data items. The intent would not be casual browsing of patient records, but rather identification of data items that are potential violations (privacy, personal, or safety).

Information and examples would be available to explain in laymen's terms how

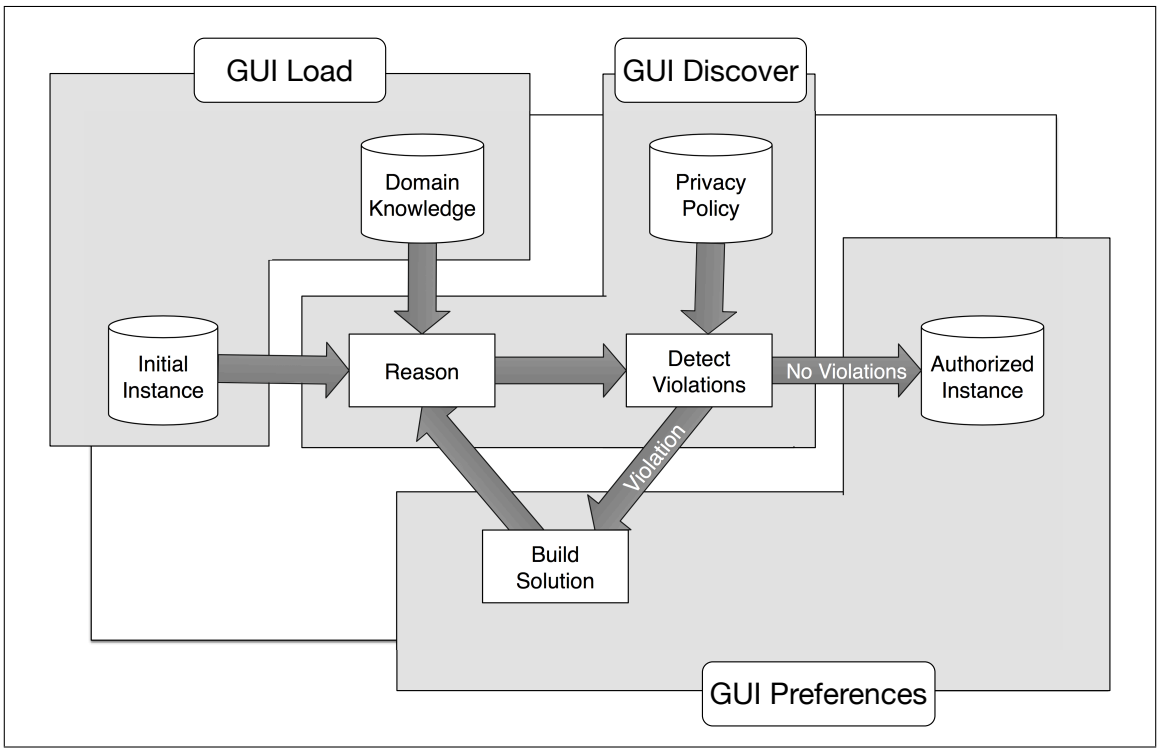


Figure 7.1 Graphical User Interface integration with the PIA framework.

an attacker might leverage data items to their advantage.

Lastly, the GUI prototype would allow the user to set personal preferences for release of their data. These preferences would be included in the extended privacy model (Chapter 6).

7.1 High-Level Design

The user interface of our GUI prototype is constrained to one window with launching of external viewers for additional information content (PDF, web pages, videos). The interface consists of a tool bar along the left edge of the window and a primary viewing area (see Figure 7.2).

Buttons on the tool bar provide functions that trigger various actions: review information / tutorials, load patient data, discover new information, provide program information, review / set patient preferences, and exit application. The primary

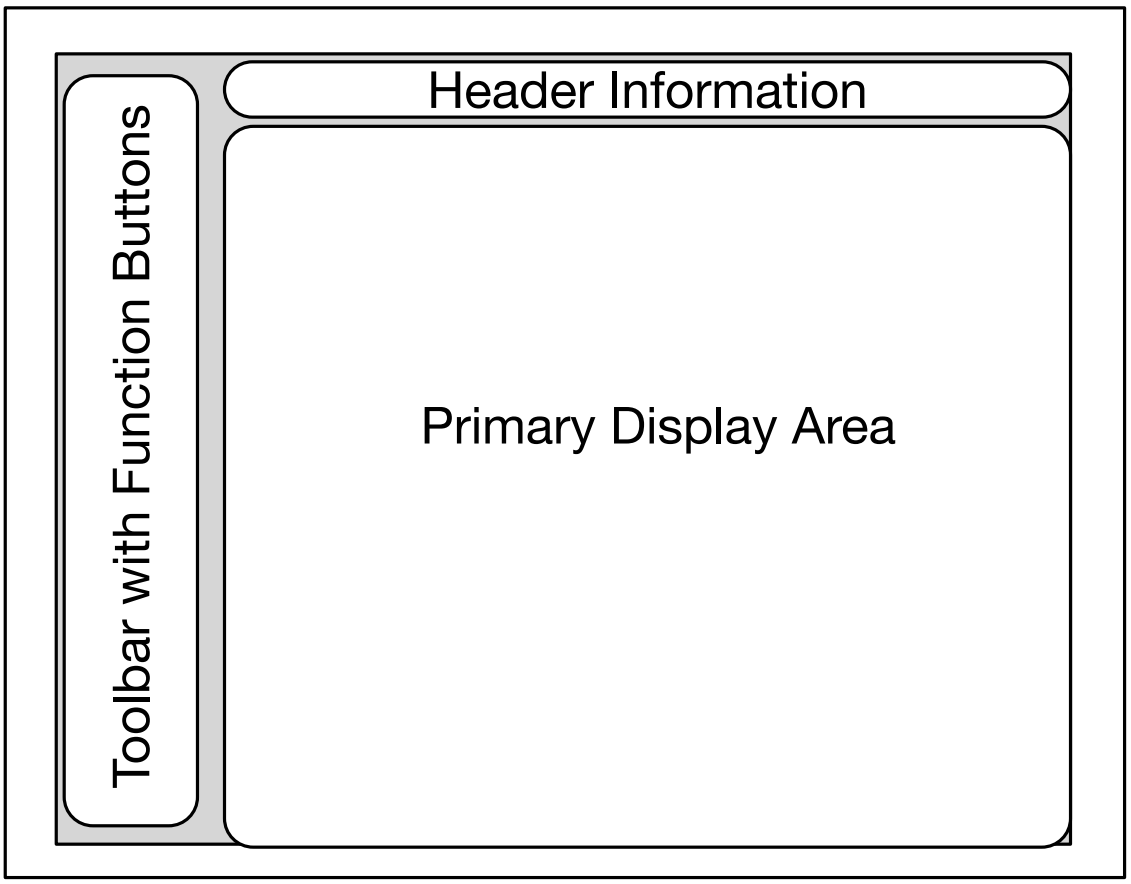


Figure 7.2 Graphical User Interface. Basic display layout.

viewing area provides a panel that is context-aware and displays the selected information. The initial implementation has functions to display various types of reference information, allow viewing of a dynamic ontology tree (tree is enhanced with data loaded and inferred from reasoning), patient preference management, and a list of common reasons for data release. Additional functions may be easily added as well as additional information content.

The high-level functions associated with each the buttons on the toolbar are as follows:

- "Privacy and Knowledge Information " - When this button is selected, the user is presented high level description of the four categories of information provided

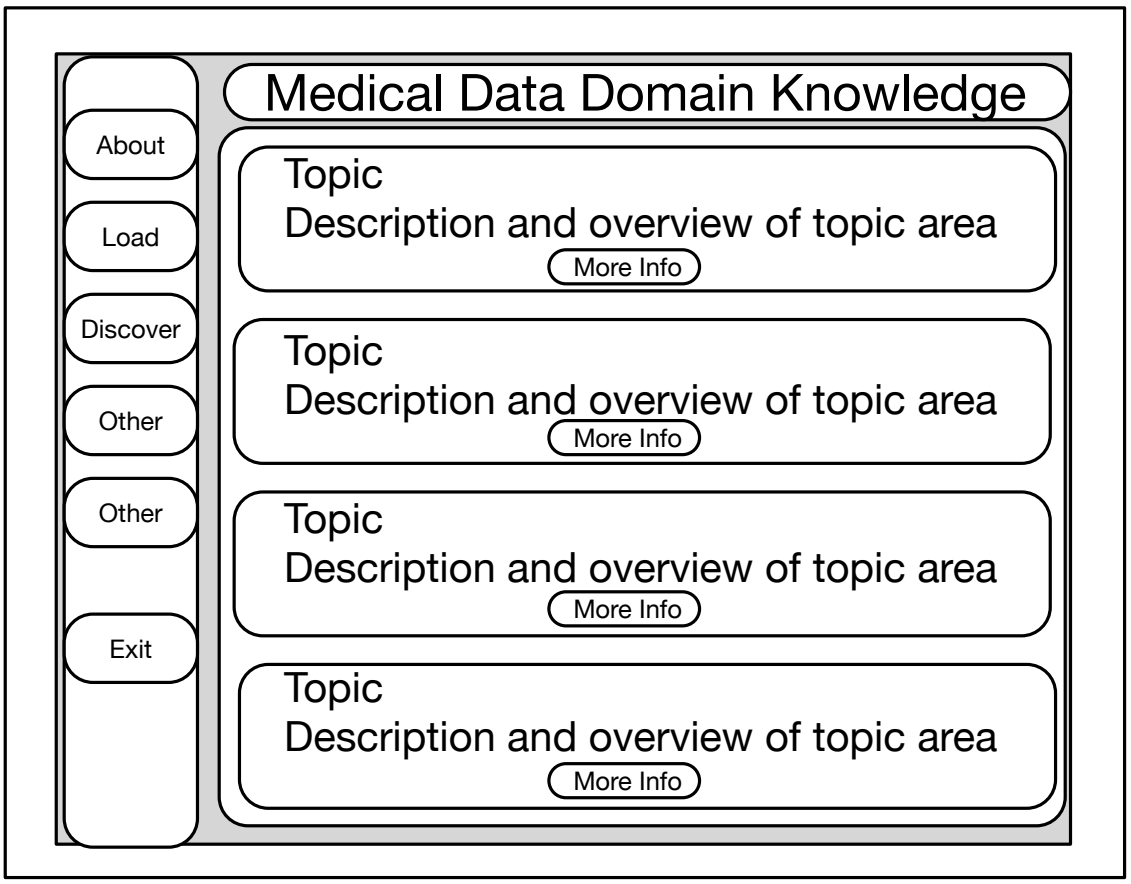


Figure 7.3 Graphical User Interface. Description of tutorials and reference material areas with button to access information.

(see Figure 7.3). The four information categories are: "Data Privacy Regulations", "Consent Information", "Healthcare Institution Information", and "Healthcare Domain Knowledge". Each category has a narrative description along with a button to "Learn More". Clicking the "Learn More" button will launch a category-specific window with buttons to access additional content. Currently each of the categories has three content buttons. These buttons may be programmed to display content from PDF files, web pages, or videos. They may also bring up a application generated display in the main viewing area. Most content is currently displayed by launching content-specific native view-

ers for the content type.

- "Places your Data is Sent" - When this button is selected, the user is presented with a table giving information on typical types of data sets released from a healthcare facility, the reason that data is released, who within the healthcare facility is responsible for the release, and what agency / entity it is released to. There is also an explanation of accountability of disclosure and what it means to the patient.

- "Load Patient Data" - When this button is selected, an ontology diagram is displayed (see Figure 7.4). There is initially no patient data incorporated in the diagram and no inferred data is shown. The user is presented a load file dialog. Once a file is selected, patient data is loaded and the ontology diagram updated to show patient data incorporated as additional facts (vertices with edges to show relationships to ontology concepts). The ontology diagram is an active graph – clicking on a node or vertex will instantiate a popup window that gives a brief explanation of the data item.

- "Discover Information" - When this button is selected, the ontology diagram is updated to incorporate any generated data coming from the inference process along with appropriate relationships to patient data and ontology concepts. If the patient data has not been loaded, the user is prompted to load patient data before running discover information. Any data items identified as privacy violations will be colored red and all participation facts will be colored yellow. Additional nodes and vertices on the ontology graph are activated to give explanations if selected.

- "Patient Privacy Preferences" - When this button is selected, the ontology diagram with patient data incorporated will be shown. If patient has not been loaded, user is prompted to load patient data before running discover. Any

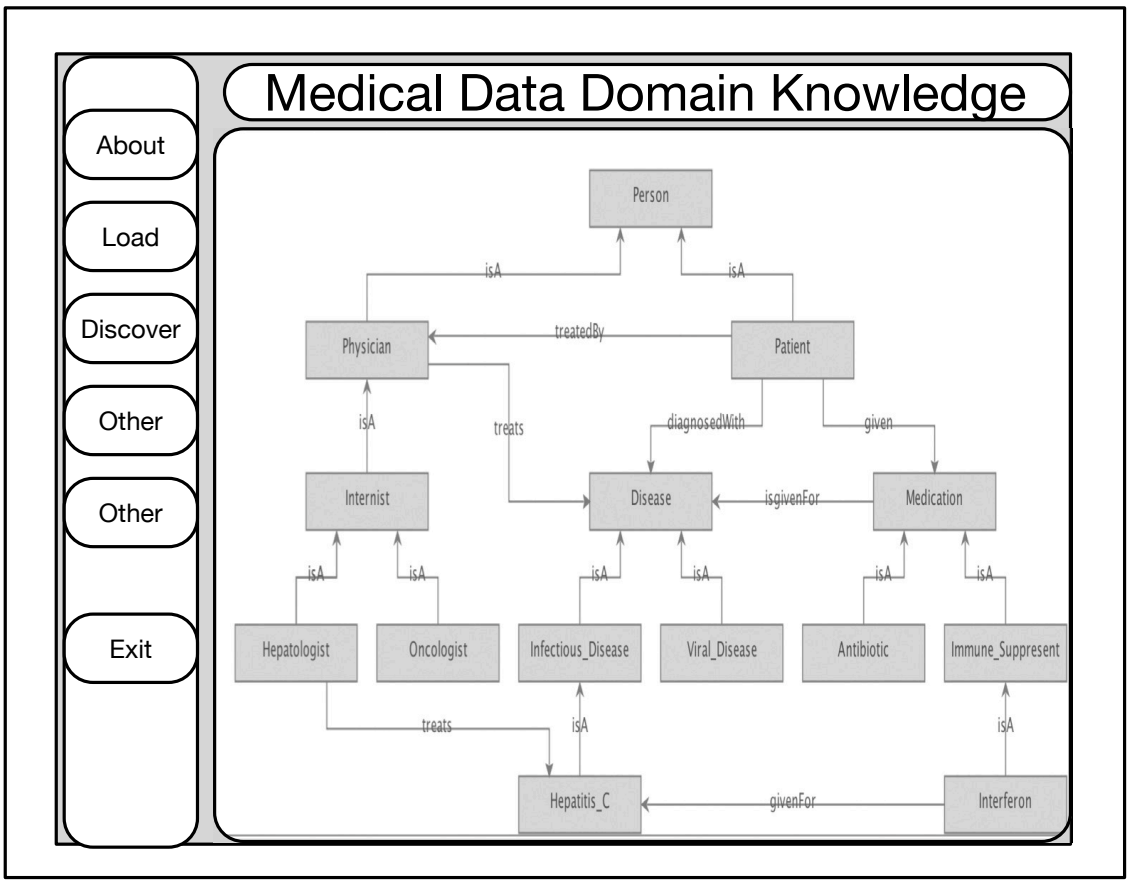


Figure 7.4 Graphical User Interface. Initial view of the ontology with no data loaded or inferred.

facts that have had their preference previously set will be shown. Previous preferences are ready from a file at the time of patient data load. User will have the option to set or change preferences and save information.

- "About" – When this button is selected, an informational window is displayed that provides application information.

- "Exit" - Selecting the exit button will exit the application.

The GUI prototype is a stand-alone application, but leverages services from the PIA framework (see Figure 7.1). Since both applications are developed in Java, the

GUI interface is able to access all PIA public methods by including the framework prototype package when being compiled. As shown in the diagram, PIA services are leveraged at several points in the GUI workflow. When ontologies and patient data are loaded, GUI utilizes the PIA functions that support instance database and domain knowledge loading. When we discover new information in the GUI, we utilize the PIA functions that manage privacy policy, reason over patient data and domain knowledge and detect security violations. Lastly, PIA and GUI share the same patient preference files. These files are also used by PIA to determine cost, which drives construction of minimal disruption covers. The GUI also uses PIA output to make patient preference suggestions.

Once GUI prototype application has been started, use cases are as follows:

- View Information - PDF file, Webpage, Video or application generated content
  - Select "Privacy and Knowledge Information" button on toolbar
  - Main viewing area updated to show content topics
  - Press “More Information” button in Main Viewing Area corresponding to desired content
  - A window with content options will be presented. Select the desired content.
  - External viewer will be launched or main viewing area content changed
  - Video will start playing or static content will be displayed
  - Close external viewer (if launched) when finished
- Review places data may be sent
  - Select “Places Your Data is Sent” button tab on toolbar
  - Main viewing area is updated to show table of common places patient data is sent

- Load Patient Data
  - Select “Load Patient Information” button tab on toolbar
  - Press Load Data button in toolbar
  - Select Patient data file
  - Ontology is automatically updated to show patient data
- Discovered Data
  - Load Patient Data (above)
  - Select “Discover Information” button tab on toolbar
  - Ontology is shown and automatically updated to show generated data and violations
- Patient Privacy Preferences
  - Load Patient Data (above)
  - Select “Patient Privacy Preferences” button tab on toolbar
  - Main viewing area is updated to ontology with patient data incorporated
  - Suggested preference setting colored green
  - Current preferences colored yellow
  - Select relationship (graph edge) to update preference on patient data (not the ontology)
  - Select save to store preferences
- Prototype Application
  - Press the “About the Application” button on the toolbar
  - Information about the application will be displayed in a popup window

- Exit the application
  - Press the “Exit” button on the toolbar
  - Confirm exit

7.2 Prototype Implementation

The GUI prototype implementation is an extension of the Java project used for the PIA efficient framework prototype from Chapter 5. Separate Java packages were created to hold components of the prototype. We approached the GUI design using a Model View Controller (MVC) architecture, each being a separate Java package in the project.

- The model package manages all data. Data structures are built to hold patient, ontology and reference information. This package also managed the interface with the efficient framework for data loading, model rendering, and inference discovery / path interrogation.
- The view package manages all graphical aspects of the prototype. The prototype’s primary window is divided up into a row of buttons and an information frame. Buttons are used to load and discover information as well as provide version information and exit the prototype. Panels in the main display area are used to display reference information as well as provide a view of the ontology graph and impact of the ontology and inference process. All graph vertices and edges are active and provide information when selected.
- The controller manages provides a generic interface between the model and view packages. This allows each package to communicate with the other without having to understand its methods.

We use the Java Swing libraries to construct the visual components of the prototype. The Java package mxGraph (JGraphX) [20] is used to manage and render the ontology graph. Vertices and edges are managed as objects so they can be expanded as data is loaded and reasoning is performed.

Selected screen shots from execution of the GUI prototype are show as follows:

- Figure 7.5 shows the initial screen displayed at startup. This screen is linked to the Privacy and Knowledge Information button on the toolbar
- Figure 7.6 shows a loaded ontology. All vertices and edges are active and provide additional information if clicked.
- Figure 7.7 shows the loaded ontology once data has been loaded and reasoning performed.
- Figure 7.9 shows a table of sample destinations that patient data may be sent to.
- Figure 7.9 shows the loaded ontology with patient data. Patient preferences are indicated and patients can update the preference values.

7.3 Patient Reference Information

Part of the objective of the GUI prototype is to provide patients with information on various privacy aspects of their healthcare data. We envision this data describing three topic ares: US National and State privacy regulations, various types of patient consent, and local healthcare facility privacy policy and associated research rules. We have included samples of information from these three areas in the GUI prototype. When possible, we are using hyperlinks to pull in active content. This reduces maintenance, provides the most up-to-date information, and makes sure we do not misinterpret an agency's official stand on patient privacy. When active content is not

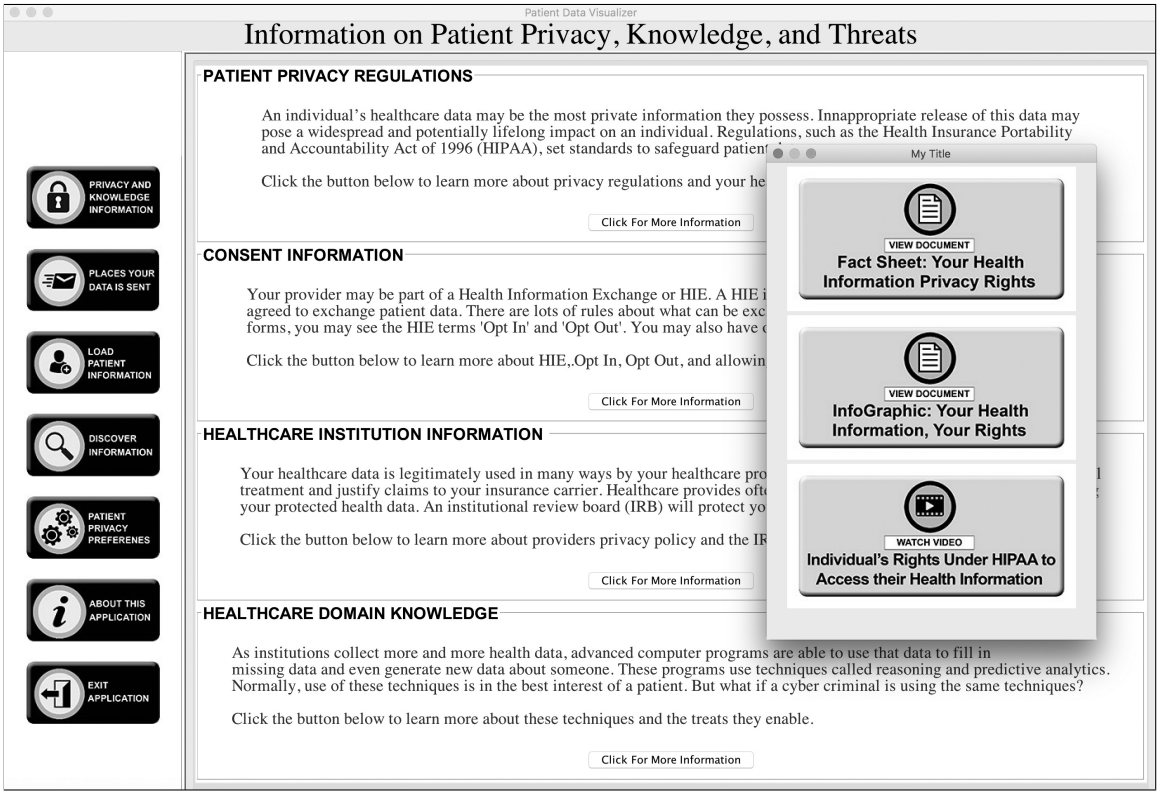


Figure 7.5 Graphical User Interface. View of educational information.

available, we construct sample content based on information from discussions with healthcare data analysts responsible for release of information and accountability of disclosure activities. Non-active content is shown only as a sample of what could be presented and is the author's best-effort collection of data from several sources. This information is not intended to be authoritative or complete and may not reflect the letter of the law, regulation, or policy. All data sources should be edited for content and appropriate reading levels before being released to the general public. The following information is representative of relevant content and is "linked to" in the GUI prototype:

- US National Healthcare Data Privacy
  - Easy-to-read fact sheet entitled "Your health information privacy rights"

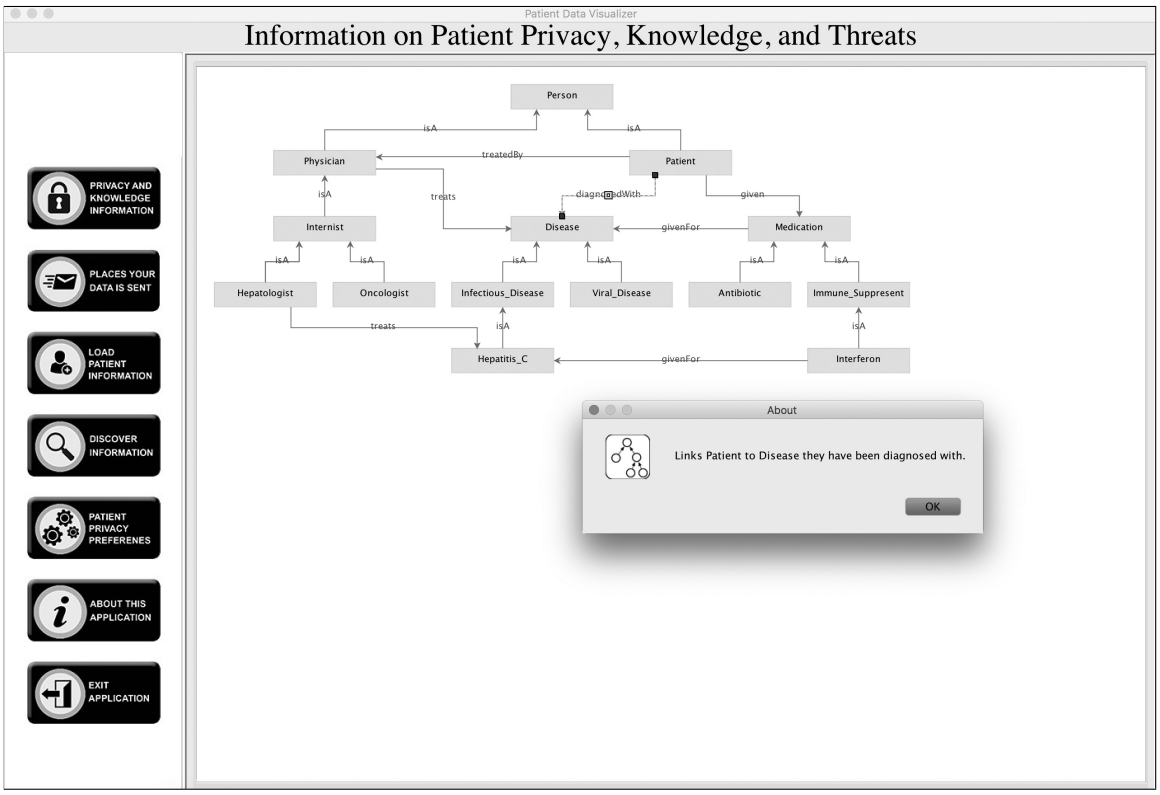


Figure 7.6 Graphical User Interface. Initial view of ontology with no data loaded or inferred.

  - An infographic entitled “Your Health Information, Your Rights”
  - Video on “Individual’s Rights under HIPAA to Access their Health Information”

- Patient Consent
  - Web content on “Meaningful Consent”
  - Health Information Exchange - “Opt In Opt Out” Video
  - Web content on “Informed Consent for Human Research"

- Healthcare Facility -
  - example notice of privacy practices overview
  - example notice of privacy practices agreement

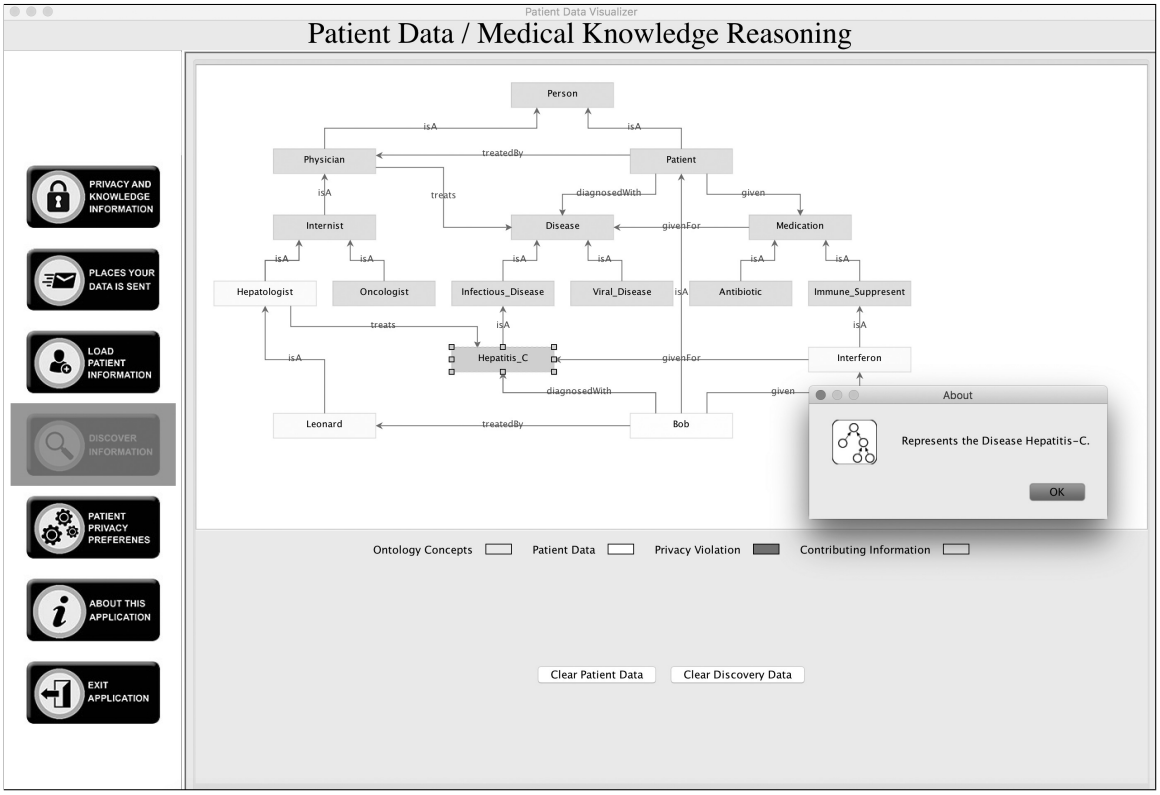


Figure 7.7 Graphical User Interface. View of ontology with patient data loaded and inferred.

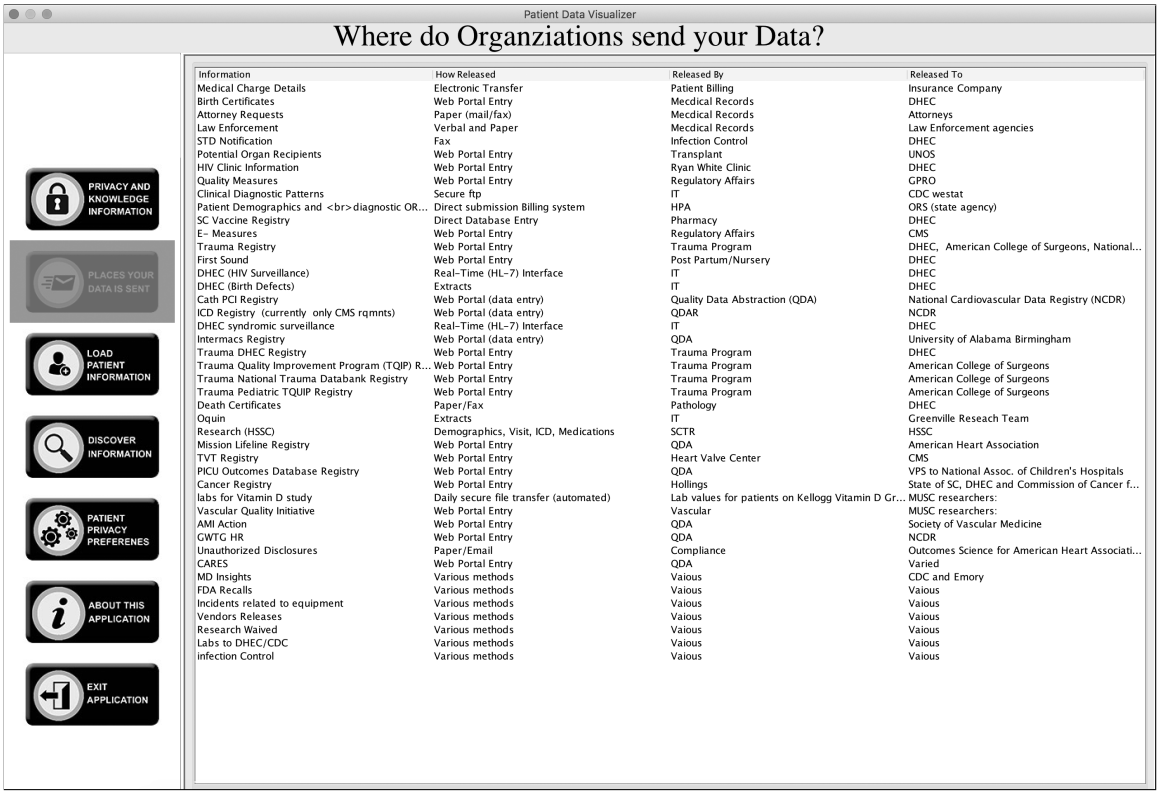
  - IRB policies and procedures

- Healthcare Domain Knowledge

  - Information on domain knowledge and reasoning. Examples given to help understand the semantic reasoning process.

  - Information on the inference problems and how this process can generate undesired inferences. Examples of the type of information that can be generated and ways that information can be used against them.

  - Presentation of a sample ontology graph that can be explored by drilling down on vertices and edges.



Information	How Released	Released By	Released To
Medical Charge Details	Electronic Transfer	Patient Billing	Insurance Company
Birth Certificates	Web Portal Entry	Mecdical Records	DHEC
Attorney Requests	Paper (mail/fax)	Mecdical Records	Attorneys
Law Enforcement	Verbal and Paper	Mecdical Records	Law Enforcement agencies
STD Notification	Fax	Infection Control	DHEC
Potential Organ Recipients	Web Portal Entry	Transplant	UNOS
HIV Clinic Information	Web Portal Entry	Ryan White Clinic	DHEC
Quality Measures	Web Portal Entry	Regulatory Affairs	GPRO
Clinical Diagnostic Patterns	Secure ftp	IT	CDC westat
Patient Demographics and diagnostic OR...	Direct submission Billing system	HPA	ORS (state agency)
SC Vaccine Registry	Direct Database Entry	Pharmacy	DHEC
E- Measures	Web Portal Entry	Regulatory Affairs	CMS
Trauma Registry	Web Portal Entry	Trauma Program	DHEC, American College of Surgeons, National...
First Sound	Web Portal Entry	Post Partum/Nursery	DHEC
DHEC (HIV Surveillance)	Real-Time (HL-7) Interface	IT	DHEC
DHEC (Birth Defects)	Extracts	IT	DHEC
Cath PCI Registry	Web Portal (data entry)	Quality Data Abstraction (QDA)	National Cardiovascular Data Registry (NCDR)
ICD Registry (currently only CMS rqmnts)	Web Portal (data entry)	QDAR	NCDR
DHEC syndromic surveillance	Real-Time (HL-7) Interface	IT	DHEC
Intermacs Registry	Web Portal (data entry)	QDA	University of Alabama Birmingham
Trauma DHEC Registry	Web Portal Entry	Trauma Program	DHEC
Trauma Quality Improvement Program (TQIP) R...	Web Portal Entry	Trauma Program	American College of Surgeons
Trauma National Trauma Databank Registry	Web Portal Entry	Trauma Program	American College of Surgeons
Trauma Pediatric TQUIP Registry	Web Portal Entry	Trauma Program	American College of Surgeons
Death Certificates	Paper/Fax	Pathology	DHEC
Oquin	Extracts	IT	Greenville Reseach Team
Research (HSSC)	Demographics, Visit, ICD, Medications	SCTR	HSSC
Mission Lifeline Registry	Web Portal Entry	QDA	American Heart Association
TVT Registry	Web Portal Entry	Heart Valve Center	CMS
PICU Outcomes Database Registry	Web Portal Entry	QDA	VPS to National Assoc. of Children's Hospitals
Cancer Registry	Web Portal Entry	Hollings	State of SC, DHEC and Commission of Cancer f...
labs for Vitamin D study	Daily secure file transfer (automated)	Lab values for patients on Kellogg Vitamin D Gr...	MUSC researchers:
Vascular Quality Initiative	Web Portal Entry	Vascular	MUSC researchers:
AMI Action	Web Portal Entry	QDA	Society of Vascular Medicine
GWTG HR	Web Portal Entry	QDA	NCDR
Unauthorized Disclosures	Paper/Email	Compliance	Outcomes Science for American Heart Associati...
CARES	Web Portal Entry	QDA	Varied
MD Insights	Various methods	Vaious	CDC and Emory
FDA Recalls	Various methods	Vaious	Vaious
Incidents related to equipment	Various methods	Vaious	Vaious
Vendors Releases	Various methods	Vaious	Vaious
Research Waived	Various methods	Vaious	Vaious
Labs to DHEC/CDC	Various methods	Vaious	Vaious
infection Control	Various methods	Vaious	Vaious

Figure 7.8 Graphical User Interface. View of places a healthcare facility may send your data.

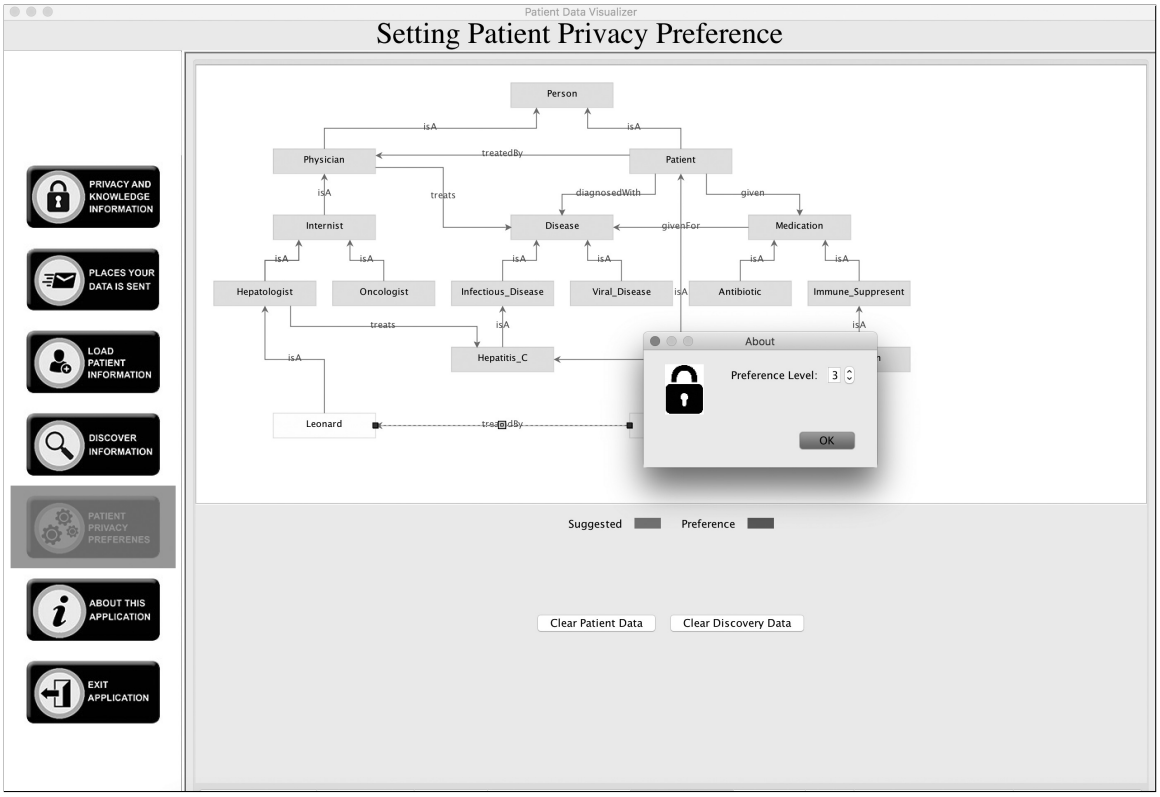


Figure 7.9 Graphical User Interface. View of ontology with preference selector active for a relation.

Chapter 8

Conclusions & Future Research

Conclusions

In this dissertation, we investigate privacy violations occurring when non-confidential patient data is combined with medical domain ontologies to disclose a patient's protected health information (PHI). We have shown that reasoning over non-sensitive patient data and domain knowledge can generate a threat to patient privacy. We have developed a formal framework to detect and eliminate privacy threats. We have also implemented our privacy protection methods.

Our optimal solution uses an exhaustive method and evaluates the cost of every possible solution that eliminates undesired inferences. We propose ontology-guided data item alteration for inference removal. We have proven that it is possible to detect and block all undesired inferences that resulted from combining non-sensitive data with domain knowledge. We also showed that our approach provided maximal data availability. Our theoretical contributions include: 1.) development of a Pre-Release Inference Analyzer framework, 2.) inference removal methods based on optimal and heuristic-based approaches, and 3.) graphical user interface to support patient-centric privacy. We propose a cost measurement based on the number of data items that must be altered to eliminate undesired inferences, the number of non-violating inferences that are impacted by the alterations, patient preferences, and medical safety characteristics.

We have built a proof-of-concept prototype of our models. Our empirical results show that privacy protection against undesired inferences can be achieved. Our op-

timal cost solution, i.e., exhaustive approach, was not computationally efficient and did not scale. We developed heuristics to improve the performance efficiency of our approach while still providing a cost-effective solution. We proposed methods using heuristics to reduce potential candidate solutions and quickly determined a feasible solution. Our empirical results using heuristics reduced the computational complexity by several orders of magnitude. We also enhanced our privacy model to incorporate safety and patient privacy preferences. The performance characteristics of our prototype implementation were not impacted by the extended cost function. Lastly, we developed a graphical tool to support patients' understanding of the privacy settings. The tool provides tutorials as well as the ability for a patient to view their data along with data generated by the inference process.

Future Research

Future research areas include the following:

- Investigate the practical aspects of integrating the PIA framework with commercial healthcare applications. Medical databases, such as Epic, Cerner, and Meditech, thoroughly restrict data-flows among trust domains. However, recent announcements, such as the Apple's inclusion of a patient Heath Record module on their iPhone, indicates that this strictly controlled environment will change.

- Investigate integration options between the GUI prototype and commercial Electronic Medical Record (EMR) systems. Specifically, investigate options for use of the HL7 Fast Healthcare Interoperability Resources (FHIR) standard. FHIR is quickly becoming available on must EMR platforms.

- Investigate incorporation of publicly available data in domain knowledge reasoning. Patients are, sometimes unknowingly, allowing personal devices, sensors

and applications to collect and store data relevant to their health. If accessible, this data could be leveraged to broaden the unintended inference threat.

- Investigate options for integration of the PIA framework and the GUI prototype with larger multi-ontology domain knowledge bases. Use existing data to calculate entropy of data items and concepts.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant, *Privacy-preserving data mining*, SIGMOD Rec. **29** (2000), no. 2, 439–450.

[2] Apache, *Apache jena*, 2017.

[3] Lauren B Becnel, Smita Hastak, Wendy Ver Hoef, Robert P Milius, MaryAnn Slack, Diane Wold, Michael L Glickman, Boris Brodsky, Charles Jaffe, Rebecca Kush, et al., *Bridg: a domain information model for translational and clinical protocol-driven research*, Journal of the American Medical Informatics Association.

[4] A. Brodsky, C. Farkas, and S. Jajodia, *Secure databases: constraints, inference channels, and monitoring disclosures*, IEEE Transactions on Knowledge and Data Engineering **12** (2000), no. 6, 900–919.

[5] Jacques Calmet and Anusch Daemi, *From entropy to ontology*, CYBERNETICS AND SYSTEMS 2004 - AT2AI-4: FROM AGENT THEORY TO AGENT IMPLEMENTATION, 2004, p. 2004.

[6] Ellick M. Chan, Peifung E. Lam, and John C. Mitchell, *Understanding the challenges with medical data segmentation for privacy*, Presented as part of the 2013 USENIX Workshop on Health Information Technologies (Washington, D.C.), USENIX, 2013.

[7] Feixiong Cheng and Zhongming Zhao, *Machine learning-based prediction of drug–drug interactions by integrating drug phenotypic, therapeutic, chemical, and genomic properties*, Journal of the American Medical Informatics Association **21** (2014), no. e2, e278–e286.

[8] Juliana Medeiros Destro, Julio Cesar dos Reis, Ariadne Maria Brito, Rizzoni Carvalho, and Ivan Luiz Marques Ricarte, *Influence of semantic similarity measures on ontology cross-language mappings*, Proceedings of the Symposium on Applied Computing (New York, NY, USA), SAC '17, ACM, 2017, pp. 323–329.

[9] Murthy V. Devarakonda, Neil Mehta, Ching-Huei Tsou, Jennifer J. Liang, Amy S. Nowacki, and John Eric Jelovsek, *Automated problem list generation and physicians perspective from a pilot study*, International Journal of Medical Informatics **105** (2017), 121 – 129.

[10] D. Edberg, L. OMara, and J. Wendel, *Finding value while planning a statewide health information exchange*, 2014 47th Hawaii International Conference on System Sciences, Jan 2014, pp. 2778–2787.

[11] D Elliott Bell and Leonard J. La Padula, *Secure computer system: Unified exposition and multics interpretation*, Tech. Report MTR-2997, The MITRE Corporation, Bedford, MA, 03 1976.

[12] Csilla Farkas and Sushil Jajodia, *The inference problem: A survey*, SIGKDD Explor. Newsl. **4** (2002), no. 2, 6–11.

[13] Jesualdo Tomás Fernández-Breis, José Alberto Maldonado, Mar Marcos, María del Carmen Legaz-García, David Moner, Joaquín Torres-Sospedra, Angel Esteban-Gil, Begoña Martínez-Salvador, and Montserrat Robles, *Leveraging electronic healthcare record standards and semantic web technologies for the identification of patient cohorts*, Journal of the American Medical Informatics Association **20** (2013), no. e2, e288–e296.

[14] Office for Civil Rights (OCR), *Summary of the hipaa privacy rule*, U.S. Department of Health and Human Services, July 2013.

[15] Fabien Gandon and Guus Schreiber, *RDF 1.1 XML syntax*, W3C recommendation, W3C, February 2014, http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[16] Vaibhav Gowadia and Csilla Farkas, *Rdf metadata for xml access control*, Proceedings of the 2003 ACM Workshop on XML Security (New York, NY, USA), XMLSEC '03, ACM, 2003, pp. 39–48.

[17] Leonardo H. Iwaya, Fausto Giunchiglia, Leonardo A. Martucci, Alethia Hume, Simone Fischer-Hübner, and Ronald Chenu-Abente, *Ontology-based obfuscation and anonymisation for privacy*, pp. 343–358, Springer International Publishing, Cham, 2016.

[18] Amit Jain and Csilla Farkas, *Secure resource description framework: An access control model*, Proceedings of the Eleventh ACM Symposium on Access Control

Models and Technologies (New York, NY, USA), SACMAT '06, ACM, 2006, pp. 121–129.

[19] Amit Jain and Csilla Farkas, *Ontology-based authorization model for xml data in distributed systems*, ch. 3, pp. 57–82, IGI Global, Hershey, PA, USA, 2010.

[20] JGraph Ltd, *mxgraph*, 2018.

[21] Adeeb Noor, Abdullah Assiri, Serkan Ayvaz, Connor Clark, and Michel Dumontier, *Drug-drug interaction discovery and demystification using semantic web technologies*, Journal of the American Medical Informatics Association **24** (2017), no. 3, 556–564.

[22] Charles P. Pfleeger and Shari Lawrence Pfleeger, *Security in computing*, 4 ed., Prentice Hall, 2007.

[23] Manuel Quesada-Martínez, Jesualdo Tomás Fernández-Breis, and Robert Stevens, *Extraction and analysis of the structure of labels in biomedical ontologies*, Proceedings of the 2Nd International Workshop on Managing Interoperability and compleXity in Health Systems (New York, NY, USA), MIXHS '12, ACM, 2012, pp. 7–16.

[24] Z. Rashid, A. Basit, and Z. Anwar, *Trdbac: Temporal reflective database access control*, 2010 6th International Conference on Emerging Technologies (ICET), Oct 2010, pp. 337–342.

[25] Adam Wright, Skye Aaron, and David W. Bates, *The big phish: Cyberattacks against u.s. healthcare systems*, Journal of General Internal Medicine **31** (2016), no. 10, 1115–1118.

[26] Adam Wright, Justine Pang, Joshua C Feblowitz, Francine L Maloney, Allison R Wilcox, Harley Z Ramelson, Louise I Schneider, and David W Bates, *A method and knowledge base for automated inference of patient problems from structured data in an electronic medical record*, Journal of the American Medical Informatics Association **18** (2011), no. 6, 859–867.