To appear in the *Proceedings of the First International Conference on Knowledge Discovery and Data Mining, Montreal, Canada, 1995.*

Estimating the Robustness of Discovered Knowledge*

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

Abstract

This paper introduces a new measurement, *robustness*, to measure the quality of machine-discovered knowledge from real-world databases that change over time. A piece of knowledge is robust if it is unlikely to become inconsistent with new database states. Robustness is different from predictive accuracy in that by the latter, the system considers only the consistency of a rule with unseen data, while by the former, the consistency after deletions and updates of existing data is also considered. Combining robustness with other utility measurements, a system can make intelligent decisions in learning and maintenance of knowledge learned from changing databases. This paper defines robustness, then presents an estimation approach for the robustness of Horn-clause rules learned from a relational database. The estimation approach applies the Laplace law of succession, which can be efficiently computed. The estimation is based on database schemas and transaction logs. No domain-specific information is required. However, if it is available, the approach can exploit it.

Introduction

Databases are evolving entities. Knowledge discovered from one database state may become invalid or inconsistent with a new database state. It is important to know whether a piece of knowledge is *robust* against database changes. *Predictive accuracy*, commonly used in inductive learning and knowledge discovery as a measurement, is not appropriate for databases that change. Although data tuples look alike in a database and in an example set of inductive learning applications, they should be treated differently. In inductive learning, a tuple represents a static state, an *example* of experience, whereas in a database, a tuple represents a dynamic state of the world that changes over time. Moreover, data may be deleted or updated in a database, but predictive accuracy measures only the probability that learned knowledge is consistent with new data.

In a changing environment like a database, estimating robustness of learned knowledge enables a system to make intelligent decisions in order to learn and maintain knowledge economically. Consider a tour guide who leads groups of tourists. From a particular group of tourists, the tour guide might learn that the age of these tourists is more than 55. He might also learn that tourists from Quebec speak fluent French. Obviously, knowledge about age is not robust, because it changes depending on particular groups, but knowledge about language usage is robust. The tour guide may try to remember the knowledge about language usage but forget the knowledge about age right after he sends the group home. However, the knowledge about age may still be useful for the guide to decide how often he should let his tourists take a break.

Estimating the robustness of learned knowledge is especially useful for applications that learn from databases so as to improve their performance. An example of those applications is learning for semantic query optimization (Siegel 1988; Hsu & Knoblock 1994; 1995). Semantic query optimization (SQO) (King 1981; Hsu & Knoblock 1993b) 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 (Hsu & Knoblock 1993a; 1994; 1995), we have developed a learning approach that achieves an optimization saving above 43 percent 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,

*The research reported here 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. F30602-91-C-0081. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of NSF, DARPA, RL, the U.S. Government, or any person or agency connected with them.

Schema:

```
geoloc(name,glc_cd,country,latitude,longitude),
seaport(name,glc_cd,storage,silo,crane,rail).
```

Rules:

R1: ?latitude $\geq$ 35.89 $\Leftarrow$ geoloc(_,_,"Malta",?latitude,_).
R2: seaport(_,?glc_cd,_,_,_,_) $\Leftarrow$ geoloc(_,?glc_cd,"Malta",_,_).
R3: ?storage > 2000000 $\Leftarrow$ seaport(_,?glc_cd,?storage,_,_,_) $\wedge$ geoloc(_,?glc_cd,"Malta",_,_).

Table 1: Schema of a geographic database and semantic rules

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

Table 2: A database fragment

the learning system can estimate the robustness of candidate rules and learn those rules with high estimated robustness. When the database is changed, a maintenance system can be used to update the robustness and delete those rules with low robustness. Meanwhile, the learning system can keep learning new rules from new database states. This way, the system can autonomously maintain a set of effective and consistent rules for optimization.

In this paper, we first establish the basic terminology on databases and rules. Next, we present both informal and formal definitions of robustness. Following that, we review the Laplace law of succession and then describe how it is used to estimate the robustness of a Horn-clause rule. Finally, we conclude with a summary of the contributions and directions for future work.

Databases and Rules

This section briefly introduces the basic database and knowledge discovery terminology that will be used throughout this paper. We are particularly interested in estimating the robustness of Horn-clause rules derived from a relational database, because Horn-clause rules and relational databases are widely used in practice. In this paper, a *database* consists of a set of relations. A *relation* is then a set of instances. Each *instance* is a vector of attribute values. 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 two relations and their attributes. In this database, the relation `geoloc` stores data about geographic locations, and the attribute `glc_cd` is a geographic location code.

Knowledge is expressed in Horn-clause rules in this paper. Table 1 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). In Table 1, rule `R1` states that the latitude of a Maltese geographic location is greater than or equal to 35.89. `R2` states that all Maltese geographic locations are seaports. `R3` states that all Maltese seaports have a storage capacity greater than 2,000,000 ft^3.

A *database state* at a given time `t` is the collection of the instances presented in the database at the time `t`. We use the *closed-world assumption* (CWA) to interpret the semantics of a database state. That is, information not explicitly presented 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 antecedent of the rule also satisfy the consequent of the rule. A straightforward approach to identifying an inconsistent rule is to transform a rule of the form $C \leftarrow A$ into a query $\neg C \wedge A$. If the query returns an answer that is not empty, then the rule is inconsistent. 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 tuple in `seaport` with a common `glc_cd` value.

A database can be changed by *transactions*. 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. A transaction can be considered as a mapping from a database state to a new database state.

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 – intuitive definition) *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 $Robust(r) = \Pr(D)$.*

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,

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

There are two problems with this estimation. 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. An alternative definition can be derived from the observation that the likelihood of database states is determined by a current database state and the probability of certain transactions on that state. A rule become inconsistent when a transaction that results in a new state inconsistent with the rule is performed. The robustness of a rule can then be defined as the complement of the probability that such a transaction is performed. In other words, for a given rule and a current database state, if the transactions that will invalidate the rule are unlikely to be performed, then the rule is robust. We present this reformulated definition as follows:

Definition 2 (Robustness) *Given a database state d and a rule r that is consistent with d, let t denote the transactions on d that result in new database states inconsistent with r. The robustness of r in the database state d is $Robust(r|d) = 1 - \Pr(t|d)$.*

This definition retains our intuitive notion of robustness, but allows us to estimate robustness without estimating the probability of possible database states. The main difference is in that the intuitive definition defines the robustness regardless of how databases are changed. In Definition 2, the robustness of a rule is different in different database states. Since databases are changed over time, the robustness of a rule should change accordingly. Definition 2 captures this idea.

Laplace Law of Succession

This section introduces two useful estimates for the probability of the outcomes of a repeatable random experiment. They will be used to estimate the robustness of rules.

Theorem 1 (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 $\frac{r+1}{n+k}$.*

Detailed description and a proof of the Laplace law of succession can be found in (Howson & Urbach 1988). The Laplace law applies to any *repeatable* experiments that can be performed as many times as required. An example of a repeatable experiment is tossing a coin. The Laplace law is a special case of a modified estimate called *m-Probability* (Cestnik & Bratko 1991). A prior probability of outcomes can be brought to bear in this more general estimate.

Theorem 2 (m-Probability) *Let r, n, and C be as in Theorem 1. Suppose $\Pr(C)$ is known as the prior probability that the experiment has an outcome C, and m is an adjusting constant that indicates our confidence in the prior probability $\Pr(C)$. The probability that the outcome of the next experiment will be C can be estimated as $\frac{r + m \cdot \Pr(C)}{n+m}$.*

The idea of *m-Probability* can be understood as a weighted average of known relative frequency and prior probability:

$$\frac{r + m \cdot \Pr(C)}{n+m} = (\frac{n}{n+m}) \cdot (\frac{r}{n}) + (\frac{m}{n+m}) \cdot \Pr(C)$$

where n and m are the weights. The Laplace law is a special case of the m-probability estimate with $\Pr(C) = 1/k$, and $m = k$. The prior probability used here is that k outcomes are equally probable. The m-probability estimate has been used in many machine learning systems for different purposes. Convincing results in handling noisy data and pruning decision trees have been achieved (Cestnik & Bratko 1991; Lavrač & Džeroski 1994). We will use these theorems to estimate the probability of transactions and the robustness of rules.

Estimating Robustness

Our problem 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 transactions that may invalidate a rule and estimating the probability of those transactions. This section illustrates our approach with an example.

Consider `R1` in Table 3 as an example. Transactions that map the current database state to a state that satisfies the negation of `R1` will invalidate `R1`. The negation of `R1` is:

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

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 not equal to "Malta" is updated such that its country = "Malta".

Table 3: Transactions that invalidate R1

$\exists$?latitude: ?latitude < 35.89 $\wedge$ geoloc(_,_,"Malta",?latitude,_).

Note that variables in a Horn-clause rule are universally quantified. Table 3 lists three transactions that will invalidate R1 because in the database states yielded by them, there exists a tuple of geoloc that satisfies the negation of R1. Since T1, T2, and T3 are mutually exclusive, we have Pr(T1 $\vee$ T2 $\vee$ T3) = Pr(T1) + Pr(T2) + Pr(T3). The probability of these transactions, and thus the robustness of R1, can be estimated from the probabilities of T1, T2, and T3.

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. We first decompose the transaction T1 into a conjunction of more primitive statements such that constraints of different degrees of detail are excerpted:

- a_1: a tuple is updated.
- a_2: a tuple of geoloc is updated.
- a_3: a tuple of geoloc with its country = "Malta" is updated.
- a_4: a tuple of geoloc whose latitude is updated.
- a_5: a tuple of geoloc whose latitude is updated to a value less than 35.89.

These statements specify constraints, respectively, of the type of the transaction, which relation, which tuples, which attributes, and how the update is performed. From the probability theory, we have

$$\begin{aligned}\Pr(\texttt{T1}) &= \Pr(a_1 \wedge a_2 \wedge a_3 \wedge a_4 \wedge a_5) \\ &= \Pr(a_1) \cdot \Pr(a_2|a_1) \cdot \Pr(a_3|a_2 \wedge a_1) \cdot \\ &\quad \Pr(a_4|a_3 \wedge a_2 \wedge a_1) \cdot \Pr(a_5|a_4 \wedge a_3 \wedge a_2 \wedge a_1)\end{aligned}$$

We estimate each conditional probability using the Laplace law or the m-probability theorem. They are applicable because transactions of a database are random repeatable events. Since we decompose a complex transaction into conditional events, information such as database schema, transaction logs, and domain knowledge such as expected size of relations, expected distribution, range of attribute values, etc., can be easily included in the estimation. When no information is available, we use the principle of indifference and treat all possibilities as equally probable. We now describe how these conditional probabilities can be estimated.

- *A tuple is updated:*

$$\Pr(a_1) = \begin{cases} \frac{1}{3} & \text{no information available} \\ \frac{t_u+1}{t+3} & \text{transaction log available} \end{cases}$$

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. 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:*

$$\Pr(a_2|a_1) = \begin{cases} \frac{1}{R} & \text{no information available} \\ \frac{t_{u,geoloc}+1}{t_u+R} & \text{transaction log available} \end{cases}$$

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 $\Pr(a_1)$, 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:*

$$\Pr(a_3|a_2 \wedge a_1) = \begin{cases} \frac{I_{a3}}{G} & \text{no information available} \\ \frac{t_{u,a3}+1}{t_{u,geoloc}+R} & \text{transaction log available} \end{cases}$$

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 (Ullman 1988; Piatetsky-Shapiro 1984) to estimate this number.

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

$$\begin{aligned}\Pr(a_4|a_3 \wedge a_2 \wedge a_1) &= \Pr(a_4|a_2 \wedge a_1) \\ &= \begin{cases} \frac{1}{A} & \text{no information available} \\ \frac{t_{u,geoloc,latitude}+1}{t_{u,geoloc}+A} & \text{transaction log available} \end{cases}\end{aligned}$$

R2: seaport(_,?glc_cd,_,_,_,_) ⇐ geoloc(_,?glc_cd,"Malta",_,_).

T1: One of the existing tuples of geoloc with its country = "Malta" is updated such that its glc_cd value is not equal to any glc_cd values of seaport tuples.
T2: A set of tuples of seaport sharing their glc_cd values is updated such that their glc_cd values are not equal to any glc_cd values of geoloc tuples with their country = "Malta".
T3: A new tuple of geoloc with its country = "Malta" and glc_cd not equal to any glc_cd values of seaport tuples is inserted into the database.
T4: A set of tuples of seaport sharing their glc_cd values, which are equal to any glc_cd values of geoloc tuples with their country = "Malta", is deleted.

Table 4: Transactions that invalidate R2

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 a_4 and a_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:*

$$\Pr(a_5|a_4 \wedge a_3 \wedge a_2 \wedge a_1) = \begin{cases} 0.5 & \text{no information available} \\ 0.398 & \text{with range information} \end{cases}$$

Without any information, we assume that the attribute will be updated to any value with uniform chances. 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 relation geoloc is 80, four of them with country = "Malta", without any transaction log information, and from the example schema and the database state (see Table 1 and Table 2), we have 2 relations in the database, 5 attributes for geoloc relation. Therefore,

$$\Pr(\texttt{T1}) = \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{4}{80} \cdot \frac{1}{5} \cdot \frac{1}{2} = 0.008$$

Similarly, we can estimate Pr(T2) and Pr(T3). Suppose that Pr(T2) = 0.241 and Pr(T3) = 0.001, then the robustness of the rule can be estimated as $1-(0.008+0.241+0.001) = 0.75$.

The estimation accuracy of our approach may depend on available information, but even given only database schemas, our approach can still come up with a reasonable estimation. 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.

Implementation

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. Therefore, we can manually derive a set of templates of transactions for different classes of knowledge specification. In our case of Horn-clause rules, there are two classes that need different templates:

1. A rule with a built-in literal as its consequent (e.g., R1). Templates of transactions that invalidate these rules can be generalized from those for R1 shown in Table 3.
2. A rule with its consequent a database literal (e.g., R2). Templates of transactions that invalidate these rules can be generalized from those for R2 shown in Table 4.

To estimate robustness efficiently, each transaction in a template must be minimal in the sense that no redundant conditions are specified. For R1, a transaction 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 redundant; thus the transaction is not minimal. Also, transactions should be mutually exclusive so that no transaction covers another. For any two transactions t_a and t_b, if t_b covers t_a, then $\Pr(t_a \vee t_b) = \Pr(t_a)$ and it is redundant to list t_b for probability estimation. Again, for R1, a transaction that deletes all geoloc tuples and then inserts tuples invalidating R1 does not need to be considered, because it covers T2 in Table 3.

Furthermore, we can derive the templates of the equations to compute robustness estimation for each type of rules. Then the system can estimate the robustness of rules by retrieving necessary parameters

(e.g., the size of a relation) and applying the equations directly. The system can even link parameters with rules. When the database is changed, the system can update the robustness of rules by increasing those parameters being affected and recompute the probability. This way, the estimated robustness is able to evolve with databases and thus supports rule maintenance systems in decision making.

Discussion

Robustness is an appropriate and practical measurement for knowledge discovered from databases that are changed frequently. An efficient estimation approach for robustness enables effective learning and knowledge maintenance. This paper has defined robustness as the complement of the probability of rule-invalidating transactions, and described an estimating approach.

Robustness in its more general sense can be used to guide the learning of *drifting concepts* from dynamic environments (Helmbold & Long 1994; Widmer & Kubat 1993). So far, approaches to learning drifting concepts assume a world that changes gradually, and focus on incremental modification of learned rules. Learning robust rules may increase their tolerance to the world changes and reduce the need of modification effort.

We are currently working on applying our approach to learning for semantic query optimization, as described earlier in this paper. The approach can also be applied to other database applications, such as view management, intelligent database caching (Arens & Knoblock 1994), and learning for the integration of heterogeneous multidatabases (Ambite & Knoblock 1995). These applications require the system to extract a compressed description (e.g., a view definition) of data, and the consistency of the description with the database is important. Robustness can guide the system to extract robust descriptions so that they can be used with minimal maintenance effort. Our future work includes empirical comparisons of robustness and other measurements of learned knowledge. Another direction for future work is to extend the definition and estimation approach of robustness to probabilistic rules.

Acknowledgments

We wish to thank the SIMS project members, Yigal Arens, Wei-Min Shen, Chin Y. Chee, José-Luis Ambite, and Sheila Tejada, for their help on this work. Thanks also to Shiela Coyazo and the anonymous reviewers for their valuable comments.

References

Ambite, J.-L., and Knoblock, C. A. 1995. Reconciling distributed information sources. In *Working Notes of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments.*

Arens, Y., and Knoblock, C. A. 1994. Intelligent caching: Selecting, representing, and reusing data in an information server. In *Proceedings of the Third International Conference on Information and Knowledge Management.*

Cestnik, B., and Bratko, I. 1991. On estimating probabilities in tree pruning. In *Machine Learning – EWSL-91, European Working Session on Learning.* Berlin, Germany: Springer-Verlag. 138–150.

Helmbold, D. P., and Long, P. M. 1994. Tracking drifting concepts by minimizing disagreement. *Machine Learning* 14:27–45.

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

Hsu, C.-N., and Knoblock, C. A. 1993a. Learning database abstractions for query reformulation. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases.*

Hsu, C.-N., and Knoblock, C. A. 1993b. Reformulating query plans for multidatabase systems. In *Proceedings of the Second International Conference on Information and Knowledge Management.* Washington, D.C.: ACM.

Hsu, C.-N., and Knoblock, C. A. 1994. Rule induction for semantic query optimization. In *Proceedings of the Eleventh International Conference on Machine Learning.*

Hsu, C.-N., and Knoblock, C. A. 1995. Using inductive learning to generate rules for semantic query optimization. In Piatetsky-Shapiro, G., and Fayyad, U., eds., *Advances in Knowledge Discovery and Data Mining.* MIT Press. Chapter 17.

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: 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.

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. 371–385.

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

Widmer, G., and Kubat, M. 1993. Effective learning in dynamic environments by explicit context tracking. In *Machine Learning: ECML-93.* Berlin: Springer-Verlag.