Speeding Up Multi-Relational Data Mining

Anna Atramentov, Vasant Honavar
Artificial Intelligence Research Laboratory,
Computer Science Department, Iowa State University,
226 Atanasoff Hall, Ames, IA 50011-1040, USA,
{anjuta, honavar}@cs.iastate.edu

Abstract

We present a general approach to speeding up a family of multi-relational data mining algorithms that construct and use *selection graphs* to obtain the information needed for building predictive models (e.g., decision tree classifiers) from relational database. Preliminary results of our experiments suggest that the proposed method can yield 1-2 orders of magnitude reductions in the running time of such algorithms without any deterioration in the quality of results. The proposed modifications enhance the applicability of multi-relational data mining algorithms to significantly larger relational databases that would otherwise be not feasible in practice.

1 Introduction

Recent advances in high throughput data acquisition, digital storage, and communications technologies have made it possible to gather very large amounts of data in many scientific and commercial domains. Much of this data resides in relational databases. Even when the data repository is not a relational database, it is often convenient to view heterogeneous data sources as if they were a collection of relations [Reinoso-Castillo, 2002] for the purpose of extracting and organizing information from multiple sources. Thus, the task of learning from relational data has begun to receive significant attention in the literature [Blockeel, 1998; Knobbe *et al.*, 1999a; Friedman *et al.*, 1999; Koller, 1999; Krogel and Wrobel, 2001; Getoor, 2001; Kersting and De Raedt, 2000; Pfeffer, 2000; Dzeroski and Lavrac, 2001; Dehaspe and Raedt, 1997; Jaeger, 1997; Karalic and Bratko, 1997].

Recently, [Knobbe *et al.*, 1999a] outlined a general framework for multi-relational data mining which exploits structured query language (SQL) to gather the information needed for constructing classifiers (e.g., decision trees) from multi-relational data. Based on this framework, several algorithms for multi-relational data mining have been developed. Experiments reported by [Leiva, 2002] have shown that MRDTL – a multi-relational decision tree learning algorithm is competitive with other approaches to learning from relational data. One common feature of all algorithms based on the multi-relational data mining framework proposed by [Knobbe *et al.*, 1999a] is their use of *selection graphs* to query the relevant databases to obtain the information (e.g., statistics) needed for constructing a model. Our experiments with MRDTL revealed that the execution of queries encoded by such selection graphs was a major bottleneck in terms of the running time of the algorithm. Hence, this paper describes an approach for significantly speeding up some of the most time consuming components of such algorithms. Preliminary results of our experiments suggest that the proposed method can yield one to two orders of magnitude speedups in the case of MRDTL. We expect similar speedups to be obtained with other multi-relational data mining algorithms which construct and use selection graphs.

The rest of the paper is organized as follows: in Section 2 we overview multi-relational data-mining framework, in Section 3 we describe the speed up scheme for this framework and in Section 4 we show the experimental results that we obtained applying the scheme.

2 Multi-Relational Data Mining

2.1 Relational Databases

A relational database consists of a set of tables $D = \{X_1, X_2, ...X_n\}$, and a set of associations between pairs of tables. In each table a row represents description of one record. A column represents values of some attribute for the records in the table. An attribute A from table X is denoted by $X.A$.

Definition 2.1 *The* domain *of the attribute $X.A$ is denoted as $DOM(X.A)$ and is defined as the set of all different values that the records from table X have in the column of attribute A.*

Associations between tables are defined through primary and foreign key attributes in D.

Definition 2.2 *A* primary key *attribute of table X, denoted as $X.ID$, has a unique value for each row in this table.*

Definition 2.3 *A* foreign key *attribute in table Y referencing table X, denoted as $Y.X\_ID$, takes values from $DOM(X.ID)$.*

An example of a relational database is shown in Figure 1. There are three tables and three associations between tables. The primary keys of the tables GENE, COMPOSITION, and INTERACTION are: GENE_ID, C_ID, and

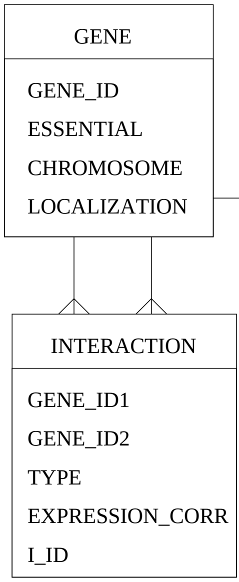

Figure 1: Example database

I_ID, respectively. Each COMPOSITION record references some GENE record through the foreign key COMPOSITION.GENE_ID, and each INTERACTION record references two GENE records through the foreign keys INTERACTION.GENE_ID1 and INTERACTION.GENE_ID2.

In this setting, if an attribute of interest is chosen, it is called *target attribute*, and the table in which this attribute is stored is called *target table* and is denoted by T_0.

Each record in T_0 corresponds to a single object. Additional information about an object is stored in other tables of the database, which can be looked up, when following the associations between tables.

2.2 Multi-Relational Data Mining Framework

Multi-relational data mining framework is based on the search for interesting patterns in the relational database, where multi-relational patterns can be viewed as "pieces of substructure encountered in the structure of the objects of interest" [Knobbe *et al.*, 1999a].

Definition 2.4 *A multi-relational object is* covered *by a multi-relational pattern iff the substructure described by the multi-relational pattern, in terms of both attribute-value conditions and structural conditions, occurs at least once in the multi-relational object. ([Knobbe* et al.*, 1999a])*

Multi-relational patterns also can be viewed as subsets of the objects from the database having some property. The most interesting subsets are chosen according to some measure (i.e. information gain for classification task), which guides the search in the space of all patterns. The search for interesting patterns usually proceeds by a top-down induction. For each interesting pattern, subputterns are obtained with the help of refinement operator, which can be seen as further division of the set of objects covered by initial pattern. Top-down induction of interesting pattern proceeds recursively applying such refinement operators to the best patterns.

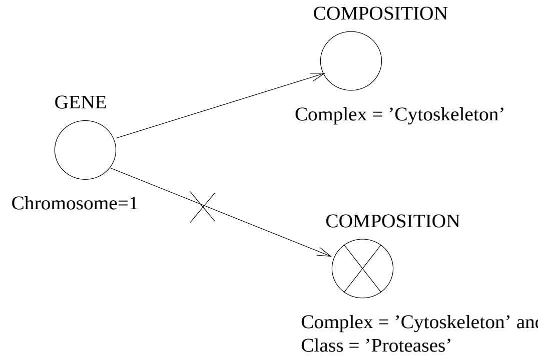

Figure 2: Selection graph, corresponding to those GENE(s) that belong to chromosome number 1, that have at least one COMPOSITION record whose complex value is 'Cytoskeleton', but for which none of the COMPOSITION records have complex value 'Cytoskeleton' and class value 'Proteases' at the same time.

Multi-relational pattern language is defined in terms of *selection graphs* and *refinements* which are described in the following sections.

2.3 Selection Graphs

Multi-relational patterns are expressed in a graphical language of selection graphs [Knobbe *et al.*, 1999b].

Definition 2.5 *A* selection graph S *is a directed graph* $S = (N, E)$*.* N *represents the set of nodes in* S *in the form of tuples* (X, C, s, f)*, where* X *is a table from* D*,* C *is the set of conditions on attributes in* X *(for example,* X*.color = 'red' or* X*.salary* $>$ *5,000),* s *is a flag with possible values* open *and* closed*, and* f *is a flag with possible values* front *and* back*.* E *represents edges in* S *in the form of tuples* (p, q, a, e)*, where* p *and* q *are nodes and* a *is a relation between* p *and* q *in the data model (for example, X.ID = Y.X_ID), and* e *is a flag with possible values* present *and* absent*. The selection graph should contain at least one node* n_0 *that corresponds to the target table* T_0*.*

An example of the selection graph for the data model from Figure 1 is shown in Figure 2. This selection graph corresponds to those GENE(s) that belong to chromosome number 1, that have at least one COMPOSITION record whose complex value is 'Cytoskeleton', but for which none of the COMPOSITION records have complex value 'Cytoskeleton' and class value 'Proteases' at the same time. In this example the target table is GENE, and within GENE the target attribute is LOCALIZATION.

In graphical representation of a selection graph, the value of s is represented by the presence or absence of a cross in the node, representing the value *open* and *closed*, respectively. The value for e, in turn, is indicated by the presence (*present* value) or absence (*absent* value) of a cross on the corresponding arrow representing the edge. An edge between nodes p and q chooses the records in the database that match the joint condition, a, between the tables which is defined by the relation between the primary key in p and a foreign key in q, or the other way around. For example, the join condition, a, between table GENE and COMPOSITION in selection graph from Figure 2 is GENE.GENE_ID = COMPOSITION.GENE_ID.

A *present* edge between tables p and q combined with a list of conditions, $q.C$ and $p.C$, selects those objects that match the list of conditions, $q.C$ and $p.C$, and belong to the join between p and q, specified by join condition, $e.a$. On the other hand, an *absent* edge between tables p and q combined with a list of conditions, $q.C$ and $p.C$, selects those objects that match condition $p.C$ but do not satisfy the following: match $q.C$ and belong to the join between tables at the same time.

Flag f is set to *front* for those nodes that on their path to n_0 have no closed edges. For all the other nodes flag f is set to *back*.

[Knobbe *et al.*, 1999b] introduces the algorithm (Figure 3) for translating a selection graph into SQL query that returns the records in the target table covered by this selection graph, where subgraph(S, $j.q$) procedure returns the subgraph of the selection graph S starting with the node q as the target node, with label s reset to *open*, removing the part of the graph that was connected to this node with the edge j and reseting all the values of flag f at the resulting selection graph by definition of f. Notation $j.q$.key means the name of the attribute (primary or foreign key) in the table q that is associated with the table p in relation $j.a$.

```
TRANSLATE(S, key)
Input Selection graph S, key (primary or foreign) in the target node of S
Output SQL query for creating sufficient information about graph S
table_list := ''
condition_list := ''
join_list := ''
for each node i in S do
    if (i.s = 'open' and i.f = 'front')
        table_list.add(i.table_name + 'T' + i)
        for each condition c in i do
            condition_list.add(c)
for each edge j in S do
    if (j.e = 'present')
        if (j.q.s = 'open' and j.q.f = 'front')
            join_list.add(j.a)
    else
        join_list.add(
            j.p + '.' + j.p.primary_key + ' not in ' +
            TRANSLATE( subgraph(S, j.q), j.q.key))
return 'select distinct' + 'T0.' + key +
    ' from ' + table_list +
    ' where ' + join_list + ' and ' + condition_list
```

Figure 3: Translation of selection graph into SQL query

Using this procedure the graph in Figure 2 translates to the SQL statement shown in Figure 4.

```
select  distinct T0.gene_id
from    GENE T0, COMPOSITION T1
where   T0.gene_id = T1.gene_id
        and T0.chromosome = 1
        and T1.complex = 'Cytoskeleton'
        and T0.gene_id not in
        ( select T0.gene_id
        from  COMPOSITION T0
        where  T0.complex = 'Cytoskeleton'
            and T0.class = Proteases)
```

Figure 4: SQL query corresponding to the selection graph in Figure 2

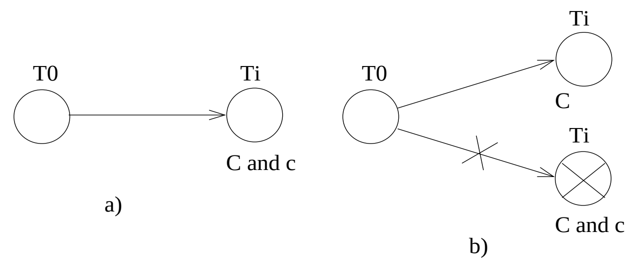

Figure 5: Complement refinements for adding condition to the node: a) positive condition, b) negative condition

2.4 Refinements of the Selection Graphs

Multi-relational data mining algorithms search for and successively refine interesting patterns and select promising ones based on some impurity measure (e.g. information gain). The set of refinements introduced by [Knobbe *et al.*, 1999b] are given below. Note that all of these refinements can only be applied to the *open*, *front* nodes in the selection graph S.

- Add positive condition (Figure 5 a)). This refinement will simply add a condition c to the set of conditions C in the node T_i of selection graph S without actually changing the structure of S.
- Add negative condition (Figure 5 b). If the node which is refined is not n_0, this refinement will introduce a new absent edge from the parent of the selection node in question. The condition list of the selection node will be copied to the new closed node, and will be extended by the new condition. This node will also get the copies of the children of the selection graph in question and open edges to those children will be added. If the node which is refined does represent the target table, the condition is simply negated and added to the current list of conditions for this node. This refinement is complement to the "add positive condition refinement", in the sense that it covers those objects from the original selection graph which were not covered by corresponding "add positive condition" refinement.
- Add present edge and open node (Figure 6 a)). This refinement introduces a present edge together with its corresponding table to the selection graph S.

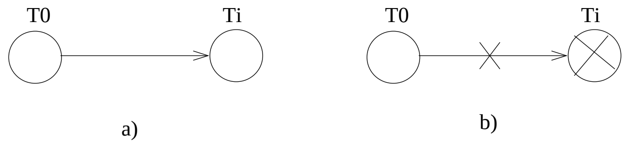

Figure 6: Complement refinements for adding edge to selection graph: a) adding present edge and open node, b) adding absent edge and closed node

- Add absent edge and closed node (Figure 6 b). This refinement introduces an absent edge together with its corresponding table to the selection graph S. This refinement is complement to the "add present edge and open node", in the sense that it covers those objects from the original selection graph which were not covered by "add present edge and open node" refinement.

It is important to note that only through the "add edge" refinements the exploration of all the tables in the database is done. We can consider "add condition" refinement on some attribute from some table only after the edge to that table has been added to the selection graph. What happens if the values of the attributes in some table are important for the task but the edge to this table can never be added, i.e. adding edge doesn't result in further split of the data covered by the refined selection graph? Look ahead refinements, which are a sequence of several refinements, are used for dealing with this situation. In the case when some refinement doesn't split the data covered by the selection graph, the next set of refinements is also considered as refinements of the original selection graph.

3 Speeding Up Multi-Relational Data Mining

Let S be some selection graph. Any refinement of S covers the subset of instances covered by S. Since all the refinements of S usually need to be examined, storing intermediate results obtained from S will reduce the amount of time needed to examine all its refinements.

The goal of this section is to show what intermediate information should be stored for each selection graph S so that the instances covered by each of its refinements can be recovered quickly.

The knowledge of the structure of the selection graph S is enough to restore all the objects in the database corresponding to any refinement R of S. This can be done by first applying the refinement to S to obtain a refined selection graph $R(S)$, which in turn is then transformed into an SQL query as described in Subsection 2.3. The size of the resulting SQL query increases with the complexity of the graph, resulting in the corresponding increase in the execution time of the query.

It is possible to substantially speed up this step of the algorithm as follows. For each object covered by selection graph S we store only its class label and the primary key values from the tables corresponding to the *open*, *front* nodes in the selection graph S. We call the resulting table the sufficient table for S and denote it by I_S.

The procedure that transforms selection graph S into SQL query for creating sufficient table I_S is shown in Figure 7.

```
SUF_TABLE(S)
Input Selection graph S
Output SQL query for creating sufficient table I_S
   table_list, condition_list, join_list :=
        extract_from(TRANSLATE(S))
   primary_key_list := 'T_0.target_attribute'
   for each node i in S do
        if (i.s = 'open' and i.f = 'front')
             primary_key_list .add(i.ID)
   return 'create table I_S as ' +
        ' (select ' + primary_key_list +
        ' from ' + table_list +
        ' where ' + join_list +
        ' and ' + condition_list + ')'
```

Figure 7: Algorithm for generating SQL query corresponding to the sufficient table I_S of the selection graph S

Given a sufficient table I_S, we can restore all the records from the target table that are covered by the selection graph S, by applying the following SQL query on table I_S:
select distinct T_0.primary_key **from** I_S.

The sufficient table I_S stores all the records corresponding to the selection graph S, i.e., all the records satisfying the constraints imposed by S, even though these constrains are not explicit anymore.

Let R be a refinement of the selection graph S, and $R(S)$ a new selection graph resulting from refining S with R. The procedure for obtaining the sufficient table for $R(S)$ given I_S is shown in Figure 8.

The sufficient table for a selection graph contains all the information necessary to obtain the database objects that are covered by the selection graph and any of its refinements.

Proposition 1 *Given a selection graph S, its sufficient table I_S, and a refinement R, the table constructed by REFINEMENT_SUF_TABLE(I_S, R) will contain the same records as the table constructed by SUF_TABLE($R(S)$)*

Proof sketch: Selection graph can be viewed as multi-relational pattern consisting of two subpatterns. The one that corresponds to all the *open, front* nodes in the selection graph, and the complement one. Let's denote the first subpattern as EXPLICIT subpattern, and the latter as IMPLICIT subpattern. The sufficient table contains the information about the EXPLICIT subpattern. Information about IMPLICIT subpattern is hidden in the sufficient table. It is important to note though, that objects stored in sufficient table still match IMPLICIT subpattern. Refinements can be applied only to the *open*, *front* nodes. Let's consider applying either 'add positive condition' refinement or 'add present edge' refinement. The resulting refined selection graph consists of extended (refined) EXPLICIT subpattern and unchanged IMPLICIT subpattern. This means that applying the refinement only to the sufficient table (as it is done in Figure 8) will result in objects matching to the extended EXPLICIT subpattern and inherently matching to the IMPLICIT subpattern, which means that they are matching to the refined selection graph. Similar

```
REFINEMENT_SUF_TABLE(I_S, R)
Input Sufficient table I_S for selection graph S,
refinement R
Output SQL query for sufficient table for R(S)
    table_list := 'I'_S
    condition_list := ''
    join_list := ''
    primary_key_list := primary_keys(I_S)
    if R == add positive condition, c, in table T_i
        table_list += 'T'_i
        condition_list += 'T_i.c'
        join_list += T_i.ID+' ='+I_S.T_i_ID
    else if R == add negative condition, c, in table T_i
        condition_list += T_0.ID + 'is not in
            ( select distinct' + I_S.T_0_ID +
            ' from' + I_S, T_i +
            ' where' + T_i.c + 'and' + T_i.ID +
              ' =' +I_S.T_i_ID+')'
    else if R = add present edge, e, from T_i to T_j
        table_list += T_i+','+T_j
        join_list += T_i.ID+' =' +I_S.T_i_ID+
            ' and ' + e.a
        primary_key_list += T_j.ID
    else if R == add closed edge, e from T_i to T_j
        condition_list += T_0.ID + 'is not in
        ( select distinct' + I_S.T_0_ID +
        ' from ' + I_S+','+T_i+','+T_j +
        ' where ' + T_i.ID+' =' +I_S.T_i_ID+
            ' and ' + e.a+')'
    return 'create table I_R as ' +
        '(select ' + primary_key_list +
        ' from ' + table_list +
        ' where ' + join_list +
        ' and ' + condition_list + ')'
```

Figure 8: Algorithm for generating SQL query corresponding to sufficient table $I_{R(S)}$

argument can be used for the case of other refinements. ∎

Note that REFINEMENT_SUF_TABLE procedure always returns a query of the constant size, i.e. the number of tables that need to be joint and the number of conditions that need to be applied is constant, which means that the time needed for executing this query doesn't increase with the size of the selection graph. On the other hand, the time needed for the execution of the TRANSLATE(S) function increases considerably with the size of the selection graph.

The above discussion can be extended to the look-ahead refinements, since they are a sequence of two refinements.

4 Experimental Results

We illustrate how the proposed approach can speed up a multi-relational data mining algorithm by considering multi-relational decision tree learning (MRDTL) algorithm, which constructs a decision tree for classifying a target attribute from a target table in a given database.

This algorithm proposed in [Knobbe *et al.*, 1999b] and implemented in [Leiva, 2002] is an extension of the logical decision tree induction algorithm called TILDE proposed by [Blockeel, 1998]. Essentially, MRDTL, like the propositional version of the decision tree algorithm [Quinlan, 1993], adds decision nodes to the tree through a process of successive refinement until some termination criterion is met (e.g., correct classification of instances in the training set). The choice of the decision node to be added at each step is guided by a suitable impurity measure (e.g., information gain). MRDTL starts with the selection graph containing a single node at the root of the tree, which represents the set of all objects of interest in the relational database. This node corresponds to the target table T_0. The algorithm iteratively considers every possible refinement that can be made to the current pattern (selection graph) S with respect to the database D and selects, in a greedy fashion, the optimal refinement (i.e., the one that maximizes information gain) and its complement.

Each candidate refinement is evaluated in terms of the split of the data induced by it with respect to the target attribute, as in the case of the propositional version of the decision tree learning algorithm [Quinlan, 1993]. Splits based on numerical attributes are handled using a technique similar to that of C4.5 algorithm [Quinlan, 1993] with modifications proposed in [Fayyad and Irani, 1992; Quinlan, 1996].

The hypothesis resulting from the induction of the relational decision tree algorithm described above can be viewed as a set of SQL queries associated with the selection graphs that correspond to the leaves of the decision tree. Each selection graph (query) has a class label associated with it. If the corresponding node is not a pure node, (i.e., it misclassifies some of the training instances that match the query), the label associated with the node can be based on the classification of the majority of training instances that match the corresponding selection graph. Alternatively, we can use probabilistic assignment of labels based on the distribution of class labels among the training instances that match the corresponding selection graph. The complementary nature of the different branches of a decision tree ensures that a given instance will not be assigned conflicting labels. It is also worth noting that it is not necessary to traverse the entire tree in order to classify a new instance; all the constraints on a certain path are stored in the selection graph associated with the corresponding leaf node. Instances that do not match the selection graphs associated with any of the leaf nodes in the tree are assigned unknown label and are counted as incorrectly classified when evaluating the accuracy of the tree on test data.

We have implemented MRDTL in Java using Oracle relational database and tested it on different databases. We have also implemented the speedup scheme for this algorithm. The resulting algorithm is shown in Figure 9.

We conducted our experiments on the data for prediction gene localization from KDD Cup 2001 [Cheng *et al.*, 2002]. Our current implementation of MRDTL assumes that the target table has a primary key, therefore it was necessary to normalize one of the initial tables given in this task. This normalization was achieved by creating tables named GENE, INTERACTION, and COMPOSITION as shown in Figure 1. For the gene/protein localization task, the target table is

Tree_Induction(D, S, I_S)
Input Database D, selection graph S, sufficient table I_S
Output The root of the tree, T

ALL := all_refinements(S)
R := optimal_refinement(I_S, D, ALL)
if stopping_criteria(I_S)
  return leaf
else
  $T_{left} := Tree\_Induction(D, R(S), R(I_S))$
  $T_{right} := Tree\_Induction(D, \bar{R}(S), \bar{R}(I_S))$
  return node(T_{left}, T_{right}, R)

Figure 9: MRDTL algorithm with speed up

	o_r_min	o_r_max	o_r_all	all
WOSU	0.04	70.642	3838.512	4764.15
WSU	0.00	3.656	65.241	416.74

Table 1: Experimental results. Here o_r_min denotes the shortest running times (in seconds) spent by the algorithm on a single call of *optimal_refinement* procedure, o_r_max denotes the longest running times (in seconds) spent by the algorithm on a single call of *optimal_refinement* procedure, o_r_all denotes the running time (in seconds) spent by the algorithm on all calls of the *optimal_refinement* procedure, all denotes the overall running time (in seconds) of the algorithm, WOSU denotes the results for the run of the algorithm without speed up scheme implemented, and WSU denotes the results for the run of the algorithm with speed up scheme implemented.

GENE and the target attribute is LOCALIZATION. The resulting training set consists of 862 genes and the test set consists of 381 genes. We constructed a classifier using all the training data and test the resulting classifier on the test set.

We have recorded the running times of the algorithm with and without speedup scheme proposed in the paper. We also measured the amount of time spent on the function *optimal_refinement*.

Experimental results are shown in Table 1, where o_r_min denotes the shortest running times (in seconds) spent by the algorithm on a single call of *optimal_refinement* procedure, o_r_max denotes the longest running times (in seconds) spent by the algorithm on a single call of *optimal_refinement* procedure, o_r_all denotes the running time (in seconds) spent by the algorithm on all calls of the *optimal_refinement* procedure, all denotes the overall running time (in seconds) of the algorithm, WOSU denotes the results for the run of the algorithm without speed up scheme implemented, and WSU denotes the results for the run of the algorithm with speed up scheme implemented.

The overall running time spent on querying the database in training phase was decreased by a factor of around 59. The running time improvement by a factor of 11 was observed in the overall running time for the MRDTL algorithm on this database. Some calls of *optimal_refinement* procedure had running time improvement up to a factor of 1000.

5 Conclusion

In this paper we present a general approach to speeding up a class of multi-relational data mining algorithms. We have incorporated the proposed method into MRDTL algorithm. Preliminary results of our experiments have shown that the proposed method yields one to two orders of magnitude reductions in the running time of the algorithm. The proposed modifications make it feasible to apply multi-relational data mining algorithms to significantly larger relational databases. Our work in progress is aimed at:

- Incorporation of sophisticated methods for handling missing attribute values into MRDTL
- Incorporation of sophisticated pruning methods or complexity regularization techniques into MRDTL to minimize overfitting and improve generalization
- More extensive experimental evaluation of MRDTL on real-world data sets
- Development of ontology-guided multi-relational decision tree learning algorithms to generate classifiers at multiple levels of abstraction (based on the recently developed prepositional decision tree counterparts of such algorithms [Zhang *et al.*, 2002]
- Development of variants of MRDTL for classification tasks where the classes are not disjoint, based on the recently developed propositional decision tree counterparts of such algorithms [Caragea *et al.*, in preparation]
- Development of variants of MRDTL that can learn from heterogeneous, distributed, autonomous data sources based on recently developed techniques for distributed learning [Caragea *et al.*, 2001b; 2001a] and ontology-based data integration [Honavar et al., 2001; Honavar et al., 2002; Reinoso-Castillo, 2002].
- Application of multi-relational data mining algorithms to data-driven knowledge discovery problems in bioinformatics and computational biology.

6 Acknowledgements

This research was supported in part by a grant from the National Science Foundation (NSF ITR 021969) and a research assistantship funded by the Iowa State University Graduate College. The paper has benefited from discussions with Hector Leiva and Doina Caragea of the Iowa State University Artificial Intelligence Research Laboratory.

References

[Blockeel, 1998] Hendrik Blockeel. *Top-down induction of first order logical decision trees*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1998.

[Caragea *et al.*, 2001a] D. Caragea, A. Silvescu, and V. Honavar. Decision tree learning from distributed data. Technical Report TR, Iowa State University, Ames, IA, 2001.

[Caragea *et al.*, 2001b] D. Caragea, A. Silvescu, and V. Honavar. *Invited Chapter. Toward a Theoretical Framework for Analysis and Synthesis of Agents That Learn from Distributed Dynamic Data Sources*. Berlin: Springer-Verlag, 2001.

[Caragea *et al.*, in preparation] D. Caragea, A. Silvescu, and V. Honavar. Learning decision tree classifiers when the classes are not disjoint, in preparation.

[Cheng *et al.*, 2002] J. Cheng, M. Krogel, J. Sese, Hatzis C., S. Morishita, H. Hayashi, and D. Page. Kdd cup 2001 report. In *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations*, volume 3, 2002.

[Dehaspe and Raedt, 1997] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 125–132. Springer-Verlag, 1997.

[Dzeroski and Lavrac, 2001] S. Dzeroski and N Lavrac. Relational data mining. Springer-Verlag, 2001.

[Fayyad and Irani, 1992] U. M. Fayyad and K. B Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8, 1992.

[Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, , and Pfeffer. Learning probabilistic relational models. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*. Morgan Kaufman, 1999.

[Getoor, 2001] L Getoor. Multi-relational data mining using probabilistic relational models: research summary. In *Proceedings of the First Workshop in Multi-relational Data Mining*, 2001.

[Jaeger, 1997] M Jaeger. Relational bayesian networks. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-1997)*, 1997.

[Karalic and Bratko, 1997] Karalic and Bratko. First order regression. *Machine Learning 26*, 1997.

[Kersting and De Raedt, 2000] K. Kersting and L. De Raedt. Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 2000.

[Knobbe *et al.*, 1999a] J. Knobbe, H. Blockeel, A. Siebes, and Van der Wallen D. Multi-relational data mining. In *Proceedings of Benelearn 99*, 1999.

[Knobbe *et al.*, 1999b] J. Knobbe, H. Blockeel, A. Siebes, and Van der Wallen D. Multi-relational decision tree induction. In *Proceedings of the 3rdEuropean Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD 99*, 1999.

[Koller, 1999] D Koller. Probabilistic relational models. In S. Dzeroski and P. Flach, editors, *Proceedings of 9th International Workshop on Inductive Logic Programming (ILP-99)*. Springer, 1999.

[Krogel and Wrobel, 2001] M. Krogel and S. Wrobel. Transformation-based learning using multirelational aggregation. In Celine Rouveirol and Michele Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[Leiva, 2002] Hector Ariel Leiva. A multi-relational decision tree learning algorithm. *M.S. thesis. Deparment of Computer Science. Iowa State University*, 2002.

[Pfeffer, 2000] A. Pfeffer. A bayesian language for cumulative learning. In *Proceedings of AAAI 2000 Workshop on Learning Statististical Models from Relational Data*. AAAI Press, 2000.

[Quinlan, 1993] R. Quinlan. *C4.5: Programs for Machine Learning*. 1993.

[Quinlan, 1996] R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4, 1996.

[Reinoso-Castillo, 2002] Jaime Reinoso-Castillo. Ontology-driven information extraction and integration fromheterogeneous distributed autonomous data sources. *M.S. Thesis. Department of Computer Science. Iowa State University*, 2002.

[Zhang *et al.*, 2002] J. Zhang, A. Silvescu, , and V. Honavar. Ontology-driven induction of decision trees at multiple levels of abstraction. In *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation (SARA-2002)*, Kananaskis, Alberta, Canada,, 2002.