An Enhanced Block Notation for Discrimination Network Optimisation

Christoph Terwelp, Karl-Heinz Krempels and Fabian Ohler
Information Systems, RWTH Aachen University, Aachen, Germany

Keywords: Rule-based Systems, Discrimination Networks.

Abstract: Because of their ability to efficiently store, access, and process data, Database Management Systems (DBMSs) and Rule-based Systems (RBSs) are used in many information systems as information processing units. A basic function of a RBS and a function of many DBMSs is to match conditions on the available data. To improve performance, intermediate results are stored in Discrimination Networks (DNs). The resulting memory consumption and runtime cost depend on the structure of the DN. A lot of research has been done in the area of optimising DNs. In this paper, we focus on re-using network parts considering multiple rule conditions and exploiting the characteristics of equivalences. Hence, we present an approach incorporating the potential of both concepts as an enhancement to previous work.

1 INTRODUCTION

Because of their ability to efficiently store, access, and process data, Database Management Systems (DBMSs) and Rule-based Systems (RBSs) are used in many information systems as information processing units (Brownston et al., 1985; Forgy, 1981). A basic function of a RBS and a function of many DBMSs is to match conditions on the available data. Checking all data repeatedly every time some data changes performs badly. It is possible to improve performance by saving intermediate results in memory introducing the method of dynamic programming. A common example for this approach is the Discrimination Network (DN). Different DN optimization techniques are discussed in (Forgy, 1982), (Miranker, 1987), and (Hanson and Hasan, 1993). These approaches only address optimisations limited to single rules. Further improvement is possible by optimising the full rule set of a RBS. By exploiting the characteristics of equivalences, additional performance improvements are possible. In this paper, we will introduce an approach extending (Ohler and Terwelp, 2015) and (Ohler et al., 2016) incorporating the potential of both concepts.

This paper is organized as follows: In Section 2, we introduce DNs and in Section 3, we explain the concept of re-using network parts for different rules. Section 4 describes the potential of binding variables in rule conditions. In Section 5, we discuss the arising problems in the field of node sharing. Existing work in the area of DN and query optimisation are presented in Section 6. The identified problems are then addressed in Section 7 by introducing an advanced version of the block notation. Section 8 comprises the conclusion and gives an outlook on future work.

2 DISCRIMINATION NETWORKS

Rules in RBSs and DBMSs both comprise a condition and actions. The actions of a rule must only be executed, if the data in the system matches the condition of the rule. DNs are an efficient method of identifying rules to be executed employing dynamic programming trading memory consumption for runtime improvements. Rule conditions are split into their atomic (w. r. t. conjunction) sub-conditions. In the following, such sub-conditions are called filters.

DNs apply these filters successively joining only the required data. Intermediate results are saved to be reused in case of data changes. Each filter is represented by a node in the DN. Additionally, every node has a memory, at least one input, and one output. The memory of a node contains the data received via its inputs matching its filter. The output is used by successor nodes to access the memory and receive notifications about memory changes. Data changes are propagated through the network along the edges. The atomic data unit travelling through a DN is called fact. Changed data reaching a node is joined with the data saved in nodes connected to all other inputs of the node. So only the memories of affected nodes have to be adjusted. Each rule condition is represented by a terminal node collecting all data matching the complete rule condition.

data input nodes serve as entry points for specific types of data into the DN. They are represented as diamond shaped nodes.

filter nodes join the data from all their inputs and check if the results match their filters. They are represented as inverted triangle shaped nodes.

terminal nodes collect all data matching the conditions of the corresponding rules. They are represented as triangle shaped nodes. The action part of a rule should be executed for each data set in its terminal node.

3 NODE SHARING

The construction of a DN that exploits the structure of the rules and the facts to be expected in the system is critical for the resulting runtime and memory consumption of the RBS. To avoid unnecessary re-evaluations of partial results, an optimal network construction algorithm has to identify common subsets of rule conditions. In the corresponding DN, these common subsets may be able to use the output of the same network nodes. This is called node sharing and was already described in (Brant et al., 1991).

Despite the fact, that there is a lot of potential to save runtime and memory costs, current DN construction algorithms mostly work rule by rule (cf. Section 6). This way it will not always be possible to exploit node sharing to its full extent, e. g., if the nodes were constructed in a way, that the network is (locally) optimal for the single rule it was constructed for, but prevents node-sharing w. r. t. further rules and might therefore thwart finding the (globally) optimal DN for all rules in case sharing the nodes would have reduced costs (cf. example 3.1).

Example 3.1. Assume there are two filters: filter f_1 uses facts of type a and b, filter f_2 uses facts of type b and c. Furthermore, there are two rules: rule r_1 using f_1 and rule r_2 using f_1 and f_2. Then filter f_1 is used in both rules and we can construct a DN where both rules use the same node to apply f_1 to the input (see Figure 1).

If we were to construct rule r_2 first and would have decided to construct the node f_2 as an input for f_1, sharing f_1 with r_1 afterwards would have been impossible, since the output of the node for f_1 is also already filtered by f_2.

It is therefore advisable to construct the DN taking into account the set of rules as a whole.

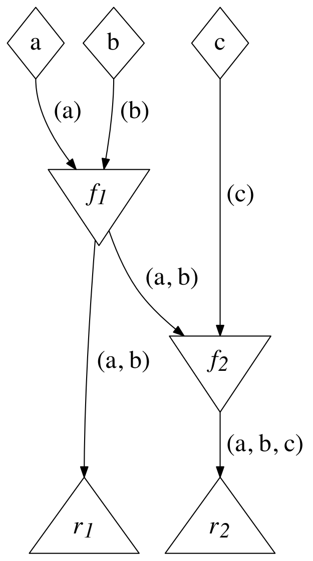

Figure 1: Simple node sharing example network.

4 EQUIVALENCE CLASSES

The common rule description languages resemble the Domain Relational Calculus (DRC) such that variable symbols that appear multiple times (e. g., within different relations or comparable constructs) implicitly cause that the condition is only true if the values of all symbol occurrences are the same. Considering the following condition in DRC $\{a,b,c \mid X(a,b) \wedge Y(a,c) \wedge a > 20\}$ one may choose whether the test $a > 20$ is applied to the data of X or Y (or both). The occurrences of a variable symbol in locations where the variables can be bound to values are collected in what we will from now on call *equivalence classes*. The resulting freedom in choosing an element of the equivalence class for filters can be considered within DN construction algorithms. Additionally, a minimal set of tests to ensure the equality of all elements of an equivalence class can be chosen freely.

5 CHALLENGES

Since node sharing is beneficial in most situations, DN construction algorithms should be presented the necessary data to maximise the potential savings in runtime cost and memory consumption. This section will present the challenges associated with generating these information.

Sadly, identifying common subsets of rule conditions isn't sufficient to make use of node sharing in network construction. This can be seen by extending the previous example.

Example 5.1. Assume there is an additional third rule r_3 using only the filter f_2. Now f_1 is part of r_1 and r_2 while f_2 is part of r_2 and r_3. Despite the fact that there are two non-trivial rule condition subsets, we can't share both filters between the three rules in an intuitive way. The rule r_2 requires a network that applies the filters f_1 and f_2 successively. Yet, the rule r_1 (r_3) needs the output of a node applying nothing but f_1 (f_2), meaning the corresponding nodes receive unfiltered input. Thus, we need two nodes for the two filters side by side at be beginning of the network and some additional node to satisfy the chained application of the two filters. There are three result networks still applying node sharing to some extent: We can either share f_1 and duplicate f_2, share f_2 and duplicate f_1, or re-use both nodes for r_2 by introducing an additional node that selects only those pairs of facts that contain identical b-typed facts in both inputs.

Formalising the phenomenon just observed, we say that two filters are *in conflict* if they use the same facts. Since in (Ohler and Terwelp, 2015) it has been shown that the runtime costs of the third network in example 5.1 are always higher than those of the other two networks, we will not consider such networks here. The decision which of the two remaining networks performs better depends on the data to be expected.

Furthermore, there may be situations where node sharing is not beneficial. For example, two rules sharing a filter that all facts pass should not share that filter if they have other (more selective) filters that could be applied to the data first. Sharing the filter would require to apply that filter first resulting in a high maintenance cost for the corresponding node. Applying the filter last could lead to very low maintenance costs as very few facts reach the node such that even the twofold costs are lower than the costs in the sharing situation. Detecting these situations requires information about e. g., filter selectivities, but can continue to improve the quality of the resulting network.

Finally, integrating the degree of freedom introduced by the equivalence classes as mentioned in Section 4 into the network construction is a further aspect considered here.

6 STATE OF THE ART

There are several DN construction algorithms creating different types of networks such as Rete (Forgy, 1982), TREAT (Miranker, 1987), and Gator (Hanson et al., 2002). Yet, they all consider the rules one after another so that the degree of sharing network parts is governed mainly by the order in which the rules are considered and the order of the filters within the rule conditions. Moreover, the optimisation potential introduced by the equivalence classes is neglected and all variables are assumed to be bound or are bound in a preliminary consideration.

An approach for query optimisation for in-memory DRC database systems is presented in (Whang and Krishnamurthy, 1990). They exploit the concept of equivalence classes, but only consider left-deep join plans and look at each query on its own without evaluating node-sharing.

In (Aouiche et al., 2006), the authors apply a data-mining technique to decide which views to materialise during the processing of a set of queries in a relational database system. Here, several queries are considered together and grouped by a similarity heuristic. Columns relevant for materialisation are identified by a cost function and re-used as much as possible to prevent repeated evaluations. In doing so, the filters to be applied are reduced to the ones relevant to all queries involved. Thereby, they do not identify the problem of conflicts as such and decisions are made based on columns to be materialised instead of filters as done here.

A first version of the block notation was presented in (Ohler and Terwelp, 2015), which was a rather simple approach not yet considering equivalence classes, existentials and predicates occurring more than once per rule among other aspects. It was further developed and evaluated in (Ohler et al., 2016). The new version integrates the gaps mentioned before, but still suffers from some shortcomings. Predicates occurring multiple times per rule could not be fully exploited for sharing and some situations, in which sharing would be possible, could not be taken advantage of since all occurrences of an equivalence class within a block were restricted in the same way.

7 APPROACH

Previously, we referred to different types of facts, which we will now call templates. A template resembles a class and its fields are called slots. All facts are instances of templates. More specifically, we will use the term *fact binding* to be able to distinguish between several facts of the same template. Every fact in the resulting fact tuple of a rule condition corresponds to a fact binding and vice versa. *Equivalence classes* as introduced in Section 4 contain fact bindings, slot bindings (bindings to a slot of a fact binding), constants, and functional expressions (i. e. $?x+?y$). A filter comprises a predicate (the test to be executed) and the equivalence classes to be used as parameters.

Existential parts of a condition have to be processed in a special way. If an equivalence class contains bindings originating from two different scopes, it is split into two classes containing the corresponding elements. Additionally, equivalence classes in child scopes know of their corresponding equivalence class in parent scopes. New scopes are created by existentials, which can also be nested. Filters appearing within existential parts can then be divided into three categories:

1. filters using only equivalence classes belonging to the current scope
2. filters using only equivalence classes belonging to parent scopes
3. filters using equivalence classes belonging to the current and parent scopes

The filters of the first two categories can be processed separately and have to be applied to the data prior to those of the third category. When applying the filters of the third category, the corresponding join merges the regular data with the existential data and implements the existential semantics. In a pre-processing step, all filters of the third category are merged into one filter, which we call the final filter of an existential condition part. It also contains the tests for equality of equivalence classes contained in the surrounding as well as in the existential scope. The parameters of this predicate are marked according to whether they are regular, existential or negated existential ones. As a consequence, existential condition parts can be integrated into the concepts to be presented.

We will now introduce a directed graph containing different types of nodes for different concepts represented. For this, we use the following example explaining the key parts.

Example 7.1. Consider the following rule condition with two template instances (i. e. facts) and a filter with three arguments:

```
(t (s ?x))
(t' (s' ?x) (s'' ?y))
(test (f ?x ?y ?y))
```

In the so called assignment graph, we create a node for every filter (f) and every template instance (t and t'). We let F denote the set of filter nodes and T denote the set of template instance nodes. As the rule contains three slot bindings, we create three corresponding nodes b for `t::s`, b' for `t'::s'`, and b'' for `t'::s''`. These nodes are gathered in the set of fact and slot bindings B_T. The variable `?x` can be bound to `t::s` and `t'::s'`, so the corresponding equivalence class contains those two slot bindings. For every element of an equivalence class, we create a corresponding node in the assignment graph gathered in the set O_i of implicit equivalence class occurrence (more on that later). Thus, for the equivalence class corresponding to `?x`, there are the two nodes o_i for `t::s`, and o_i' for `t'::s`. Since the variable `?y` can only be bound to `t'::s''` and the corresponding equivalence class only contains one element, we omit the creation of a corresponding node as for all trivial equivalence classes. Finally, we create a node for every parameter of a filter (o_f, o_f' and o_f''). These nodes are gathered in the set O_F of equivalence class occurrences within filters.

Figure 2 shows the assignment graph with the corresponding directed edges. One can identify four layers the way the graph is depicted. The top layer contains the filters (F). Below, there are the equivalence class occurrences (O). Every filter is connected to its occurrences via edges. The third layer contains the bindings (B). The occurrences are connected to the bindings contained in the corresponding equivalence classes via edges. In the bottom layer the template instances are contained (T) and connected to the corresponding bindings in the third layer via edges.

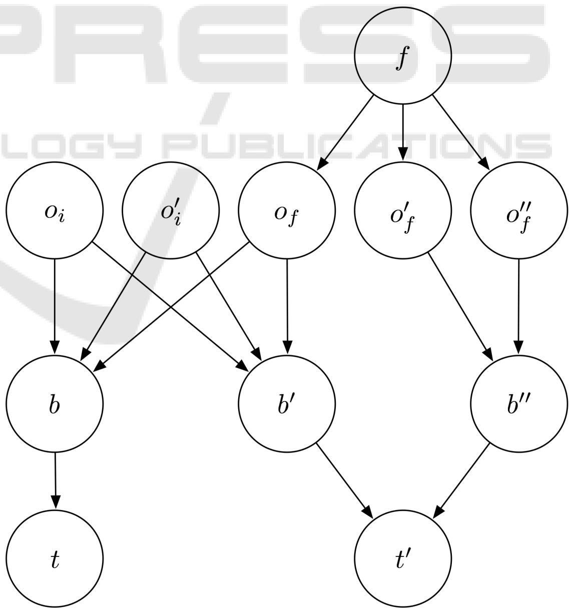
$$\begin{tikzcd}
 & & & f \arrow[dl] \arrow[d] \arrow[dr] & \\
o_i \arrow[d] \arrow[drr] & o_i' \arrow[dl] \arrow[dr] & o_f \arrow[dll] \arrow[d] & o_f' \arrow[dr] & o_f'' \arrow[d] \\
b \arrow[d] & & b' \arrow[dr] & & b'' \arrow[dl] \\
t & & & t' &
\end{tikzcd}$$

Figure 2: Assignment graph example.

The four layers already described i. e. the corresponding sets of nodes form the node set $V = F \dot\cup O \dot\cup B \dot\cup T$ of the assignment graph $G = (V,E)$. The set of occurrences $O = O_I \dot\cup O_F \dot\cup O_V$ additionally contains the set O_V of occurrences within functional expressions and the set of bindings $B = B_T \dot\cup B_C \dot\cup B_V$ additionally contains the set B_C of constant bindings and the set B_V of functional expression bindings. Note that constants occurring in multiple rules are mapped to one node per rule they are contained in. O_I contains one node per equivalence class element.

The set of edges $E = E_F \dot\cup E_T \dot\cup E_O$ of the assignment graph comprise the edges E_F from the filters $f \in F$ to the corresponding occurrences in $O_f \subset O_F$, the edges E_T to the template instances $t \in T$ originating from the corresponding slot and fact bindings in $B_t \subset B_T$, and the edges E_O from the occurrences $O_Q \subset O$ of an equivalence class Q to its bindings $B_Q \subset B$.

Every rule is represented in the graph by at least one connected component of the assignment graph. Thus there are no edges between rules.

A non-empty subset $Z \subset E(G)$ of the edges of the assignment graph G is called a *block row* if the following conditions are fulfilled:

- If an edge adjacent to a filter is in Z, all edges in G adjacent to that filter have to be in Z.
- The edge to a non-implicit occurrence is in Z iff at least one edge from that occurrence to a binding is in Z.
- The edge from a fact or slot binding to the corresponding template instance is in Z iff at least one edge from an occurrence to a binding of this template instance is in Z.
- If an edge from an occurrence to a functional expression binding is in Z, all edges originating from that binding have to be in Z.
- If an edge from a functional expression binding is in Z, at least one edge to that binding has to be in Z.
- An edge from an implicit occurrence to its corresponding binding is in Z iff at least one other edge to that binding is in Z.
- If the edges (o,b), (o,b'), and (o',b) for occurrences o, o' and bindings b, b' are in Z, then (o',b') has to be in Z.
- For all occurrences in functional expressions with adjacent edges in Z, it holds that there are paths in Z from these occurrences to slot, fact, or constant bindings.
- If an existential edge (f,o) adjacent to a filter f is in Z, the connected component originating from removing all existential edges adjacent to f that contains o has to be a subset of Z.

Two block rows are *compatible* iff both block rows are disjoint and no edge in the one block row is adjacent to an edge in the other block row.

A non-empty subset $S \subset E(G)$ of the edges of the assignment graph G is called a *block column* if the following conditions are fulfilled:

- All edges in S are pairwise non-adjacent
- For the set of start or target nodes V' of the edges in S one of the following conditions holds:
  - V' contains filters only and they all apply the same predicate having the same parameters marked as (negated) existential
  - V' only contains implicit occurrences
  - V' only contains non-implicit occurrences representing the same position in the list of parameters of a filter or functional expression.
  - V' only contains bindings to the same constant.
  - V' only contains bindings to slots of the same name. The equality of the template is assured via the compatibility of block columns (see below).
  - V' only contains fact bindings.
  - V' only contains bindings to functional expressions and they all use the same function.
  - V' only contains template instances (facts) of the same template.
- If all start nodes of the edges in S are implicit occurrences, either all or none of the edges lead to the corresponding binding.

Two different block columns S and S' are *compatible* iff at most one pair of the sets of start and target nodes of S and the sets of start and target nodes of S' are identical and all others are disjoint.

We also refer to the number of the elements of a block column as the height of the block column.

A set of pairwise compatible block rows $\mathcal{Z}$ together with a set of pairwise compatible block columns $\mathcal{S}$ is called a *block* iff the set of all edges of the block rows is identical to the set of all edges of the block columns and the amount of block rows corresponds to the height of the block columns.

A block with more than one row represents the possibility of sharing its included filters. So only one row of the block has to be implemented in the DN and can be reused for the implementation of the other rows. Implicit occurrences allow sharing of implicit equalities represented by equivalence classes.

Two blocks $X = (\mathcal{Z},\mathcal{S})$, $Y = (\mathcal{Z}',\mathcal{S}')$ are in *conflict* iff none of the following conditions are satisfied:

- $T \cap T' = \emptyset$ with T and T' being the template instances of the blocks X and Y, respectively.
- $\forall Q \in \mathcal{Q}\ \exists R \in \mathcal{R} : R \subset Q$ for $\mathcal{Q},\mathcal{R} \in \{\mathcal{S},\mathcal{S}'\}$ as well as $\mathcal{Q},\mathcal{R} \in \{\mathcal{Z},\mathcal{Z}'\}$ with $\mathcal{Q} \neq \mathcal{R}$ in both cases.

A block set free of conflicts between blocks is useful to prevent conflicts between filters.

We say that a set of blocks is complete, if every node of the assignment graph is contained in at least one block, none of the blocks contains all elements of another block partitioned the same way, and no block can be extended further.

Constructing the DN for a complete, conflict-free block set can be done by materialising the filters in the blocks starting with the blocks containing the fewest columns. Within the set of blocks containing an equal number of columns the order is arbitrary, since none of these blocks can be the input of another block in that set (otherwise they would "overlap" and would have been in conflict). A more detailed explanation of how to construct the network (part) for a block including considerations about the optimisation potential and what to keep in mind w. r. t. filters with existential parameters is given in (Ohler et al., 2016).

The construction order is relevant only if blocks contain the same nodes. Since the block set is conflict-free and complete, if one block overlaps with another block, the columns of one of the blocks are a subset of the columns of the other block. As the one with fewer columns is constructed first, its output can be used to construct the larger (w. r. t. column count) block.

8 CONCLUSION & OUTLOOK

We presented a concept for an optimisation of DNs for RBSs considering node-sharing and integrating the degree of freedom emerging from being able to choose between elements that are supposed to be equal. This block concept is able to formalise the problems of node-sharing, i. e. which network parts would compete against each other. Equivalence classes were integrated into the block concept to allow for a free choice of which element to use for which filter and of how to check the equality among the elements efficiently, e. g., using a minimal spanning tree.

Based on the notation presented, we are currently developing optimisation algorithms considering several rules at once. The output of such an algorithm should be a conflict-free set of blocks, where no block can be extended and no block contains all elements of another block partitioned the same way. An optimising DN construction algorithm can then use this information to decide, whether node-sharing is beneficial in terms of runtime cost and memory consumption w. r. t. the data to be expected. Developing such an algorithm with acceptable runtime costs – despite the fact that it has to look at a set of rules instead of a single one – is pending.

ACKNOWLEDGEMENTS

This work was funded by German Federal Ministry of Economic Affairs and Energy for project Mobility Broker (01ME12136).

REFERENCES

Aouiche, K., Jouve, P.-E., and Darmont, J. (2006). Clustering-based materialized view selection in data warehouses. In Manolopoulos, Y., Pokorný, J., and Sellis, T. K., editors, *Advances in Databases and Information Systems*, volume 4152 of *Lecture Notes in Computer Science*, pages 81–95. Springer Berlin Heidelberg.

Brant, D. A., Grose, T., Lofaso, B., and Miranker, D. P. (1991). Effects of database size on rule system performance: Five case studies. In Lohman, G. M., Sernadas, A., and Camps, R., editors, *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 287–296.

Brownston, L., Farrell, R., Kant, E., and Martin, N. (1985). *Programming expert systems in OPS5*. Addison-Wesley Pub. Co., Inc., Reading, MA.

Forgy, C. L. (1981). OPS5 User's Manual. *Tech. Report CMU-CS-81-135. Carnegie-Mellon Univ. Pittsburgh Dept. Of Computer Science*.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37.

Hanson, E. N., Bodagala, S., and Chadaga, U. (2002). Trigger condition testing and view maintenance using optimized discrimination networks. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):261–280.

Hanson, E. N. and Hasan, M. S. (1993). Gator : An Optimized Discrimination Network for Active Database Rule Condition Testing. *Tech. Report TR93-036, Univ. of Florida*, pages 1–27.

Miranker, D. P. (1987). TREAT: A Better Match Algorithm for AI Production Systems; Long Version. Technical report, Austin, TX, USA.

Ohler, F., Krempels, K.-H., and Terwelp, C. (2016). Randomised optimisation of discrimination networks considering node-sharing. Manuscript submitted for publication.

Ohler, F. and Terwelp, C. (2015). A notation for discrimination network analysis. In *Proceedings of the 11th International Conference on Web Information Systems and Technologies*, pages 566–570.

Whang, K.-Y. and Krishnamurthy, R. (1990). Query optimization in a memory-resident domain relational calculus database system.