

Query Rewriting and Search in CROQUE

Joachim Kröger[◊], Regina Illner[†], Steffen Rost[†], Andreas Heuer[◊]

[◊] University of Rostock
Computer Science Department
Database Research Group
D-18051 Rostock
Germany

[†] c/o IBM Deutschland Entwicklung GmbH
Böblingen Programming Lab
Department 3303/3301
D-71032 Böblingen
Germany

{jo, ah}@informatik.uni-rostock.de

{illner, srost}@de.ibm.com

<http://wwwdb.informatik.uni-rostock.de/Research/CROQUE.engl.html>

Abstract

One of the hard aspects of query optimization is the query rewriting process. Using rewrite rules, a given query will be transformed into an equivalent execution plan that is cheaper than the straightforwardly assigned plan according to some cost model. Finding the cheapest of all equivalent execution plans is a challenge since the rewriting of complex queries on the basis of a large set of rewriting rules may potentially span a very large space of equivalent plans. Consequently, one has to either use search strategies to explore (parts of) the search space or some heuristics to prune this space thus making it efficiently traversable.

This paper presents the use of search strategies in the CROQUE project in detail. The adaptation of some common strategies led to the development of a very simple but surprisingly powerful heuristics which is demonstrated by examples executed in the CROQUE prototype. The proposed heuristics can support any random-based search strategy or can be used stand-alone. It may be integrated seamlessly into most of the present query optimizers without almost any effort.

Keywords: Query optimization, rule-based query rewriting, search strategies, heuristics

1 Introduction

The CROQUE project¹ [HK96, SGG⁺97] is concerned with different aspects of the optimization and evaluation of object-oriented queries. Starting points of all our considerations are queries in ODMG's OQL ([Cat96], formalized in [RS97]), that are first represented internally in a hybrid approach of calculus and algebra [GKG⁺97] before being transformed using a rule- and cost-based optimizer. The already developed cost model [KJH98] is not further discussed in this paper.

In CROQUE, rule-based rewriting spans a space of equivalent query execution plans. The cost model allows a comparative assessment of all these plans thus defining the search space of

¹CROQUE (*Cost- and Rulebased Optimization of object-oriented QUERies*) is a joint research project of the universities of Rostock and Konstanz funded by the German Research Association (DFG) under contracts He 1768/5-2 and Scho 554/1-2.

CROQUE. By the use of search strategies, this search space is explored. The search is controlled by a heuristics to first explore the most interesting parts of the search space.

In this paper, we present the use of search strategies in the CROQUE project in detail. The adaptation of the common random-based strategies such as random walk, iterative improvement, simulated annealing and two-phase optimization led to the development of a very simple but surprisingly powerful heuristics that we demonstrate using examples executed in the prototypical implementation of CROQUE. The proposed heuristics is independent of the CROQUE project itself and can therefore in principle support any random-based search strategy or be used stand-alone. The heuristics may be integrated seamlessly into most of the present rule-based query optimizers without almost any effort.

Certainly, the discussed heuristics is not for free. Based on the results presented here, we developed another heuristics. Comparing the optimization effort of both approaches obviously leads to a decision against the heuristics proposed here in the case of ad-hoc-query optimization. But considering the relative strengths of both approaches we are thoroughly convinced that merging the ideas behind the two heuristics will result in a yet more powerful outcome. We will have a short look on the mentioned second approach that is presented in more detail in [KPH98a, KPH98b] and discussed in [KPH98c]. The fusion of both heuristics will only be sketched in this paper since this is an open problem we are just planning to address in our future work.

In the following presentation, we restrict ourselves to the traditional two level approach consisting of a logical and a physical algebra (as introduced in [GD87]), since the calculus rewriting is done in a more goal-oriented fashion just using a small set of rewriting rules.

The paper proceeds as follows. We begin in Section 2 by discussing the given search space and the adaptations that were necessary for being able to implement the search strategies in our CROQUE prototype. In Section 3, we present our heuristics and some experimental results obtained from examples executed using the CROQUE prototype implementation. We compare our work with related work in Section 4 focussing on the suitability of applied optimization effort. Finally, we conclude in Section 5 also giving an outlook on further work in the CROQUE project.

2 Search in CROQUE

Query rewriting on logical algebra expressions potentially spans a space of equivalent query expressions and additionally a space of equivalent execution plans, if also a physical rewriting is realized and even physical execution alternatives are considered (as e.g. done in CROQUE with respect to materialized views [GGMS97]). This space of equivalent solutions has to be traversed as fast and as cheap as possible by means of search strategies to find a plan as quickly as possible.

In most cases, it is not really necessary to find the best of all equivalent plans but we are interested in avoiding the selection of the worst plans, i.e. we are satisfied if a sufficiently cheap plan is found. Moreover, sometimes it is yet not possible to find the best of all plans in an acceptable amount of search time.

Therefore, we are searching for the plan that has to be used for evaluating the given user query. The process of searching for this plan in CROQUE as presented here is done after the search space has been generated completely. The decision to do so was made because it is easy

to implement the search in this way thus allowing for a rapid prototyping as well as allowing to demonstrate and verify our heuristics easily.

In the following subsections we first have a look on the shape of the search space in CROQUE, then concentrate on presenting the adaptations of the used common search strategies necessary to meet the requirements of this special search space, and finally summarize our experiences with using these search strategies in our prototype system.

2.1 The Search Space

Essentially, we identified three possibilities to select the plan that has to be used for evaluating the given user query:

- *Exhaustive generation of the whole space and then using a search strategy to select a plan.* The exhaustive generation not followed by the use of a search strategy is not considered as a possibility of its own here because the space may be very large and therefore the plan assessment according to the cost model might be too expensive since cost models tend to use very complex functions.
- *Heuristic generation of the search space.* This alternative describes a pruning of the potential space during its generation, thus cutting off some subspaces on the basis of heuristic information. The aim is to be able to completely ignore some less promising subspaces and therefore to reduce the necessary effort and get a more manageable space. In this way, it might even become unnecessary to use a search strategy. Being cautious, we assume that a search strategy still is necessary. This is important from our point of view, because there is no guarantee that the used heuristics works well in each case and it may thus happen that a space generated in this way is too large and not really manageable without using a search strategy.
- *Cost-based generation of the search space.* Actually this means a plan assessment according to the given cost model during the space generation. A search strategy may only be used to decide which rewriting alternatives have to be considered in each step, i.e. in which way the potential search space is traversed. Like the heuristic generation, only subspaces are traversed but the cost-based generation already results in the selection of exactly one plan, thanks to the fact that the phases of generation and search as well as the plan assessment are integrated into one single algorithm.

The last kind of solution was implemented in EXODUS [GD87] but has shown there to be inefficient and ineffective at least for this special implementation. Therefore, we do not pursue this alternative at the moment. Further research concerning this topic is left for future treatment trying to overcome the inefficiency problem.

The exhaustive generation offers the advantage of supporting rapid prototyping and the evaluation of the implemented search strategies as already mentioned. Moreover, a close inspection of the whole search space is possible with negligible effort resulting in a better understanding of how the strategies really work. It provided us with some insights into the weaknesses of the strategies we took into consideration. Beyond this, we were able to compare the strategies on the basis of some different search spaces resulting in an assessment of space shapes.

The obvious drawback of the exhaustive generation is the effort necessary to generate all possible equivalents. Investigating large spaces as needed for the evaluation of some strategies

leads to an enormous consumption of resources. Therefore, the heuristic generation should be preferred if an ad-hoc-query optimization is desired.

Since we are concerned with an offline-evaluation of the search strategies, we nevertheless decided to start with an implementation of the exhaustive generation. The following reflection supported our decision: extending this approach in direction of a heuristic generation is still possible without further work. Thus, we got in the position to be able to combine the advantages of an exhaustive generation with the possible extension of this implementation to the very efficient heuristic generation without losing results of work done before.

We designed our search space to have two dimensions according to our heuristics presented in Section 3: one dimension of equivalent logical algebra expressions and a second dimension spanned by equivalent physical plans belonging to each logical expression. In the first prototype of our query optimizer, which provided the basis for the discussion presented here, there have been no physical choices, i.e. each logical expression was transformed in a one-to-one manner into a physical plan. In effect, our search space only consists of one plan dimension. This shape of search space results in the need to adapt the used common search strategies.

2.2 Adaptation of Common Search Strategies

Our first prototype has been implemented using the optimizer generator described in [FMS93]. This optimizer generator is based on CRML [She93, HS93], an extension of the functional language SML [Har93, MTH90, Pau96].

In [SMK93], the well-known search algorithms are divided into four categories: deterministic, random-based, genetic, and hybrid algorithms. We started investigating random walk, iterative improvement, simulated annealing, and two-phase optimization as random-based search strategies. These four have been implemented and are also presented in the following subsections. Thereafter, we decided to have a look on the class of search strategies called genetic algorithms. Genetic algorithms are exceptionally well suited to traverse large search spaces of higher dimensions [SP96] but the realization of this strategy is not possible straightforwardly in contrast to the other strategies used.

Adapting genetic algorithms requires coding the query plans in a suitable fashion. Such a coding function may easily be defined for relational join order optimization but is rather less obvious for complex query expressions as treated in CROQUE. Thus we renounce on further discussing genetic algorithms and concentrate our attention on the adaptation of the random-based strategies to the one-dimensional shape of our search space.

Within all of the algorithms treated in the following, branch-and-bound pruning and hashing of already visited plans (or marking of those plans) can be done. Therefore, we only have to consider plans that are within a given cost limit (mostly: cheaper than plans considered earlier). Moreover, this ensures that no plan is considered twice.

2.2.1 Random Walk (RW)

This is the most simple search strategy: in every step, one plan is randomly chosen. The choice does not depend on any former steps. The best plan found during the search process is delivered as result. The search may be restricted by a time limit or through the maximum number of plans visited (either a constant value or depending on the search space size, i.e. defined as a percentage value of this size).

RW could easily be implemented in CROQUE by generating a random value and then selecting the plan at the position specified by this value from the list of equivalent plans.

2.2.2 Iterative Improvement (II)

Iterative improvement (e.g. described in [Kan91]) starts at a random state in the search space and improves the solution by repeatedly accepting downhill moves (visiting neighbored plans characterized by lower costs) until it reaches a minimum. This is assumed to be a local minimum, if after a given number of trials there has not been found a neighbored plan with lower costs. It is worth to be noted that this common algorithm may even fail in selecting a local minimum.

The adaptation of II is done in the following way: plans are neighbored to each other iff they are at neighbored positions in the list of equivalent plans. After starting at a random state, it is decided by a random choice to go left or right in the list of equivalent plans. If the algorithm is not able to accept the first plan in the chosen direction due to its higher cost value, the other direction is pursued. In contrast to the generic algorithm our approach does guarantee that a local minimum is found at least.

2.2.3 Simulated Annealing (SA)

The algorithm simulated annealing [IW87, Ing95] may be imagined by a ball jumping in a landscape (e.g. represented in Figure 1). Like II, SA goes down the (cost) hills but SA also does accept uphill moves with a certain probability. This probability is a function depending on the number of moves done before, steadily decreasing by time.

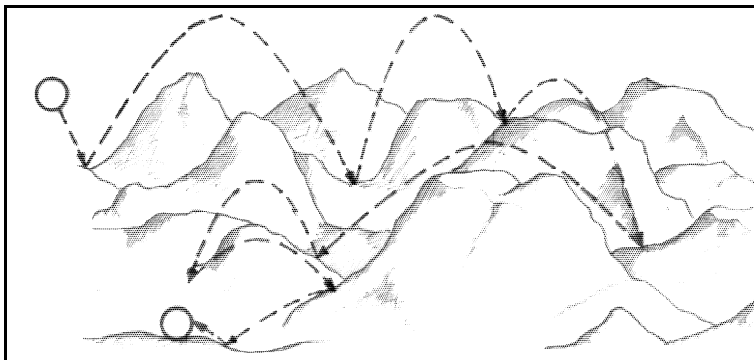


Figure 1: SA: ball jumping in a landscape

SA originates from the idea to reproduce the process of crystallization. Starting from a certain temperature a liquid is gradually cooled down to a minimum value. Purity and size of the produced crystals depend on the temperature development. The main characteristics of this method are the starting temperature and the cooling factor corresponding to starting energy and energy loss in the more figurative imagination of a jumping ball.

We defined the starting temperature to be twice the starting plan cost value. The cooling factor was defined depending on the starting temperature, the actual number of equivalent plans in the space, and the percentage of plans that shall be visited in maximum.

The implementation of SA on a one-dimensional search space is very similar to the one of II. Whenever a move in the chosen direction is not possible in the first step, the other direction is

pursued. The algorithm stops if further moves are not possible. The number of moves is used to control the probability of accepting uphill moves. Similar to II, SA is guaranteed to find a local minimum at least.

2.2.4 Two-Phase Optimization (2PO)

Two-phase optimization [Kan91] is a combination of the two strategies II and SA. In the first phase, II is used to find a local minimum and in the second phase SA searches the surroundings, being able to do some uphill moves (refer to Figure 2).

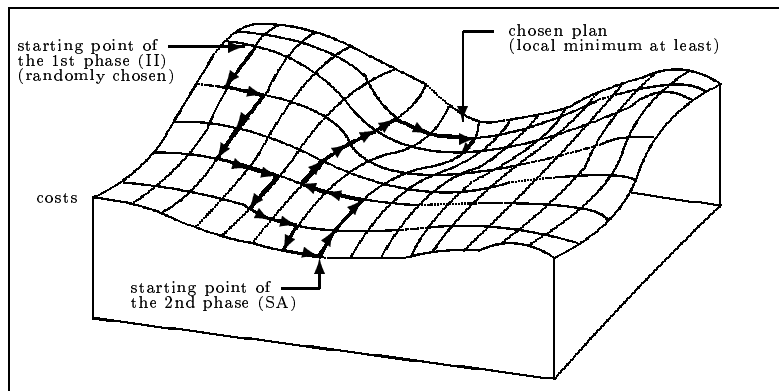


Figure 2: Two-phase optimization

2PO may be implemented for one-dimensional search spaces by only prolonging the starting downhill phase of SA, i.e. not considering these moves in the question whether being able to do uphill moves or not. In effect, 2PO would degenerate to a special version of SA. A more favourable solution is to keep track of the right- and leftmost plan visited in the current run. This enables a search in both directions in every step. In principle, this could also be done in SA, thus enlarging the potentially traversed space. The drawback of this kind of solution is the additional overhead demanding for the maintenance of global variables.

2.3 Summary

We search for the plan that has to be used for evaluating the given user query. This search is done after the search space has been generated completely. Due to the lack of physical choices in our first prototype, our search space has only one plan dimension. This shape of search space results in the need to adapt the used common search strategies.

As could be seen from the above discussion, the adaptation of the strategies is not as complicated as was feared first. This is due to the fact that the definitions of these strategies are universally valid since they are given in generic terms. By determining how the term “neighbourhood” should be defined it is possible to change the generic strategies into algorithms for every kind of search space shape.

Our first prototype comprising the random-based search strategies random walk, iterative improvement, simulated annealing, and two-phase optimization has been implemented using an optimizer generator [FMS93] based on CRML as described in [Ros96, Ill96].

The investigations done using our implementations led to the following observations according to the shape of given search spaces.

- RW is not really satisfying in most cases.
- Search spaces containing some plateaus of the same cost value are not good for II.
- If the cost values mostly alternate, II degenerates to the most simple strategy, RW.
- SA and 2PO are very well suited for large search spaces containing many local minima.
- Search spaces comprising some plateaus of the same cost value and spaces containing cost values alternating most times are bad for SA and 2PO.
- 2PO is more complex but its usage has priority over SA.

These observations motivate our interest in finding heuristics supporting the search as the one presented in the following section.

3 Our Heuristics

All search strategies except random walk heavily depend on the shape of the given search space. A given space can be characterized by two properties. First of all, every search space is obviously characterized by its shape, i.e. the cost value distribution and the number of dimensions. In general, it is not possible to ensure a specific mix of plateaus of the same cost value and the distribution of increasing or decreasing cost values. The number of dimensions may be easily fixed as every value in between one and the maximum number of rewriting rules that are applicable in parallel to a given query expression. Nevertheless, the advantage of a high number of dimensions is questionable as can be seen from the above description of search strategies. At least with regard to random-based search strategies, a one-dimensional search space is more promising since the discovery of local minima can be guaranteed. The shape of a space is none the less hard to influence in a goal-oriented manner.

Another crucial point characterizing a space is the definition of the neighbourhood relation inherently influencing the shape of the space due to its effect on the space generation.

Our heuristics is based on the idea to define a suitable neighbourhood relation whose effect on the space is exploited by a skillful choice of search strategy starting points. In CROQUE, these starting points are the random numbers consumed by the random-based search strategies. All in all, we thus control the search to be as effective as possible.

In the following subsections we first introduce our heuristics. Afterwards, we demonstrate its successful use by some examples, showing how the search may be controlled to become much more effective. We finally summarize our experiences in heuristically controlling the search, thereby also discussing the known drawbacks of our approach.

3.1 The Idea

Often, the neighbourhood relation is directly defined by the application of rewriting rules: two plans are neighbored iff the first plan is the result of applying one rule to the second plan. Whenever two neighbored plans are considered, it is assumed that they are very similar by shape and costs. But this neighbourhood relation is neither complete nor always correct. The

same plan may be produced by very different transformation steps. Moreover, the described assumption fails for very effective rewriting rules.

We propose a slightly different way of defining the neighbourhood relation not solving these two problems but being surprisingly powerful as will be shown in Subsection 3.2 by examples.

In general, equivalence rules may be applied in both directions. Since in most cases a heuristically preferred direction of application can easily be distinguished, we only consider directed rules (transformation rules) in the framework of our CROQUE project similar to the VOLCANO approach [GM93]. The idea of our heuristics is that using more rules for generating a plan will result in a better plan than using less rules for generation because the rules are only applied in the most favourable direction. More rule applications, i.e. improvements, thus result in better plans.

Therefore, our neighbourhood relation is based on the number of rule applications: two plans are closely neighbored if they are produced by the same number of transformation steps.

Implementing this heuristics is done by ordering the search space during its generation according to the number of realized transformation steps. Every plan therefore contains an integer attribute expressing the number of rules successfully applied during the generation of this special plan. The produced plans are ordered by decreasing factors, i.e. the heuristically better plans are in front (or left) and the original plan is the rightmost one.

Instead of choosing pure, uniformly distributed random values as search strategy starting points, the ordering of plans is exploited by using random values obeying a triangular distribution. By this distribution, the more promising (i.e. ordered more left) plans are privileged.

Our experimental results show some performance gains, i.e. the implementation of our heuristics is more effective than search not supported by our heuristics.

3.2 Experimental Results

In this subsection we present some experimental results gained using the implementation of the concepts described above. Due to space restrictions we will neither mention details of our cost model nor present the queries but only use some search spaces for illustration purposes. The results presented here have been confirmed by further tests.

The search strategy evaluation was done on the basis of three criteria:

- How often does the search result in finding the global optimum?
This is an essential criterion expressing the quality of the search. Using random-based search strategies, the result of this criterion depends on the shape of the space, the implemented algorithm, and the generated random values, i.e. especially the quality of the random value generator.
- Of which quality are the other plans selected for execution?
It is very unlikely traversing only a small part of the search space anyway finding the global minimum every time. Therefore, we are also interested in how often the next best plans are selected. Although the global optimum may have been missed, search results and therefore search strategies, too, may be rated high if a plan nearby the optimum is found most times.
- How many plans have been assessed?
The number/percentage of plans being visited during search as well as the development of

this number if the search space is enlarged may be used to estimate the overall effort of the strategy.

The used search spaces are represented graphically (in Figures 3 and 4) by drawing curves connecting the discrete cost values belonging to the listed plans. Both figures consist of two such curves: in case a) plans produced by the CRML optimizer generator are registered in the search space just in the order they were generated. In case b) the correct ordering according to our heuristics is ensured during list generation. The number of transformation steps done during production of each plan is noticed on the x axis of type b) figures. In all four figures, the best plan is marked by a circle. In type b) figures, an additional line is drawn, showing how the local minima get worse to the right hand side of the figures.

Assessing the algorithms is done by carrying out one thousand calls of each algorithm on the unordered and one thousand calls on the ordered search space. The results of these tests are summarized in tables by counting how often plans of which cost value have been selected for execution.

This means, in the tabular representations (in Table 1, 2 and 3) the plans are grouped according to their cost values. The first column (“rank”) is used to rank the cost values. Thus, rank value one indicates the global optimal plan(s) found in each space. The column “plans/rank” shows the number of plans per rank, i.e. the number of plans for which the same cost value has been estimated. All the table entries indicate hit rates, i.e. the number of how often plans of this rank have been selected from the considered search space (not ord. = unordered space; ordered = space ordered according to our heuristics) by the considered search strategy (RW, II, and SA).

In the following subsections, three search spaces are used to investigate our heuristics in combination with different search strategies.

3.2.1 Search space containing 44 plans (RW, II, and SA)

The search space contains two global optimal plans. These are very close to each other in Figure 3 a) and b). Rewriting the starting plan uses five rules at most. The additional line in Figure 3 b) shows that the local minima at the left hand side are of smaller values than those on the right hand side except for the two optimal plans that have been produced by only three transformation steps.

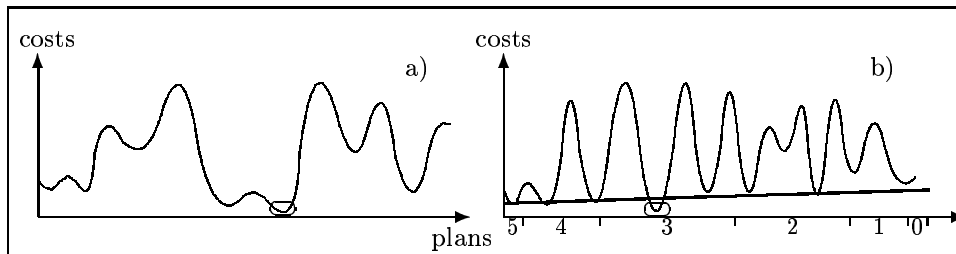


Figure 3: a) Unordered and b) ordered search space containing 44 plans

Having a look on Figure 3 b) exhibits that there are six more local minima having cost values insignificantly higher than the minimum value.

Table 1 presents the search results for all three search strategies. Random Walk and Iterative Improvement both contain columns labeled “RV” and “ Δ RV” subdividing the search on an ordered space into two cases using uniformly distributed random values and triangular distributed random values, respectively.

rank	plans/ rank	RW			II			SA	
		not ord.	ordered		not ord.	ordered		not ord.	ordered
			RV	Δ RV		RV	Δ RV		
1	2	295	279	333	237	339	319	439	248
2	4	405	395	462	308	332	409	356	323
3	2	106	113	205	19	81	272	30	429
4	2	77	79	0	135	58	0	86	0
5	2	34	47	0	10	64	0	25	0
6	2	44	24	0	9	63	0	14	0
7	2	15	24	0	145	5	0	28	0
8	2	13	21	0	0	11	0	0	0
9	2	7	8	0	0	0	0	0	0
10	2	2	6	0	84	21	0	12	0
11	2	2	2	0	0	21	0	1	0
12	2	0	1	0	45	5	0	9	0
13	2	0	1	0	1	0	0	0	0
14	2	0	0	0	7	0	0	0	0
15..21	14	0	0	0	0	0	0	0	0

Table 1: Search result on the space of Figure 3

Comparing the unordered case and the RV column of RW shows the fact that an ordering of plans does not worsen the situation because there is nearly no effect in all. The same holds for II, too. Doing the last step using Δ RV in comparison to RV or the unordered space emphasizes the advantage of our heuristics. All three search strategies clearly benefit from the plan ordering if triangular distributed random values are used since they all result anytime in plans of the best three cost ranks thus avoiding the selection of worse plans.

Note that plans of some cost ranks (e.g. cost rank nine for II in the RV column of Table 1) are never selected neither by II nor by SA. This will happen every time if all the plans of this cost rank are local maxima, since both algorithms ensure the discovery of local minima at least.

Random Walk was adjusted to visit 15% of all plans. II visited 23% and SA 25% (eleven plans) in average. The reason for the higher number of visited plans is that both, II and SA, were programmed by two loops, whereby only the outer one was controlled by a breaking condition related to the overall number of visited plans (15% was the breaking condition here, too). Moreover, the search space investigated by this test is really very small. Using some spaces consisting of more plans showed that II and SA approximated the 15% border value well.

Therefore, we stress that the search result may be positively influenced by exploiting heuristic information to control the search as our approach does according to the number of transformation steps.

3.2.2 Search space consisting of 224 plans (only RW and II)

This search space (Figure 4) contains two global optimal plans that are very close to each other in both figures, too. There are six plans whose cost values are estimated to be very close to the global optimum. The rewriting takes 6 rules at most. The additional line in Figure 4 b) shows, that the local minima at the left hand side are of smaller values than those on the right hand side except for the two optimal plans that have been produced by only four transformation steps.

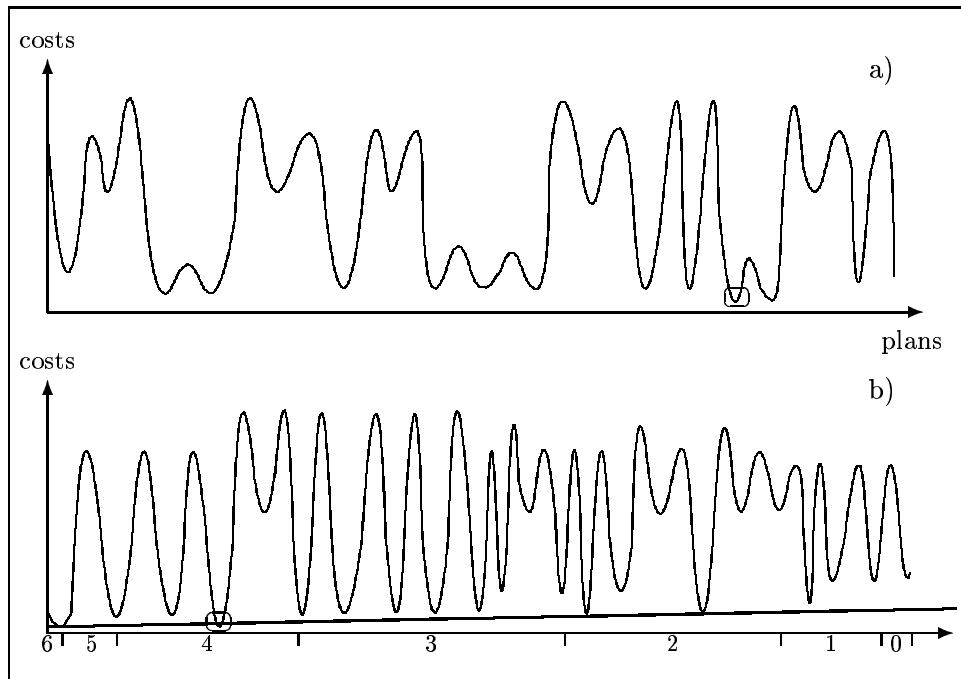


Figure 4: a) Unordered and b) ordered search space containing 224 plans

Again, the tabular representation (in Table 2) indicates the superiority of our approach. This example is yet more expressive because the search space comprises more plans whereas the plan selection according to the ΔRV column again yields one out of the three best cost values every time.

Moreover, the quality of the result is better every time. A plan of the best rank is found much more often than in the unordered case or by the use of uniformly distributed random values. This is also valid for the second-best and the third-best plan in both cases, RW and II.

Again, Random Walk was adjusted to visit 15% of all plans. Now, II visited 17% of the plans in average. Thus, the inner loop running some time along plans of the same cost value is of minor effect in contrast to the outer loop.

3.2.3 Another search space consisting of 224 plans (only SA)

The search space investigated here is slightly different than the one in the last subsection due to minor changes to the cost model and the rule base but the results are the same as presented before confirming all our statements given above. Table 3 presents the search result of SA.

rank	plans/ rank	RW			II		
		not ord.	ordered		not ord.	ordered	
			RV	Δ RV		RV	Δ RV
1	2	260	260	408	196	192	261
2	4	388	364	410	163	257	365
3	2	94	108	182	7	111	374
4	2	67	86	0	72	72	0
5	2	51	61	0	9	6	0
6	16	68	54	0	272	151	0
7	8	37	40	0	0	0	0
8	8	35	26	0	179	145	0
9	8	0	1	0	9	0	0
10	2	0	0	0	56	30	0
11	2	0	0	0	5	22	0
12	1	0	0	0	0	3	0
13	8	0	0	0	7	0	0
14	8	0	0	0	5	1	0
15	2	0	0	0	2	6	0
16	2	0	0	0	15	4	0
17	1	0	0	0	0	0	0
18	1	0	0	0	0	0	0
19	1	0	0	0	0	0	0
20	1	0	0	0	0	0	0
21	1	0	0	0	0	0	0
22	1	0	0	0	2	0	0
23	1	0	0	0	1	0	0
24..45	140	0	0	0	0	0	0

Table 2: Search result on the space of Figure 4

On average, SA visited 40 (17.4%) of the plans in the space. This confirms the claim that II and SA approximate the 15% border value given in the outer loop as breaking condition whenever a larger search space is investigated. This example visibly supports all of our statements since SA selects a plan out of the best two cost ranks in each case.

It is true that SA more often selects a plan of the best cost rank from the unordered space using uniformly distributed random values (30%) than from the ordered space using random values obeying a triangular distributed (19.4%). But at the same time it is true that SA only selects a plan of the best two cost ranks in 56.4% (in comparison to 100% in the other case).

3.3 Summary

As illustrated by the given examples, exploiting our heuristics results in selecting better plans more often than this would happen without the use of this heuristics. But as can also be seen for these “good” search spaces, the pure number of rule applications is not sufficient as a means for plan ordering since the global optimal plan never was the one created using the greatest number of rule applications. This is due to the fact that in this approach all rules are weighted uniformly which does not seem very naturally. How to overcome this inaccuracy is described in [KPH98a, KPH98b].

rank	plans/ rank	SA	
		not ord.	ordered
1	2	300	194
2	4	264	806
3	2	38	0
4	2	98	0
5	2	19	0
6	4	197	0
7	4	8	0
8	16	66	0
9	8	2	0
10	8	6	0
11	8	0	0
12	2	0	0
13	2	1	0
14	1	0	0
15	8	1	0
16..45	151	0	0

Table 3: Search result on a search space slightly different than the space presented in Figure 4

The advantage of our heuristics is the ease of implementation and its power. In fact, the plan ordering corresponds to a depth-first traversal of the potential search space. The obvious main drawback of our approach is the necessarily exhaustive search space generation. For this, a solution avoiding the demand to first span the whole space is sketched in the next section.

4 Related Work

[ONK⁺95] explains how the complexity of optimizing queries with many selection predicates is comparable to join sequence optimization in relational databases. Because there are many rules describing commutativity and selection pushing, every selection predicate may be arranged at nearly every place in a query. Thus, arranging these predicates is a combinatorial problem like, e.g. join sequence optimization. This justifies the need for far-reaching optimization in the object-oriented context, too.

Until now, we only considered the number of transformation steps that were necessary to generate a plan. This method only seems to be a good choice if the optimization is done offline, i.e. the rewriting is decoupled from query execution. In this case, the time spent for searching the plan that has to be executed is nearly irrelevant. The implementation discussed in Section 3 meets this requirement. There, we focused on the evaluation of search strategies and this task is independent of the time needed to answer the query.

Changing the perspective, a corresponding approach to online (ad hoc) query optimization would consider the quality of the applied rules instead of the quantity of rule applications. The method described in [KPH98a, KPH98b] heuristically orders rewriting rules according to their quality (the “optimization potential”) and applies them in this order. This is a more promising way for an online optimization where optimization time and query execution time are added yielding the system’s overall response time. For this method, the most costly normal form as described in [KM94] — that may be generated fast by a naive query transformer — seems to

be a good starting point. The best rules are applied first gaining the best possible rewriting success very fast, too. Therefore, fast query translation and very fast successful query rewriting result in a low response time.

[RL98] is concerned with semantic query optimization using rule graphs. Although this topic is not the same as we are concerned with, both have in common that rules are used to come to a more sophisticated solution. [RL98] states that it is probable that the best reformulated query will be found directly from the rule set rather than by taking longer in query reformulation to search the graph for transitive rules, or build them by successive application of rules to the query. This statement is very similar to the basics of the last-mentioned approach where the rule quality has been used to decide on the rule application and not the application quantity. Therefore, the results of semantic query optimization are partly transferable.

Cascades [Gra95] proposes an offline transitive closure computation to exclude rules from consideration. Therefore, this optimizer generator introduces promise functions to guide the search for the rules that have to be applied. This approach is very similar to ours but in [Gra95] the essential step of defining such functions is left to the database implementor without giving any hint how to do so.

4.1 Suitability of Optimization Effort

Offline and online optimization are uncomparable with respect to optimization effort. Considering the offline optimization effort actually prohibits the construction of a query response time since the purpose of this kind of optimization is, by definition, only to select the best plan with no respect to the optimization time. Therefore, the optimization effort is suitable if the best result (or a nearby value) is found.

The effort of an online optimization is suitable, if a good plan is found quickly. This may be characterized by the following reflection: optimization is never for free. If a query shall be optimized online, we first have to decide on the effort we are willing to invest in the worst case. This worst case happens if the straightforwardly assigned plan is the best of all in a large space of equivalent plans. Then, every attempt to optimize the query is superfluous.

[ONK⁺95] precedes as follows. The time for an exhaustive (offline) search combined with the estimated execution time for the selected plan is compared with a heuristic (online) search effort combined with the estimated execution time of the plan selected in this way. Clearly, the exhaustive search performs worse than the heuristic search. In most cases, exhaustive search will find a better plan than heuristic search. Whenever a plan has to be executed repeatedly the exhaustive search may therefore be more valuable (assuming that the plan found in this way is stored for further use). The mistake inherent to the described approach raised our interest on the suitability of optimization effort.

Being interested in an online optimization (as we are for our upcoming next prototype), it is important to fix some border values to define the maximum optimization effort. Aiming at an offline optimization only, it would at least be wise to do so, too. We identified two kinds of factors that are both well suited to control optimization effort: the number of plans visited and the time spent for optimization. The algorithm “search effort” in Figure 5 guarantees that the search will break if the number of visited plans or the elapsed search time run out of the given border values.

We may decide to visit a given total number of plans only (`max_absolute_plans` in Figure 5) or a percentage value of the number of plans in the space (`max_percentage_plans`). Another

criterion breaking the search should be the total time spent for searching (`max_absolute_time`). Additionally, we define a factor called `max_percentage_time` indicating that percentage of the estimated execution time of the best plan found, that we are willing to invest in searching.

```

max_absolute_plans = w; // number_of_plans > w > 0
max_percentage_plans = x; // x somewhere in [0..1]
max_absolute_time = y; // y > 0
max_percentage_time = z; // z somewhere in [0..1]

max_plans_to_be_visited =
    min {max_absolute_plans, number_of_plans * max_percentage_plans};

initialize (search_space);
starting_plan = chose_starting_plan_randomly (search_space);
// OR straightforwardly_assigned_plan;
best_plan_found = starting_plan;
number_of_visited_plans = 1;
optimization_time = 0;

while max_absolute_time > optimization_time and
    e_e_time (best_plan_found) * max_percentage_time > optimization_time and
    max_plans_to_be_visited > number_of_visited_plans
do {
    plan_found = search (strategy, starting_plan, search_space,
                        number_of_visited_plans);
    if e_e_time (plan_found) < e_e_time (best_plan_found)
        best_plan_found = plan_found;
    starting_plan = chose_starting_plan_randomly (search_space);
    update (optimization_time);
}

```

Figure 5: Algorithm “search effort”

The function `e_e_time()` estimates the execution time of a plan, i.e. is declared by our cost function. `search()` is called to do one run of the search `strategy` beginning at the `starting_plan` in the `search_space` updating the `number_of_visited_plans` by a side effect.

Fixing the values named `w`, `x`, `y`, and `z` in Figure 5 allows for an adaptation of the algorithm for both kinds, offline and online optimization. In this way, we are responsible for the suitability of the optimization effort by our own, not leaving the important decision about this problem to the optimizer. The algorithm is an appropriate means assuring the suitability of the optimization effort with respect to the mentioned factors.

Note, that during the increase of optimization effort done — usually accompanied by finding better plans — the remaining optimization time decreases because `e_e_time (best_plan_found)` is computed to decide on breaking the search. The method converges by the defined factors.

5 Conclusion and Future Work

We presented the use of search strategies in the CROQUE project. The adaptation of some common strategies according to the characteristics of our search space led to a very simple but surprisingly powerful heuristics that benefits have been demonstrated by a detailed discussion of some examples executed in the prototypical implementation of CROQUE.

The considerations about related work focussed on the suitability of applied optimization effort. The discussion resulted in the presentation of an algorithm assuring the suitability of the optimization effort with respect to the defined factors.

Whereas the proposed heuristics based on the quantity of rule applications may only be used after the rewriting has completed, the heuristics presented in [KPH98a, KPH98b] developed to care more for the quality of the applied rules may be used to control the rewriting. Therefore, a pruning may be achieved during search space generation. This is advantageous but also a much more complex task. To overcome the inherent inaccuracies of both approaches we plan to fuse them. In this way, a more natural consideration of a combination of quality and quantity shall be achieved.

The handling of commutativity rules still is a challenge in both approaches because those rules may have no effect in some algebra only allowing further rules to be applied afterwards.

We also plan an adaptation of the optimizer system by the help of selected statistics. Histograms as presented in [IP95] for estimation problems seem to be a good basis for such an adaptation.

Acknowledgements: We would like to thank our CROQUE project partners Marc H. Scholl, Torsten Grust, and Dieter Gluche at the University of Konstanz for lots of discussions and hints improving the concepts described here and the quality of the paper, our students Ralf Asmus, Holger Janz, Stefan Paul, Beate Porst, and Denny Priebe for implementing most parts of the first two CROQUE prototypes, and our colleague Jürgen Schlegelmilch giving many hints further improving the paper.

References

- [Cat96] R.G.G. Cattell, editor. *The Object Database Standard: ODMG-93, Release 1.2*. Morgan-Kaufmann, San Mateo, CA, 1996.
- [FMS93] L. Fegaras, D. Maier, and T. Sheard. Specifying Rule-based Query Optimizers in a Reflective Framework. In *Proc. of the 3rd Int. Conference on Deductive and Object-Oriented Databases*, pages 146–168, New York, December 1993. Springer.
- [GD87] G. Graefe and D.J. DeWitt. The EXODUS Optimizer Generator. In *Proc. of the ACM SIGMOD Int. Conference on Management of Data*, pages 160–172, San Francisco, CA, USA, May 1987.
- [GGMS97] D. Gluche, T. Grust, C. Mainberger, and M.H. Scholl. Incremental Updates for Materialized Views with User-Defined Functions. In *Proc. of the Fifth Int. Conference on Deductive and Object-Oriented Databases (DOOD'97), Montreux, Switzerland, LNCS 1341, Springer*, pages 52–66, December 1997.
- [GKG⁺97] T. Grust, J. Kröger, D. Gluche, A. Heuer, and M.H. Scholl. Query Evaluation in CROQUE — Calculus and Algebra Coincide. In *Proc. of the 15th British National Conference on Databases (BNCOD 15), London, UK, LNCS 1271, Springer*, pages 84–100, July 1997.

- [GM93] G. Graefe and W.J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *IEEE Conference on Data Engineering 4/1993*, pages 209–218, Vienna, Austria, April 1993.
- [Gra95] G. Graefe. The Cascades Framework for Query Optimization. *Bulletin of the Technical Committee on Data Engineering*, 18(3):19–29, September 1995.
- [Har93] R. Harper. *Introduction to Standard ML*. School of Computer Science, Carnegie Mellon University, Pittsburgh, 1993.
- [HK96] A. Heuer and J. Kröger. Query Optimization in the CROQUE Project. In *Proc. of the 7th Int. Conference on Database and Expert Systems Applications (DEXA '96), Zurich, Switzerland, LNCS 1134, Springer*, pages 489–499, September 1996.
- [HS93] J. Hook and T. Sheard. A Semantics of Compile-time Reflection. Technical report, Oregon Graduate Institute, Portland, Oregon, 1993.
- [Ill96] R. Illner. Employment of Simulated Annealing for a Cost-Based Optimization of Object-Oriented Queries. Master's thesis, CS Dept., University of Rostock, June 1996. In german.
- [Ing95] L. Ingber. *Adaptive Simulated Annealing*. Lester Ingber Research, McLean, 1995.
- [IP95] Y. Ioannidis and V. Poosala. Histogram-Based Solutions to Diverse Database Estimation Problems. *Bulletin of the Technical Committee on Data Engineering*, 18(3):10–18, September 1995.
- [IW87] Y.E. Ioannidis and E. Wong. Query Optimization by Simulated Annealing. In *Proc. of the 13th ACM SIGMOD Int. Conference on Management of Data*, pages 9–22, San Francisco, CA, USA, May 1987.
- [Kan91] Y.C. Kang. *Randomized Algorithms for Query Optimization*. PhD thesis, University of Wisconsin, Madison, October 1991.
- [KJH98] J. Kröger, H. Janz, and A. Heuer. About Reducing the Costs for Cost Calculation in CROQUE. Preprint, CS Dept., University of Rostock, 1998. In preparation.
- [KM94] A. Kemper and G. Moerkotte. Query Optimization in Object Bases: Exploiting Relational Techniques. In J.C. Freytag, D. Maier, and G. Vossen, editors, *Query Processing for Advanced Database Systems*, chapter 3, pages 63–98. Morgan Kaufmann Publishers, 1994.
- [KPH98a] J. Kröger, S. Paul, and A. Heuer. On the Ordering of Rewrite Rules (Extended Abstract). In *Proc. of the Second East-European Symposium on Advances in Databases and Information Systems (ADBIS'98), Poznan, Poland, LNCS 1475, Springer*, pages 157–159, September 1998.
- [KPH98b] J. Kröger, S. Paul, and A. Heuer. Query Optimization: On the Ordering of Rules. Preprint CS-09-98, CS Dept., University of Rostock, June 1998. URL: <http://wwwdb.informatik.uni-rostock.de/~jo/CS-09-98.html>.
- [KPH98c] J. Kröger, S. Paul, and A. Heuer. Query Optimization: Ordering Rules? Preprint CS-12-98, CS Dept., University of Rostock, June 1998. URL: <http://wwwdb.informatik.uni-rostock.de/~jo/CS-12-98.html>.
- [MTH90] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.
- [ONK⁺95] F. Ozcan, S. Nural, P. Koksai, M. Altinel, and A. Dogac. A Region Based Query Optimizer Through Cascades Query Optimizer Framework. *Bulletin of the Technical Committee on Data Engineering*, 18(3):30–40, September 1995.
- [Pau96] L.C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996.

- [RL98] J. Robinson and B. Lowden. Semantic Query Optimisation and Rule Graphs. In *Proc. of the 5th KRDB Workshop*, Seattle, WA, May 1998.
- [Ros96] S. Rost. Analysis of Alternative Search Strategies for a Cost-Based Optimization of Object-Oriented Queries. Master's thesis, CS Dept., University of Rostock, June 1996. In german.
- [RS97] H. Riedel and M.H. Scholl. A Formalization of ODMG Queries. In *Proc. of the 7th Int. Conference on Database Semantics (DS-7)*, Leysin, Switzerland, October 1997.
- [SGG⁺97] M.H. Scholl, D. Gluche, T. Grust, A. Heuer, and J. Kröger. Optimization of generic operations and storage structures in object databases (working report). Preprint CS-05-97, CS Dept., University of Rostock, April 1997. *Also published as:* Technical Report 32/1997 (Konstanzer Schriften in Mathematik und Informatik, Nr. 32), Faculty of Mathematics and CS, University of Konstanz. In german. URL: <http://wwwdb.informatik.uni-rostock.de/~jo/CS-05-97.html>.
- [She93] T. Sheard. Guide to using CRML – Compile-Time Reflective ML. Technical report, Pacific Software Research Center, Oregon Graduate Institute of Science & Technology, Beaverton, Oregon, October 1993.
- [SMK93] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Optimizing Join Orders. Technical Report MIP-9307, Faculty of mathematics and computer sciences, University of Passau, 1993.
- [SP96] M. Srinivas and L.M. Patnaik. Genetic Search: Analysis Using Fitness Moments. *IEEE Transaction on Knowledge and Data Engineering*, 8(1):120–133, February 1996.