Randomised Optimisation of Discrimination Networks Considering Node-sharing

Fabian Ohler, Karl-Heinz Krempels and Christoph Terwelp
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. We present an approach incorporating the potential of both concepts and balance their application in a randomised fashion. To evaluate the algorithms developed, they were implemented and yielded promising results. Shortcomings of this approach are discussed and their removal constitutes our current 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) incorporating the potential of both concepts and balance their application in a randomised fashion.

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 is presented in Section 6. The identified problems are then addressed in Section 7 by introducing the block notation and the construction algorithm using it. This algorithm is evaluated in Section 8. Section 9 comprises the conclusion and gives an outlook on future work.

2 DISCRIMINATION NETWORKS

Rules in RBSs and DBMSs both comprise conditions 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 mem-

Ohler, F., Krempels, K-H. and Terwelp, C.
Randomised Optimisation of Discrimination Networks Considering Node-sharing.
In *Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST 2016) - Volume 2*, pages 257-267
ISBN: 978-989-758-186-1

ory 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. An example DN is shown in Figure 1.

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 reevaluations 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, e. g., in (Brant et al., 1991).

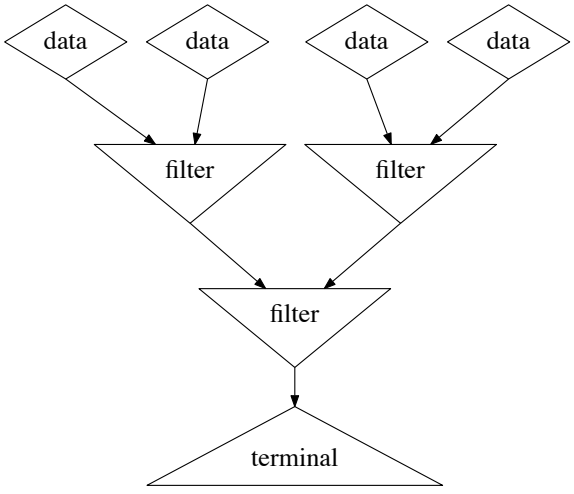

Figure 1: DN example.

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).

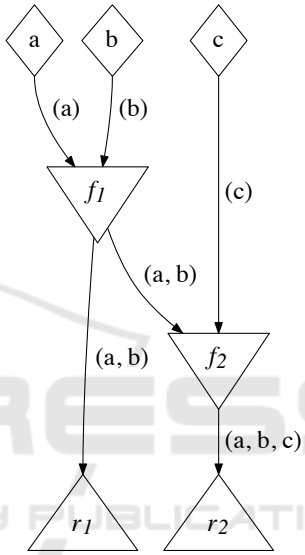

Figure 2: Simple node sharing example network.

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 2).
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.

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.

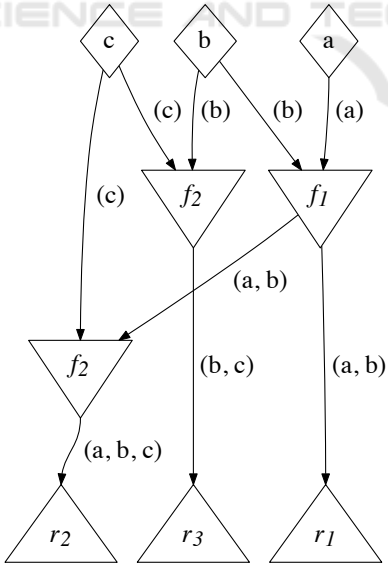

Figure 3: f_1 shared, twofold materialisation of f_2.

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

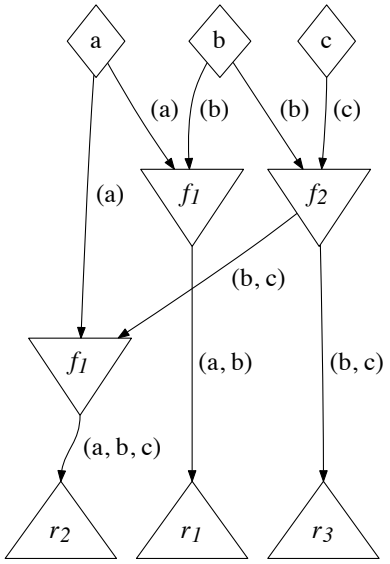

Figure 4: f_2 shared, twofold materialisation of f_1.

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 (Figure 3), share f_2 and duplicate f_1 (Figure 4), 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 (Figure 5).

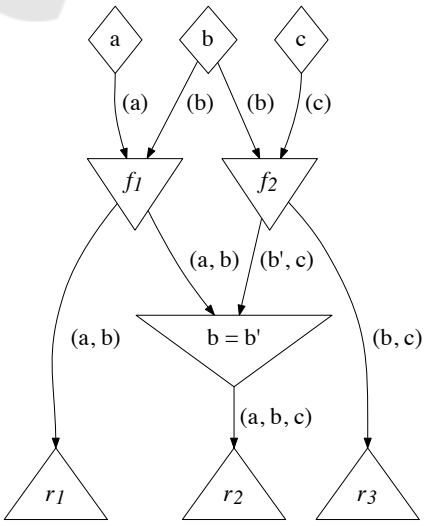

Figure 5: Sharing conflict solved using a special join.

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 network depicted in Figure 5 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. Furthermore, 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.

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 parameters to be used. We distinguish between the following two types of filters:

Explicit Filter. An explicit filter is a filter using equivalence classes as arguments.

Implicit Filter. An implicit filter tests the equality of exactly two elements of the corresponding equivalence class.

For two filters f and g we call $c(\vartheta(f), \vartheta'(g))$ the *conflict index set* w. r. t. the equivalence class restrictions ϑ and ϑ' (see below). It contains pairs of indices with the first index corresponding to a parameter position of the filter f, the second index meaning the same for g. Only those index pairs are contained, for which there is a non-empty intersection of the fact bindings in the restricted equivalence classes corresponding to the parameter determined by the indices.

Two filters f and g are *in conflict* w. r. t. ϑ and ϑ' iff $c(\vartheta(f), \vartheta'(g)) \neq \emptyset$. Given the filters a, b, f, g, we write $(a, f) \sim_c^{\vartheta} (b, g)$ instead of $c(\vartheta(a), \vartheta(f)) = c(\vartheta(b), \vartheta(g))$. A block consists of the following four components:

Equivalence Class Restriction. An equivalence class restriction (denoted ϑ) of a block is a function mapping every equivalence class occurring in the block onto the maximal subset still guaranteed by the implicit tests of the block.

Filter Partition. A filter partition is a partition of the explicit filters of a block with the following property: Every set of the partition contains filters of only one predicate and for every pair of sets in the partition it holds that every pair of elements of the

same rule has the same conflict index set as every other pair of elements of a different rule.

Fact Binding Partition. A fact binding partition is a partition of the fact bindings of the block such that every set of the partition contains bindings to only one template.

Element Partition. An element partition consists of sets of "compatible" elements comprising the elements of the restricted equivalence classes of the block. Two elements are "compatible", if they do not prevent sharing, e. g., two fact bindings belonging to the same set in the fact binding partition, two equal constants and so on. An element partition is always defined w. r. t. a fact binding partition.

Additionally, every set in all of the partitions defined contains exactly one element per rule of the block.

We now give an inductive definition of the *block* property:

1. A single explicit filter i together with an equivalence class restriction that maps the equivalence classes in i onto singleton subsets form a block. Every element of the corresponding partitions contains only one element making the partitions unique.
2. Let a, b be two different elements in an equivalence class K. Let Z be the set of equivalence classes appearing within a and b together with K. Let ϑ be an equivalence class restriction mapping K to $\{a, b\}$ and all other equivalence classes in Z to singleton subsets.
   The two filters $(= a\ b)$ and $(= b\ a)$ belonging to K together with the equivalence class restriction ϑ, a fact binding partition f of the fact bindings within the restricted equivalence classes in Z, a compatible element partition and the empty filter partition form a block. The corresponding filter partition contains singleton sets for the two filters.
3. Let B be a block and N be a set of explicit filters belonging to the same filter and to the rules in B. Let B and N be disjoint and N contain exactly one filter per rule in B. Let ϑ be an equivalence class restriction extending the equivalence class restriction of B by mapping all equivalence classes not part of the original domain to singleton subsets. Let there be at least one filter in N and one (explicit or implicit) filter in B that are in conflict w. r. t. to ϑ. Every pair of elements of the same rule consisting of one element in B and one in N has the same conflict index set as every other such pair of elements of a different rule.
   $B \cup N$ together with the equivalence class restriction ϑ (restricted to the relevant domain), an extension of the fact binding partition for the additional elements (w. r. t. ϑ) and a compatible extension of the element partition form a block. Additionally, N is to be added to the filter partition.
4. Let B be a block and P be one of the sets in the element partition of B. Let V be a set consisting of exactly one hidden (by the equivalence class restriction of the block) element per equivalence class belonging to an element in P. Let ϑ be an equivalence class restriction extending the equivalence class restriction of B by mapping all equivalence classes not part of the original domain to singleton subsets and adding the elements in V to the corresponding restricted subsets. Let T be the set of all implicit filters testing the equality between an element in V and the elements of the corresponding equivalence class restricted according to the block. Let the filter partition of B be a filter partition w. r. t. ϑ.
   $B \cup T$ together with ϑ (restricted to the relevant domain), an extension of the fact binding partition for the elements in V, a compatible extension of the element partition and the original filter partition form a block.
5. Let B be a block and R be a set of explicit filters belonging to the same rule. Let B and R be disjoint. Let A be a subset of the explicit filters in B belonging to the same rule. Let ϑ be a equivalence class restriction extending the equivalence class restriction of B by mapping all equivalence classes in R to subsets. Let there be a bijection between the fact bindings of the equivalence classes of A and the fact bindings of the equivalence classes in R w. r. t. ϑ. Analogous, let there also be bijections for the elements and explicit filters. Let f be a fact binding partition of $B \cup R$ extending the fact binding partition of B by adding for every fact binding in the equivalence classes of A the corresponding (according to the bijection) fact binding to the partition that the former is contained in. Let e be an element partition of $B \cup R$ extending the element partition of B in the same way and I be a filter partition of $B \cup R$ extending the filter partition of B in the same way.
   $B \cup R$ together with the equivalence class restriction ϑ, the fact binding partition f', the element partition e' and the filter partition I' form a block.
6. Only sets generated according to the rules given form blocks and accordingly partitions and equivalence class restrictions.

We elevate the conflict property to describe conflicts

between blocks: Two blocks X and Y are in *conflict* iff one of the following conditions is met:

- The blocks are disjoint and a filter in X is in conflict with a filter in Y.
- Let P be the filter partition of X and Q be the filter partition of Y. Let w. l. o. g. P contain at least as many sets as Q. There is a set in Q for which there does not exist a set in P that is a subset of the former set.
  Otherwise let $\mathbb{M}$ be the set of equivalence classes contained in both blocks. There is an equivalence class in $\mathbb{M}$ whose restricted version according in X is a proper subset of the restricted version in Y.

A block is *maximal* if it can not be extended in any way according to the rules given above. Furthermore, a block is contained in another block if every filter of the first block is contained within the second one. A set of blocks is called *complete* (w. r. t. a set of filters) if every filter is contained in at least one block and no block is contained in another block.

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.

As a consequence, existential condition parts can be integrated into the block notation. All filters of the existential condition part despite the final filter are considered as a separate rule. The final filter remains part of the original rule, but is treated in a different manner. Two such final filters a, b belong to the same filter if both apply the same predicate to the same arguments (specified by templates and slots) and the set of filters A (B) of the existential condition part of a (b) satisfies the following condition: There is a bijection φ between the filters in $A \cup \{a\}$ and the filters in $B \cup \{b\}$ that maps a onto b, every filter belongs to the same filter as its image and every pair of filters has the same conflict index set as the corresponding pair of images w. r. t. a non-restricting ϑ.

7.1 Construction of Maximal Blocks

Due to space limitations, a detailed description of the construction of maximal blocks can not be provided here. Thus, this section only outlines the necessary steps.

To get heterogeneous sets of implicit filter, equivalence classes are represented as follows by implicit tests:

- For every pair of elements $a \neq b$ of fact bindings, slot bindings and constants in an equivalence class two implicit filter are created: `(= a b)` and `(= b a)`.
- For every pair of elements $a \neq b$ of functional expressions in an equivalence class e, two filter are created: `(= a b)` and `(= b a)`. Additionally, if e also contains at least one fact binding, slot binding or constant, for every functional expression a two filter are created: `(= a e)` and `(= e a)`. These filter are similar to explicit filter since their arguments are equivalence classes.

By looking at all rule variations, try to find blocks consisting of one filter per rule belonging to the same filter with matching equivalence class restrictions. Every block found this way is expanded *horizontally* in a recursive manner. A block can be expanded *horizontally* by adding further elements of the rules of the block, i. e., explicit filters or elements of a contained equivalence class and the corresponding implicit filters. To mitigate the fact that the order of block expansions leading to a specific result block is not unique a stack of filters to be excluded is integrated into the recursion. Additional constraints are exploited to limit the search scope for block expansions, whose explanation would require a more detailed explanation of the actual algorithms.

The blocks found this way are maximal in their horizontal dimension or contained in a maximal block (the latter caused by the fixed set of rules in the block or the exclusion stack). All blocks contained within other blocks can be discarded. Since every variation of rules is considered, all *vertically* maximal blocks

(i. e., no further rules can be added) are contained in the result set. Additionally, every filter is contained within some block since the expansion was initiated starting at it at some time and only blocks contained in other blocks are discarded. Thus, the resulting block set is a complete block set.

7.2 Conflict Resolution

The block set acquired this way is not necessarily conflict-free. There are several ways to solve conflicts between blocks. The one used here solves a conflict between two blocks by replacing one of them with smaller blocks contained within it, but not in conflict with the other block. Those blocks either have stronger equivalence class restrictions than the original block or contain fewer explicit filters and only a subset of the original set of equivalence classes. Reducing the set of rules contained is another option, of course.

Every new block found this way is either contained within another block or can not be extended without causing further conflicts. The former are to be discarded, the latter to be added to the block set currently considered. To prevent blocks that have already been replaced to reappear in the block set, those have to be remembered. Any block on that list or contained in such a block has to be prevented from being reinserted into the block set.

Thus, all conflicts can be resolved by replacing one of the two blocks in conflict. Yet, a conflict resolution may influence further blocks in the block set. Other conflicts may be resolved as a side effect, but the newly added blocks might also introduce additional conflicts with themselves or the blocks in the block set. Hence, the order, in which conflicts are solved, is relevant to the network constructed from the conflict-free block set.

For this reason, we now try to quantify the positive and negative effects a conflict resolution. All filters in the replaced block that are in conflict with the persisting block (but not contained in it) have to be presented separately in the resulting network. A further node is necessary, if the blocks don't overlap. The number of additional nodes represents the negative effect of the conflict resolution. Conversely, the negative effects within other conflict resolutions prevented by resolving a conflict constitute the positive effect of a conflict resolution. The difference between positive and negative effects is called the *benignity* of a conflict resolution.

As long as the block set contains conflicts, apply the best conflict resolution according to their benignity.

7.3 Network Construction

To construct the network for a complete, conflict-free block set, the blocks have to be ordered. Given a block B let

$$n(B) := \frac{\text{number of filters in } B}{\text{number of rules in } B}$$

denote its characteristic number. Blocks are ordered according to their characteristic number in ascending order. Yet, the construction of final filters in existential condition parts has to be delayed until the corresponding filters of the existential condition part are constructed. Thus, blocks containing them are moved behind the final filter preserving the order within the blocks moved this way.

To construct a block contained in the ordered set of blocks, only the filters of a single rule of the block have to be considered. Regarding equivalence classes, only the equality tests for the restricted versions are relevant. Additionally, all filters already constructed can be ignored. As a preparatory measure, the (restricted variants of) equivalence classes containing constants can be identified. Occurrences of those equivalence classes can be fixed to contain those constants. All filters only using constants can now be constructed. Identify the groups of fact bindings already joined in the network (using singleton sets where appropriate). Now repeat the following steps:

- Construct all filters that can be executed on any group without further joins. For implicit tests, the symmetry and transitivity of equality can be exploited to reduce the amount of tests actually performed.
- If all filters of the block have been constructed, break the loop.
- Consider a graph having the fact binding groups as nodes. The edges are marked with the combined join selectivity of the tests that can be applied using those two groups only. If there are no edges, increase the number of groups allowed (leading to hyper-edges) until edges emerge. Identify the edge of minimal weight and merge the adjacent groups.

7.4 Randomized Optimisation

This work makes use of the two well-known randomisation algorithms *Iterative Improvement* (Nahar et al., 1986) and *Simulated Annealing* as presented in (Ioannidis and Kang, 1990). As common to all randomization heuristics they need a definition of a state, the possible moves and a cost function.

Applying randomisation in network construction can make sense at different points. The block sets constructed so far maximise sharing of network parts. This approach does not necessarily result in minimal runtime and memory costs. Using an application-dependent cost function for states allows an integration of the corresponding data into the optimisation.

While constructing a block, randomised decisions can be made in choosing which implicit filters to use and which representatives to use for equivalence classes occurring in explicit filters. Furthermore, filters may be duplicated in a randomised fashion to see whether this would be beneficial.

The order in which filters are applied (and the resulting join order) as well as which filters are grouped together into a node are other possible targets for randomisation. Finally, for a node with more than two inputs different join plans can be determined based on which input new data entered the node over.

This paper only covers a randomisation of the block set. A state comprises of a conflict-free block set, where no block in contained in another block. The following transformations of a state are possible:

- Extend the filter partition of a block by a set of explicit filters.
- Reduce a block by a set of explicit filters in the filter partition.
- Extend the equivalence class restriction of a block by adding the corresponding implicit filters to the block.
- Diminish the equivalence class restriction of a block. In doing so, remove all implicit filter instances no longer needed.
- Extend a block by a rule.
- Reduce a block by a rule.
- Create a new block.

After every such transformation remove all blocks contained in other blocks and solve newly arisen conflicts. Every transformation together with these subsequent restoring actions represents a move. Using the rating function for discrimination networks given in (Ohler et al., 2013), costs for states are determined by constructing the corresponding discrimination network and applying the rating function.

A state is intentionally not defined as a complete block set in favour of the randomisation. All filters not contained within a block are considered to be in singleton blocks.

8 EVALUATION

To evaluate the presented approach, it was implemented for the rule-based system Jamocha[1]. A thorough evaluation of the presented concepts has been performed and the essential parts are presented is this section. The benefit is shown by means of the rules of the benchmark Waltz (Winston, 1984).

8.1 Description of the Measurements

The parameters for the randomised algorithms were chosen in dependence on (Ioannidis and Wong, 1987; Swami and Gupta, 1988; Hanson et al., 2002) and are given in Table 1. The rating function described in (Ohler et al., 2013) was used as the cost function considering runtime costs only.

This rating function needs statistical information concerning the facts to be expected. Currently, Jamocha lacks a statistics component that could provide this data. Since the number of values needed is too high to be determined by hand, select values were determined explicitly and all other values were set to defaults, see Tables 2 and 3. It was assumed, that 1 000 facts fit on a memory page. As a further simplification, conditional probabilities were only incorporated in some trivial cases.

An additional command (`defrules`) was added to the CLIPS[2] language, which is used as the input language for Jamocha, allowing for the definition of several rules within one command. The construction methods described in this paper were implemented as the *ECBlocks* compiler. A simpler version of this compiler called *PathBlocks* that does not integrate equivalence classes and a *trivial* compiler considering each rule on its own were additionally implemented for the following comparison. Since the runtime of the algorithms presented grows exponentially in the number of rules, the rule base of Waltz was split into smaller groups of rules to be considered at the same time.

The rule set was constructed with each of the three algorithms and additionally using the ECBlocks compiler without the randomisation part. For every resulting network the costs according to the rating function were determined. As the randomisation was expected to disperse the results, 12 measurements were conducted per algorithm.

[1] source code available via: `git clone -b pre-partitioning-change` git://git.code.sf.net/p/jamocha/git

[2] CLIPS Project Page: http://clipsrules.sourceforge.net/

Table 1: Randomisation parameters.

Iterative Improvement	
initial state	complete, conflict-free block set
stopping condition for outer loop	iteration count equals number of fact bindings in the rule set
definition of local minimum	20 moves without improvement
Simulated Annealing	
initial temperature	5% of the costs of the best II state
initial state	best II state
stopping condition for inner loop	iteration count equals number of fact bindings in the rule set
stopping condition for outer loop	best state unchanged for 5 iterations or temperature less than the thousandth part of the initial temperature
temperature reduction	$x \mapsto 0.95 \cdot x$

Table 2: Parameters of the rating function: template-data.

template	insert/delete frequency	number of facts
stage	10	1
line	20	1 000
edge	100	2 000
junction	30	800
all other templates	10	1 000

Table 3: Parameters of the rating function: selectivities.

filter	selectivity
`(= edge::p1 edge::p1)`	0.05
`(= junction::base_point edge::p1)`	0.05
`(= edge::p1 junction::base_point)`	0.05
`(not (= edge::p1 edge::p1))`	0.95
`(not (= edge::p2 edge::p2))`	0.95
`(= edge::label nil)`	0.85
`(= edge::joined false)`	0.75
`(= edge::joined edge::joined)`	0.90
`(= junction::type *)`	0.30
cross product	1.00
all other filters	0.60

8.2 Description of the Results

The costs of the resulting networks are plotted in Figure 6 showing the three quartiles. The median determines the height of the bars, the other two quartiles are shown as error bars.

For the given rule set, the *trivial* construction algorithm produced networks with the highest costs and a low spread. Networks constructed by the PathBlocks compiler were rated better in both aspects than the trivial ones. Note that the runtime costs decrease about twice as much as the memory consumption. Additionally, the spread decreases to a negligible value. The small improvement of the ECBlocks networks without randomisation is hardly noticeable and the spread is still low. However, activating the randomisation part leads to a significant improvement. The scatter of the results are comparatively high.

8.3 Discussion

Not all steps in the transfer of a block set to a network are completely deterministic, since some decisions are to be made between options that seem to be equally beneficial. Thus, even non-randomised results are slightly dispersed. The simplifications regarding the statistical information influence the re-

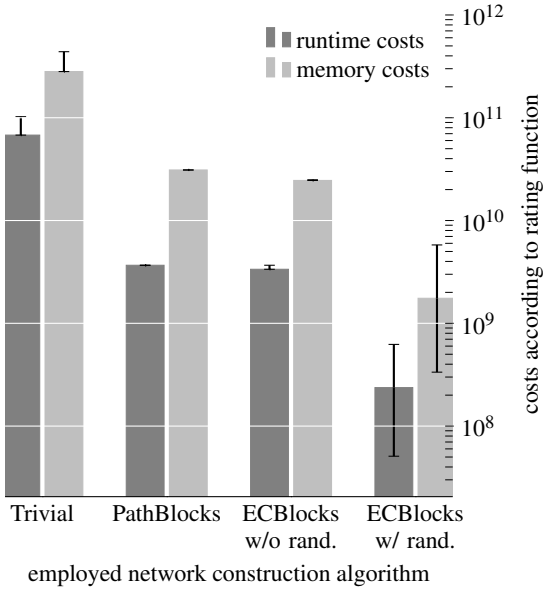

Figure 6: Rating of the networks for Waltz.

sults of the rating function. With respect to the construction without randomisation networks may have been rated incorrectly. Additionally, through incorrect ratings some decisions made during the randomisation may have in fact been adverse.

The way the logic behind the rules in Waltz works is similar to a state machine. Thus, the rule set can be partitioned w. r. t. states they belong to. In CLIPS, this is implemented via a `stage`-template added to all rules conditions to check whether the system currently is in the corresponding state. Thus, it suggests itself to only keep those parts of the discrimination network up-to-date that belong to active state. This could be realised to a large extent by having the stage-template in the first join (i. e. a cross product) of every rule resulting in an empty successor network for all inactive states. Since every state is active only once, this would reduce the maintenance costs considerably. Yet, Jamocha uses the connectivity heuristic which delays cross products as much as possible. Integrating a mechanism allowing for a maintenance of partial networks only is considered as future work.

9 CONCLUSION & OUTLOOK

We presented a concept for a randomised 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. Possible solutions of these conflicts were presented and cast into algorithms. 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. Via the randomisation, a balance between node-sharing and degree of freedom is to be established and situations are to be identified, in which reduced sharing increases the performance.

Our evaluations show promising results even though we could only consider small groups of rules at a time for runtime reasons.

The most runtime-intensive task is the construction of a maximal block set. This could be mitigated by either using a heuristic approach to find a block set sufficiently close to the maximal one or applying the randomisation starting with an empty block set. For the latter to yield acceptable results, the method of extending blocks within the randomisation has to be improved and we are currently working on this problem. Alternatively, there are approaches in the area of parallel programming to speed up the task, e. g., Transactional Memory (Herlihy and Moss, 1993; Adl-Tabatabai et al., 2006). They can exploit the fact that most blocks found are already contained in another block and can be discarded. Only in case two newly found blocks have to be inserted into the result set, a synchronisation has to be performed.

Filters occurring multiple times within a rule were not exhaustively considered for sharing in this paper. This calls for decisions to be made especially in those cases where it occurs more than once per rule since there are mutual dependencies. Being able to consider these requires a change in the block definition since not equally many filters are contained per rule. How to solve conflicts in this scenario remains open, too.

Restricting an equivalence class in different ways for different occurrences of the corresponding variable symbol within one block would allow for increased sharing opportunities. Enabling this without a further blow-up of the construction runtime is subject to future extensions of the work presented here.

ACKNOWLEDGEMENT

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

REFERENCES

Adl-Tabatabai, A., Kozyrakis, C., and Saha, B. (2006). Unlocking concurrency. *ACM Queue*, 4(10):24–33.

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.

Herlihy, M. and Moss, J. E. B. (1993). Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture. San Diego, CA, May 1993*, pages 289–300.

Ioannidis, Y. E. and Kang, Y. C. (1990). Randomized algorithms for optimizing large join queries. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990.*, pages 312–321.

Ioannidis, Y. E. and Wong, E. (1987). Query optimization by simulated annealing. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pages 9–22.

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

Nahar, S., Sahni, S., and Shragowitz, E. (1986). Simulated annealing and combinatorial optimization. In *DAC*, pages 293–299.

Ohler, F., Schwarz, K., Krempels, K., and Terwelp, C. (2013). Rating of discrimination networks for rule-based systems. In *Proceedings of the 2nd International Conference on Data Technologies and Applications*, pages 32–42.

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.

Swami, A. N. and Gupta, A. (1988). Optimization of large join queries. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988.*, pages 8–17.

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

Winston, P. H. (1984). *Artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.