Knowledge Organization for Exploration

Amihai Motro and Sylvie Goullioud

Department of Information and Software Systems Engineering
George Mason University
Fairfax, VA 22030-4444

Abstract. To support applications, such as efficient browsing in large knowledge bases and cooperative knowledge discovery in large databases, the concept of rule similarity is essential. In this paper we define such a measure, called *distance*, and then put in the service of various knowledge exploration processes. Rule distances allow us to place all the available knowledge on a "map", in which proximity reflects similarity. Users can then *browse* in the knowledge, by iteratively visiting rules and examining their neighborhoods. In our cooperative knowledge *discovery* process, users suggest a hypothetical rule that has been observed. If the hypothesis is verified, the system may suggest other, even better, rules that hold; if refuted, the system attempts to focus the hypothesis until it holds; in either case, the system may offer other rules that hold and are close to the hypothesis. Our work is within the formal framework of logic databases, and we report on an experimental system that implements our approach.

1 Introduction

Much database research today is concerned with systems that incorporate *knowledge*. The prevailing approach is to model each knowledge item with a *rule*. The repository of rules is often unorganized. If organization is imposed, it is usually a tool for *indexing* the rules, whose purpose is to locate rules quickly in processes such as inference or integrity verification. Such organization, however, is unsuitable when different knowledge items need to be related according to their *semantic similarity*. One possible approach, which we advocate in this paper, is to define a *distance* between every two knowledge items. Such distance measures how "similar" or "different" these items are. It is then possible to consider questions, such as "what are the knowledge items similar to a given item?".

An application made possible by the availability of distance is knowledge *browsing*. User interfaces may be developed that will permit users to examine an item of the knowledge, then proceed to examine items that are in its *neighborhood*. Selecting an item from this neighborhood, users may proceed to examine items in this item's neighborhood, and so on. Another application is a cooperative system for *discovering* new knowledge in databases. In this system, a user

This work was supported in part by NSF Grant No. IRI-9007106 and by an ARPA grant, administered by ONR under Grant No. N0014-92-J-4038.

begins by suggesting a hypothetical rule that has been observed. If the hypothesis is verified, the system may suggest other, even "better", rules that hold; if refuted, the system attempts to focus the hypothesis until it holds; in either case, the system may offer other rules that hold and are "close" to the hypothesis.

An important concept used throughout this paper is *subsumption*. Intuitively, one rule subsumes another rule, if the latter is a restricted version of the former. If both rules hold, then the specialized rule is redundant. Another useful concept is rule *complexity*, which is a measure that expresses preference (low scores) for rules that are simple. Subsumption and complexity are used to define rule *distance*. Intuitively, the distance between two rules is the number of "elementary transformations" necessary to transform one rule into the other. Subsumption, complexity, and distance are defined and discussed in Section 2.

Section 3 is devoted to knowledge browsing and interactive knowledge discovery, two applications made possible by the concepts developed in Section 2. We then discuss the commonalities of these two processes and show how they can be integrated in a single process of *knowledge exploration*. Section 4 concludes this paper with a discussion of an experimental system that implements our approach, and of several problems that are still open to research.

To place our effort in the context of other research, we consider briefly here three subjects: information distance (the subject of this work), and information browsing and knowledge discovery (two applications of our work).

The concept of information distance has been researched in information retrieval systems [15] and in database systems [8]. In both types of systems, the motivation for introducing this concept has been to enable *approximate* matching of information to requests. With the help of a distance measure, the stored information can be modeled by a metricized space. As retrieval requests describe hypothetical information items, they too may be placed in this space. A retrieval request can then be satisfied by the item closest to it, or by the items in its immediate neighborhood. The information stored and manipulated by these systems is essentially *data*, and the concept of distance they define is strictly limited to data items. The authors are unaware of any effort to extend this concept to knowledge-rich databases (i.e., logic databases), that store and manipulate both data and knowledge.

Browsers are user interfaces to information systems that are intended for exploratory searches, often by naive users. Thus, they usually employ simple conceptual models and offer simple intuitive commands. Ideally, browsing should not require familiarity with the particular database being accessed, or even preconceived retrieval targets. As their conceptual model, most browsers assume some kind of network structure that connects the stored information [7, 3, 1, 6]. In this network, *adjacency* reflects *relationship*; e.g., the objects *John* and *Manufacturing* would be linked because John works in the Manufacturing department. A different approach is taken in [14] (and to some degree in [7]), where objects are placed in a space, so that their mutual proximity reflects their *affinity* (as computed by some measure). Users can then navigate among the objects with the help of a two dimensional "map" that visualizes the distances among objects.

A common organization for knowledge bases is a network, in which two rules are connected if the consequent of one matches the premise of the other; i.e., adjacency reflects *causality*. This organization supports efficient look-up of rules during the inference process, and with a simple browsing interface, users can also explore the various possible chains of rule applications [17]. Yet, this organization cannot help users to search for rules that are similar to a particular rule. The browsing process defined in this paper uses the distance measure in a way reminiscent of an affinity browser.

The topic of knowledge discovery in databases has received much attention lately [13, 12]. In general, it is concerned with methods for discovering abstract behavior patterns in a given extension of a database. Two essential issues of knowledge discovery are (1) what is a good rule (e.g., neither incidental nor trivial), and (2) how to enhance the efficiency of the discovery process through interaction (as opposed to just combing the database for all possible rules). Our work pertains to both issues. With respect to the first issue, various methods have been suggested to measure goodness. Usually, rules are allowed to be "almost true", and the issue of goodness focuses on whether a given rule is statistically significant [10, 16]. In this work, we consider only rules that are "always true", and our measures focus on other criteria; i.e., generality and simplicity. With respect to the second issue, several interactive systems have been developed for knowledge discovery. Such systems use elaborate interactions to assist their users in focusing their domain of interest [11, 5]. However, the authors are unaware of an approach in which users propose hypothetical rules (these rules could reflect actual observations that are suspected to be true, or they could be viewed as guidance to the system as to the kind of knowledge being sought), and the system attempts to refine these initial approximations into "better" or "similar" rules.

2 The Model

2.1 Basics

We review briefly several basic definitions of logic databases [2]. A *predicate* is a Boolean-valued function, and an *argument* is either a variable or a constant. A *literal* is a predicate symbol, possibly negated, with a list of arguments. A *Horn clause* is a disjunction of literals with at most one positive literal. It follows that a Horn clause has one of these three forms:

1. $q \longleftarrow p_1 \wedge \cdots \wedge p_n \ (n \geq 1)$
2. $\longleftarrow p_1 \wedge \cdots \wedge p_n \ (n \geq 1)$
3. $q \longleftarrow$

where q and each p_i are positive literals, and all variables are quantified universally. q is the *head* of the clause, and $p_1 \wedge \cdots \wedge p_n$ is its *body*. We require that all variables in the head appear in the body (*safety*). It follows that clauses of the third form have no variables. Such clauses are called *facts*. The other two forms are referred to as *rules*. We also require that each rule is connected; i.e., its literals cannot be split into disjoint sets which do not share any arguments.

Examples of facts are $french(pierre)$ and $owner(pierre, renault)$, which state that Pierre is French and owns a car made by Renault. An example of a rule of the first form is $owner(X, renault) \longleftarrow french(X)$, which states that all French citizens own a car made by Renault. An example of a rule of the second form is $\longleftarrow french(X) \wedge italian(X)$, which states that a person cannot be a citizen of both France and Italy.

A rule is *consistent* with a set of facts, if every substitution of its variables that satisfies its body (i.e., the substituted body literals are facts) also satisfies its head (the substituted head literal is a fact). For example, the rule $owner(X, renault) \longleftarrow french(X)$ is consistent with a set of facts, if for all constants a, if $french(a)$ is a fact, then $owner(a, renault)$ is also a fact. A set of rules is *(mutually) consistent*, if there exists at least one set of facts in which every rule is consistent.

A *database* is a set of Horn clauses (rules and facts), whose rules are safe, connected, consistent with the facts, and mutually consistent. The rules correspond to *integrity constraints*, and are referred to as the *knowledge*. The facts correspond to a relational *database instance*, and are referred to as the *data*.

Our model is similar to Datalog, except that in Datalog the clause $q \longleftarrow p_1 \wedge \cdots \wedge p_n$ defines the predicate q (i.e., it *generates* facts), while here such a clause imposes a constraint on the facts of the predicates $p_1, \ldots, p_n$ and q.[1]

The work discussed here is concerned with rules that are either *declared* or *discovered*. A declared rule is part of the database definition, and is used by the database system to assure that various real-world relationships are satisfied by the stored facts. A discovered rule is a relationship that is observed in the stored facts. Declared rules are guaranteed to be consistent in *any* database extension. Discovered rules are guaranteed to be consistent only in the present extension.

2.2 Rule Subsumption

Some rules are obviously more informative than others. Consider, for example, the rules $driver(X) \longleftarrow owner(X, renault)$ (every owner of a car made by Renault is a driver) and $driver(X) \longleftarrow owner(X, Y)$ (every car owner is a driver). Clearly, if both rules are consistent, then the latter is preferred, because it is more general and more informative; indeed, the former rule is redundant. The latter rule is said to *subsume* the former rule [2]. Rule R_1 subsumes rule R_2, if there is a substitution of its variables by variables and constants that transforms it into a subclause of R_2. For example, consider the rules

$$R_1 : owner(X, renault) \longleftarrow french(X) \wedge student(X, Y)$$
$$R_2 : owner(Y, renault) \longleftarrow french(Y) \wedge student(Y, sorbonne) \wedge female(Y)$$

R_1 subsumes R_2 with the substitution $\{X/Y, Y/sorbonne\}$. As another example, consider the rules

$$R_1 : \longleftarrow young(X) \wedge owner(X, Y)$$
$$R_2 : driver(Y) \longleftarrow french(Y) \wedge young(Y) \wedge owner(Y, renault)$$

[1] We use the term *rule* in the sense of a law that is maintained throughout the system, not in the sense of a rule of inference.

R_1 subsumes R_2 with the substitution $\{X/Y, Y/renault\}$. Note that R_1 is consistent only when there are no young car owners. In this case, there would be no young French owners of cars made by Renault, either. Hence, R_2 is consistent because its premise is always false.

When one rule subsumes another, the former rule could be considerably more general than the latter, or it could be only slightly more general. A rule R_1 is an *immediate generalization* of a rule R_2 (R_2 is an *immediate specialization* of R_1) if R_1 differs from R_2 either by a substitution of a single variable by a constant, or by a substitution of single variable by an existing variable, or by a single unconstrained literal.[2] As an example, consider these four rules:

$$R_1 : owner(X, renault) \longleftarrow french(X)$$
$$R_2 : owner(X, renault) \longleftarrow french(X) \wedge student(Y, Z)$$
$$R_3 : owner(X, renault) \longleftarrow french(X) \wedge student(X, Z)$$
$$R_4 : owner(X, renault) \longleftarrow french(X) \wedge student(X, sorbonne)$$

R_3 immediately subsumes R_4 with the substitution $\{Z/sorbonne\}$, R_2 immediately subsumes R_3 with the substitution $\{Y/X\}$, and R_1 immediately subsumes R_2 by not having the unconstrained literal $student(Y, Z)$.

Immediate subsumption can also be approached by constraints. A *constraint* in a rule is either (1) a constant argument, (2) two identical variable arguments, or (3) an unconstrained literal. A rule R_1 is an immediate generalization of a rule R_2, if R_2 is obtained from R_1 by *adding a single constraint*. In the above example, the constraints added to R_1 are, in this order, $student(Y, Z)$, $X = Y$, and $Z = sorbonne$.

When gradually generalizing a rule that holds, one eventually reaches a critical point of a *last* rule that holds. Given a rule R_1 that holds, a rule R_2 is its *maximal consistent generalization (MCG)*, if R_2 holds, R_2 subsumes R_1, and any rule that subsumes R_2 and is not a variant of R_2 (a rule obtained from R_2 by renaming its variables) fails. Similarly, when specializing a rule that holds, one eventually reaches a critical point of a *first* rule that holds. Given a rule R_1 that fails, a rule R_2 is its *minimal consistent specialization (MCS)*, if R_2 holds, R_1 subsumes R_2, and any rule that subsumes R_2 and is not a variant of R_2 fails.

2.3 Rule Complexity and Rule Distance

Subsumption suggests that one rule may be preferred over another. But because it is only a partial order on rules, it cannot be used to pick the "best" rule from an arbitrary set of rules. We now define a measure that assigns every rule a numeric score, thus allowing rules to be compared on the basis of their scores.

All other considerations being equal, simpler rules are usually preferred over more complex rules. Our definition of rule simplicity considers its total number of constraints. To each literal we assign a *complexity factor*, that reflects the number of constraints it represents, and is computed as follows.

[2] A literal is unconstrained, if all its arguments are variables, that are different from each other and none appears elsewhere in the rule. Such a literal imposes a constraint that the corresponding predicate must have at least one fact associated with it.

1. Every constant argument contributes 1 to the factor.
2. Every occurrence of a variable argument contributes $(n-1)/n$ to the factor, where n is the total number of occurrences of this variable in the rule.[3]
3. The literal itself contributes 1 to the factor.

As an example, consider the literal $p(a, b, X, X, Y, Z)$, and assume that X occurs a total of four times in the rule, Y occurs twice, and Z occurs only once (this occurrence). The complexity factor of this literal is 5.

The *complexity* of a rule R, denoted $|R|$, is the sum of the complexity factors of its literals. Clearly, more general rules are have lower complexity: if R' subsumes R, then $|R'| < |R|$. Below are some example rules and their complexity.

$$\begin{array}{rll} & \longleftarrow & 0 \\ & \longleftarrow young(X) & 1 \\ & \longleftarrow young(X) \wedge owner(X, Y) & 3 \\ owner(X, renault) & \longleftarrow french(X) \wedge young(X) \wedge student(X, sorbonne) & 9 \end{array}$$

Intuitively, each rule may be regarded as if it is constructed from the null rule by adding constraints, while observing the required syntactical structure. The complexity of the final rule is the total number of constraints thus added to the null rule (whose complexity is 0).

Assume that R_1 subsumes R_2. The difference in their complexities provides us with a measure of *how much* R_1 generalizes R_2. For example, consider the following three rules and their complexities.

$$\begin{array}{lll} owner(X, renault) & \longleftarrow french(X) & 4 \\ owner(X, renault) & \longleftarrow french(X) \wedge student(X, sorbonne) & 7 \\ owner(X, renault) & \longleftarrow french(X) \wedge young(X) \wedge married(X, Y) \wedge young(Y) & 10 \end{array}$$

The first rule subsumes both the second and third rules. However, it generalizes the third to a much larger degree than it generalizes the second (3 vs. 6).

The definition of subsumption implies that every rule is subsumed by the null rule $\leftarrow$. Hence, every two rules are guaranteed a common generalization. Moreover, every two rules R_1 and R_2 have a *least common generalization (LCG)*, denoted $R_1 \wedge R_2$.[4] Finally, The *distance* between two rules R_1 and R_2 is defined as the sum of the complexity differences between each of the rules and their LCG: $d(R_1, R_2) = |R_1| + |R_2| - 2|R_1 \wedge R_2|$. In particular, if R_1 subsumes R_2, then the distance between them is simply $|R_2| - |R_1|$. The complexity of a rule R is simply its distance from the null rule: $|R| = d(R, \leftarrow)$

Intuitively, one could think of $R_1 \wedge R_2$ as being *transformed* into R_1 or R_2 by the gradual *addition* of constraints. Conversely, one could think of R_1 or

[3] The occurrence of a variable n times in a rule represents a total of $n-1$ constraints. Thus, each occurrence contributes $(n-1)/n$ constraints.

[4] Note that a common specialization is not guaranteed, because to derive a common specialization requires adding new constraints to each of the rules, until they converge. This, may require having two literals in the head of the common specialization, creating a rule which is not a Horn clause.

R_2 as being transformed into $R_1 \wedge R_2$ by the gradual *removal* of constraints. Altogether, one rule is transformed into the other rule, first by the gradual removal of constraints, and then by the gradual addition of constraints. These transformations are reminiscent of the generalizations of queries in cooperative answering systems [9, 4]. Obviously, by transforming one rule into the LCG, and then transforming the LCG into the other rule, the *shortest* chain of transformations from one rule into the other is realized. The *length* of this chain is the distance between the two rules.

With the three previous rules, it can be shown that the first rule is the LCG of the other two rules. Hence, the distance between the second and the third rule is $10 + 7 - 2 \cdot 4 = 9$. The third rule is generalized in six transformations to the first rule, and is then specialized in three transformations to the second rule.

3 Applications

We now apply this model in two knowledge browsing and knowledge discovery processes. We then discuss the commonalities of these two processes and show how they can be integrated in a single process of *knowledge exploration*.

3.1 Browsing

When the knowledge in a database is of substantial size and complexity, there is frequent need for browsing in it. For efficient browsing, however, meaningful organization is required, preferably one in which *proximity* reflects *similarity*. The distance defined earlier helps establish such an organization.

For knowledge browsing, we assume a database as defined in Section 2.1; i.e., a set of Horn clauses, where the rules are safe, connected, consistent with the facts and mutually consistent. In such an environment it would be reasonable to assume that the rules are variant-free and subsumption-free. The distance between every two rules in the knowledge base is computed. A standard *radius* is then established for the knowledge base, and the *neighborhood* of a given rule is defined as the rules that are within radius distance from the given rule.

A browsing session is a sequence of *visits*, that begins at an arbitrary rule. When visiting a rule, the user is presented with the rules in its neighborhood. These rules are sorted by their distance from the visited rule. The user may then select a rule from this neighborhood and visit that rule, or the user may "extend" the present visit, by increasing the radius and thus broadening the neighborhood. By repeating this process, users are able to "navigate" in the knowledge base. The underlying structure perceived by users is a two-dimensional "map", where rules are placed according to their mutual distances.

3.2 Cooperative Knowledge Discovery

Cooperative knowledge discovery is an iterative process that begins with a hypothetical rule H suggested by the user. H may be regarded as an initial approximation of the desired knowledge, or as a "marker" for the area of interest.

The system begins by testing the consistency of H (whether it *holds*). The task of verifying consistency with the database *facts* is relatively simple, because a rule $\beta \longleftarrow \alpha$ is consistent with a set of facts, if and only if the answer to the query $\alpha \wedge \neg\beta$ is null. The task of verifying consistency with the database *rules* is more complex. However, a rule, shown to be consistent with the facts, might be inconsistent with the rules, only when its premise α is never satisfied. Thus, the second verification task is necessary only if the query α is also null.

Assume first that H holds. It is possible that an even more general rule holds. For example, the hypothesis $driver(X) \longleftarrow french(X) \wedge young(X)$ (all young French are drivers) could hold, when the more general rule $driver(X) \longleftarrow french(X)$ (all French are drivers) would hold as well. Because they imply the hypothesis, more general rules that hold are superior, and the system always seeks to *improve* a hypothesis that holds, by searching for its MCG. Note that there could be several such rules.

Assume now that H fails. It is possible that a somewhat more specific rule holds. For example, the hypothesis $driver(X) \longleftarrow french(X)$ (all French are drivers) might fail, but the more specific rule $driver(X) \longleftarrow french(X) \wedge young(X)$ (all young French are drivers) could hold. Because they are implied by the hypothesis, more specific rules that hold are desirable, and the system always seeks to *focus* a hypothesis that fails, by searching for its MCS. Again, there could be several such rules.

MCG and MCS may be considered *reformed* versions of the hypothesis H. In addition, whether H holds or not, the system searches for consistent rules that are close to H, but are neither its generalizations nor its specializations; such rules are called *closest consistent siblings (CCS)* of H. Altogether, the system presents the user with the *neighborhood* of H, consisting of reformations and siblings. This process can be summarized as follows:

1. User suggests hypothesis H.
2. System tests H.
3. If H holds, then system searches for maximal H-subsuming rules and closest sibling rules that hold.
4. If H fails, then system searches for minimal H-subsumed rules and closest sibling rules that hold.

This process employs three knowledge discovery tasks. Each task starts with a rule, and searches for rules that are its (1) maximal consistent generalizations, (2) minimal consistent specializations, and (3) closest consistent siblings.

Whereas the search for MCG need not have an "upper bound",[5] for the search for MCS a "lower bound" is often needed: a rule may have to be specialized considerably before it holds, and by then may lose its relevance. For example, if Pierre is the only French driver, then $driver(pierre) \longleftarrow french(pierre)$ would be the minimal specialization of the hypothesis $driver(X) \longleftarrow french(X)$. Such lower bounds may be established with the distance measure.

[5] Of course, the empty rule, which never holds, provides the ultimate upper bound.

A hypothesis $H : \beta \longleftarrow \alpha$ that fails, either holds occasionally (i.e., some substitutions that satisfy α also satisfy β), or it never holds (i.e., substitutions that satisfy α never satisfy β). In the first case, H can be specialized into a rule that holds. In the second case, the rule $H' : \longleftarrow \alpha \wedge \beta$ holds. If the radius of neighborhoods is large enough, H' will be discovered during the search for close siblings of H. For example, $owner(X, renault) \longleftarrow italian(X)$ (all Italians own cars made by Renault) is transformed in six steps into $\longleftarrow italian(X) \wedge owner(X, renault)$ (no Italians own cars made by Renault). A radius of at least 6 would allow the latter rule to be included in the neighborhood of the former.

In conclusion, every hypothesis H, either holds, or can be specialized into a rule that holds, or can be transformed into a rule that states that H never holds. Thus, every hypothesis evolves into a rule that holds.[6] However, the eventual discovery of this rule depends on the parameters of the neighborhood: the bound for reforming the hypothesis and the radius for determining its siblings. As in the browsing process, these parameters may be increased if the user so desires.

3.3 Integrated Browsing and Discovery

Browsing and discovery are both *exploratory* search processes. In either case, the search begins at an arbitrary object (a known rule in the former case, a hypothesized rule in the latter case), and the system retrieves close rules that are known to hold. However, in a browsing process, this neighborhood is extracted from the *stored* knowledge base, whereas in a discovery process, it is extracted from the *ultimate* knowledge base: the complete set of rules that hold, including both known and unknown rules.

This commonality suggests that these two processes can be integrated in a single process of *knowledge exploration*. In this process, neighborhoods are extracted from the ultimate knowledge base, and therefore include rules of three kinds: declared, already-discovered, and yet-undiscovered. Let $\mathcal{K}$ denote the variant- and subsumption-free base of declared and already-discovered rules.

As before, the exploratory process begins with a user hypothesis H, which the system then attempts to verify. This is accomplished first by looking up H in $\mathcal{K}$, and then, if it is not there, by actually testing it. If H holds, then $\mathcal{K}$ is searched for the MCG of H; otherwise, $\mathcal{K}$ is searched for the MCS of H. In either case, $\mathcal{K}$ is also searched for the CCS of H. The rules thus obtained comprise the *neighborhood* of H. It is extracted entirely from the available knowledge. Upon examining this neighborhood, the user may either select a rule from this neighborhood and retrieve its own neighborhood, or choose to enlarge the present neighborhood. Neighborhoods may be enlarged in two ways: they may be *broadened*, and they may be *deepened*. Broadening means that a wider neighborhood within $\mathcal{K}$ is to be searched. Deepening means that the same neighborhood is to be searched for new, yet-undiscovered rules. All rules thus discovered are added to $\mathcal{K}$.

[6] Though, as mentioned earlier, a specialized rule might be too focused to be of interest.

With these options, users may combine browsing in the available knowledge with attempts to discover new knowledge. By suggesting a hypothesis H, and then proceeding to deepen its neighborhood, this exploratory search is similar to our *discovery* process. If the option to deepen a neighborhood is never exercised, this exploratory search is conducted entirely within the available knowledge, and is similar to our *browsing* process.

4 Conclusion

4.1 Experimentation

To test our approach, an experimental system was designed and implemented. Its purpose was to obtain feedback on the soundness of the concepts (e.g., complexity, distance, neighborhood), and the adequacy of the knowledge exploration processes. There was only limited effort at applying *efficient* knowledge discovery techniques. The technique used for the derivation of MCG, MCS, and CCS (the three knowledge exploration tasks) may be described as essentially naive, yet with several significant refinements.

Our strategy is as follows. Given a rule that holds, its MCG are detected by gradually generalizing it, removing one constraint at a time, until a critical point is reached, where a rule holds, but all its immediate generalizations fail. Similarly, given a rule that fails, its MCS are detected by gradually specializing it, adding one constraint at a time, until a critical point is reached, where a rule fails, but some its immediate specializations hold. Because rules may have multiple MCG and MCS, these searches must be pursued in possibly many different directions.

To guard against "combinatorial explosion" of the number of rules generated and tested in these processes, we refined this brute force technique in several critical aspects. Most of these refinements are in the process of specialization, which is much less restricted than the process of generalization, simply because there are many more possibilities of *adding* a constraint to a rule, than of *removing* a constraint from a rule. Towards this end, the model was extended with the concepts of *types* and *domains*. A type is a label that is assigned to each argument of a predicate. A domain is the set of values that are possible for an argument of given type. One kind of immediate specialization is the identification of two different variables. By using types, the number of possible identifications is reduced considerably. Another kind of immediate specialization is the substitution of a variable by a constant. By considering only elements of the appropriate domain, the number of possible substitutions is reduced as well. Because domains could be large, the latter kind of specialization might still cause a rule to diverge into many different specializations. Here, we consider only domains that are relatively small. For example, the predicate $owner(X, Y)$ would be specialized by attempting specific car makers in Y, but not specific individuals in X. In addition to reducing the number of specialized rules, this restriction is justified by the fact that, assuming uniform distributions, substitutions of constants for variables of large domains tend to reduce the applicability of a rule (the number of occurrences that satisfy it) much too drastically.

4.2 Further Research Issues

We conclude by briefly discussing several issues that require further research.

The unavailability of existential quantification in the head limits the expressiveness of the knowledge. For example, it is impossible to express the rule that every person has a parent, which requires existential quantification in the head: $\forall X\ \exists Y person(X) \Rightarrow parent(Y, X)$. Further work is required to extend our work to rules that permit such quantification.

Our rules are interpreted as *integrity constraints*, rather than as *rules of inference*. This interpretation was motivated only by our desire to apply our results to knowledge discovery, where a discovered rule is a law that is maintained throughout the database, not a generator of new facts. However, all our results are immediately applicable to systems that interpret rules as inference agents or to systems that permit both kinds of rules. In addition, we foresee no great difficulties in allowing *built-in* predicates, to express arithmetic relationships.

Our definition of a consistent rule required that the head of the rule is satisfied by *every* substitution that satisfies its body. In practice, however, knowledge discovery applications often allow rules that are consistent in *most* instances. Like complexity, the *certainty* of a given rule would be another important measure of goodness, and the exploratory searches described in Section 3 would have to be modified to accommodate this new measure. For example, the concept of deepening a neighborhood would be extended to involve a parameter of *depth*, indicating the level of uncertainty that would be tolerated. Thus, neighborhoods would be controlled by a breadth parameter, to limit the distance from the hypothesis, and by a depth parameter, to limit the *distance from certainty*.

The aim of this work has been to develop concepts that would enable effective knowledge exploration, and we avoided any discussion of the efficiency of the proposed processes. Efficiency is less critical in processes that browse in the available knowledge, because all searches are confined to the stored rule base. By calculating in advance all distances, the system can prepare a "map" of the knowledge, thus enabling quick, real-time explorations. On the other hand, in processes that explore the yet-undiscovered knowledge, efficiency is critical. Here, we propose to apply knowledge discovery methodologies that have been proposed elsewhere, and which address efficiency concerns. For example, rather than search for *all* the sibling rules that are within a prescribed distance from the hypothesis, the system could combine precompiled statistics and further input from the user, to focus the search and quickly detect *some* of the more interesting rules. In general, our discovery processes have the potential of efficient implementation, because the hypothesis that launches them constraints the search considerably.

Finally, the notion of subsumption, which is at the core of this work, could be extended to define a rule $\beta \longleftarrow \alpha$ as a generalization of a rule $\delta \longleftarrow \gamma$, whenever $\alpha \longleftarrow \gamma$ and $\delta \longleftarrow \beta$. For example, if $french(X) \longleftarrow parisian(X)$ (all Parisians are French) and $driver(X) \longleftarrow owner(X)$ (all car owners are drivers), then $owner(X) \longleftarrow french(X)$ (all French own cars) is a generalization of $driver(X) \longleftarrow parisian(X)$ (all Parisians are drivers). This new generalization relationship reflects more *semantics* than the subsumption relationship.

References

1. R. Agrawal, N. H. Gehani, and J. Srinivasan. OdeView: The graphical interface to Ode. In *Proc. ACM-SIGMOD Int. Conf. Management of Data*, pp. 34–43, 1990.
2. S. Ceri, G. Gottlob, and L. Tanka. *Logic Programming and Databases.* Springer-Verlag, Berlin, Germany, 1990.
3. A. D'Atri, A. Motro, and L. Tarantino. ViewFinder: an object browser. Tech. Rep., Dept. Information and Software Systems Eng., George Mason Univ., 1992.
4. T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. In *Proc. Workshop Nonstandard Queries and Answers*, Vol. 2, pp. 101–120, 1991.
5. W. Klosgen. Visualization and adaptivity in the statistics interpreter EXPLORA. In *Proc. AAAI-91 Workshop Knowledge Discovery in Databases*, pp. 25–34, 1991.
6. T. Learmont and R. G. G. Cattell. An object-oriented interface to a relational database. Tech. Rep., Information Management Group, Sun Microsystems, 1987.
7. A. Motro. BAROQUE: A browser for relational databases. *ACM Trans. Office Information Systems*, 4(2):164–181, April 1986.
8. A. Motro. VAGUE: A user interface to relational databases that permits vague queries. *ACM Trans. Office Information Systems*, 6(3):187–214, July 1988.
9. A. Motro. FLEX: A tolerant and cooperative user interface to databases. *IEEE Trans. Knowledge and Data Engineering*, 2(2):231–246, June 1990.
10. G. Piatetsky-Shapiro. Discovery, analysis and presentation of strong rules. In *Knowledge Discovery in Databases*, pp. 229–248. AAAI Press, 1991.
11. G. Piatetsky-Shapiro. Knowledge discovery workbench: An exploratory environment for discovery in business databases. In *Proc. AAAI-91 Workshop Knowledge Discovery in Databases*, pp. 11–24, 1991.
12. G. Piatetsky-Shapiro, editor. *Proc. AAAI-91 Workshop Knowledge Discovery in Databases*, 1991.
13. G. Piatetsky-Shapiro and W. Frawley, editors. *Knowledge Discovery in Databases.* AAAI Press, 1991.
14. X. Pintado and D. Tsichritzis. An affinity browser. In *Active Object Environments*, pp. 51–60. Centre Universitaire d'Informatique, Université de Genève, 1988.
15. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, NY, 1983.
16. P. Smyth and R. M. Goodman. Rule induction using information theory. In *Knowledge Discovery in Databases*, pp. 161–176. AAAI Press, 1991.
17. S. R. Turner. Nexpert object (product review). *IEEE Expert*, 6(6):72–77, December 1991.