Multi-Relational Decision Tree Induction

Arno J. Knobbe
Syllogic B.V.
Hoefseweg 1
3821 AE Amersfoort
The Netherlands
a.knobbe@syllogic.com

Arno Siebes
CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands
arno@cwi.nl

Daniël van der Wallen
Syllogic B.V.
Hoefseweg 1
3821 AE Amersfoort
The Netherlands
d.van.der.wallen@syllogic.com

Abstract

Discovering decision trees is an important set of techniques in KDD, both because of their simple interpretation and the efficiency of their discovery. One of their disadvantages is that they do not take the structure of the mining object into account. By going from the standard single-relation approach to the multi-relational approach as in ILP this disadvantage is removed. However, the straightforward generalization loses the efficiency of the standard algorithms. In this paper we present a framework that allows the efficient discovery of multi-relational decision trees through the exploitation of the domain knowledge encoded in the data model of the database.

Introduction

The induction of decision trees has been getting a lot of attention in the field of Knowledge Discovery in Databases over the past few years. This popularity has been largely due to the efficiency with which decision trees can be induced from large datasets, as well as to the elegant and intuitive representation of the knowledge that is discovered. However, traditional decision tree approaches have one major drawback. Because of their propositional nature, they can not be employed to analyse relational databases containing multiple tables. Such databases can be used to describe objects with some internal structure, which may differ from one object to another. The means to describe groups of such objects in terms of occurrence of a certain substructure are simply not available in propositional (attribute-value) decision trees.

This limitation of propositional decision trees has been overcome by an algorithm called Tilde [2, 3]. The approach taken there is to go from propositional to first order representations of objects, and to use first order logic to represent decisions in the tree. The trees that this algorithm produces are called first order logical decision trees. Because of the richer formalism that is used, first order logical decision trees do allow the representation of patterns within a set of structured objects. However, Tilde assumes that objects are represented in first order logic rather than as collections of records in a relational database, and thus does not benefit from the opportunities for optimisation that are provided by a relational representation. In this paper, we present an alternative approach that does provide the means to induce decision trees from structural information. We call such decision trees multi-relational decision trees, in line with a previously proposed multi-relational data mining framework [8, 9].

In order to be able to induce decision trees from a large relational database efficiently, we need a framework with the following characteristics:

1. Both attribute-value and structural information are included in the analysis.
2. The search space is drastically pruned by using the constraints that are available in the data model. This means that we are considering only the structural information that is intended by the design of the database, and we are not wasting time on potentially large numbers of conceptually invalid patterns.
3. The concepts of negation and complementary sets of objects are representable. Decision trees recursively divide the data set up into complementary sets of objects. It is necessary that both the positive split, as well as the complement of that, can effectively be represented.
4. Efficiency is achieved by a collection of primitives that can be used to summarise both attribute-value and structural information. The use of data mining primitives, an effective and accepted means of achieving efficiency in many propositional approaches, should be generalised in order to work with the new structural information.
5. The framework can be implemented by a dedicated client/server architecture. This requires that a clear separation between search process and data processing can be made. This enables the data processing part (usually the main computational bottleneck) to be implemented on a scalable server.

Only two of these requirements are met by algorithms for inducing first order logical decision trees, as described in [2, 3]. Specifically the items 1. and 3. are addressed by this approach, but little attention has been giving to efficient implementations. The concepts addressed in item 3. are partially solved by representing the whole decision tree as a decision list in Prolog that depends heavily on the order of clauses and the use of cuts. By doing so, the problem of representing individual patterns associated with internal nodes or leafs of the tree is circumvented. Our previous work on a multi-relational data mining framework, described in [8, 9], covers four of these five characteristics. The problem of handling negation and complementary sets of objects (item 3.) has not been addressed, and will be considered in more detail in this paper.

Propositional decision trees typically divide some set of objects into two complementary subsets, in a recursive manner [15]. This decision is made by applying a simple propositional condition to the current set of objects. One branch of the tree represents the set of objects that adhere to the conjunction of the original list of conditions and the new condition, whereas the other branch selects the set of objects for which the conjunction of the original list of conditions and the negation of the new condition holds. This demonstrates that in an attribute-value environment complementary patterns are produced by simply negating the additional condition. In a multi-relational environment, producing such a complement is less trivial. Say we are considering people and the structure of their household, for example. If we have established that the set of people who have at least one child is interesting, we might want to split this set into the set of people who have a son, and the complement of that set. Clearly, this complement is not equal to the set of people who have a daughter, as this does not exclude people who have both sons and daughters. Rather, the complement is equal to the set of people who do have a child, but for whom none of the children are male.

The outline of this paper is as follows. In **Multi-Relational Data Mining** we introduce the basic concepts involved with knowledge discovery in a relational database. The issue of representing patterns within such databases is addressed in **Multi-Relational Patterns**. The extended graphical language for describing such patterns is defined in **Selection Graphs**. How these selection graphs support the induction of decision trees is described in **Multi-Relational Decision Trees**, where we present a generic algorithm for multi-relational decision tree induction. We end with some conclusions and future work in **Conclusion**.

Multi-relational data mining

The attribute-value paradigm, which includes many common Data Mining algorithms, only allows the analysis of fairly simple objects. It requires that each object can be described by a fixed set of attributes each of which can only have a single (unstructured) value. To be able to represent more complex and structured objects, one has to employ a relational database containing multiple tables. Each object can be described by multiple records in multiple tables. It is clear that in order to be able to handle the structural information that is represented by these records, changes will have to be made to existing algorithms.

We will assume that the data to be analysed is stored in a relational database [4, 17]. A relational database consists of a set of tables and a set of associations (i.e. constraints) between pairs of tables describing how records in one table relate to records in another table. Both tables and associations are also known as relations, so we will use the former terminology to be able to distinguish between the two concepts. An association between two tables describes the relationships between records in both tables. The nature of this relationship is characterised by the *multiplicity* of the association. The multiplicity of an association determines whether several records in one table relate to single or multiple records in the second table. Also, the multiplicity determines whether every record in one table needs to have at least one corresponding record in the second table. More formally, we define the following two predicates related to the multiplicity of an association A between two tables P and Q.

Definition 1 $Multiple(A, P)$ iff every record in Q may correspond to multiple records in P.

Definition 2 $Zero(A, P)$ iff a record in Q may have no corresponding record in P.

Note that these predicates are defined both for P and Q, and both predicates may or may not hold for either P or Q. This way there are 16 different possible multiplicities for association A.

A special case of an association between two tables is a *foreign key relation*. A foreign key relation from a foreign key in table P to a primary key in table Q is an association for which the following hold:

$$Multiple(A, P),\ Zero(A, P),\ \text{not}(Multiple(A, Q)),\ \text{not}(Zero(A, Q))$$

Most associations in a physical data model will be foreign key relations.

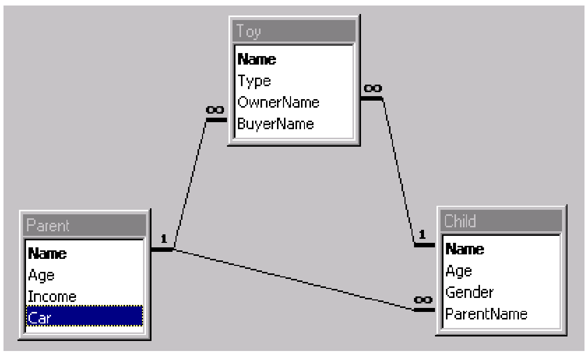


Figure 1 The data model of our example database.

Example 1 Figure 1 shows an example of a data model that describes parents, children and toys, as well as how each of these relate to each other. We will be referring to this example throughout this paper. The data model shows that parents may have zero or more children, children may have zero or more toys, and parents may have bought zero or more toys. Note that toys owned by particular child may not necessarily have been bought by their parents. They can be presents from other parents. Also note that children have one parent (for simplicity).

Even though the data model consists of multiple tables, there is still only a single kind of objects that is central to the analysis. You can choose the kind of objects you want to analyse, by selecting one of the tables as the *target table*. Each record in the target table, which we will refer to as t_0, will now correspond to a single object in the database. Any information pertaining to the object which is stored in other tables can be looked up by following the associations in the data model. If the data mining algorithm requires a particular feature of an object to be used as a dependent attribute for classification or regression, we can define a particular *target attribute* within the target table.

Example 2 In figure 1, the highlighting of the attribute *Car* in the *Parent* table indicates that *Parent* is the target table and *Car* is the attribute of interest, the target attribute. This means that our primary interest is in the parents, and that we will be considering different sets of parents in order to come up with good indicators for classifying parents as car owners or not.

The idea of mining from multiple tables is not a new one. It is being studied extensively in the field of Inductive Logic Programming (ILP) [6, 11]. However, these approaches are mostly based on data stored as Prolog programs, and little attention has been given to data stored in relational database and to how knowledge of the data model can help to guide the search process [1, 12, 19]. Nor has a lot of attention been given to efficiency and scalability issues. Conceptually, however, there are many parallels between ILP and multi-relational data mining. Where ILP can be seen as learning from a set of predicates, multi-relational data mining can be seen as learning from a relational database.

Extensional predicates, i.e. those predicates for which only ground facts exist, are the counterparts of tables in a relational database. Intentional predicates, i.e. those predicates for which also rules are given, correspond to the concept of views. However, multi-relational data mining differs from ILP in three aspects. Firstly, it is restricted to the discovery of non-recursive patterns. Secondly, the semantic information in the database is exploited explicitly. Thirdly, the emphasis on database primitives ensures efficiency.

Multi-relational patterns

In our search for knowledge in relational databases we want to consider not only attribute-value descriptions, as is common in traditional algorithms, but also the structural information which is available through the associations between tables. We will refer to descriptions of certain features of multi-relational objects as multi-relational patterns. We can look at multi-relational patterns as small pieces of substructure which we wish to encounter in the structure of the objects we are considering. For example, if we are considering a database of chemical compounds, a multi-relational pattern might describe a subgroup of the compounds by listing some of the elements and bonds between them [5]. In the context of occurrence of a particular substructure within a multi-relational object, we introduce the following definition.

Definition (cover) 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.

An alternative view on multi-relational patterns is that of selections within the relational database. A multi-relational pattern implies a subset of the available multi-relational objects, and a range of multi-relational patterns might be considered, in order to examine the interestingness of the associated subset. The interestingness of the subset may be defined in terms of its size, or in terms of the distribution of some dependent attribute, for classification for example. With respect to the size of the associated subset we define the following.

Definition (support, frequent) The *support* of a multi-relational pattern within a relational database is the number of objects within this database that are covered by the multi-relational pattern. Patterns with a large support, usually above some predefined threshold, will be referred to as *frequent*.

As was explained in [8, 9], we view multi-relational data mining as the search for interesting multi-relational patterns. The multi-relational data mining framework allows many different search paradigms, each of which are multi-relational generalisations of well-known attribute-value search algorithms. Within the search paradigms we can distinguish two approaches:

- searching for local models (nuggets). This approach involves searching for interesting patterns which are good descriptions of some subset of the whole database. Each pattern may be part of a rule which only makes a statement about the elements of this subset, i.e. which is local model. Usually there is no relation between each of the discovered local models. This means that the local models do not necessarily cover the complete database, may be overlapping, and may even be contradictory.

- Searching for global models. This approach involves searching for sets of interesting patterns which together form a complete description or explanation of the whole database. The resulting patterns are strongly related, and are usually organised in a decision tree of some sorts. This means that the whole database is progressively divided into complementary and non-overlapping subsets. We saw in previous sections that being able to express such complementary patterns is essential and nontrivial.

Both approaches towards knowledge discovery are elegantly supported by a top-down approach [2, 3, 5, 7, 8, 9, 13, 15, 19]. Some examples of a top-down approach to local model searching in the context of multi-relational data mining are given in [5, 8, 9, 13, 19]. Similarly, the induction of tree like structures from relational databases is introduced in [2, 3, 15]. Each of these top-down approaches share the idea of a refinement operator. Whenever a promising pattern is discovered, a list of refinements will be examined. When we speak about refinement of a multi-relational pattern, we are referring to an extension of the actual description of the pattern, which results in a new selection of objects which is a subset of the selection associated with the original multi-relational pattern. Recursively applying such refinement operators to promising patterns results in a top-down algorithm which zooms in on interesting subsets of the database.

Taking into account the above discussion of multi-relational data mining and top-down approaches, we can formulate the following requirements for a multi-relational pattern language. In the following section we will define a language which satisfies these requirements. Descriptions of multi-relational patterns should

- reflect the structure of the relational model. Sticking close to the structure of the relational model allows for easier understanding of the pattern. Also, it is easier to enforce that the pattern makes sense given the referential constraints in the data model. On an implementation level, it is easier to check the coverage of an object if the underlying data structure corresponds to the structure of a pattern.
- be intuitive. Structural or first order logical descriptions, notably those involving lots of structural parts and negation, are notoriously hard to interpret by human beings. Any step possible to improve human understanding should be taken.
- support atomic, local refinements. Because refinements produce subsets which are somewhat similar to the original set, we prefer the actual descriptions, i.e. the multi-relational patterns, to be similar to the original pattern. This means that any refinement operator should preferably be some small change rather than the complete re-writing of the original pattern.
- allow refinements which are complementary. This means that if there is a particular refinement to a multi-relational pattern which produces a certain subset, there should also be a complementary refinement which produces the complementary subset.

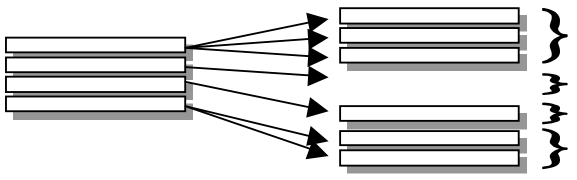
Selection graphs

In order to describe the constraints related to a multi-relational pattern, we introduce the concept of *selection graphs*:

Definition 4 A *selection graph G* is a directed graph (*N*, *E*), where *N* is a set of triples (*t*, *C*, *s*), *t* is a table in the data model and *C* is a, possibly empty, set of conditions on attributes in *t* of type *t.a operator c*; the *operator* is one of the usual selection operators, =, > etc. *s* is a flag with possible values *open* and *closed*.
E is a set of tuples (*p*, *q*, *a*, *e*) called *selection edges*, where *p* and *q* are selection nodes and *a* is an association between *p.t* and *q.t* in the data model. *e* is a flag with possible values *present* and *absent*. The selection graph contains at least one node n_0 that corresponds to the target table t_0.

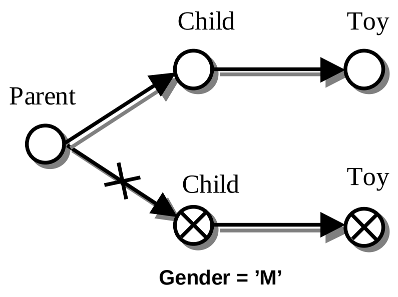
Selection graphs can be represented graphically as labelled directed graphs. The value of *s* is indicated by the absence or presence of a cross in the node, representing the value *open* and *closed* respectively. The value of *e* is indicated by the absence or presence of a cross on the arrow, representing the value *present* and *absent* respectively.

Intuitively, selection graphs can be interpreted as follows. Every edge between *p* and *q*, together with the associated conditions *q.C*, imposes some constraints on how multiple records in table *q.t* relate to a single record in table *p.t*. The association between *p.t* and *q.t* imposes some grouping on the records in *q.t*, and the combination of edge and conditions selects some of these groups in *q.t*, and thus selects some records in *p.t*. Multiple edges coming from one node simply indicate multiple conditions on the records belonging to that node. Working with statements about groups of records is exactly what makes multi-relational data mining more powerful than propositional approaches. Such a grouping is demonstrated below.



A *present* edge combined with a list of conditions selects those groups for which there is at least one record that respects the list of conditions. An *absent* edge combined with a list of conditions selects only those groups for which there is not a single record that respects the list of conditions. The selection associated with any subgraph is the combined result of all such individual constraints within the subgraph on groups of records. This means that any subgraph that is pointed to by an *absent* edge should be considered as a joint set of negative conditions. The flag *s* associated with nodes of the selection graph has no effect on the selection. Rather, it is used to indicate whether a particular node in a selection graph is a candidate for refinement.

Example 3 The following selection graph selects those parents that have at least one child with a toy, but for whom none of these children are male:



The following algorithm shows how selection graph can be translated to SQL. The translation provides a formal semantics for selection graphs. Resulting SQL statements will not be used in actual implementations, as special-purpose calls in a dedicated architecture can be processed far more efficiently. The algorithm will produce a list of tables *table_list*, a list of join conditions *join_list*, and a list of conditions *condition_list*, and combine these to produce an SQL-statement. The algorithm effectively produces a join of all the tables associated with an *open* node. For each subgraph attached to an *absent* edge, a subquery is produced by calling the *translate_subgraph* procedure. The fact that we state `select distinct` in the main query rather than `select` is caused by the fact that a record in t_0 may be covered by the selection graph in multiple ways. Since the target table represents our objects, they should be counted only once.

```
translate(S : Selection Graph)

        table_list := ''
        condition_list := ''
        join_list := ''
        for each node i in S do
                if (i.s = 'open')
                        table_list.add(i.table_name + ' T' + i)
                        for each condition c in i do
                                condition_list.add('T' + i + '.' + c)
        for each edge j in S do
                if (j.e = 'present')
                        if (j.right_node.s = 'open')
                                join_list.add(
                                        j.left_node + '.' +
                                        j.left_attribute + ' = ' +
                                        j.right_node + '.' +
                                        j.right_attribute)
                else
                        join_list.add(
                                j.left_node + '.' +
                                j.left_attribute + ' not in ' +
                                translate_subgraph(subgraph(S, j.right_node), j.right_attribute))
        return 'select distinct' +
                ' T0.' + n_0.primary_key +
                ' from ' + table_list +
                ' where ' + join_list +
                ' and ' + condition_list
```

```
translate_subgraph(S : Selection Graph, K : Key)

    table_list := ''
    condition_list := ''
    join_list := ''
    for each node i in S do
        table_list.add(i.table_name + ' T' + i)
        for each condition c in i do
            condition_list.add( 'T' + i + '.' + c)
    for each edge j in S do
        join_list.add(
            j.left_node + '.' +
            j.left_attribute + ' = ' +
            j.right_node + '.' +
            j.right_attribute)
    return 'select ' +
        ' T0.' + K +
        ' from ' + table_list +
        ' where ' + join_list +
        ' and ' + condition_list
```

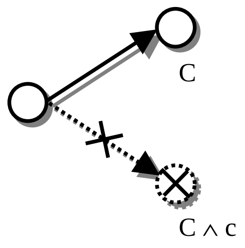
Example 4 The algorithm *translate* will produce the following SQL statement for the selection graph given in example 3:

```
select distinct T0.Name
from Parent T0, Child T1, Toy T2
where T0.Name = T1.ParentName and T1.Name = T2.OwnerName
and T0.Name not in
      (select T3.ParentName
       from Child T3, Toy T4
       where T3.Name = T4.OwnerName and T3.Gender = 'M')
```

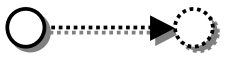
Refinements As was described earlier, the selection graphs will be used to represent sets of objects belonging to nodes or leafs in a decision tree. Whenever a new split is introduced in the decision tree, we are in fact refining the current selection graph in two ways. We will be using the following refinement operators of a selection graph *G* as potential splits in the multi-relational decision tree. The refinements are introduced in pairs of complimentary operations:

- add positive condition. This refinement will simply add a condition to a selection node in G without actually changing the structure of *G*.

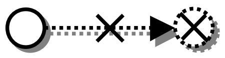
- add negative condition. In case the node which is refined does not represents the target table, this refinement will introduce a new *absent* edge from the parent of the selection node in question. The condition list of the selection node will be copied to the new *closed* node, and will be extended by the new condition. 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.



- add *present* edge and *open* node. This refinement will instantiate an associations in the data model as a *present* edge together with its corresponding table and add these to *G*.



- add *absent* edge and *closed* node. This refinement will instantiate an associations in the data model as an *absent* edge together with its corresponding table and add these to *G*.



This set of refinements is the main source of efficiency for our multi-relational data mining framework. By only allowing the addition of edges when consistent with the data model, unnecessary and invalid patterns will be pruned from the search space. The four refinements exploit the existence of associations between tables for optimisation. Apart from using information about the existence of associations, we also use the multiplicity of available associations in order to further prune the search space. This is explained in more detail in [8, 9].

Multi-relational decision trees

The induction of decision trees in first order logic has been studied by several researchers [2, 3, 10, 18]. Each of these approaches share a common Divide and Conquer strategy, but produce different flavours of decision trees. For example [10] discusses the induction of regression trees, whereas [3] discusses the induction of decision trees for clustering. In [2] an overview is given of potential uses of decision trees within a single framework. However, these papers have largely focused on induction-parameters such as the choice of splitting criterion or stopping criterion. None of these papers provide a good solution for the representation of patterns associated with the leaves and internal nodes of the decision tree. In this section we give a generic algorithm for the top-down induction of multi-relational decision trees within the multi-relational data mining framework. It illustrates the use of selection graphs, and specifically the use of complementary selection graphs in the two branches of a split.

Top-down induction of decision trees is basically a Divide and Conquer algorithm. The algorithm starts with a single node at the root of the tree which represents the set of all objects in the relational database. By analysing all possible refinements of the empty selection graph, and examining their quality by applying some interestingness measure, we determine the optimal refinement. This optimal refinement, together with its complement, are used to create the patterns associated with the left and the right branch respectively. Based on the stopping criterion it may turn out that the optimal refinement and its complement do not give cause for further splitting, a leaf node is introduced instead. Whenever the optimal refinement does provide a good split, a left and right branch are introduced and the procedure is applied to each of these recursively.

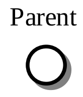
```
build_tree($T$ : tree, $D$ : database, $P$ : pattern)

$R$ := optimal_refinement($P$, $D$)
**if** stopping_criterion($R$)
        $T$ := leaf($P$)
**else**
        $P_{left}$ := $R(P)$
        $P_{right}$ := $R_{compl}(P)$
        build_tree(left, $D$, $P_{left}$)
        build_tree(right, $D$, $P_{right}$)
        $T$ := node(left, right, $R$)
```

The function *optimal_refinement* takes the current pattern *P* and considers every possible generic positive refinement with respect to the data model of *D*. For each of these generic refinements a multi-relational data mining primitive call is sent to the server that handles the data, and the necessary statistics are retrieved. The statistics describe the support of all possible refinements derived from the current generic refinement. However, these counts also imply the support of the complementary negative refinements. These two sets of counts are used to compute the interestingness measure of choice. This process is repeated for every possible generic positive refinement and the optimal refinement is reported. Which refinements are candidates is governed by the structure of the data model of *D*, and notably the multiplicity of the associations within this data model, as is explained in more detail in [8, 9].

The function *stopping_criterion* determines whether the optimal refinement leads to a good split based on the statistics associated with the optimal refinement and its complement. A range of stopping criteria can be used, depending on the induction paradigm (classification, clustering, regression, etc.), but the actual choice is immaterial to the present discussion.

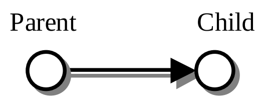
Example In order to illustrate the workings of the above algorithm we apply it to a classification problem within our example database shown in figure 1. Assume Parent is our target table and we wish to determine what influences the ownership of a car, i.e. the value of the attribute Car. We start with an empty selection graph which represents all parents in our database.

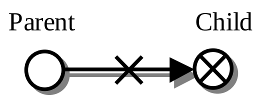


By taking into account the current selection graph and the available attributes and associations in the data model, we get the following list of possible generic positive refinements:

- add condition Parent.Age > x
- add condition Parent.Income > x
- add edge and node from Parent to Child
- add edge and node from Parent to Toy

Every refinement is tested and it turns out that having a child or not (the penultimate refinement) is the best indicator for owning a car or not. The following two complementary selection graphs are created for the left and the right branch respectively.

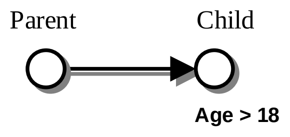




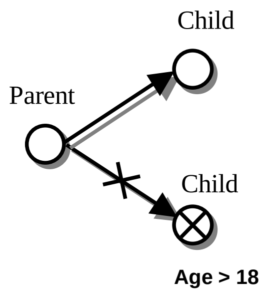
The induction process is now continued recursively for the set of parents who have a child, and for the set of parents who don't (strictly speaking not really parents) respectively. We will only demonstrate the effect for the left branch. At this point in the tree the previous list of refinements is still valid, and completed with the following extra refinements:

- add condition Child.Age > x
- add condition Child.Age < x
- add condition Child.Gender = female
- add condition Child.Gender = male
- add edge and node from Child to Toy

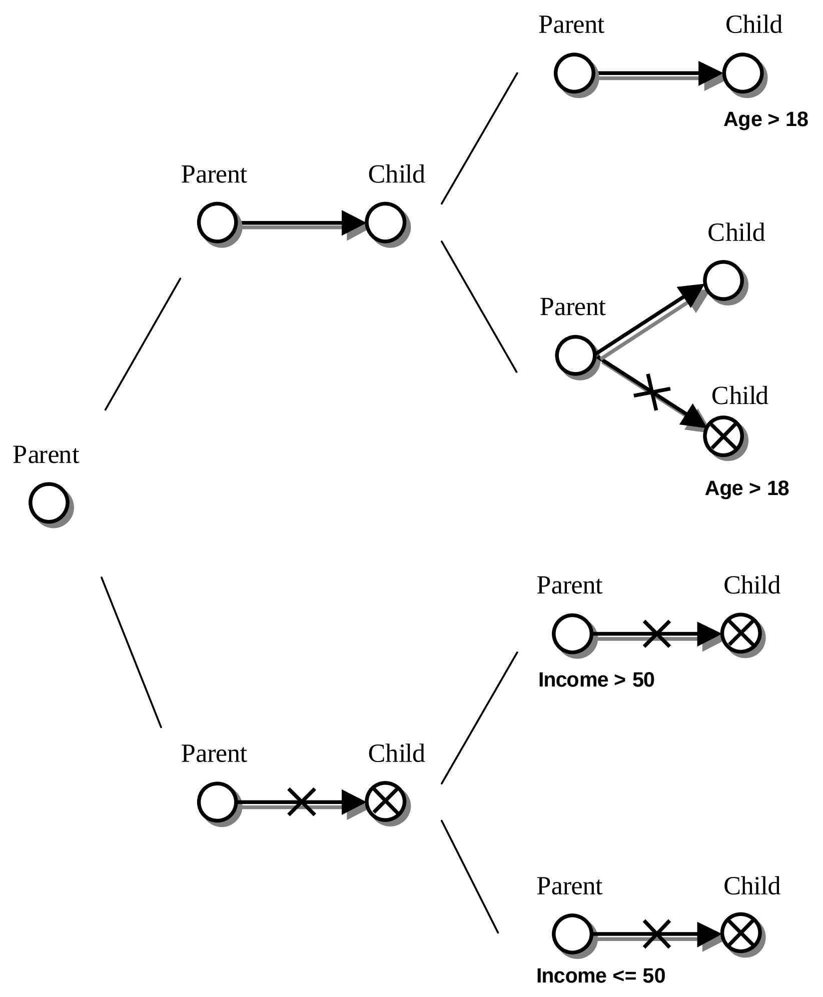
The same process of finding the optimal refinement is repeated, and this time a condition on the age of the child is introduced. The left branch will now represent the set of parents who have an adult child:



The right branch will represent the set of parents who do have a child, but for whom none of the children are adults:



The resulting decision tree will be as follows:



Conclusion

In this paper we have presented a framework that allows the efficient discovery of multi-relational decision trees. The main advantage above the standard decision tree algorithms is the gain in expressiveness. The main advantage above the ILP approach towards decision trees is the gain in efficiency achieved by exploiting the domain knowledge present in the data model of the database. One of the main remaining challenges is to extend this framework such that the selection graphs may contain cycles. Such an extension would, e.g., allow to refine into parents that have not bought their own children toys.

References

[1] Blockeel, H., De Raedt, L. *Relational knowledge discovery in databases,* In Stephen Muggleton, editor, Proceedings of the Sixth International Workshop on Inductive Logic Programming (ILP'96), 199-211, Volume 314 of Lecture Notes in Artificial Intelligence, Springer Verlag, 1996

[2] Blockeel, H., De Raedt, L. *Top-down induction of first order logical decision trees,* Artificial Intelligence 101 (1-2):285-297, June 1998

[3] Blockeel, H., De Raedt, L., Ramon, J. *Top-down induction of clustering trees,* In Proceedings of the 15th International Conference on Machine Learning (ICML'98), 55-63, 1998, `http://www.cs.kuleuven.ac.be/~ml/PS/ML98-56.ps`

[4] Date, C.J. *An Introduction to Database Systems,* Volume I, The Systems Programming Series, Addison-Wesley, 1986

[5] Dehaspe, L., Toivonen, H., King, R.D. *Finding frequent substructures in chemical compounds*, In Proceedings of the Fourth International Conference on Knowledge Discovery & Data Mining (KDD'98), 30-36, 1998

[6] Dzeroski, S. *Inductive Logic Programming and Knowledge Discovery in Databases,* Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996

[7] Knobbe, A.J. *Towards Scalable Industrial Implementations of ILP,* ILPNet2 Seminar on ILP & KDD, Caminha, Portugal, 1998

[8] Knobbe, A.J., Blockeel, H., Siebes, A., Van der Wallen, D.M.G. *Multi-Relational Data Mining,* submitted

[9] Knobbe, A.J., Blockeel, H., Siebes, A., Van der Wallen, D.M.G. *Multi-Relational Data Mining,* technical report CWI.

[10] Kramer, S. *Structural regression trees,* In Proceedings of the 13th National and Conference on Artificial Intelligence (AAAI'96) , 1996

[11] Lavrac, N., Dzeroski, S. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994

[12] Lindner, G., Morik, K. *Coupling a relational learning algorithm with a database system,* Workshop Notes of the MLNet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, 163-168, 1995

[13] Mannila, H., Toivonen, H. *On an algorithm for finding all interesting sentences*, In Proceedings of the 13th European Meeting on Cybernetics and Systems Research (EMCSR'96), 973-978, 1996

[14] Martin, L., Moal, F., Vrain, C. *A Relational Data Mining Tool Based on Genetic Programming,* In Proceedings PKDD '98, Nantes, France, 130-138, 1998

[15] Quinlan, R.J., *C4.5: Programs for Machine Learning,* Morgan Kaufmann, 1993

[16] Ribeiro, J.S., Kaufman, K.A., and Kerschberg, L. *Knowledge discovery from multiple databases,* In Proceedings of the First International Conference on Knowledge Discovery & Data Mining (KDD'95), 240-245, 1995

[17] Ullman, I.D. *Principles of Databases and Knowledge-Based Systems,* Volume I, Computer

Science Press, 1988

[18] Watanabe, L., Rendell, L. *Learning structural decision trees from examples*, In Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91), 770-776, 1991

[19] Wrobel, S. *An algorithm for multi-relational discovery of subgroups,* In Proceedings of Principles of Data Mining and Knowledge Discovery (PKDD'97), 78-87, 1997

[20] Yao, I., Lin, H. *Searching Multiple Databases for Interesting Complexes,* In Proceedings of the 1997 Pacific and Asia Conference on Knowledge Discovery and Data Mining: Techniques and Applications (PAKDD'97), 198-210, 1997