Discovering Robust Knowledge from Dynamic Closed-World Data*

Chun-Nan Hsu and Craig A. Knoblock
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{chunnan,knoblock}@isi.edu
`http://www.isi.edu/sims/`

Abstract

Many applications of knowledge discovery require the knowledge to be consistent with data. Examples include discovering rules for query optimization, database integration, decision support, etc. However, databases usually change over time and make machine-discovered knowledge inconsistent with data. Useful knowledge should be *robust* against database changes so that it is unlikely to become inconsistent after database changes. This paper defines this notion of robustness, describes how to estimate the robustness of Horn-clause rules in closed-world databases, and describes how the robustness estimation can be applied in rule discovery systems.

Introduction

Databases are evolving entities. Knowledge discovered from one database state may become invalid or inconsistent with a new database state. Many applications require discovered knowledge to be consistent with the data. Examples are the problem of learning for database query optimization, database integration, knowledge discovery for decision support, etc. However, most discovery approaches assume static databases, while in practice, many databases are dynamic, that is, they change frequently. It is important that discovered knowledge is *robust* against data changes in the sense that the knowledge remains valid or consistent after databases are modified.

This notion of *robustness* can be defined as the probability that the database is in a state consistent with discovered knowledge. This probability is different from predictive accuracy, which is widely used in learning classification knowledge, because predictive accuracy measures the probability that knowledge is consistent with randomly selected unseen data instead of with an entire database state. This difference

---

Schema:
geoloc(name,glc_code,country,latitude,longitude),
seaport(name,glc_code,storage,rail,road,anch_offshore),
wharf(id,glc_code,depth,length,crane),
ship(name,class,status,fleet,year),
ship_class(class_name,type,draft,length,container_cap).
Rules:
R1: *;The latitude of a Maltese geographic location is greater ;than or equal to 35.89.*
?latitude $\geq$ 35.89 $\Leftarrow$
geoloc(_,_,?country,?latitude,_) $\wedge$?country = "Malta".

R2: *;All Maltese geographic locations are seaports.*
seaport(_,?glc_cd,_,_,_,_) $\Leftarrow$
geoloc(_,?glc_cd,?country,_,_) $\wedge$?country = "Malta".

R3: *;All ships built in 1981 belong to either* "MSC" *fleet or ;*"MSC Lease" *fleet.*
member(?R133,["MSC","MSC LEASE"]) $\Leftarrow$
ship(_,_,_,?R133,?R132) $\wedge$?R132 = 1981.

R4: *;If the storage space of a seaport is greater than 200,000 tons, ;then its geographical location code is one of the four codes.*
member(?R213,["APFD","ADLS","WMY2","NPTU"]) $\Leftarrow$
seaport(_,?R213,?R212,_,_,_) $\wedge$?R212 < 200000.

Table 1: Schema and rules of an example database

is significant in databases that are interpreted using the *closed-world assumption*. For a Horn-clause rule $C \leftarrow A$, predictive accuracy is usually defined as the conditional probability $Pr(C|A)$ given a randomly chosen data instance (Cohen 1993; 1995; Cussens 1993; Furnkranz & Widmer 1994; Lavrač & Džeroski 1994). In other words, it concerns the probability that the rule is valid with regard to a newly inserted data. However, databases also change by updates and deletions, and in a closed-world database, they may affect the validity of a rule, too. Consider the rule `R2` in Table 1 and the database fragment in Table 2. `R2` will become inconsistent if we delete the seaport instance labeled with a "*" in Table 1, because the value 8004 for variable `?glc_cd` that satisfies the antecedent of `R2` will no longer satisfy the consequent of `R2`. To satisfy the consequent of `R2` requires that there is a seaport instance with its `glc_cd` value 8004, according to the closed-world assumption.

Closed-world databases are widely used partly because of the limitation of the representation systems, but mostly because of the characteristics of application domains. Instead of being a piece of static state

*This research was supported in part by the National Science Foundation under Grant No. IRI-9313993 and in part by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under Contract No. F30602-94-C-0210.

```
geoloc("Safaqis",    8001, Tunisia, ...)
geoloc("Valletta",   8002, Malta,   ...)+
geoloc("Marsaxlokk", 8003, Malta,   ...)+
geoloc("San Pawl",   8004, Malta,   ...)+
geoloc("Marsalforn", 8005, Malta,   ...)+
geoloc("Abano",      8006, Italy,   ...)
geoloc("Torino",     8007, Italy,   ...)
geoloc("Venezia",    8008, Italy,   ...)
        ⋮
seaport("Marsaxlokk",   8003, ...)
seaport("Grand Harbor", 8002, ...)
seaport("Marsa",        8005, ...)
seaport("St Pauls Bay", 8004, ...)*
seaport("Catania",      8016, ...)
seaport("Palermo",      8012, ...)
seaport("Traparri",     8015, ...)
seaport("AbuKamash",    8017, ...)
        ⋮
```

Table 2: Example database fragment

of past experience, an instance of closed-world data usually represents a dynamic state in the world, such as an instance of employee information in a personnel database. Therefore, closed-world data tend to be dynamic, and it is important for knowledge discovery systems to handle dynamic and closed-world data.

This paper defines this notion of robustness, and describes how robustness can be estimated and applied in knowledge discovery systems. The key idea of our estimation approach is that it estimates the probabilities of data changes, rather than the number of possible database states, which is intractably large for estimation. The approach decomposes data changing transactions and estimates their probabilities using the Laplace law of succession. This law is simple and can bring to bear information such as database schemas and transaction logs for higher accuracy. The paper also describes a rule pruning approach based on the robustness estimation. This pruning approach can be applied on top of any rule discovery or induction systems to generate robust rules. Our experiments demonstrate the feasibility of our robustness estimation and rule pruning approaches. The estimation approach can also be used by a rule maintenance system to guide the updates for more robust rules so that the rules can be used with a minimal maintenance effort.

This paper is organized as follows. We establish the terminology on databases and rules in the next section. Then we define robustness and describe how to estimate the robustness of a rule. We present our rule pruning system next. Finally, we conclude with a summary of contributions and potential applications of the robustness estimation.

Terminology

This section introduces the terminology that will be used throughout this paper. In this paper, we consider *relational databases*, which consist of a set of *relations*. A relation is a set of *instances* (or *tuples*) of attribute-value vectors. The number of attributes is fixed for all instances in a relation. The values of attributes can be either a number or a string, but with a fixed type. Table 1 shows the schema of an example database with five relations and their attributes.

Table 1 also shows some Horn-clause rules describing the data. We adopt standard Prolog terminology and semantics as defined in (Lloyd 1987) in our discussion of rules. In addition, we refer to literals on database relations as *database literals* (e.g., seaport(_,?glc_cd,?storage,_,_,_)) and literals on built-in relations as *built-in literals* (e.g., ?latitude $\geq$ 35.89). We distinguish two classes of rules. Rules with built-in literals as their consequents (e.g., `R1`) are called *range rules*. The other class of rules contains rules with database literals as their consequents (e.g., `R2`). Those rules are *relational rules*. These two classes of rules are treated differently in robustness estimation.

A *database state* at a given time `t` is the collection of the instances present in the database at time `t`. We use the *closed-world assumption* (CWA) to interpret the semantics of a database state. That is, information not explicitly present in the database is taken to be false. A rule is said to be *consistent* with a database state if all variable instantiations that satisfy the antecedents of the rule also satisfy the consequent of the rule. For example, `R2` in Table 1 is consistent with the database fragment shown in Table 2, since for all `geoloc` tuples that satisfy the body of `R2` (labeled with a "+" in Table 1), there is a corresponding instance in `seaport` with a corresponding `glc_cd` value.

A database can be changed by *transactions*. A transaction can be considered as a mapping from a database state to a new database state. There are three kinds of primitive transactions — inserting a new tuple into a relation, deleting an existing tuple from a relation, and updating an existing tuple in a relation.

Estimating Robustness of Rules

This section first defines formally our notion of robustness and then describes an approach to estimating robustness. Following those subsections is an empirical demonstration of the estimation approach.

Robustness

Intuitively, a rule is robust against database changes if it is unlikely to become inconsistent after database changes. This can be expressed as the probability that a database is in a state consistent with a rule.

Definition 1 (Robustness for all states) *Given a rule r, let D denote the event that a database is in a state that is consistent with r. The robustness of r is $Robust_1(r) = \Pr(D)$.*

This probability can be estimated by the ratio between the number of all possible database states and the number of database states consistent with a rule. That is,

$$Robust_1(r) = \frac{\#\ of\ database\ states\ consistent\ with\ r}{\#\ of\ all\ possible\ database\ states}$$

There are two problems with this estimate. The first problem is that it treats all database states as if they are equally probable. That is obviously not the case in real-world databases. The other problem is that the number of possible database states is intractably large, even for a small database. Alternatively, we can define robustness from the observation that a rule becomes inconsistent when a transaction results in a new state inconsistent with the rule. Therefore, the probability of certain transactions largely determines the likelihood of database states, and the robustness of a rule is simply the probability that such a transaction is *not* performed. In other words, a rule is robust if the transactions that will invalidate the rule are unlikely to be performed. This idea is formalized as follows.

Definition 2 (Robustness for accessible states) *Given a rule r and a database in a database state denoted as d. New database states are accessible from d by performing transactions. Let t denote the transactions on d that result in new database states inconsistent with r. The robustness of r in accessible states from the current state d is defined as* $Robust(r|d) = \Pr(\neg t|d) = 1 - \Pr(t|d)$.

This definition of robustness is analogous in spirit to the notion of *accessibility* and the *possible worlds semantics* in modal logic (Ramsay 1988). If the only way to change database states is by transactions, and all transactions are equally probable to be performed, then the two definitions of robustness are equivalent. However, this is usually not the case in real-world databases since the robustness of a rule could be different in different database states. For example, suppose there are two database states d_1 and d_2 of a given database. To reach a state inconsistent with r, we need to delete ten tuples in d_1 and only one tuple in d_2. In this case, it is reasonable to have $Robust(r|d_1) > Robust(r|d_2)$ because it is less likely that all ten tuples are deleted. Definition 1 implies that robustness is a constant while Definition 2 captures the dynamic aspect of robustness.

Estimating Robustness

We first review a useful estimate for the probability of the outcomes of a repeatable random experiment. It will be used to estimate the probability of transactions and the robustness of rules.

Laplace Law of Succession *Given a repeatable experiment with an outcome of one of any k classes. Suppose we have conducted this experiment n times, r of which have resulted in some outcome C, in which we are interested. The probability that the outcome of the next experiment will be C can be estimated as* $\frac{r+1}{n+k}$.

A detailed description and a proof of the Laplace law can be found in (Howson & Urbach 1988). The Laplace

R1: ?latitude $\geq$ 35.89 $\Leftarrow$
geoloc(_,_,?country,?latitude,_) $\wedge$
?country = "Malta".

T1: One of the existing tuples of geoloc with its ?country = "Malta" is updated such that its ?latitude < 35.89.
T2: A new tuple of geoloc with its ?country = "Malta" and ?latitude < 35.89 is inserted to the database.
T3: One of the existing tuples of geoloc with its ?latitude < 35.89 and its ?country $\neq$ "Malta" is updated such that its ?country = "Malta".

Table 3: Transactions that invalidate `R1`

law applies to any *repeatable* experiments (e.g., tossing a coin). The advantage of the Laplace estimate is that it takes both known relative frequency and prior probability into account. This feature allows us to include information given by a DBMS, such as database schema, transaction logs, expected size of relations, expected distribution and range of attribute values, as prior probabilities in our robustness estimation.

Our problem at hand is to estimate the robustness of a rule based on the probability of transactions that may invalidate the rule. This problem can be decomposed into the problem of deriving a set of invalidating transactions and estimating the probability of those transactions. We illustrate our estimation approach with an example. Consider `R1` in Table 3, which also lists three mutually exclusive transactions that will invalidate `R1`. These transactions cover all possible transactions that will invalidate `R1`. Since `T1`, `T2`, and `T3` are mutually exclusive, we have $\Pr(\texttt{T1} \vee \texttt{T2} \vee \texttt{T3}) = \Pr(\texttt{T1}) + \Pr(\texttt{T2}) + \Pr(\texttt{T3})$. The probability of these transactions, and thus the robustness of `R1`, can be estimated from the probabilities of `T1`, `T2`, and `T3`.

We require that transactions be mutually exclusive so that no transaction covers another because for any two transactions t_a and t_b, if t_a covers t_b, then $\Pr(t_a \vee t_b) = \Pr(t_a)$ and it is redundant to consider t_b. For example, a transaction that deletes all `geoloc` tuples and then inserts tuples invalidating `R1` does not need to be considered, because it is covered by `T2` in Table 3.

Also, to estimate robustness efficiently, each mutually exclusive transactions must be minimal in the sense that no redundant conditions are specified. For example, a transaction similar to `T1` that updates a tuple of `geoloc` with its `?country = "Malta"` such that its `latitude` < 35.89 and its `longitude` > 130.00 will invalidate `R1`. However, the extra condition "`longitude` > 130.00" is not relevant to `R1`. Without this condition, the transaction will still result in a database state inconsistent with `R1`. Thus that transaction is not minimal for our robustness estimation and does not need to be considered.

We now demonstrate how Pr(`T1`) can be estimated only with the database schema information, and how we can use the Laplace law of succession when transaction logs and other prior knowledge are available. Since the probability of `T1` is too complex to be esti-

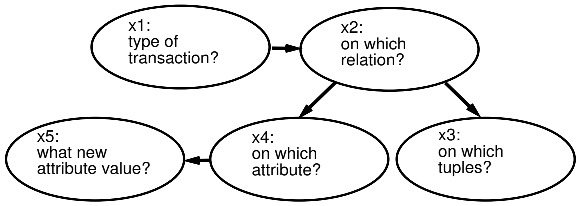

Figure 1: Bayesian network model of transactions

mated directly, we have to decompose the transaction into more primitive statements and estimate their local probabilities first. The decomposition is based on a Bayesian network model of database transactions illustrated in Figure 1. Nodes in the network represent the random variables involved in the transaction. An arc from node x_i to node x_j indicates that x_j is dependent on x_i. For example, x_2 is dependent on x_1 because the probability that a relation is selected for a transaction is dependent on whether the transaction is an update, deletion or insertion. That is, some relations tend to have new tuples inserted, and some are more likely to be updated. x_4 is dependent on x_2 because in each relation, some attributes are more likely to be updated. Consider our example database (see Table 1), the `ship` relation is more likely to be updated than other relations. Among its attributes, `status` and `fleet` are more likely to be changed than other attributes. Nodes x_3 and x_4 are independent because, in general, which tuple is likely to be selected is independent of the likelihood of which attribute will be changed.

The probability of a transaction can be estimated as the joint probability of all variables $\Pr(x_1 \wedge \cdots \wedge x_5)$. When the variables are instantiated for `T1`, their semantics are as follows:

- x_1: a tuple is updated.
- x_2: a tuple of `geoloc` is updated.
- x_3: a tuple of `geoloc` with its `?country = "Malta"` is updated.
- x_4: a tuple of `geoloc` whose `?latitude` is updated.
- x_5: a tuple of `geoloc` whose `?latitude` is updated to a new value less than 35.89.

From the Bayesian network and the chain rule of probability, we can evaluate the joint probability by a conjunction of conditional probabilities:

$$\begin{aligned}\Pr(\mathtt{T1}) &= \Pr(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5) \\ &= \Pr(x_1) \cdot \Pr(x_2|x_1) \cdot \Pr(x_3|x_2 \wedge x_1) \cdot \\ &\quad \Pr(x_4|x_2 \wedge x_1) \cdot \Pr(x_5|x_4 \wedge x_2 \wedge x_1)\end{aligned}$$

We can then apply the Laplace law to estimate each local conditional probability. This allows us to estimate the global probability of `T1` efficiently. We will show how information available from a database can be used in estimation. When no information is available, we apply the principle of indifference and treat all possibilities as equally probable. We now describe our approach to estimating these conditional probabilities.

- *A tuple is updated:*

$$\Pr(x_1) = \frac{t_u + 1}{t + 3}$$

where t_u is the number of previous updates and t is the total number of previous transactions. Because there are three types of primitive transactions (insertion, deletion, and update), when no information is available, we will assume that updating a tuple is one of three possibilities (with $t_u = t = 0$). When a transaction log is available, we can use the Laplace law to estimate this probability.

- *A tuple of* `geoloc` *is updated, given that a tuple is updated:*

$$\Pr(x_2|x_1) = \frac{t_{u,geoloc} + 1}{t_u + R}$$

where R is the number of relations in the database (this information is available in the schema), and $t_{u,geoloc}$ is the number of updates made to tuples of relation `geoloc`. Similar to the estimation of $\Pr(x_1)$, when no information is available, the probability that the update is made on a tuple of any particular relation is one over the number of relations in the database.

- *A tuple of* `geoloc` *with its* `?country = "Malta"` *is updated, given that a tuple of* `geoloc` *is updated:*

$$\Pr(x_3|x_2 \wedge x_1) = \frac{t_{u,a3} + 1}{t_{u,geoloc} + G/I_{a3}}$$

where G is the size of relation `geoloc`, I_{a3} is the number of tuples in `geoloc` that satisfy `?country ="Malta"`, and $t_{u,a3}$ is the number of updates made on the tuples in `geoloc` that satisfy `?country ="Malta"`. The number of tuples that satisfy a literal can be retrieved from the database. If this is too expensive for large databases, we can use the estimation approaches used for conventional query optimization (Piatetsky-Shapiro 1984; Ullman 1988) to estimate this number.

- *The value of* `latitude` *is updated, given that a tuple of* `geoloc` *with its* `?country ="Malta"` *is updated:*

$$\Pr(x_4|x_2 \wedge x_1) = \frac{t_{u,geoloc,latitude} + 1}{t_{u,geoloc} + A}$$

where `A` is the number of attributes of `geoloc`, $t_{u,geoloc,latitude}$ is the number of updates made on the `latitude` attribute of the `geoloc` relation. Note that x_4 and x_3 are independent and the condition that `?country ="Malta"` can be ignored. Here we have an example of when domain-specific knowledge can be used in estimation. We can infer that `latitude` is less likely to be updated than other attributes of `geoloc` from our knowledge that it will be updated only if the database has stored incorrect data.

- *The value of* `latitude` *is updated to a value less than 35.89, given that a tuple of* `geoloc` *with its* `?country ="Malta"` *is updated:*

$$\Pr(x_5|x_4 \wedge x_2 \wedge x_1) = \begin{cases} 0.5 & \text{no information available} \\ 0.398 & \text{with range information} \end{cases}$$

$\theta(?x) \Longleftarrow A(\ldots, ?x, \ldots) \wedge \mathcal{L}_1 \wedge \cdots \wedge \mathcal{L}_n.$

Transaction templates:

T1: Update $?x$ of a tuple of A covered by the rule so that the new $?x$ does not satisfy $\theta(?x)$;

T2: Insert a new tuple to a relation involved in the antecedents so that the tuple satisfies all the antecedents but not $\theta(?x)$

T3: Update one tuple of a relation involved in the antecedents not covered by the rule so that the resulting tuple satisfies all the antecedents but not $\theta(?x)$

Table 4: Templates of invalidating transactions for range rules

Without any information, we assume that the attribute will be updated to any value with uniform probability. The information about the distribution of attribute values is useful in estimating how the attribute will be updated. In this case, we know that the latitude is between 0 to 90, and the chance that a new value of latitude is less than 35.89 should be $35.89/90 = 0.398$. This information can be derived from the data or provided by the users.

Assuming that the size of the relation `geoloc` is 616, ten of them with `?country ="Malta"`, without transaction log information, and from the example schema (see Table 1), we have five relations and five attributes for the `geoloc` relation. Therefore,

$$\Pr(\texttt{T1}) = \frac{1}{3} \cdot \frac{1}{5} \cdot \frac{10}{616} \cdot \frac{1}{5} \cdot \frac{1}{2} = 0.000108$$

Similarly, we can estimate $\Pr(\texttt{T2})$ and $\Pr(\texttt{T3})$. Suppose that $\Pr(\texttt{T2}) = 0.000265$ and $\Pr(\texttt{T3}) = 0.00002$, then the robustness of the rule can be estimated as $1 - (0.000108 + 0.000265 + 0.00002) = 0.999606$.

The estimation accuracy of our approach may depend on available information, but even given only database schemas, our approach can still come up with reasonable estimates. This feature is important because not every real-world database system keeps transaction log files, and those that do exist may be at different levels of granularity. It is also difficult to collect domain knowledge and encode it in a database system. Nevertheless, the system must be capable of exploiting as much available information as possible.

Templates for Estimating Robustness

Deriving transactions that invalidate an arbitrary logic statement is not a trivial problem. Fortunately, most knowledge discovery systems have strong restrictions on the syntax of discovered knowledge. Hence, we can manually generalize the invalidating transactions into a small sets of transaction templates, as well as templates of probability estimates for robustness estimation. The templates allow the system to automatically estimate the robustness of knowledge in the procedures of knowledge discovery or maintenance. This subsection briefly describes the derivation of those templates.

Recall that we have defined two classes of rules based on the type of their consequents. If the consequent of a rule is a built-in literal, then the rule is a range rules (e.g., `R1`), otherwise, it is a relational rule with a database literal as its consequent, (e.g., `R2`). In Table 3 there are three transactions that will invalidate `R1`. `T1` covers transactions that update on the attribute value used in the consequent, `T2` covers those that insert a new tuple inconsistent with the rule, and `T3` covers updates on the attribute values used in the antecedents. The invalidating transactions for all range rules are covered by these three general classes of transactions. We generalize them into a set of three transaction templates illustrated in Table 4. For a relational rule such as `R2`, the invalidating transactions are divided into another four general classes different from those for range rules. The complete templates are presented in detail in (Hsu & Knoblock 1996a). These two sets of templates are sufficient for any Horn-clause rules on relational data.

From the transaction templates, we can derive the templates of the equations to compute robustness estimation for each class of rules. The parameters of these equations can be evaluated by accessing database schema or transaction log. Some parameters can be evaluated and saved in advance (e.g., the size of a relation) to improve efficiency. For rules with many antecedents, a general class of transactions may be evaluated into a large number of mutually exclusive transactions whose probabilities need to be estimated separately. In those cases, our estimation templates will be instantiated into a small number of approximate estimates. As a result, the complexity of applying our templates for robustness estimation is always proportional to the length of rules (Hsu & Knoblock 1996a).

Relation	geoloc	seaport	wharf
Size	616	16	18
Updates	0	1	1
Insertions	25	6	1
Deletions	0	2	1

Relation	ship	ship_class	Total
Size	142	25	
Updates	10	1	13
Insertions	22	12	66
Deletions	10	6	19

Table 5: Relation size and transaction log data

Empirical Demonstration

We estimated the robustness of the sample rules on the database as shown in Table 1. This database stores information on a transportation logistic planning domain with twenty relations. Here, we extract a subset of the data with five relations for our experiment. The database schema contains information about relations and attributes in this database, as well as ranges of some attribute values. For instance, the range of `year` of `ship` is from 1900 to 2000. In addition, we also have

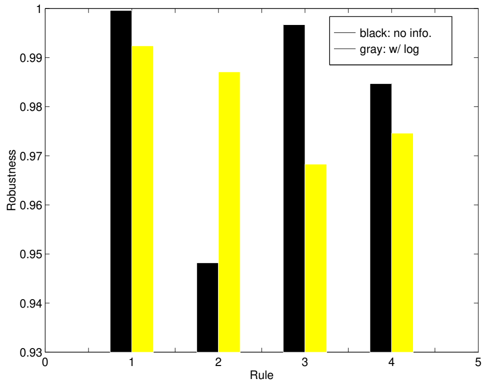

Figure 2: Estimated robustness of sample rules

a log file of data updates, insertions and deletions over this database. The log file contains 98 transactions. The size of relations and the distribution of the transactions on different relations are shown in Table 5.

Among the sample rules in Table 1, `R1` seems to be the most robust because it is about the range of `latitude` which is rarely changed. `R2` is not as robust because it is likely that the data about a geographical location in Malta that is not a seaport may be inserted. `R3` and `R4` are not as robust as `R1`, either. For `R3`, the fleet that a ship belongs does not have any necessary implication to the year the ship was built, while `R4` is specific because seaports with small storage may not be limited to those four geographical locations.

Figure 2 shows the estimation results. We have two sets of results. The first set shown in black columns is the results using only database schema information in estimation. The second set shown in grey columns is the results using both the database schema and the transaction log information. The estimated results match the expected comparative robustness of the sample rules. The absolute robustness value of each rules, though, looks high (more than 0.93). This is because the probabilities of invalidating transactions are low since they are estimated with regard to all possible transactions. We can normalize the absolute values so that they are uniformly distributed between 0 and 1.

The results show that transaction log information is useful in estimation. The robustness of `R2` is estimated lower than other rules without the log information because the system estimated that it is not likely for a country with all its geographical locations as seaports. (See Table 1 for the contents of the rules.) When the log information is considered, the system increases its estimation because the log information shows that transactions on data about Malta are very unlikely. For `R3`, the log information shows that the fleet of ships may change and thus the system estimated its robustness significantly lower than when no log information is considered. A similar scenario appears in the case of `R4`. Lastly, `R1` has a high estimated robustness as expected regardless whether the log information is used.

Applying Robustness in Rule Discovery

This section discusses how to use the robustness estimates to guide the knowledge discovery.

Background and Problem Specification

Although robustness is a desirable property of discovered knowledge, using robustness alone is not enough to guide the knowledge discovery system. The tautologies such as

False ⇒ seaport(_,?glc_cd,_,_,_,_), and
seaport(_,?glc_cd,_,_,_,_) ⇒ True

have a robustness estimate equal to one, but they are not interesting. Therefore, we should use robustness together with other measures of interestingness to guide the discovery. One of the measures of interestingness is applicability, which is important no matter what our application domains are. This section will focus on the problem of discovering rules from closed-world relational data that are both highly applicable and robust. In particular, we will use length to measure the applicability of rules. Generally speaking, a rule is more applicable if it is shorter, that is, if the number of its antecedent literals is smaller, because it is less specific.

Many systems are now able to generate a set of Horn-clause rules from relational data. These systems include inductive logic programming systems (Lavrač & Džeroski 1994; Raedt & Bruynooghe 1993), and systems that discover rules for semantic query optimization (Hsu & Knoblock 1994). Instead of generating desired rules in one run, we propose to use these existing algorithms to generate rules, and then use a rule pruning algorithm to prune a rule so that it is highly robust and applicable (short). The rationale is that rule construction algorithms tend to generate overly-specific rules, but taking the length and robustness of rules into account in rule construction could be too expensive. This is because the search space of rule construction is already huge and evaluating robustness is not trivial. Previous work in classification rule induction (Cohen 1993; 1995; Furnkranz & Widmer 1994) shows that dividing a learning process into a two-stage rule construction and pruning can yield better results in terms of classification accuracy as well as the efficiency of learning. These results may not apply directly to our rule discovery problem, nevertheless, a two-stage system is clearly simpler and more efficient. Another advantage is that the pruning algorithm can be applied on top of existing rule generation systems.

The specification of our rule pruning problem is as follows: take a machine-discovered rule as input, which is consistent with a database but potentially overly-specific, and remove antecedent literals of the rule so that it remains consistent but is short and robust.

The Pruning Algorithm

The basic idea of our algorithm is to search for a subset of antecedent literals to remove until any fur-

R5: ?length $\geq$ 1200 $\Leftarrow$
wharf(_,?code,?depth,?length,?crane) $\wedge$
seaport(?name,?code,_,_,_,_) $\wedge$
geoloc(?name,_,?country,_,_) $\wedge$
?country = "Malta" $\wedge$
?depth $\leq$ 50 $\wedge$
?crane > 0.

r7: ?length $\geq$ 1200 $\Leftarrow$
wharf(_,?code,?depth,?length,?crane) $\wedge$
seaport(?name,?code,_,_,_,_) $\wedge$
geoloc(?name,_,?country,_,_) $\wedge$
?crane > 0.

r10:?length $\geq$ 1200 $\Leftarrow$
wharf(_,?code,?depth,?length,?crane) $\wedge$
seaport(?name,?code,_,_,_,_) $\wedge$
geoloc(?name,_,?country,_,_).

Table 6: Example rule to be pruned and results

ther removal will make the rule inconsistent with the database, or make the rule's robustness very low. We can apply the estimation approach described in the previous section to estimate the robustness of a partially pruned rule and guide the pruning search.

The main difference of our pruning problem from previous work is that there is more than one property of rules that the system is trying to optimize, and these properties — robustness and length — may interact with each other. In some cases, a long rule may be more robust, because a long rule is more specific and covers fewer instances in the database. These instances are less likely to be selected for modification, compared to the case of a short rule, which covers more instances. We address this issue by a beam search algorithm. Let n denote the beam size, our algorithm expands the search by pruning a literal in each search step, preserves the top n robust rules, and repeats the search until any further pruning yields inconsistent rules. The system keeps all generated rules and then selects those with a good combination of length and robustness. The selection criterion may depend on how often the application database changes.

Empirical Demonstration of Rule Pruning

We conducted a detailed empirical study on `R5` in Table 6 using the same database as in the previous sections. Since the search space for this rule is not too large, we ran an exhaustive search for all pruned rules and estimated their robustness. The entire process took less than a second (0.96 seconds). In this experiment, we did not use the log information in the robustness estimation.

The results of the experiment are listed in Table 7. To save the space, we list the pruned rules with their abbreviated antecedents. Each term represents a literal in the conjunctive antecedents. For example, "W" represents the literal `wharf(_,?code,...)`. "Cr" and "Ct" represent the literals on `?crane` and `?country`, respectively. Inconsistent rules and rules with dangling literals are identified and discarded. A set of literals are considered dangling if the variables occurring in

Rule	Antecedents	Robustness	Remarks
R5	W S G Cr D Ct	0.9784990	
r1	W S G D Ct	0.9814620	
r2	W S G Cr Ct	0.9784990	
r3	W S G Cr D	0.9784991	
r4	W S Cr D		Inconsistent
r5	W S G Ct	0.9814620	
r6	W S G D	0.9814620	
r7	W S G Cr	0.9896200	
r8	W D C		Inconsistent
r9	W S Cr		Inconsistent
r10	W S G	0.9814620	
r11	W S D		Inconsistent
r12	W G Cr		Dangling
r13	W G D		Dangling
r14	W Cr		Inconsistent
r15	W S		Inconsistent
r16	W G		Dangling
r17	W D		Inconsistent
r18	W		Inconsistent

Table 7: Result of rule pruning on a sample rule

those literals do not occur in other literals in a rule. Dangling literals are not desirable because they may mislead the search and complicate the robustness estimation.

The relationship between length and robustness of the pruned rules is illustrated in Figure 3. The best rule will be the one located in the upper right corner of the graph, with short length and high robustness. On the top of the graph is the shortest rule `r10`, whose complete specification is shown in Table 6. Although this is the shortest rule, it is not so desirable because it is somewhat too general. The rule states that wharves in seaports will have a length greater than 1200 feet. However, we expect that there will be data on wharves shorter than 1200 feet. Instead, with the robustness estimation, the system can select the most robust rule `r7`, also shown in Table 6. This rule is not as short but still short enough to be widely applicable. Moreover, this rule makes more sense in that if a wharf is equipped with cranes, it is built to load/unload heavy cargo carried by a large ship, and therefore its length must be greater than some certain value. Finally, this pruned rule is more robust and shorter than the original rule. This example shows the utility of the rule pruning with the robustness estimation.

Conclusions

Robustness is an appropriate and practical measure for knowledge discovered from closed-world databases that change frequently over time. An efficient estimation approach for robustness enables effective knowledge discovery and maintenance. This paper has defined robustness as the complement of the probability of rule-invalidating transactions and described an approach to estimating robustness. Based on this estimation approach, we also developed a rule pruning approach to prune a machine-discovered rule into a highly robust and applicable rule.

Robustness estimation can be applied to many AI

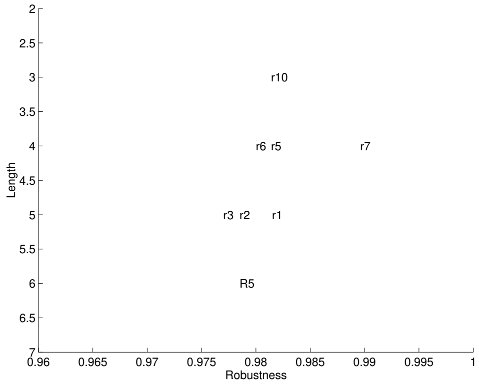

Figure 3: Pruned rules and their estimated robustness

and database applications for information gathering and retrieval from heterogeneous, distributed environment on the Internet. We are currently applying our approach to the problem of learning for semantic query optimization (Hsu & Knoblock 1994; 1996b; Siegel 1988; Shekhar *et al.* 1993). Semantic query optimization (SQO) (King 1981; Hsu & Knoblock 1993; Sun & Yu 1994) optimizes a query by using semantic rules, such as *all Maltese seaports have railroad access*, to reformulate a query into a less expensive but equivalent query. For example, suppose we have a query to *find all Maltese seaports with railroad access and 2,000,000 ft^3 of storage space.* From the rule given above, we can reformulate the query so that there is no need to check the railroad access of seaports, which may reduce execution time. In our previous work, we have developed an SQO optimizer for queries to multidatabases (Hsu & Knoblock 1993; 1996c) and a learning approach for the optimizer (Hsu & Knoblock 1994; 1996b). The optimizer achieves significant savings using learned rules. Though these rules yield good optimization performance, many of them may become invalid after the database changes. To deal with this problem, we use our rule pruning approach to prune learned rules so that they are robust and highly applicable for query optimization.

Acknowledgements We wish to thank the SIMS project members and the graduate students of the Intelligent Systems Divison at USC/ISI for their help on this work. Thanks also to Yolanda Gil, Steve Minton, and the anonymous reviewers for their valuable comments.

References

Cohen, W. W. 1993. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence(IJCAI-93).*

Cohen, W. W. 1995. Fast effective rule induction. In *Machine Learning, Proceedings of the 12th International Conference(ML-95).* San Mateo, CA: Morgan Kaufmann.

Cussens, J. 1993. Bayes and pesudo-bayes estimates of conditional probabilities and their reliability. In *Machine Learning: ECML-93*, 136–152. Berlin, Germany: Springer-Verlag.

Furnkranz, J., and Widmer, G. 1994. Incremental reduced error prunning. In *Machine Learning, Proceedings of the 11th International Conference(ML-94).* San Mateo, CA: Morgan Kaufmann.

Howson, C., and Urbach, P. 1988. *Scientific Reasoning: The Bayesian Approach.* Open Court.

Hsu, C.-N., and Knoblock, C. A. 1993. Reformulating query plans for multidatabase systems. In *Proceedings of the Second International Conference on Information and Knowledge Management(CIKM-93).*

Hsu, C.-N., and Knoblock, C. A. 1994. Rule induction for semantic query optimization. In *Machine Learning, Proceedings of the 11th International Conference(ML-94).* San Mateo, CA: Morgan Kaufmann.

Hsu, C.-N., and Knoblock, C. A. 1996a. Discovering robust knowledge from databases that change. Submitted to *Journal of Data Mining and Knowledge Discovery.*

Hsu, C.-N., and Knoblock, C. A. 1996b. Using inductive learning to generate rules for semantic query optimization. In Fayyad, U. M. et al., eds., *Advances in Knowledge Discovery and Data Mining.* AAAI Press/MIT Press. chapter 17.

Hsu, C.-N., and Knoblock, C. A. 1996c. Semantic Optimization for Multidatabase Retrieval, Forthcoming.

King, J. J. 1981. *Query Optimization by Semantic Reasoning.* Ph.D. Dissertation, Stanford University, Department of Computer Science.

Lavrač, N., and Džeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood.

Lloyd, J. W. 1987. *Foundations of Logic Programming.* Berlin, Germany: Springer-Verlag.

Piatetsky-Shapiro, G. 1984. *A Self-Organizing Database System – A Different Approach To Query Optimization.* Ph.D. Dissertation, Department of Computer Science, New York University.

Raedt, L. D., and Bruynooghe, M. 1993. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence(IJCAI-93).*

Ramsay, A. 1988. *Formal Methods in Artificial Intelligence.* Cambridge, U.K.: Cambridge University Press.

Shekhar, S.; Hamidzadeh, B.; Kohli, A.; and Coyle, M. 1993. Learning transformation rules for semantic query optimization: A data-driven approach. *IEEE Transactions on Knowledge and Data Engineering* 5(6):950–964.

Siegel, M. D. 1988. Automatic rule derivation for semantic query optimization. In Kerschberg, L., ed., *Proceedings of the Second International Conference on Expert Database Systems.* Fairfax, VA: George Mason Foundation.

Sun, W., and Yu, C. T. 1994. Semantic query optimization for tree and chain queries. *IEEE Trans. Knowledge and Data Engineering* 6(1):136–151.

Ullman, J. D. 1988. *Principles of Database and Knowledge-base Systems*, volume II. Palo Alto, CA: Computer Science Press.