Semantic and Structural Query Reformulation for Efficient Manipulation of Very Large Knowledge Bases

Sang-goo Lee
Division of Science and Mathematics
University of Minnesota
Morris, Minnesota 56267

Donghoon Shin
Samsung Advanced Institute of Technology
P.O. Box 111
Suwon, Kyungkido, Korea

Abstract

A knowledge base can be divided into two subcomponents, namely, a rule base and a fact base. An element of the rule base would be a syntactic definition of a predicate in terms of others, usually in the form of a first order Horn clause. The fact base consists of positive unit clauses. By using an additional level of rules that can define the semantic relationship between the predicates, the system can screen out meaningless answers from the answer set generated by the syntactic rules alone. Also, by allowing general unit clauses (complex objects) as persistent objects of the fact base, the functionalities of the system can be considerably increased. We propose a framework for a logic oriented knowledge base system that supports two levels of rules and efficient manipulation of complex objects in its fact base.

1. Introduction

A knowledge base can be divided into two subcomponents, namely, a rule base and a fact base. Knowledge base systems for data intensive applications have been generally assumed that they consist of a front-end inference engine that operates on the rule base and a back-end database system to manage the large amount of data objects in the fact base. This is natural since the types of operations applied to the two types of knowledge, rules and facts, are quite different. General assumption is that the set of rules related to a given query is small enough to fit into main memory while the set of facts that need to be manipulated for the query is too large to bring into main memory. We employ the same general architecture of two stages of processing but improve on the functionality of each of the stages, thus, creating a highly versatile knowledge base system.

Our rule base consists of two types of rules; syntactic rules and semantic rules. Syntactic rules correspond to the Intensional Database (IDB) rules in a deductive database. They define the syntactic structures of defined predicates or views. Most rule base systems support single type of rules which correspond to our syntactic rules. Semantic rules define the relationships and other constraints on predicates. These rules are used to refine or screen out answers generated by the syntactic rules. For example, while an English sentence can be defined syntactically as a noun phrase followed by a verb phrase, it is the semantic rules about singulars and plurals that screen out awkward sentences like "The boys sings". By allowing two levels of rules, the syntactic structure can be kept relatively simple and clear. However, query processing in a multilevel rule base is complicated because in addition to extracting the syntactic structure for a given query, the system must identify relevant semantic rules. We present a method of identifying relevant semantic rules while extracting the syntactic structure.

Complex objects do not pose serious problems in the inference stage since the resolution principle using general unification is used. However, complex objects must be given special consideration for the fact base. The first normal form restriction posed on most relational database systems forces attribute values to be non-decomposable; each data element must be atomic such as an integer, a real, a string with a limited length, or a boolean. This is considered one of the factors limiting the modeling power of current relational database systems required to support advanced data intensive applications. Thus, in recent years, there has been considerable effort directed toward database systems without the constraint on structure [1, 3, 5].

In the context of logic-oriented knowledge base systems, Zaniolo [10] proposed to extend the relational algebra for complex unit clauses with function symbols (complex facts), and described compilation of queries into a sequence of Extended Relational Algebra operations. By allowing general first order terms in persistent data objects, we can produce a data model as powerful as the non-first normal form databases [9]. We employ the surrogate file approach of [9] for the physical structure of our fact base and present a query reformulation process corresponding to this structure.

We assume that the user is familiar with general terminology in relational databases and deductive databases. The syntactic rules of our rule base correspond to the intensional database (IDB) rules and the fact base corresponds to the extensional database (EDB). We assume that the IDB rules are range-restricted, Skolem-function-free, Horn clauses where all variables are universally quantified (V). The fact base consists of positive unit clauses that may contain complex objects. A predicate that represents a database relation, either IDB or EDB, is called a relational predicate.
In section 2, we present the use of semantic rules and how queries in a multilevel rule base are reformulated. In section 3, brief introduction to the physical structure of the fact base is given with the query reformulation process required for accessing this structure. The paper is concluded in section 4.

2. The Rule Base

2.1 A Two-Level Rule Base

Semantic query optimization has been introduced as a process of utilizing information from integrity constraints to optimize a given query [2, 6, 7]. Upon receiving a query, relevant integrity constraints (IC) are searched and they are used in introducing restrictions or eliminating redundant joins [6]. Since the information is used for optimization purposes, the use is optional; that is, if the use of the restriction will not improve the efficiency, it can be discarded. However, in a two-level rule base system, the use of semantic rules are mandatory. We discuss the difference by employing the proof theoretic approach [4].

In a deductive database, the answer set is derived (denoted by the symbol \( \Rightarrow \)) from the set of EDB and IDB.

\[
EDB \cup IDB \Rightarrow \text{answer}
\]

Since in a valid state of database, IC is a consequence of EDB \( \cup \) IDB (i.e. \( EDB \cup IDB \Rightarrow IC \)), the same set of answers can be derived with IC added to the database.

\[
EDB \cup IDB \cup IC \Rightarrow \text{answer}
\]

Thus, in semantic query optimization, it can be assumed that use of IC's will not alter the answer set.

In a two-level rule base system, the use of both types of rules is mandatory. However, the set of semantic rules (SR) is not a consequence of EDB and IDB. The answer set in a two-level rule base system is defined as

\[
EDB' \cup IDB \cup SR \Rightarrow \text{answer}
\]

where EDB' is the maximal subset of EDB such that EDB' \( \cup \) IDB \( \cup \) SR is satisfiable.

Example 2.1 Consider the following database.

\[
SR = \{ \leftarrow \langle X, b \rangle \}
\]

\[
IDB = \{ \langle U, V \rangle \leftarrow q(U, V) \}
\]

\[
EDB = \{ q(a, b), q(N(c), d) \}
\]

The set of clauses \( SR \cup IDB \cup EDB \) is unsatisfiable. By deleting \( q(a, b) \) from the EDB, the set is satisfiable and so EDB' = \( \{ q(N(c), d) \} \). The answer to a query \( ?p(X, Y) \) will be \( \{ p(h(c), d) \} \).

Example 2.2 Consider the following relation employee.

employee(Name, ID, Address, Dept, personal(Sex, Race, Age))

To hide certain information as personal, we can use the following syntactic rule

\[
\text{employee}(X, Y, Z) \leftarrow \text{employee}(X, Y, Z, D, U)
\]

However, the information can be protected only when direct access to employee is prohibited by some system control. So, the user would have to specify access privileges for the relations using some system language or other means. When the number of relations is large and the rules of assigning access privileges are complicated, it will be quite difficult to rely only on the protection scheme supplied by the system.

In a two-level rule structure, the degree of freedom with which a user may specify such access privileges is limited only by the user's ability to express them in predicate logic clauses. The personal information of the above relation can be hidden by using the following semantic rule:

\[
\text{access}(UserID, \text{special}) \leftarrow \text{employee}(X, Y, Z, D, U), \text{output}(U), \text{systemid}(ID, UserID)
\]

where predicate \( \text{output} \) is a special predicate that is true only if its argument is an output variable and \( \text{system} \) is a system predicate through which system variables are bound. Relation \( \text{access} \) is a table of users and their access levels. For a query \( ?\text{employee}(X, N, Z) \), the predicate \( \text{output} \) will be false and the semantic rule is satisfied. For a query \( ?\text{employee}(X, Y, Z, U) \), the predicate \( \text{output} \) is true and the predicate \( \text{system} \) will bind the variable \( UserID \) to the ID of the user and evaluate to true. Since the body is satisfied, the semantic rule is true only when its head predicate is true. So the head predicate with its appropriate binding is added to the query resulting in

\[
?\text{employee}(X, Y, Z, U), \text{access}(UserID, \text{special})
\]

If \( \text{access}(UserID, \text{special}) \) evaluates to false, then the answer set is \( \{ \} \).

Use of semantic rules are different from the use of syntactic rules in 3 ways. First, while syntactic rules are used for generating answers, use of semantic rules can only result in restricting the answer. For example, while adding a syntactic rule for a relation \( R \) to the rule base will allow more tuples to be generated, adding a semantic rule about \( R \) to the rule base will eliminate certain tuples from \( R \).

Second, while the relevancy of a syntactic rule to a given query is determined by whether the head predicate is referenced in the query either directly or indirectly, the relevancy of a semantic rule is decided by whether the body of the rule is referenced (or satisfied) in the query either directly or indirectly. Since the body of a clause may involve more than one literal and the variable patterns may vary, deciding whether or not a semantic rule is relevant to a query is a complicated matter. The fact that queries may be expressed in a number of different forms using the syntactic rules adds to the complexity.

Third, while a relevant syntactic rule will substitute its body for the head predicate appearing in the query preserving logical equivalence, a relevant semantic rule will add its head to the query, thus, changing the logical meaning of the query.

In conclusion, a semantic rule is relevant when its body is satisfied by the predicates and instantiation of the query, and when a semantic rule is found relevant, its head is added to the query making the query a more restrictive one.

2.2 Semantic Query Reformulation

The process of identifying relevant semantic rules for a given query in a two-level rule base is identical to finding relevant integrity constraints for semantic query optimization.
pursposes in a deductive database. Semantic query optimization in deductive databases have been studied in [2, 7, 8]. As shown in [7], identifying and using information from relevant integrity constraints in a recursive database requires a different type of analysis from a nonrecursive one. So, to simplify our discussions, we will not consider recursive relations. Furthermore, we will assume that nonrelational predicates (arithmetic, comparison, or special predicates such as output, and system in the above example) can be evaluated with minimal overhead. Thus, the relevancy of a semantic rule can be decided rather simply once the pattern of the relational predicates in the body of the rule has been identified in the query. Without loss of generality, we will further assume that there are at most two relational predicates in the body of a semantic rule since other cases can be easily generalized.

**Example 2.3** Consider the following rule base where \( m, n, l, \) and \( k \) are EDB relations (nonrelational predicates are omitted form the body of the rules).

\[
\text{SR} = \{ s1. \ X1=a \rightarrow p(X, Y) ; \\
\text{s2.} \ X2= Y2 \rightarrow q(X, Y), r(W, Y) ; \\
\text{s3.} \ X3>b \rightarrow m(X, U), n(U, Y3) ; \\
\text{s4.} \ X4=d \rightarrow n(X, Y), r(W, Y) \\
\}
\]

\[
\text{IDB} = \{ p(X, Y) \leftarrow q(X, W), r(W, Y) ; \\
q(X, Y) \leftarrow m(X, Y) ; \\
r(X, Y) \leftarrow m(X, Y) ; \\
r(X, Y) \leftarrow n(X, Y) \\
\}
\]

For a query \( Q = \text{ ?p(a, Y)} \), by the IDB rules, we have the following forms of the query, each of which is called an extension of the query \( Q \):

\[
p(a, Y) = q(a, W) \land r(W, Y) \quad (Q1) \\
\equiv (m(a, W) \lor n(a, W)) \land r(W, Y) \quad (Q2) \\
\equiv (m(a, W) \land (l(W, Y) \lor k(W, Y))) \land r(W, Y) \quad (Q3) \\
\equiv (m(a, W) \lor (l(W, Y) \lor k(W, Y))) \quad (Q4)
\]

Semantic rule \( s1 \) is relevant to the query since it involves the relation \( p \) and by applying the substitution to \( s1 \) that unifies the body of \( s1 \) with the query, the head becomes \( a=a \). This head would be added to the query if it were not trivially true. Rule \( s2 \) is relevant in \( Q1 \). Applying appropriate substitutions, the head of \( s2 \), which becomes \( a<Y \), is added to the query.

Although predicates \( m \) and \( n \) appear in \( Q2 \) and \( Q4 \), they do not match the pattern in the body of \( s3 \). Thus \( s3 \) is not relevant to this query. In \( Q4 \), predicates \( n \) and \( l \) do not exactly match the pattern of \( s4 \). However, it is obvious that the tuples of the two relations will eventually be joined, and by \( s4 \), they must be restricted to certain values. By applying distributive law to \( Q4 \), we get

\[
(m(a, W) \land (l(W, Y) \lor k(W, Y))) \lor (m(a, W) \land (k(W, Y))) \lor (m(a, W) \land (l(W, Y))) \\
\land (m(a, W) \land (k(W, Y))) \lor (m(a, W) \land (l(W, Y))) \lor (m(a, W) \land (k(W, Y)))
\]

The head of \( s4 \) becomes false \( (a=d) \) with the substitutions from the underlined portion, and so we can delete the join term of \( n \) and \( l \). Then the transformed query becomes

\[
(m(a, W) \land (l(W, Y) \lor k(W, Y))) \lor (m(a, W) \land (k(W, Y)))
\]

A brute force method of matching every semantic rule against every extension of the query is not practical since the number of comparisons can be prohibitively large. The method presented in [2] reduces this number of semantic rules that need to be tested but fails to reduce the complexity itself.

**Figure 2.1**

We employ a method which utilizes the structure of the query graph. A tree structure (AND/OR tree) can be constructed from the query and IDB rules. This tree is called the query tree. A query tree for the above query is shown in Figure 2.1. A rectangular node (OR node) represents a relation and an oval node (AND node) represents a definition of a predicate. If a relation is defined by two IDB rules, it has links to the two AND nodes that define the relation. Each AND node has links to the OR nodes whose predicates appear in the body of the rule. We present a labeling method which assigns sequential numbers to branches of a query tree. With the labeling, the relative position of a node in a query tree is identified by its AND and OR traces. By comparing the traces of two nodes, we can determine the relationship of the nodes.

**Algorithm 1 (Branch Labeling)**

\[
\text{Label the branches of a query tree } T(Q) \text{ of query } Q \}
\]

**Step 0. (Initialize)** let \( k = 0 \);

**Step 1. (Labeling)**

for each node \( N \) of \( T(Q) \) do

\[
\text{case} \text{ Number_of_branches_from } (N) \text{ of}
\]

1. label the branch with \( \epsilon \) (the null string) ;

\[
n, \text{ where } n > 1: \text{ label branches from } k1 \text{ to } kn \}
\]

\[
k := k + 1;
\]

**case end**

**Definition 2.1** The AND Trace (OR Trace, resp.) of a node \( P \) denoted as \( \alpha_P \) (\( O_P \), resp.) is the concatenation of labels of branches out of AND nodes (OR nodes, resp.) in the path from the root node to the node \( P \).

**Example 2.4** The branches of the query tree of Figure 2.1 are labeled using Breadth-first traversal. Corresponding AND/OR traces of OR nodes are shown below.

\[
\begin{align*}
\alpha_p &= \epsilon \\
O_p &= \epsilon \\
\alpha_m &= \alpha_l \\
O_m &= \epsilon \\
\alpha_w &= l_1 \\
O_w &= l_1 \\
\alpha_n &= 2_1 \\
O_n &= 2_2 \\
\alpha_l &= l_2 \\
O_l &= 3_1
\end{align*}
\]
Theorem Let $S$ be a semantic rule with relational predicates $p$ and $q$ in its body. If the nonrelational predicates of the body are satisfied by the query instantiations, then

1. $S$ is relevant and the head can be added to the query directly if, and only if, $O_p = O_q$, and neither of $\alpha_p$ and $\alpha_q$ is a prefix of the other.
2. $S$ is relevant but needs restructuring of the tree to obtain the formula $(p \land q)$ if, and only if,
   a. $O_p \neq O_q$, and neither of $\alpha_p$ and $\alpha_q$ is a prefix of the other, and
   b. one of $O_p$ and $O_q$ is a prefix of the other, or
   if $k_i$ and $k_j$ are the leftmost differing labels of $O_p$ and $O_q$, then $k_i \neq k_j$.

The significance of the theorem is that by merely comparing the AND and OR traces of the relational predicates of the body of a semantic rule, we can identify whether the rule is relevant or not (proof can be found in [8]). The query tree is extracted from the set of IDB rules when compiling the query into an expression involving no IDB predicates. In the process, we are forced to visit each node in the tree. Hence, the labeling can be done at the time of extraction. So, the time incurred exclusively by the semantic rule collecting process is the time used in visiting the semantic rules. Since we assumed that the number of relational predicates in the body of a rule is linear to the number of semantic rules, which makes it practical to have two-level rules. The heads of relevant semantic rules are added to the query, which will require expanding the query tree. The process continues until no more semantic rules can be collected.

3. The Fact Base

The reformulated query is sent to the next stage of query processing where the fact base is accessed. In this section, we describe the storage structure for the fact base and reformulation of queries passed from the inference stage into a sequence of relational operations on normalized sub-objects.

3.1 Storage Structure for Complex Unit Clauses

Definition 3.1: Let $P(A_1, A_2, \ldots, A_k)$ be a unit clause with arity $k$. An argument of $P$, $A_i$, is a simple argument if $A_i$ is a constant. Otherwise, it is a complex argument.

Definition 3.2: A complex unit clause is a unit clause where at least one argument is complex.

We adopt a normalized storage model for query reformulation where a complex unit clause is decomposed into a number of normalized subobjects. A unique identifier (UID) is assigned to each unit clause so that, after normalization, subobjects can be related to each other, and the actual knowledge base can be accessed via the UID. We can identify subobjects of a complex unit clause by assigning a local address within the clause to each complex argument. Consider, for example, the following complex unit clause

$$r(a, p(b, q(c(X))), e) \quad \text{UID} = 579.$$
entry of the subrelation table. Details of the physical file structure is beyond the scope of this paper, and the reader is have been inserted.

<table>
<thead>
<tr>
<th>Subrelation Table</th>
<th>Tag Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>Addr</td>
</tr>
<tr>
<td>54 0</td>
<td>r</td>
</tr>
<tr>
<td>54 0.2</td>
<td>p</td>
</tr>
<tr>
<td>54 0.2</td>
<td>q</td>
</tr>
<tr>
<td>57 0</td>
<td>s</td>
</tr>
<tr>
<td>57 0</td>
<td>k</td>
</tr>
</tbody>
</table>

3.2 Query Reformulation

In this section we describe the transformation of queries on complex objects into a sequence of relational operations on normalized subrelations by using the structural information stored in the subrelation table and the tag table. Although reformulated queries from the inference stage of section 2 will usually involve unions and joins, we will concern ourselves with only single predicate queries since more general ones can be answered by applying relational operations on answers from simple subqueries.

3.2.1 PNS and NSR Plan Generation

Complex object processing is based on the decomposition of a query into a sequence of relational operations on normalized subrelations. The normalized subrelations required for constructing the query are identified by accessing the subrelation table and the tag table. The identification of required subrelations is recursively performed since objects can be arbitrarily complex. Note that, if an argument of a query is a constant, the corresponding argument of the normalized subrelation is known. We call such information a Partial Normalized Subrelation (PNS). Variables in PNS's are bound to appropriate constants by consulting the subrelation table and the tag table. When two or more tags represented by the same variable are bound to the complex tag type (i.e. 01), new PNS's are created in order to construct the next level subrelations. We call such an event complex tag sharing. New PNS's are generated until no more complex tag sharing occurs.

The required relational operations can also be recognized for PNS's and NSR's from the query. The constant keys for each subquery are associated with a selection operation denoted by $\sigma$. Consider, for example, the subquery $\leftarrow p(a, X)$ and the corresponding PNS, $P_1 = (p, 3, 0, **, V_j)$. From the subquery the selection operation on $P_1$, $\sigma_{P_1} = \alpha P_1$ can be obtained. We use "R.a" to represent the UID field of a normalized relation $R$. A projection operation is represented as $P_{a,i} = \alpha P$. An intersection operation $(R \cap S)$ is required between two subrelations with the same RID, and a union operation $(R \cup S)$ is recognized when a PNS generates two or more NSR's. A join operation between two subrelations $R$ and $S$ is represented by $R \times_{a,i} S_{a,j} \ldots S_{a,k}$.

From the given query, we can obtain a number of PNS's and the relational operations on the PNS's. The sequence of relational operations defined on PNS's is an intermediate query processing plan called the PNS plan. By accessing the subrelation table and the tag table, we can obtain NSR's by replacing the don't care match indicators and uninstantiated variables in the PNS's with constant values, and relational operations defined on PNS's are expanded using NSR's. A NSR plan consists of the NSR's obtained from the subrelation table and the tag table, and the sequence of relational operations on them. Each relational operation for NSR's is further expanded to operations on physical schema.

<table>
<thead>
<tr>
<th>Subrelation Table</th>
<th>Tag Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>Addr</td>
</tr>
<tr>
<td>54 0</td>
<td>p</td>
</tr>
<tr>
<td>54 0.2</td>
<td>q</td>
</tr>
<tr>
<td>54 0.2</td>
<td>r</td>
</tr>
<tr>
<td>54 0.3</td>
<td>s</td>
</tr>
<tr>
<td>54 0.2</td>
<td>s</td>
</tr>
<tr>
<td>54 0.3</td>
<td>g</td>
</tr>
<tr>
<td>54 0.3</td>
<td>h</td>
</tr>
<tr>
<td>78 0</td>
<td>p</td>
</tr>
<tr>
<td>78 0</td>
<td>s</td>
</tr>
<tr>
<td>78 0</td>
<td>k</td>
</tr>
<tr>
<td>19 01</td>
<td>00</td>
</tr>
<tr>
<td>19 01</td>
<td>00</td>
</tr>
</tbody>
</table>

Figure 3.1 Subrelation Table and Tag Table

Example 3.1 Let the given query be $\leftarrow p(a,q(b,X),X)$. The tables be as shown in Figure 3.1. We can obtain the following PNS's by decomposing the query:

$P_1 = (p, 3, 0, V_j, '00 01 V_3')$

$P_2 = (q, 2, 0.2, V_j, '00 V_3')$
And the following plan on the PNS's can be obtained from the query:
\[
\pi_0 \left( (\sigma_{s = s^{P1}} P1) \times (\sigma_{s = s^{P2}} P2, \sigma_{s = s^{P3}} P3, \sigma_{s = s^{P4}} P4) \right)
\]

The following NSR's are obtained from the PNS's and the tables:
- N1 = \((p, 3, 0, 54, '00 01 00')\) from P1
- N2 = \((p, 3, 0, 45, '00 01 01')\) from P1
- N3 = \((q, 2, 0.2, 54, '00 00')\) from P2
- N4 = \((q, 2, 0.2, 54, '00 01')\) from P2

The common variable \(V_2\) in P1 and P2 causes a complex tag sharing, and thus the next level subrelations of N2 and N4 must be considered. From N2 and N4, we can obtain the following PNS's:
- P3 = \((V_3, V_4, 0.3, 54, V_5)\)
- P4 = \((V_3, V_4, 0.2, 54, V_3)\)

and we can obtain the following NSR's from P3 and P4 using the subrelation table:
- N5 = \((s, 2, 0.3, 54, '00 00')\) from P3
- N6 = \((s, 2, 0.2, 54, '00 00')\) from P4

Since there exists no more complex tag sharing, the NSR plan for the ENLQ can be obtained as follows:
\[
\pi_0 \left( (\sigma_{s = s^{N1}} N1) \times (\sigma_{s = s^{N2}} N2, \sigma_{s = s^{N3}} N3) \right) \cup \\
\left[ \pi_0 \left( (\sigma_{s = s^{N4}} N4) \cap \pi_0 \left( (\sigma_{s = s^{N5}} N5, \sigma_{s = s^{N6}} N6) \right) \right) \right]
\]

4. Conclusions

We have presented a framework for a knowledge base system that supports complex objects and two-level rules. By assuming that the portion of a rule base that is related to a query is small enough to fit in main memory, the bottleneck of the inference stage is not in unifying or managing complex objects but in identifying relevant rules for the query. However, efficient storage and manipulation of complex objects is critical in the physical database access stage where the fact base consists of large number of general objects. Consequently, we have divided the system into two virtually independent stages.

An obvious application of a two-level rule base is in semantic query optimization in which case, the integrity constraints will be the semantic rules and application of restrictions from them are optional. By supplying a number of special system predicates, the two-level rule base can be used to control the activities of the knowledge base. The two levels of rules naturally map to rules and meta rules in artificial intelligence applications.

The choice of normalized storage model (NSM) for our basic storage model is based on the assumption that queries in a very large knowledge base system are highly selective, that is, queries generally contain many search keys and generate a relatively small number of results. We have also assumed that most join operations are accompanied with some searching keys (partial match joins), or with some integrity constraints so that the operations to be performed are highly selective, that is, the results consists of only a fraction of the data. When this is the case, our scheme provides good performance in complex object management. In addition, updating operations of NSM are more efficient than those of fully decomposed storage model (FDSM). For queries based on a relatively small number of search keys, however, the use of FDSM with inverted indexes may outperform the NSM. The combination of NSM and FDSM based on some heuristics is subject to future research as the selection of an optimal indexing scheme for queries is known to be very difficult.

Acknowledgements

The authors would like to express their sincere gratitude to Professor Lawrence Henschen of Northwestern University and Professor Bruce Berra of Syracuse University for their contributions on earlier versions of this work.

References


