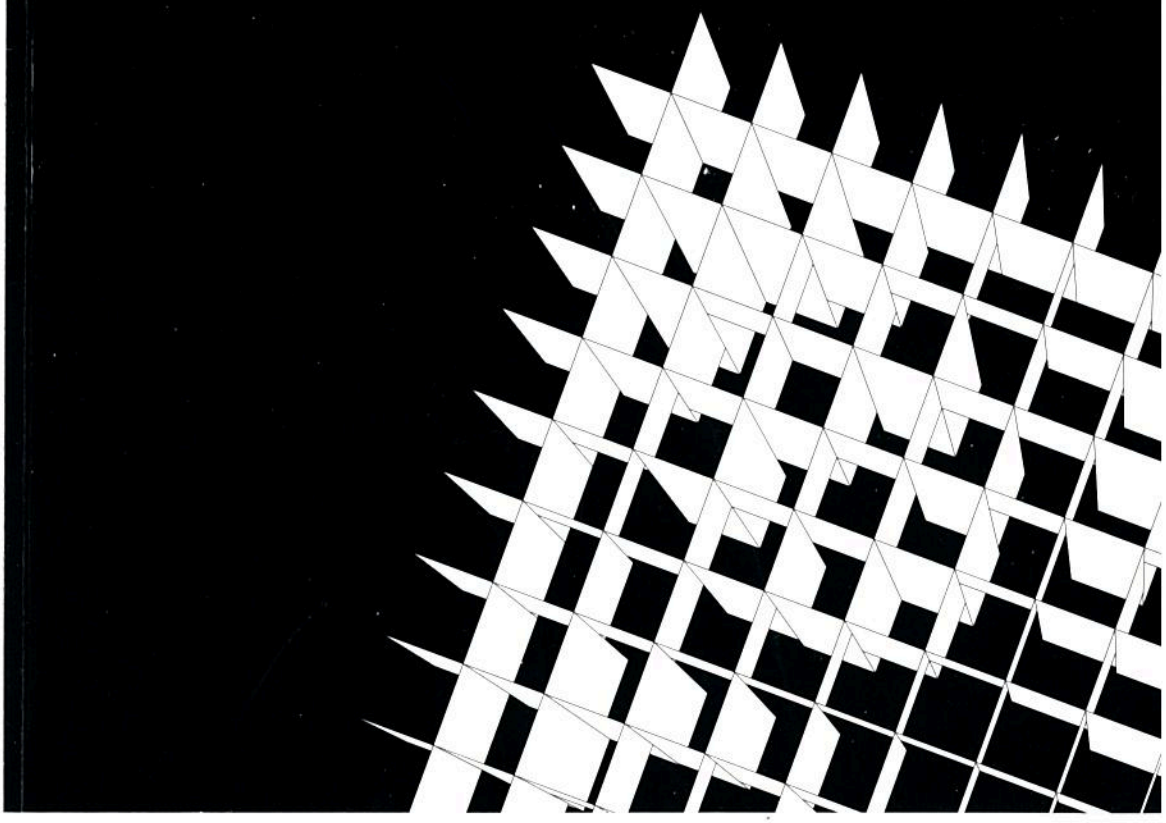
Florian Waas

Principles of Probabilistic Query Optimization



Principles of Probabilistic Query Optimization

Florian Waas

Principles of Probabilistic Query Optimization

Academisch proefschrift
ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam,
op gezag van de Rector Magnificus
prof. dr J.J.M. Franse
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op vrijdag 3. november 2000, te 10.00 uur
door Florian Michael Waas
geboren te Cham

Promotor: prof. dr. M. L. Kersten
Faculteit: Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam
Kruislaan 403
1098 SJ Amsterdam

The research reported in this thesis has been carried out at CWI, the Dutch national research laboratory for mathematics and computer science, within the theme *Data Mining and Knowledge Discovery*, a subdivision of the research cluster *Information Systems*.

SIKS Dissertation Series No. 2000-9

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

ISBN 90-6196-494-6

Contents

Acknowledgements

As you might know from other theses, a book like this is not just *written* by the author—it is rather the product of a highly complicated and tedious process which requires most notably a proper setup and a continuous flow of energy in form of motivation. What matters most are of course the people who make up the kind of biotope a thesis needs for its development. Additional lures like conference travels are as vital an ingredient as are sporadic social events.

Most of all, I'm grateful to my supervisor Martin Kersten who always got me back on track when I lost sight of the goal; he has proven an excellent sounding board. Not last we share the fascination for unconventional solutions. Cesar Galindo-Legaria was the other major source of inspiration. I am proud having written some papers together with them.

I am grateful to Peter Apers, Peter van Emde Boas, Louis Hertzberger, Paul Klint, and Patrick Valduriez for being on my committee. Their detailed comments helped improve the manuscript substantially.

I am indebted to my colleagues at CWI who were a lot of fun to work with, discuss "cultural differences", and study the subtleties of Belgian beer. Continuing the Berlin tradition not only with respect to research was a particular pleasure. Special thanks are also due to the department for game theory for taking care of my social life at countless occasions and to the department for computer-human interaction for visions and enlightenment. The colleagues concerned with my *v*oicing efforts hopefully noticed progress—thanks for ha*v*ing an eye on that. One colleague took the pain to test each single *gelateria* in Northern Italy. The wealth of test data together with detailed footage are truly appreciated. Thanks are also due to the technical support department. Making things better wouldn't be a challenge if there wasn't a sound portion of resistance.

The annual gatherings with old pals from student times were integral part of the project—special thanks to the one of them who talked me into all that.

Bologna, September 2000

1

Introduction

Once a stand alone application databases emerged as an integral part of today's operating systems, be it Linux that comes with the complete Postgres, or future releases of Microsoft Windows have been announced to include some native database support. Other vendors of operating systems announced similar developments. This strategy does not come as a surprise but reflects the enormous penetration of almost all fields of computing by database technology over the past years.

Practically all major applications that involve some kind of data management use database back-ends via application programming interfaces or query language interfaces using standard query languages like SQL. In such an architecture, applications determine and retrieve relevant data sets by posing queries that are ran against the database. Figure 1.1 shows a prototypical architecture of a query processor and its coupling. Let us briefly review the single steps it takes to process a query. The query is submitted in a declarative form in that properties and constraints the data must fulfill are specified but no information how to retrieve the data from the storage is suggested. First, the query is rewritten by a preprocessor, which simplifies the original SQL expression and transfers it into some sort of internal representation. This representation is passed on to the query optimizer who's task is to find a cost effective procedural execution plan for the query. Unlike the declarative query, the execution plan is a procedural description how to retrieve the data. It is composed of operators of the relational algebra which implement the standard set operators but also provide extended functionality like sorting etc. The resulting plan is evaluated by the execution engine which retrieves the data from the storage. Finally, the result data is returned to the client application.

Query optimization is a central task in the processing cycle—selecting an appropriate plan is immensely performance critical. However, the query optimization problem is known to belong to the class of NP-hard problems. Also often referred to as *intractable*, these problems defy any efficient algorithm for a solution. Its financial volume and the widespread use of databases render query optimization one of the most important and most

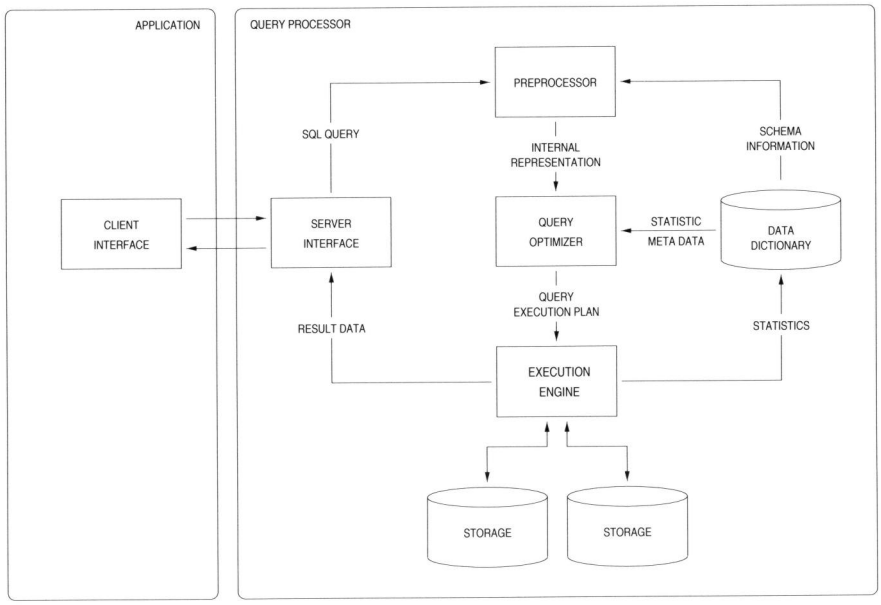


Figure 1.1. Outline of the query processing architecture

frequently tackled NP-hard problems in today's computing. Consequently, it received continuous attention ever since. The proposed techniques for approaching this problem reach from exact methods like dynamic programming [SAC+79] to genetic algorithms [BFI91, SMK97], from various heuristics [KBZ86, SI93] to randomized search strategies like Simulated Annealing [IK90] to mention just a few. Interestingly, only a few heuristics and the exact methods, both in form of dynamic programming and transformation based frameworks, made it all the way into commercial products.

This reluctance—partly based on the monopoly like position of some vendors in the field—has lead many researchers to consider the problem solved by agreeing that typical instances will not exceed the capabilities of exact methods. Especially since the projected query sizes that would require capabilities to optimize large and even very large join queries as predicted a decade ago in the wake of object-oriented databases and decision support systems failed to come.

Nevertheless, the requirements in terms of query size grew gradually but steadily and together with an ever increasing variety of operator implementations today's application often reach the limits of exact optimization methods indeed and even exceeded them. The techniques to face those

challenges reach from restricting the search space to certain kinds of processing plans at the expense of possibly failing to find the optimal and putting up with a sub-optimal yet acceptable plan, to ignoring the problem at all. Therefore, practitioners in the database development business consider the problem anything but solved, as I learned from my visits at Microsoft's SQL Server Group and IBM's Almaden Research Center.

But not only the technical requirements changed, the market situation did so too. With new fields of applications like the world wide web which opened a whole new market segment and new players in the field the competition was enlivened which in turn contributed to a higher readiness to adapt research results much quicker than in the past. Ideas of optimization techniques like randomized algorithms could now possibly fall on fertile ground as opposed to a few years ago.

However, practitioners are cautious with deploying non-exact methods mainly because the records of such algorithms are slightly tainted by inconsistent results published so far. Research papers in this field typically proposed a costing technique, and a framework of a randomized algorithm. The performance of the algorithm would be assessed by corroboration with an individually assembled set of queries—usually comparing the proposed method with other algorithms of this category or techniques used in previous work. However, different cost models and different queries as well as different parameters for the algorithms were used for the comparisons leading not seldom to results contradicting previous research, see e.g. [SG88, IK90]. Such proceeding rendered the superiority of the newly proposed techniques rather a matter of believe. Especially intuitive explanations seem to run the risk of fallacious conclusions.

With regard to a practical deployment of those techniques we need deeper insight in both the principles of the particular algorithms and their mode of action as well as—and this might be of even higher importance—into the basic properties of cost based query optimization and the possibilities of exploiting those features. To this end we develop a search space analysis based on the occurring cost values, that is, we essentially scrutinize the ratio of good to bad solutions to the problem. This approach has been inspired by two recent trends in the field of combinatorial optimization yet independent from query optimization. One is concerned with new ways to determine a problem's difficulty apart from its theoretic worst case complexity, the other deals with the applicability and fundamental properties of blind search algorithms. We briefly describe the highlights of both in the following.

Cheesman *et al.* in a noteworthy work detailed that the typical characteristic of a problem's difficulty by means of classic complexity theory often proofs too coarse a measure [CKT91]. In other words, typical instances of some NP-hard problems are fairly easy to solve despite their theoretical worst case complexity. Prior to this work a similar claim has been made by Turner concerning the very particular problem of k-colorability of graphs [Tur88]. The concept of phase transition, according to which a

problem is characterized by a parameter for which two disjoint ranges can be identified—for values within one of the ranges the problem is almost always easy to solve, for values of the other range, the problem is usually hard—has been applied to a variety of decision problems. As the most prominent example, satisfiability received special attention and the search for the precise numerical value where the phase transition and thus its difficulty toggles has meanwhile become a discipline on its own. While this concept takes the credit of having revived the discussion of a problem's "difficulty", its applicability to problems other than decision problems is unclear. Cheesman *et al.* present a shot at the Traveling Salesman Problem as one of the traditional and probably best understood optimization problems, however, the results are disputable. Though close relatives, decision problems differ from optimization problems when it comes to the notion of difficulty. In the former, there is only one correct answer an algorithm has to find—yes or no. In practice however, optimization problems are of an approximative nature. In most cases a solution close to the optimum is *good enough*. So the difficulty depends on the optimization goal, i.e., it is a function of the distance between the minimal quality required and the optimal solution.

The second trend is the currently vigorously fought debate about the efficacy of blind search algorithms in general. The label blind search covers all those optimization algorithms that do not take advantage from any knowledge about the problem itself but rather require an abstract framework of manipulators like transformation rules and a mechanism to assess the quality of a single solution. The algorithm then tries to find acceptable solutions only by using the manipulators controlled by the feedback from the quality function. Genetic Algorithms and Simulated Annealing are typical representatives of blind search techniques. The No-Free-Lunch theorems by Wolpert and Macready proof that we cannot expect any such algorithm to outperform any other algorithm of this class on average. In other words, if genetic algorithms are the best optimization technique for problem A, then there exists a problem B where another algorithm performs better. While a powerful construct by themselves—provoking an enormous paper trail—those theorems unfortunately do not provide any clues as to what algorithm should be used for a particular problem.

Clearly, query optimization is tightly intertwined with both those fields, the difficulty of NP-hard problems on the one hand, and the foundations of the applicability of blind search algorithms on the other. Thus, this work is concerned with a synthesis of the different concepts not only with respect to its application to query optimization but also providing deeper insight in previous work, explaining the discrepancies observed.

1.1 Research Objectives

Combinatorial optimization problems in practice usually differ from their synthetic theoretical counterparts as e.g. described in [GJ79] in that their objective function is much more complex. In the case of query optimization, a cost function of industrial standard covers a multitude of parameters that are intended to describe both the state of the database and the hardware the query is processed on. The accuracy of the cost model is obviously vital to the successful functioning and therefore belongs to the very core of proprietary code in commercial databases. The more complex the cost model gets, the more difficult its analysis becomes. Commercial cost functions cannot possibly be captured by practical and useful mathematical models and so a simplified, tangible version is needed. We will present several models and discuss the question as to whether the analysis of a simplified model provides insights that are also valid for its complex counterparts. This question is not only of importance for our further considerations but overshadowed most of previous research in this field. We will see that cost distributions, that is, the statistical distribution of cost values in the search space, are characteristic for an optimization problem. Not only do those distributions provide an intuitive measure of difficulty by detailing the ratio of good to bad solutions, they are also the key to an analysis of applicability of certain algorithmic principles and techniques. A large part of this work is concerned with techniques to obtain cost distributions from both the simple models and a commercial database system. Contrasting these results with each other provides also means to validate the simplifications.

Once obtained, we will scrutinize the distributions in order to identify the major features and their generality, i.e., we address the question as to whether the features found depend on the particular instance and if so, to what degree? The question central to this work is: How difficult is query optimization from a practical point of view? We will try to answer this question with respect to different optimization algorithms and the algorithmic principles deployed. Stated differently, we investigate which methods are the most promising ones.

Since we base our research on cost distributions only, the results are not necessarily restricted to the problem at hand but offer the possibility of transfer to other combinatorial optimization problems.

1.2 Organization of this Thesis

This work consists of two major parts. First, after introducing a basic model for the problem in Chapter 2, we review costing techniques and present a brief overview on related work. To obtain cost distributions by enumeration and sampling we develop different techniques that reach from enumeration methods for labeled binary trees and non-isomorphic process-

ing trees to techniques for transformation-based optimizers in Chapter 3. The latter has been implemented in Microsoft SQL Server which allows the analysis of cost distribution in a holistic query optimization context rather than in restricted scenarios which capture for example join ordering only. We conclude the first part with a discussion in Chapter 4 of the cost distributions found and the question of the difficulty of the problem in Chapter 5.

The second part is devoted to the consequences which arise from the preceding results. In Chapter 7 we present a close look at random join ordering and a similar analysis for evolutionary algorithms is given in Section 6. Both Chapters also re-evaluate previous work on the respective topics. The second part is completed with a look at biased sampling of join orders, uniform sampling in the context of unrestricted query optimization, and a practical take on random query optimization in Chapter 8.

Chapter 9 finally summarizes this thesis and contains suggestions for further research.

2

Model for the Problem

As we have seen in Figure 1.1, we distinguish different phases in the processing of a query. The first step, the rewriting and simplification of the original, *declarative* query is, technically speaking, already part of the process of *improving* the query. At this stage, the query is typically specified in a declarative language like SQL. The term *query optimization*, however, usually refers to the subsequent process of converting the result of the rewriting phase into a cost-effective, if possible cost optimal, *procedural* query execution plan which can be evaluated by the query engine. The resulting plan consists of operators of the *relational algebra* which implements set theoretic operations but also contains extensions to facilitate non-algebraic operations like sort orders or the selection of the first n elements of a sorted set. For a detailed discussion of aspects of relational algebra operators see e.g. [AHV95]. The number of extensions varies from one database product to another as does the expressiveness these operators.

The question what an "optimal" execution plan is cannot possibly be answered in simple terms, but will rather accompany us throughout the whole work. We will see different facets of this question in different contexts. However, for the moment an intuitive notion of the optimization goal is satisfactory. Clearly, the effort it takes the engine to evaluate the plan should be as little as possible.[1] On the other hand, we do not want to invest too much time in the optimization. The case that optimization time dominates the query processing costs, which may occur especially for small queries, must be avoided. Thus a trade-off is sought, which takes all costs arising in the whole process into account.

In today's commercial database systems different techniques are deployed to assure a certain balance between the components. IBM's DB2 for example gives the user the possibility to switch off the standard adjustments and control the effort put in the optimization himself. This can be beneficial if the user is a seasoned database expert, but may be diffi-

[1] Though optimization goals other than response time have been proposed in the literature, the response time is clearly the predominant objective.

cult to manage by the lesser experienced user leading to poor results. In the Microsoft SQL Server, developers pursued a different approach using several timeout controlled levels of optimization that produce simple and usually sub-optimal fall back solutions before investigating further optimization possibilities with more and more sophisticated implementations. As opposed to the previous case, no user adjustment is permitted and—in theory—not needed either.

Keeping those practical details in mind we now focus on the optimization process itself. Relentless efforts in this area provided a wealth of research papers addressing all kinds of relational algebra operators and their use as well as specific optimization techniques. It turned out early, that joins, the algebraic operator which combines two database tables—essentially building the cross product and simultaneously applying a filter—, is one of the operators most crucial to the optimization process. Not only does it rank among the most expensive operations in terms of execution costs, regarding both time and resource consumption, it is also one of the most frequently used operations. Every non-trivial query uses more than one table, i.e., requires a join operator in general. Other expensive operations include sorting and aggregation operators, however, joins usually outnumber them by far. Therefore, focusing on the *join ordering problem* is a restriction not uncommon in this field of research [IK84, IK90, GLPK94]. In the further course of this work, we will address the limited problem as well as the unrestricted, general case. Specifically, we will first address the simplified model and try a step-by-step transfer of the knowledge obtained to establish an understanding of the complex case. The following example illustrates the basic concepts.

2.1 Example

Consider a database used to maintain a college's organization consisting of tables that store information about professors, students, and courses (see Fig. 2.1). We omitted further attributes and show only the information used later on. Underscored attributes are keys, e.g. title of a course. For professors and students we store an id and the name. The courses table comprises title and the id of the lecturer who is giving this course. The last table, contains the information what student is enrolled in what course.

To find out what professors the student "Sam White" meets the next term, we pose the following query:

```
SELECT *
FROM   Professors P, Students S, Enrolled E, Courses C
WHERE  S.Name = "Sam White" AND
       S.SID = E.SID AND
       E.Title = C.Title AND
       C.By = P.PID
```

Professors

PID	Name
1023	Jeff Barnes
1035	Jerry Smith
1036	Daryll Smith
1037	Allen Steward
1039	Mary James
...	...

Students

SID	Name
43023	Dave Parker
43924	Sam White
44101	Jeniffer Clerk
44102	Susan Lowry
44105	Fred Hanson
...	...

Courses

Title	By
Databases	1023
Game Theory	1039
VLSI Layout	1039
Graph Theory	1037
Topology	1036
...	...

Enrolled

Title	SID
Databases	43023
Databases	44102
Graph Theory	43023
Topology	44102
VLSI Layout	44105
...	...

Figure 2.1. Database schema

We translate this query into a procedural execution plan using binary join operators that combine tables pairwise applying a predicate as a filter. This corresponds theoretically to building the Cartesian product of both tables and filtering the result afterwards. Practically, however, a join can be implemented in more sophisticated ways using indexes, sort orders or hash tables. For a detailed discussion of possible implementation techniques see for instance [Gra93]. Figures 2.2a and 2.2b show two possible execution plans for this query. The plans are to be read bottom-up. The leaves of the trees are the base tables as shown in Figure 2.1; the numbers on the edges denote the sizes—i.e., the number of rows—of tables and intermediate results, respectively. That is, in Figure 2.2a, Enrolled and Courses are joined first under the condition E.Title=C.Title. The resulting intermediate table has 131739 rows. This is combined with the table Professors and finally joined with the result of filtering the student table. For simplicity we assume there is only one single student named Sam White. The result table, at the root of the tree, contains only 8 rows. Plan 2.2b differs only in so far as the tables are joined in an alternative order.

Although we focus on join ordering, we usually need additional operators for instance to restrict Students to "Sam White", in our example. Such restrictions are applied as early as possible, thus not interfering with the join ordering problem. That is, restrictions as well as other operators do not need to be taken care of when determining a join order. This proceeding is not only a common assumption in previous work, but also practice in commercial systems.

While semantically equivalent, the plans differ in sizes of intermediate results as indicated. The sizes relate to the *costs* of a plan since they reflect the amount of work that has to be done at each operator as well as total resource allocation, e.g. main memory, etc. Accordingly, plan 2.2b is preferable.

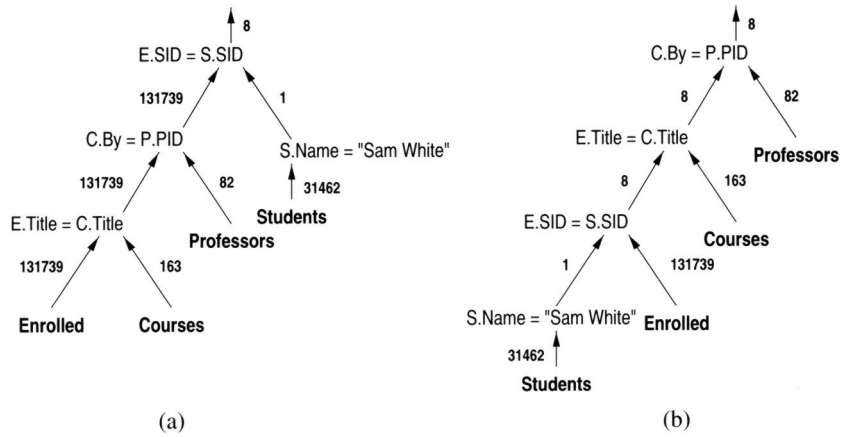


Figure 2.2. Semantically equivalent execution plans

2.2 Formal Model

For a formal description of the problem, we decompose it into three separate components, *join graphs*, *execution plans*, and *cost computation*.

2.2.1 Join Graphs

Given a query in a declarative query language like SQL we identify a *join* or *query graph* $G(V, E)$. The nodes V represent the tables involved in the query. The edges E denote which tables are to be combined by a join predicate [Ull89]. Figure 2.3 shows the join graph for the previous example. Every query corresponds to exactly one join graph.

In the literature, certain shapes of join graphs received special attention for two reasons. Firstly, certain shapes are characteristic for certain applications. Secondly, for database evaluation, synthetic work loads are often used where the shape of the query graph is used as parameter for the random generation of queries.

A common restriction is to consider tree shaped graphs only, i.e., graphs without cycles, since the majority of queries in practice are of this kind. Among the tree shaped queries, further differentiation is applied. Star graphs where all nodes are to be joined with one single, distinguished node occur in data warehousing applications when one, usually very large, fact table is combined with additional information. Another frequently encountered shape is the chain graph in which all nodes have degree less or equal

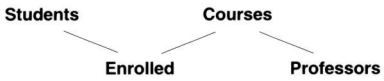


Figure 2.3. Join graph

to 2. Chain graphs are typical for path expressions for instance in object oriented database systems (see e.g. [KM94]), but also database systems that use vertical fragmentation [Bat79, CK85, BK99]. The combination of star and chain graph is known as snowflake graph where chains are adjacent to one center node. The chains themselves may also contain further stars.

In the field of cyclic query graphs, further differentiation is not common, though certain patterns can be identified from a geometric point of view, such as grids etc. However, they do not correspond to frequently encountered queries in practice. In practical examples of cyclic queries, the graphs can be easily decomposed to a tree shaped one with 1 or 2 additional edges. The graph theoretic notion of connectivity is not very helpful either, since connectivity larger than 1 means *every* node is part of at least one cycle. Besides, for queries of trivial sizes such a scenario is rather unrealistic. Particularly, the pattern of a clique graph where every node is connected with every other node is—again trivial size excluded—very unlikely to correspond to any real-world query.

2.2.2 Execution Plans

Execution plans are also often referred to as *execution tree*, *processing plans*, or *processing trees*. We will use the term synonymously in the remainder of this work.

The following definition describes valid processing trees formally:

Definition 2.2.1
Let $G(V, E)$ be a query graph. A *valid processing tree* t *over* G is a binary tree with $|V|$ leaves such that:

a) the leaves of t correspond to the nodes of G

b) the leaves of every subtree of t form a connected component in G. $\diamond$

The set of all possible execution plans spans the *search space*. The size of the search space grows exponentially in the size of G due to combinatorial explosion. For a more detailed study of this issue in combination with tree shaped graphs see [GLPK94, GLPK95]. The problem of efficiently determining the search space size in case the underlying graph is cyclic, is unsolved yet, i.e., it is unknown whether this problem is $\#P$-complete.

We introduce execution plans as binary trees. Other models have been proposed and actual implementation may use n-ary trees. However, from the complexity point of view, n-ary joins can always be decomposed into a set of binary joins with the same complexity. The agglutination to n-ary joins is used in multi-stage optimization where for example at one stage the particular join order is irrelevant or not determined yet. Such proceeding is helpful in transformation-based optimizers, where all stages of the optimization use fully assembled execution plans—not yet optimized join clusters are carried on to the next stage as n-ary joins.

A further general restriction we seem to have applied in the above definition is the limitation to trees. This is due to the direct translation of relational algebra operators which, like any other functional expression, have a tree shaped evaluation graph. In the context of object-oriented databases, Kemper *et al.* suggested cyclic execution plans where data can bypass a number of operators reducing execution time [KMPS94]. The decision how to split data streams is made at run time. This technique can be particularly beneficial when other join operators are bypassed [SPMK95]. In practice, the cases where these methods apply are rare however. Furthermore, using these so called bypass plans poses a difficult problem for the execution engine: for tree shaped plans, very efficient evaluation techniques have been developed and refined. The assumption that plans are tree shaped is unfortunately a necessary prerequisite for their functioning. If the execution plan is a DAG, the execution paradigm needs major modifications. In [6], we proposed a general model for this problem together with an extension of common evaluation techniques that overcomes this problem. In the present work, however, we will not consider these cases since they are not yet of any practical relevance.

Another scenario exceeding the limitation to trees is encountered in parallel database systems. Data streams are partitioned for executing partial plans in parallel. Usually, the parallel parts are identical in their function so that the optimizer can handle them like one single plan executed on hardware with accordingly scaled parameters [14]. Such simplification can be simply taken care of by the costing techniques without influencing the actual optimization process. This notion of *transparent parallelism* can be extended to plan segments of arbitrary size [13, 11, 10, 9] as well as to arbitrary shapes of the plan [8]. In either case, the techniques proposed allow a reduction to a sequential, tree shaped plan during optimization.

2.2.3 Cost Functions

We indicated above that different execution plans require different amounts of effort to be evaluated. The objective function for the query optimization problem assigns every execution plan a single non-negative value. This value is commonly referred to as *costs* in the query optimization business, but also in the context of other combinatorial optimization problems where the objective function is to be minimized.

Cost functions belong to the core of proprietary code of a database vendor. Their accurate tuning and alignment with all other database components requires a high level of expertise and knowledge of both hardware and database components. Cost modeling has become a line of research in its own right over the last decade [Yao77, Bat86, AKK95, ZL96, LN96, BF97, BMK99, BMK00].

In this section, we briefly describe the basic components as far as necessary for the understanding of the concepts presented later. We will not go into the gory details, not only because it would lead too far away, but also because we want to identify common schemas on a level of abstraction which allows a generalization—one of the questions we will have to face later is of course as to whether our generalization is justified and particularly where its limits are.

Cost functions can be split into two parts: a *logical* component that deals with the analysis of the algorithms used, and a *physical* one that reflects characteristics of the hardware deployed.

Logical Cost Component

Due to the strong encapsulation offered by relation algebra operators, the particular algorithms that implement different operators can be analyzed largely independently. A first criterion in terms of costs is the operator's complexity—i.e., the algorithm that implements its functionality—in the classical sense of complexity theory. Most unary operators are in $O(n)$, like selections or $O(n \log n)$, like sorting; n being the size of the input in the form of a table. Binary operators can be in $O(n)$ like the union of sets that does not eliminate duplicates, or, more often, in $O(n^2)$, as for instance join operators. The strong limitation to polynoms of low degree will be of importance when we discuss a generalization of the concepts in Section 4.

Though the relational operators input and output tuples, there is usually no direct break down of tuples to the storage schemes used in database systems. Thus, the granularity needs to be enlarged to cover pages of either the underlying operating system's memory management or, and this is the usual case in commercial database systems, the pages of the database's own buffer management. As a result, costs are computed in terms of page or buffer I/Os.

The standard database literature provides a large variety of costing formulas on the basis of buffer I/O for the most frequently used oper-

ators and their implementations. We refer the interested reader e.g. to [KS91, EN94, KE96].

The costs of all operators used in one execution plan are computed in a bottom-up fashion, and summed up to obtain a total cost value. This schema may be slightly violated if sideway information passing, for instance in the form of bit filters, is used, where costs may also be propagated top-down within usually small parts of the plan (see e.g. [CHY93]). Such situations need a special treatment. In all other cases, an operator's cost represent the total cost of the partial execution plan rooted in it leading to a *hierarchical* decomposition of the plan. We will discuss this property with respect to the optimization process below.

Physical Cost Component

Obviously, the time needed for writing or reading a given set of pages may take different times in different hardware environments. The resulting time is the sum of the times the different subsystems needed—from buses to disk latencies etc. Those figures do not scale easily from one system to another since bottlenecks shift too.

It seems tempting to separate all sensitive hardware parameters and use them for an instantiation of a general costing schema. That is, in order to transfer the costing technique to a new environment, all we need is the set of hardware specific figures. After instantiating the optimization framework with this set, the optimizer generated should need no further adaption to the new environment. This approach has been pursued in Exodus and its successor Volcano [GD87, GM93].

2.2.4 Complexity

Join ordering as a sub-problem of query optimization has been proven NP-hard under the restriction of using only linear execution plans by Ibaraki and Kameda [IK84]. Cluet and Moerkotte have proven a more general case, where also cartesian products are considered to be NP-hard too [CM95]. Finally, Scheufele and Moerkotte established the proof for the general case where cartesian products are allowed and no limitations are imposed on the shape of the execution plan [SM97]. This includes also trees that consist of cartesian products only—a problem which may seem easier than join ordering at first sight.

2.2.5 Some Practical Considerations

In order to achieve a reasonable trade-off between the time spent on the optimization and the quality of the result, we have to relax the optimization goal and try to find a solution of acceptable quality rather than the absolute optimum [Swa89a].

However, another point, often neglected, is the fact that costing a query plan involves uncertainty caused by the cost model which is based on estimates in two ways:

1. In the example above, we pointed out that the number of qualifying tuples has direct impact on the costs. Cost functions in real database systems, however, have to consider far more parameters than only the table and intermediate result sizes, since the costs aim to give as accurate as possible an assessment how long it takes to execute the query on a certain hardware configuration. Those parameters reflect the way data is stored on disk (e.g. sorted, clustered, or indexed), different kinds of join implementations (e.g. nested loop, merge sort or hash joins) as well as hardware parameter (e.g. I/O bandwidth, CPU clock speed and type). All those values are obtained by calibration, i.e., they are means determined experimentally. A number of important parameters like cache misses and alike are difficult if not impossible to incorporate into a model [BMK99]. On the other hand, modeling too many details, or details at too fine a granularity may contribute to larger errors.

2. In the example, we gave the exact sizes of the intermediate results, tacitly omitting the fact that this information is not available during the optimization phase but only after the execution of the query. What *is*, however, available at the time of optimization are statistics about the data. Based on statistics about the base tables, the selectivities of the joins—i.e., the ratio of qualifying to non-qualifying rows—are estimated. Clearly, joins which operate not only on base tables but the output of preceding joins will be affected also by estimation errors, which occurred earlier. Those estimation errors compound exponentially throughout the execution plan [IC91].

The consequence is a limited *resolution* of the cost function. That is, cost differences of a few per cent only are insignificant, since the numerical error may outweigh them—we cannot determine which of them will actually be executed quicker. Consequently, the tree with the least estimated costs is not necessarily the optimal tree. To allow for this deficiency we consider two plans t_1 and t_2 of *similar* quality if their costs $c(t_1)$ and $c(t_2)$ differ less than the resolution ρ, i.e., $|c(t_1) - c(t_2)| \leq \rho$. Practically, the optimization goal shifts to find a plan with costs less than $c_{min} + \rho$. We will discuss the role of ρ and suitable values for it later.

2.3 Query Optimization Techniques

Standard techniques in today's commercial database systems are based on dynamic programming and heuristics. Stochastic techniques are still confined to research prototypes. In this section we give a brief overview on

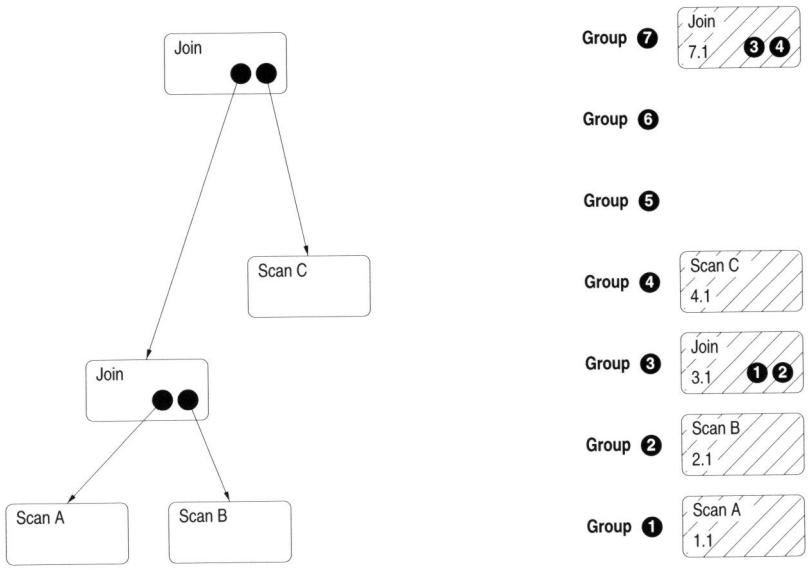


Figure 2.4. Copying the initial plan into the MEMOstructure

the major representatives in the field. We will extend and analyze some of them later in more detail.

2.3.1 Exhaustive Optimization

Exhaustive methods, often also referred to as *exact methods*, are based on a partial enumeration of the search space. The techniques we describe here are based on dynamic programming. Though the term 'dynamic programming' is often used to reference the bottom-up enumeration schema used in IBM DB2, we will use this term in its general meaning [Ber87].

MEMO-based Enumeration

Using the hierarchical decomposition into subtrees as sketched above, the optimal plan consists of optimal subtrees. By optimal subtrees we mean the tree with least cost of all equivalent trees. We sketch the approach pursued in Microsoft SQL Server and Tandem Non-Stop SQL. Both descended from the research prototype Cascades which in turn is an enhancement of the Volcano optimizer [McK93, GM93, BMG93, GCD+94, Gra94].

Since this type of optimizer is transformation based, a first initial plan must be provided by the preprocessing phase. Such an initial plan can be generated as a canonical form where all tables referenced in the From clause are joined (cf. n-ary joins above) without specifying physical join operators, together with a subsequent application of select and aggregate operators. Subqueries if they have not been un-nested in the preprocessing are separately transformed and become a subtree of the initial plan. Consider a join query that joins three tables A,B, and C. A simple initial plan is given in Figure 2.4. The initial plan consists of logical operators that describe only the algebraic properties of the operator but do not contain any implementation details.

Every operator belongs to a *group* of equivalent subtrees, the *root nodes* of equivalent subtrees. During a *copy-in* phase, the operators of the initial tree are assigned to the respective groups. The original links between operators are substituted by references to groups. The system of groups is called MEMO Structure in the following [GCD+94]. Groups are identified by their group number. The group containing the root operator of the initial tree is referred to as *root group*. To facilitate the description of further operation we label operators in the MEMO with an id tag of the form ⟨groupno.id⟩ relative to the group. References to the groups that implement possible subtrees of an operator are given by the number in the lower right corner of the operator. We anticipate a reorganization of the system of groups in our example and arrange the group numbers in a way that allows for additional groups in a graphic way.

Once the initial plan is copied into the MEMO, the actual optimization process commences. According to a control strategy that includes the aforementioned timeout and fall-back mechanisms, different sets of rules are applied to the operators in the MEMO. The concept of rules is based on the work by Freytag [Fre87] and has been advanced and refined McKenna *et al.* [McK93, BMG93]. Often, rules are also referred to as transformations. We will use both terms in this work.

A rule, when applied to a logical operator verifies a set of conditions that must be fulfilled for a successful transformation. Conditions include type and algebraic properties of the operator—but may also include specifications concerning its children—, physical properties like sort orders on certain attributes, cost bounds, etc. Provided all conditions of a rule are fulfilled an alternative for the original operator is generated. The result of a rule application can be:

- a logical operator in the same group, e.g. join(A,B) → join(B,A);
- a physical operator in the same group, e.g. join → hash join;
- a set of logical operators that form a connected sub-plan; the root goes to the original group, other operators may go to any group, including the creation of new groups as necessary, e.g. join(A,join(B,C)) → join(join(A,B),C).

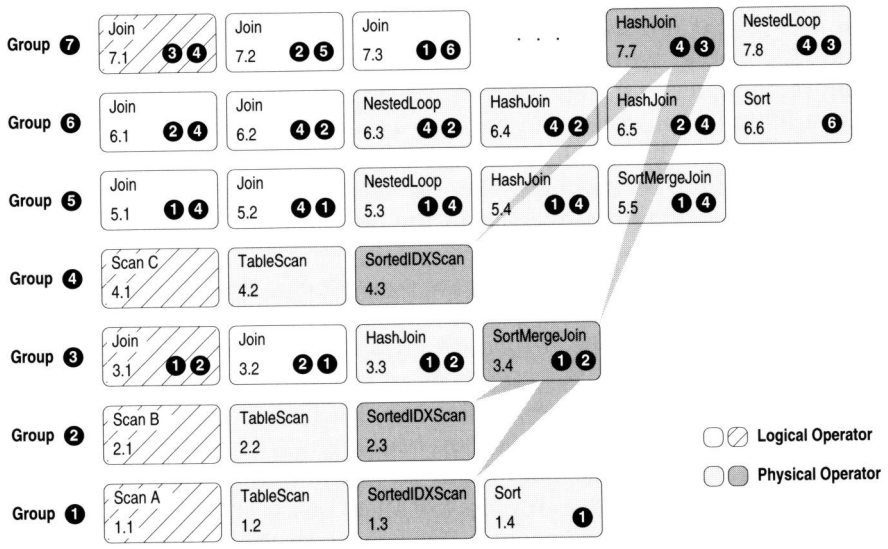


Figure 2.5. MEMO structure representing alternative solutions

The resulting operators may reference any available group in the MEMO as new children. The MEMO framework provides scheduler primitives for the rule application as well as mechanisms to detect and eliminate multiple entries. Moreover, in every group the currently cheapest operator—and thus the subtree rooted in it—is marked up.

Figure 2.5 shows the MEMO structure form the previous example after applying several transformations, now filled with both logical and physical operators. An execution plan must consist of physical operators only. The optimization ends as soon as no new operators can be derived or secondary stopping criteria like timeouts are fulfilled.

The root group is distinguished as all operators in it are possible root operators for the final execution plan—i.e., they encode an alternative execution plan. Conversely no operator of any other group can be root of the final execution plan.

Thus, the least costly physical operator in the root group is the root of the final execution plan. Its children are recursively determined by selecting the least costly physical operator of its child groups, observing the physical properties such as sort orders. In Figure 2.5 a possible final execution plan is indicated by darkened arrows.

System R

In System R a similar approach is implemented. However, instead of using an initial plan and applying transformations, the plans are generated bottom-up from scratch. The resulting sub-plans are stored in a lookup table comparable to the MEMO. The optimizer of System R and its successors, Starburst and DB2 have been scrutinized concerning their complexity and both Pellenkoft *et al.* as well as Vance and Maier showed that it actually does not meet its theoretically lower bounds. Vance and Maier proposed an enhanced version to overcome these drawbacks [VM96]. The improvements by Pellenkoft *et al.* solve the problem in the context of a transformation based framework as sketched above [PGLK97a, PGLK97b].

2.3.2 Non-exhaustive Optimization

Ibaraki and Kameda suggested an algorithm that computes the optimal join order with nested-loop joins for acyclic query graphs under certain assumption concerning the cost function [IK84]. These restriction are usually too strong for practical application. However, the basic elements of this technique can be transferred to the more general case on the expense of optimality. Such a transfer can be of interest for generating initial solutions for transformation based optimizers.

Similar to the previous approach, Krishnamurthy *et al.* proposed a ranking technique, named *KBZ*, that computes the optimal join order under a set of conditions different from the one above [KBZ86]. Again, the conditions are not fulfilled in practical cases in general but as mentioned before, a relaxation of some of the restrictions provides a heuristic. Steinbrunn *et al.* established a quantitative assessment using this heuristic for optimizing large join queries. Their results suggest decreasing performance with increasing query size [SMK97].

Swami and Iyer modified the previous algorithms and introduced a random component altering the nature of the algorithm substantially as we will see later [SI93]. Their new algorithm, called *AB* outperforms the original KBZ in almost every case.

2.3.3 Probabilistic Optimization

During the last decade probabilistic algorithms gained popularity in many fields of combinatorial optimization. Especially in fields like VLSI layout, where they are part of a user feedback controlled refinement process, such techniques belong to the standard repertoire. The basic setup in query optimization differs in two vital details from the one in VLSI layout: Queries have to be optimized in (a) on-line fashion and (b) without user feedback. As opposed to off-line applications a reliable self tuning mechanism is required.

Most probabilistic algorithms can be proven to converge to the optimum, an infinite amount of running time permitted, but the conditions ensuring success are not useful in practical applications; in a user feedback controlled process, much more aggressive tuning can be applied leading to enormous performance improvements, in terms of running time.

One of the main advantages of probabilistic algorithms, however, is their black box or blind search approach: The algorithm alters query plans only by means of a set of transformations and checks the resulting quality by calling the cost function. It does not need any further knowledge of the semantic of the changes. That way, the optimization strategy is highly extensible and can be adapted to any kind of set of relational operators.

This ground has been broken notably by Ioannidis *et al.* [IW87, IK90, IK91], and Swami *et al.* [SG88, Swa91, SI93]. A detailed discussion of their work is given in Chapter 7.

Also genetic algorithms have been applied though delivering mediocre results only. We will scrutinize the major approaches [BFI91, SS96, SMK97] in Section 6.

2.4 Summary

Query optimization is an NP-hard combinatorial problem, which has been the subject of numerous probes during the last two decades. As a result of this process, the view on it of theoretists and practitioners differ substantially: Academic researchers regard it largely as solved, database implementors experience the opposite on an almost daily basis [Cha97].

The reason for this difference of opinion is the difference in the models for the problem. In academic research usually some interesting aspects of the problem are separated from a larger context and analyzed, tackled, and solved in isolation. However, the integration of the solution back into the original setup usually has to sacrifice efficiency and effectiveness of the new approach to a considerable degree.

Practitioners, on the other hand, often seem too much concerned about minor technical details loosing the view for the picture as a whole. Consequently, the state of the art in commercial products did not progress significantly over the last decade.

In this chapter, we outlined the basic ingredients of the problem. Especially the multitude of different implementations of relational operators together with the uncertainties involved by the cost computation make the transfer of theoretical results particularly difficult. Exhaustive algorithms based on dynamic programming are therefore the algorithms of choice in commercial database systems to date. Though randomized algorithms can exceed the strict limitations of dynamic programming and may be used to solve problem instances of way larger sizes, they have not been employed in comprehensive query optimization as many effects are not fully understood yet.

3

Enumeration Techniques

In Chapter 1, we briefly sketched the notion of cost distributions already indicating that they are the key to the analysis of both the problem itself and the application of certain optimization techniques. The primary goal will be the analytical modeling of distributions. But in order to verify the generality of those results, we need means to extract actual cost distributions from search spaces used in both simplified and full-blown query optimizers. Having such a verification mechanism will also be handy to test for the limitations of the modeling techniques.

In this chapter we present three techniques that enable enumeration and sampling of search spaces and to obtain actual cost distributions. The three search spaces being scrutinized are (a) the cross product optimization problem, (b) join ordering—including problems with cyclic join graphs—and (c) full-blown query optimization as implemented in Microsoft SQL Server.

The enumeration and sampling framework in the third case is not only of interest for our experiments concerning cost distributions, but has also a very practical application: the verification of the query optimizer and the execution engine in the development of commercial database products. It extends the regular optimizer in a way that potentially all plans, the optimizer has to consider for a particular query, can be obtained. The optimality of the regular optimization result can be easily verified but more important, optimizer components like transformations and optimization strategies, which are otherwise hard to assess, can be tested: selecting plans other than the optimal provides a multitude of test cases for the execution engine. Plans the optimizer would only choose if the underlying catalog fulfills certain conditions can now be tested independently of whether or not these conditions—possibly hard to trigger otherwise—are fulfilled.

3.1 Labeled Binary Trees

The Cross Product Optimization Problem, in the following abbreviated as XOPT, is a special case of the general Join Order Problem in that all operand selectivities are equal to one. The relations are only characterized by their sizes as other parameters like attribute value distribution, are irrelevant. The cost function reduces to a pairwise multiplication of the size of the relations involved. Thus every solution can be split into two components: a binary tree, which implements the multiplication schema and the assignment of leaves.

Let us first focus on the binary tree structure and devise counting and generating schemas.

Definition 3.1.1
Let t_1 and t_2 be binary trees with n leaves. t_1 and t_2 are *isomorphic* if there exists a bijective mapping h between the nodes of t_1 and t_2 such that

$$w_j \text{ is son of } w_i \text{ in } t_1 \Leftrightarrow h(w_j) \text{ is son of } h(w_i) \text{ in } t_2.$$

We denote tree isomorphism with $t_1 \cong t_2$, or $t_1 \cong_h t_2$ if we want to specify a mapping explicitly. ◇

Isomorphic trees do not add any additional information to the cost distribution as every tree has the same number of isomorphic duplicates. Hence, we can simply omit them in the enumeration schema and focus on the set of non-isomorphic trees only. We denote the set of non-isomorphic trees with n leaves by M_n.

3.1.1 Counting

To count the number of trees in M_n we use a local argument that considers only the root of a tree and applies recursively to its subtrees. The formula is a well-known result in combinatorics, however, we think it helpful to describe the approach in detailed manner as we will use a similar argument to generate these trees.

Considering the root of a non-isomorphic tree, it is characterized by the number of leaves in its left and its right subtree, n_r and n_l respectively. Apparently, there is more than one tree per such configuration, in general. The number of trees with n leaves in total and n_l in its left subtree, which induces n_r immediately, can be denoted by the recurrence formula

$$B(n_l) \cdot B(n - n_l)$$

where $B(n)$ is the number of non-isomorphic trees with n leaves. To avoid isomorphic duplicates we need to consider only the cases

$$n_l \geq n - n_l.$$

In case n is an odd number, $B(n)$ is

$$B(n) = \sum_{1 \le i < n-i} B(n-i)B(i)$$

with $B(0) = 0$ and $B(1) = 1$.

In case n is even, we have to correct this expression by a term that determines the number of non-isomorphic combinations of trees with $\frac{n}{2}$ leaves. We will give a more detailed explanation of this term when we discuss the unranking of trees.

The number of non-isomorphic trees is

$$B(n) = \begin{cases} \sum_{\substack{i=1 \\ i<n-i}} B(n-i) \cdot B(i), & n \text{ is odd} \\ \sum_{\substack{i=1 \\ i<n-i}} B(n-i) \cdot B(i) + \frac{B(\frac{n}{2})}{2} \cdot (B(\frac{n}{2}) + 1), & n \text{ is even.} \end{cases}$$

with $B(0) = 0$ and $B(1) = 1$. Using a lookup table, $B(n)$ can be computed in $O(n^2)$.

3.1.2 Generating Trees

Unranking.

The idea behind unranking is to have a mechanism, which, given a rank and the number of leaves, creates a tree recursively by deciding locally, i.e., for the root node of each subtree, how many leaves are to be in the left subtree, and how many in the right. Then, subranks for both subtrees are derived and the unranking is called recursively. A recursion terminates, returning a leaf only, when the procedure is called with $r = 0$. Figure 3.1 shows an outline of the algorithm.

Given n and r we determine n_l as

$$n_l = \max\{i | \sum_{1 \le i \le n-1} B(i)B(n-i) \le r+1\}.$$

The subranks r_r and r_l for the right and the left subtree respectively compute to

$$r_r = \left\lfloor \frac{r}{B(n_l)} \right\rfloor$$

and

$$r_l = r - r_r B(n_l).$$

Recursive application unfolds the complete tree of rank r. Once we set up the lookup table for $B(n)$ unranking a pair (n, r) is in $O(n^2)$. The recursion terminates properly as $n \le 3$ implies a rank of zero. We refer to the resulting tree as $t_n(r)$, or simply as $t(r)$ if n is clear from the context.

```
Algorithm   UNRANK
Input       n number of leaves, r rank
Output      t tree with n leaves and rank r

if r = 0 then
    t ←LINEARTREE(n)
else
    compute n_l, n_r
    compute r_l, r_r
    t ← MAKETREE(UNRANK(n_l, r_l),UNRANK(n_r, r_r))
endif
return t
```

Figure 3.1. Generic unranking schema

Ranking.

Conversely, the rank of a given tree $b \in M_n$ computes to:

$$r(b) = r_r B(n_l) + r_l.$$

We can now close the gap of computing $B(n)$ in case n is even. Consider a tree with $\frac{n}{2}$ leaves in both subtrees. Non-isomorphic combinations are only those with

$$r_l > r_r$$

as further restriction. With $0 \leq r_l < B(\frac{n}{2})$ we get

$$\sum_{i=1}^{B(n/2)} i$$

as the total number of possible combinations when $n_l = n_r = B(n/2)$. Rewriting the term gives

$$\frac{1}{2}B(\frac{n}{2})(B(\frac{n}{2}) + 1).$$

3.1.3 Labeling

For a tree t of M_n, a function f on the leaves of t into $\{1, \ldots, n\}$ is called *labeling* of t. A labeling can be interpreted as a permutation of the leaves. In principle, there are $n!$ possibilities to label a tree with n leaves, however,

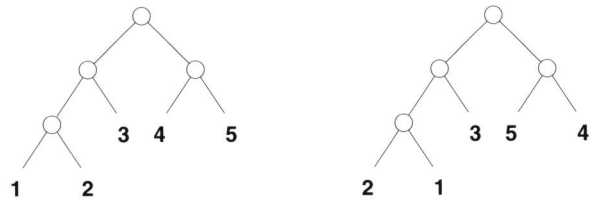


Figure 3.2. Isomorphic labeling

due to the symmetry of subtrees, we may encounter duplicates as shown in Figure 3.2.

The left tree can be transformed into the right one by means of commutative exchange of (isomorphic) subtrees.

Definition 3.1.2
Let t_1 and t_2 be isomorphic trees, and f a labeling of t_1. f is called *isomorphic labeling*, if there exists a bijective mapping h such that

1. $t_1 \cong_h t_2$
2. if w is leaf in t_1, $f(h(w)) = f(w)$

◇

For every labeling there exist 2^k isomorphic labelings, k being the number of inner nodes with isomorphic subtrees. In Figure 3.3 these nodes are indicated by circles. There are $\frac{n!}{2^k}$ non-isomorphic labelings for each tree. During enumeration of all labelings, isomorphic combinations can be avoided by skipping permutations.

For every pair of isomorphic subtrees rooted in the same node let N_1 and N_2 denote the labels of the left and the right subtree respectively. We call a labeling *monotonic* if

$$\min N_2 < \min N_1$$

holds for all such pairs. Enumerating monotonic labelings only is therefore equal to enumerating non-isomorphic ones.

If we think of the labels as id numbers of the database relations, we are able to count and generate execution plans consisting of cross products only. To obtain the cost distribution we could either enumerate the whole set M_n or sample form it with uniform probability by generating random numbers between 0 and $B(n) - 1$ and unrank the associated tree.

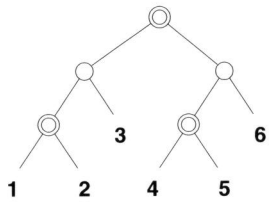


Figure 3.3. Isomorphic subtrees; roots highlighted

3.2 Non-isomorphic Processing Trees

In the next step, we leverage the previous model with the information encoded in the join graph. Instead of enumerating all cartesian products we now generate only those trees that solely consist of join operators. The set of *non-isomorphic processing trees* is a subset of M. Equality between the sets holds only if the join graph is a completely connected graph. In practice, the sets differ substantially.

The previous approach was based on determining and grouping sets of leaves, which was sufficient as any relation can be freely combined with any other in a cross product. Here, we use additional information describing the inner nodes of the tree. Our algorithm is based on edges of the join graph. Every edge corresponds to an inner node of the tree, though not uniquely. First, we develop an algorithm that turns a sequence of edges into a processing tree. Through this connection, the concept of isomorphism extends immediately from trees to sequences. Using this close connection it is sufficient to enumerate non-isomorphic sequences to obtain the desired set of processing trees.

3.2.1 Sequences

For a set $E = \{e_1, \ldots, e_n\}$ a *sequence L over E* is denoted by

$$L = \langle e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(n)} \rangle = \langle e_{\pi(i)} \rangle$$

where π is a permutation function on $\{1, \ldots, n\}$. The set of all sequences over E is denoted by E^* and contains $n!$ elements. If E is the empty set E^* contains only the *empty sequence* $\langle\rangle$. Sequences are permutations of a set. We prefer the term sequence, however, since it induces intuitively the notion of subsequences, which we will use extensively below.

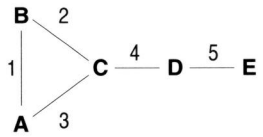


Figure 3.4. Query graph

For convenience, we introduce the following operations in analogy to sets. $|L|$ denotes the length of a sequence; $L \setminus E'$ is the sequence without the elements of E' but retaining the order of the residual elements. As an example, consider $\langle e_1, e_2, e_3 \rangle \setminus \{e_2\}$ which is $\langle e_1, e_3 \rangle$. Finally, we use :: to describe the concatenation of a sequence with either a single element or another sequence.

In order to avoid any confusion of the indices of sequences with those of sets, we introduce a labeling $\eta : E \to \{1, \ldots, n\}$. Throughout all examples we will refer to the e_i by their labels, i.e., for $L = \langle e_1, e_2, e_3, e_4 \rangle$ with $\eta(e_1) = 3, \eta(e_2) = 1, \eta(e_3) = 4, \eta(e_4) = 2$ we simply write $L = \langle 3, 1, 4, 2 \rangle$.

To establish an order on the set E^*, we introduce a ranking which assigns each sequence a unique number as follows:

Definition 3.2.1
For a sequence $L = \langle e_1, \ldots, e_n \rangle$ of length n, the *rank* of L to a *base* b with $b \geq n$ is

$$r_b(L) = \sum_{i=1}^{n} \eta(e_i) \cdot b^{n-i}$$

The rank of the empty sequence is 0. ◇

For instance, the rank of $L = \langle 3, 1, 4, 2 \rangle$ to the base $b = 10$ computes to $r_{10}(L) = 3 \cdot 10^3 + 1 \cdot 10^2 + 4 \cdot 10^1 + 2 \cdot 10^0 = 3142$.

Since we demand $b \geq n$, every rank function is injective and rank functions to different bases define the same order on E^*, i.e., we can omit the explicit notation of b.

Finally, we call a sequence *η-sorted sequence*, denoted by $\widetilde{E}$, if elements with higher η value occur later in the sequence than elements with a lower one, i.e., $\eta(e_i) < \eta(e_j) \Rightarrow \pi(i) < \pi(j)$. For completeness, we define the η-sorted sequence of the empty set to be the empty sequence.

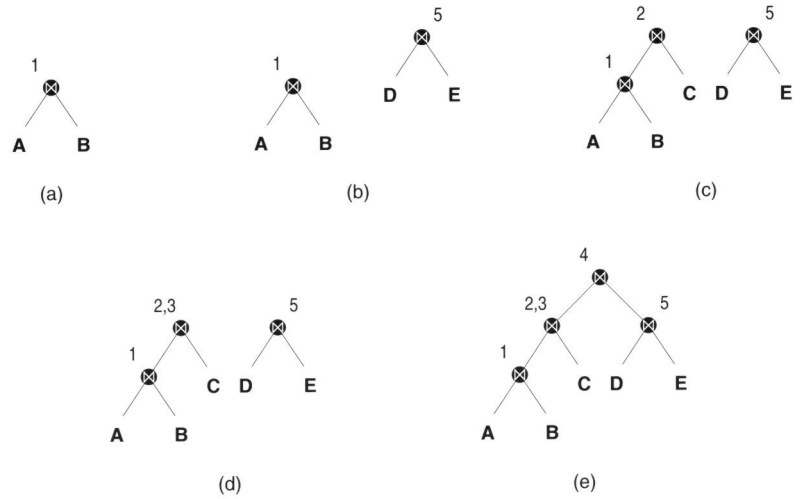


Figure 3.5. Converting the sequence $\langle 1, 5, 2, 3, 4 \rangle$ into a processing tree

3.2.2 Merging Processing Trees

Given a query graph $G(V, E)$, a sequence over E—i.e., a sequence of query predicates—can be turned into a processing tree as follows. For each edge of the sequence we add the respective join operator to the tree. If the edge's relations are already connected by a join added earlier, we only extend the predicate of that join.

Figure 3.5 shows the single steps necessary to convert the sequence $\langle 1, 5, 2, 3, 4 \rangle$ using the query graph of Figure 3.4.

A detailed description of the algorithm is given in Figure 3.6. Starting with a forest of $|V|$ trivial trees that consist of root nodes only, trees are merged pairwise by adding a common root node, indicated by $\oplus$. For simplicity of presentation, we omitted the trivial trees in Figure 3.5. After every step, T_i contains a forest. We refer to the output $T_{|L|}$ as $T(L)$.

Lemma 3.2.2
For a sequence L over *all* edges of a query graph, algorithm MERGETREES constructs one single tree only, i.e., $T(L)$ contains only one tree.

P r o o f : Assume, to the contrary, $T(L) = \{t_1, t_2, \ldots, t_m\}$. Thus, no edges between leaves of $t_1, \ldots, t_m$ were in L, which means, that G was not connected which contradicts the definition of query graphs. □

Note, in case L is only a sequence over a subset of E, the result of MERGETREES $T(L)$ will be a forest.

Lemma 3.2.3
The processing tree computed by MERGETREES is a valid tree in the sense of Definition 2.2.1.

P r o o f : T_i always contains valid processing trees for disjoint subgraphs of G:

$\underline{i = 0:}$ all elements t_j of T_0 are trivial trees of height 0, thus valid trees for the Graphs $G_{t_j}(\{v_j\}, \emptyset)$.

$\underline{i \to i+1:}$ let (n_a, n_b) be the next edge that is to be added and t_a and t_b be elements of T_i with leaves n_a and n_b, respectively. Note, that T_i contains at least a tree of height 0 with leaf v for every possible leaf. If the edge connects nodes, that are leaves of the same tree the shape of the processing tree does not change, according to its definition, and thus the proposition holds. Otherwise, the edge connects the two graphs G_{t_a} and G_{t_b}, with $G_{t_i} = G(\{\text{leaves of } t_i\})$. Therefore, G_{t_a}, G_{t_b} and $G_{t_a \oplus t_b}$ fulfill the conditions of Definition 2.2.1 which completes the proof. □

Proposition 3.2.4
For a query graph $G(V, E)$, MERGETREES converts every sequence over E into a valid processing tree.[1]

P r o o f : Follows from Lemma 3.2.2 and 3.2.3. □

The concept of isomorphism immediately extends to sequences: Two sequences L_1 and L_2 are isomorphic if $T(L_1) \cong T(L_2)$. In the previous example $T(\langle 1,5,2,3\rangle) \cong T(\langle 5,1,2,3\rangle)$ holds, since both sequences yield the forest of Figure 3.5d. On the other hand, $T(\langle 1,5,2,3\rangle) \ncong T(\langle 2,1,5,3\rangle)$.

During the enumeration we want to select one sequence of each class of isomorphic sequences. As a simple criterion for this selection, we can use the rank of the sequences as follows.

Definition 3.2.5
A sequence L over E is *rank-minimal* iff

$$\forall L' \in E^*, L' \neq L: \quad T(L) \cong T(L') \Rightarrow r(L) < r(L')$$

i.e., every isomorphic sequence has greater rank. ◇

In our example, $\langle 2,1,5\rangle$ is rank-minimal, since the only other isomorphic sequence over the same subset of E is $\langle 5,1,2\rangle$.

Monotonicity of the sequence elements implies *prefix monotonicity* of rank-minimality as follows:

[1] We present a discussion of the time complexity for all algorithms in Section 3.2.5.

Algorithm MERGETREES
Input L sequence of edges,
n_i nodes of the query graph
Output $T_{|L|}$ processing tree

$i \leftarrow 0$
for j=0 **to** $|L|$ **do**
    $t_j \leftarrow n_j$
done
$T_0 \leftarrow \bigcup_j t_j$
$L_0 \leftarrow L$
while $|L_i| > 0$ **do**
    let $e = (n_a, n_b)$ be the first element of L_i
    let $t_a \in T_i$ be tree where n_a is leaf
    let $t_b \in T_i$ be tree where n_b is leaf
    if $t_a \neq t_b$ **do**
        $t \leftarrow t_a \oplus t_b$
        $T_{i+1} \leftarrow (T_i \setminus \{t_a, t_b\}) \cup t$
    done
    annotate deepest common ancestor of n_a and n_b
        with predicate of e
    $L_{i+1} \leftarrow L_i \setminus \{e\}$
    $i \leftarrow i + 1$
done

Figure 3.6. Algorithm MERGETREES

Corollary 3.2.6
Let L be sequence over $F \subset E$ and $e \in E \setminus F$ with $\eta(e) > \eta(f)$ for all f in L then

$$L \text{ is rank-minimal} \Leftrightarrow L :: e \text{ is rank-minimal}$$

holds.

Consequently, the η-sorted sequences are always rank-minimal.

As pointed out earlier, the concept of rank-minimality is the key to a proper enumeration: Every class of isomorphic trees corresponds to exactly one single rank-minimal sequence. Therefore, enumerating the rank-minimal sequences only, would suffice.

In the next section we discuss criteria, on which we can identify prefixes that can be excluded from further considerations.

3.2.3 Redundancies

The fact that not every edge of a sequence prompts MERGETREES to change the shape of some tree already suggests that sequences may contain certain redundancies (cf. Fig. 3.5c and d). Besides the equivalences shown in previous examples which were caused solely by permutation of two elements of a sequence, cyclic graphs additionally contain redundancy.

Definition 3.2.7
For a sequence L the set of redundant edges for every element is given by:

$$\rho_L(e_i) = \{e_j | \eta(e_i) < \eta(e_j), T(L) \cong T(L \setminus \{e_j\})$$

$R(L) = \bigcup_{e \in L} \rho_L(e) \cup \bigcup_{e \notin L} \rho_{L::e}(e)$ is called the *redundancy* of L. ◇

The first part means, that every edge e can be replaced with every element of $\rho_L(e)$ while $T(L)$ remains unchanged.

The redundancy of a sequence, however, comprises more than just the redundant edges within the sequence itself. The rational behind this is as follows. When building a sequence incrementally, we need to know which of the edges that is *not yet* part of the sequence has any redundancy. Edges that are not adjacent to $G(L)$ cannot induce further redundancy. Hence, $R(L)$ is the set of edges either redundant to edges of L or edges adjacent to $G(L)$.

The redundancy of $\langle 1,2,4,5 \rangle$ is $\{3\}$, for instance. For the sequence $R(\langle 1 \rangle)$ the redundancy computes to $\{3\}$ since 3 is redundant to an adjacent edge of $G(\langle 1 \rangle)$, namely 2. Analogously, $R(\langle 3 \rangle) = 2$.

Furthermore, the redundant-edge property is transitive, i.e., for two edges e_i and e_j in L the following holds:

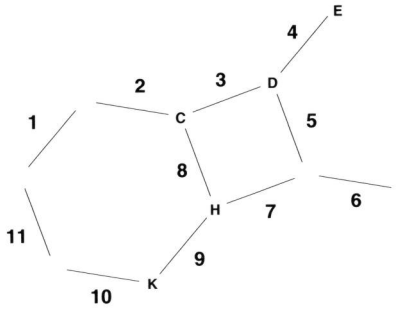
$$\rho_L(e_i) \cap \rho_L(e_j) \neq \emptyset \quad \Rightarrow \quad e_j \in \rho_L(e_i) \vee e_i \in \rho_L(e_j)$$

Removing the redundancy from a sequence does not affect the shape of the resulting tree, i.e., $T(L) \cong T(L \setminus R(L))$. Hence, redundancy is prefix monotonic, in the following sense $R(\langle e_1, \ldots, e_{n-1} \rangle) \subseteq R(\langle e_1, \ldots, e_{n-1}, e_n \rangle)$. Figure 3.7 shows three examples for different situations where redundancies may occur.

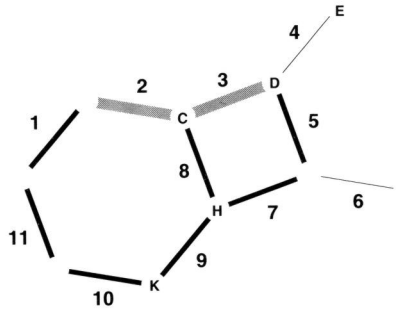
Proposition 3.2.8
Let L be a sequence over $F \subset E$ and $e \in E \setminus F$. Algorithm COMPUTEREDUNDANCY (see Fig. 3.8) computes $R(L :: e)$.

P r o o f : Since $R(L)$ is input parameter, only the redundancy added by e has to be computed. Redundancy occurs if there is more than one edge between one component of $G(L)$ and G_e, and V' and G_e, respectively. For every component of $G(L)$ the algorithm checks all remaining edges in $E \setminus L \setminus R(L)$ whether they connect the graphs and all but the first fulfilling the condition are added to R'. Thus R' is $R(L :: e)$. □

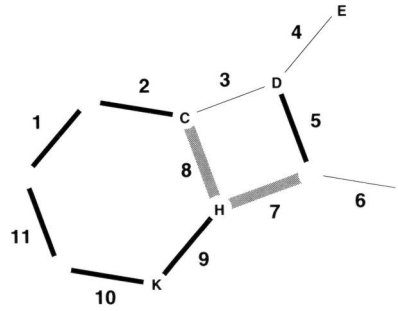
Original join graph



$L = \langle 5, 7, 8, 9, 10, 11, 1 \rangle$
$\rho(1) = 2, \rho(1) = 3$
$R = \{2, 3\}$



$L = \langle 1, 2, 5, 9, 10, 11 \rangle$
$\rho(1) = 8, \rho(2) = 8, \rho(5) = 7$
$R = \{7, 8\}$



$L = \langle 1, 5, 8, 9, 10, 11 \rangle$
$\rho(1) = 2, \rho(5) = 7$
$R = \{2, 7\}$

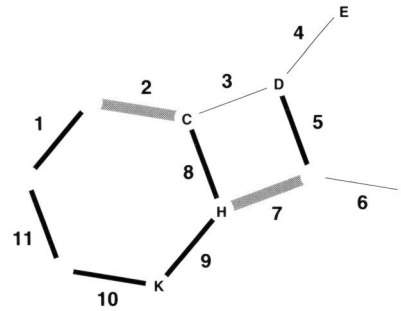


Figure 3.7. Examples of redundancies for different sequences L; Elements of sequence indicated by thick lines; Elements of redundancy indicated by gray lines

```
Algorithm   ComputeRedundancy
Input       L sequence of edges
            R set of redundant edges w.r.t. L,
            e edge with e ∈ E \ R

let G_e be subgraph of G(L :: e) containing e
let V' be set of nodes not connected to G(L :: e)
foreach connected subgraph G' ∈ V' ∪ (G(L :: e) \ G_e) do
    found ← false
    foreach f ∈ E \ L \ R do
        if f connects G' with G_e do
            if found do
                R' ← R' ∪ {f}
            done
            found ← true
        done
    done
done
```

Figure 3.8. Algorithm ComputeRedundancy

3.2.4 Generating the Trees

In this section, we assemble the techniques presented so far. To generate all non-isomorphic processing trees for a given query graph G, we generate the set $\hat{L}$ of non-isomorphic edge sequences. This set is given by

$$\hat{L} \subseteq E^*, \qquad \text{with} \quad L_1, L_2 \in \hat{L} \Rightarrow T(L_1) \not\equiv T(L_2)$$

An algorithmically more practical form is the following one:

$$\begin{aligned}\hat{L} &= \{L \in E^* \mid L \text{ rank-minimal}\} \\ &= \{L \in E^* \mid L = L' \setminus R(L') :: \widetilde{R}(L'), \\ &\qquad\qquad L' \in E^*, \quad L' \setminus R(L') \text{ rank-minimal}\}\end{aligned}$$

The algorithm RapidEnumeration, given in Figure 3.9, generates the sought sequences. The main loop iterates over the number of edges and calls itself recursively up to n times. The deepest recursion is reached once the sequence is either complete, or can be completed immediately by adding the redundancy as a η-sorted suffix. Since the basic design of the loop allows for the maximal possible set of $n!$ sequences, the algorithm has only to avoid the generation of equivalents, that is, ignore redundant edges and discard not rank-minimal prefixes. According to Corollary 3.2.6, rank-

minimality is prefix monotonic, thus we can judge for each edge e whether $L :: e$ is a valid prefix or not.

Lemma 3.2.9
Let L be a rank-minimal sequence over $F \subset E$, and $e \in E \setminus R(L)$ with $\eta(e) > \eta(f)$ where f is the edge of L with $\eta(f) = \max_{e' \in F} \eta(e')$. Then $L :: e$ is rank-minimal.

Proof: Since rank-minimality is prefix monotonic, $L :: e$ is the rank-minimal sequence containing all elements of L and e. □

Lemma 3.2.10
Let L be a rank-minimal sequence over $F \subset E$, and $e \in E \setminus R(L)$ with $\eta(e) < \eta(f)$ where f is defined as above. The following holds

$$L :: e \text{ rank-minimal} \Leftrightarrow e \text{ is adjacent to } G_f$$

where G_f denotes the component of $G(L)$ that covers f.

Proof: Let e_i be the elements of L, i.e., $L = \langle e_1, \ldots, e_{|L|} \rangle$.

Firstly, assume, to the contrary, e is not adjacent to G_f. Let e_k be $\min_i \{\eta(e_i) \in L | \eta(e) < \eta(e_i)\}$, and G_a, G_b the components of G connected by e. Furthermore, w.l.o.g. $|G_a| \leq |G_b|$.
$\underline{|G_b| = 1}$: e is not connected to any of $G(L)$ components and inserting it at any position in the sequence does not change $T(L)$.

Thus, $r(\langle e_1, \ldots, e_{k-1}, e, e_k, \ldots, e_{|L|} \rangle) < r(L :: e)$, i.e., $L :: e$ is not rank-minimal.
$\underline{|G_b| > 1}$: Let $\langle b_1, \ldots, b_{|G_b|} \rangle$ be the subsequence that defines G_b only, i.e., $L \setminus (E \setminus G_b)$. If $|G_a| > 1$, $\langle a_1, \ldots, a_{|G_a|} \rangle$ is defined analogously. Otherwise, let $a_{|G_a|}$ be $b_{|G_b|}$. The first position in L where e can occur without affecting the equivalence is after both $b_{|G_b|}$ and $a_{|G_a|}$. Since both G_a and G_b are not connected to G_e, $r(\langle e_1, \ldots, e, e_{|L|} \rangle) < r(\langle e_1, \ldots, e_{|L|}, e \rangle) = r(L :: e)$, i.e., $L :: e$ is not rank-minimal.

To show the opposite direction, assume $L :: e$ is not rank-minimal. Then, a sequence L' with $T(L :: e) \cong T(L')$ exists where e is not the last element, i.e., $L' = \langle e_1, \ldots, e, \ldots, e_{|L|} \rangle$. However, inserting e before $e_{|L|}$ does not effect the equivalence of $T(L') \cong T(L :: e)$ unless e is adjacent to G_f. □

Proposition 3.2.11
Algorithm RAPIDENUMERATION computes all sequences of $\hat{L}$ with prefix L.

Proof: The inner loop of the procedure potentially generates all prefixes. For every prefix, built incrementally, edges that are not already in the prefix or element of the redundancy of this prefix are checked for being added to the prefix. Thus, we show that only rank-minimal redundancy-free prefixes are generated by induction over the prefix length.
$\underline{i = 1}$: trivial.

Algorithm RAPIDENUMERATION
Input L sequence of edges,
R set of redundant edges w.r.t. L

if $E \setminus L \setminus R = \emptyset$ **do**
    MERGETREES($L :: \widetilde{R}$)
    return
done

$\eta_{\max} \leftarrow \max_i \{\eta(e_i) | e_i \in L\}$
let $G_{\eta_{max}}$ be subgraph of $G(L)$ covering $e_{\eta_{max}}$
foreach $e \in E \setminus L \setminus R$ **do**
    if ($\eta(e) > \eta_{\max} \vee$ e is adjacent to $G_{\eta_{max}}$) **do**
        $L' \leftarrow L :: e$
        $R' \leftarrow$ COMPUTEREDUNDANCY(L', R, e)
        RAPIDENUMERATION(L', R')
    done
done

Figure 3.9. Algorithm RAPIDENUMERATION

$\underline{i \rightarrow i+1}$: Let $e \in E \setminus L \setminus R(L)$ be the edge we check. Either $\eta(e) > \eta(e_i)$ for all e_i that are in L so far. The proposition follows with Lemma 3.2.9. Otherwise, if e is adjacent to the component covering η_{max} and Lemma 3.2.10 completes the proof. □
Invoking RAPIDENUMERATION with all possible prefixes $L_i = \langle e_i \rangle$ yields $\hat{L}$.[2]

3.2.5 Discussion

In this section we scrutinize the techniques presented with respect to an efficient implementation and present a quantitative assessment.

Complexity of the Algorithms

One of the critical elements in the algorithm is the membership test for sets. But as our sets are limited to small sizes we can represent them by bit-vectors, which reduces both test and insert/remove operations to $O(1)$.

MERGETREES can transform a sequence into its corresponding processing tree within $O(|E|^2)$ using a directory of leaves. The construction is in $O(|E|)$ and requires no updates at run time. Another critical issue is

[2]The actual implementation, in fact, expands all sequences from the empty sequence. However, for simplicity we omitted the parts necessary for proper treatment of $\langle\rangle$.

the traversal of connected components in COMPUTEREDUNDANCY. Naively identifying the components anew with every invocation is in $O(|E|)$. However, the incremental nature of the changes to the components—adding an edge leaves all other members of the component unchanged—suggests a directory of components. The overhead caused by its maintenance is in $O(|V|)$. This reduces the cost of COMPUTEREDUNDANCY from $O(|E|^2)$ to $O(|V| \cdot |E|)$. Hence, the construction of a processing tree is in $O(|V| \cdot |E|^2)$.

We deliberately used $O(|V|)$ and $O(|E|)$ despite the fact that $O(|E|)$ is in $O(|V|^2)$ since the number of edges exceeds the number of nodes only marginally in typical database applications. Queries that correspond to clique graphs are of virtually no practical impact and solely used as worst-case scenarios. In the majority of all cases $|E|$ is close to $|V|$.

Finally, what is not expressed by the time complexity is the extent of re-use. For simplicity we presented the single procedures as separate from each other as possible. However, to gain the necessary performance, the call to MERGETREES in RAPIDENUMERATION should not be postponed until the sequence is complete and—as the notation suggests—be discarded afterwards, but handled incrementally. We modify MERGETREES in a way that single edges can be added and removed from the tree or forest, respectively. The adding operation is then called before, the removing after the recursive invocation. With this modification, one tree is incrementally built, subsequently pruned, and merged again. Large parts of the tree and of the sequence prefix are not modified when going on to the next processing tree. In contrast to other enumeration techniques, the re-use is inherent in the method and does not require any additional memory nor running time spent on lookups.

In contrast to other enumeration techniques with exponential space requirements, RAPIDENUMERATION needs only space in $O(|E|)$, since only one single tree is built and modified.

Sizes of Search Spaces

The major advantage of the techniques presented is the reduction of the search space's size by the factor 2^k. In Figure 3.10, the sizes of search spaces consisting of non-isomorphic trees only are contrasted with the ones including also isomorphic trees. The queries used in this experiment were taken from the query suite proposed in [GLPK94]. Note, search spaces of cyclic and tree-shaped query graphs are comparable in size, as long as the number of edges does not exceed the number of nodes substantially. The reduction by factor 2^k applies independent of the particular search space or query graph. We experimented with a large number of further queries. Since they are exactly in the line of the above we omit the results here.

Corollary 3.2.12

The number of non-isomorphic trees enumerated by RAPIDENUMERATION

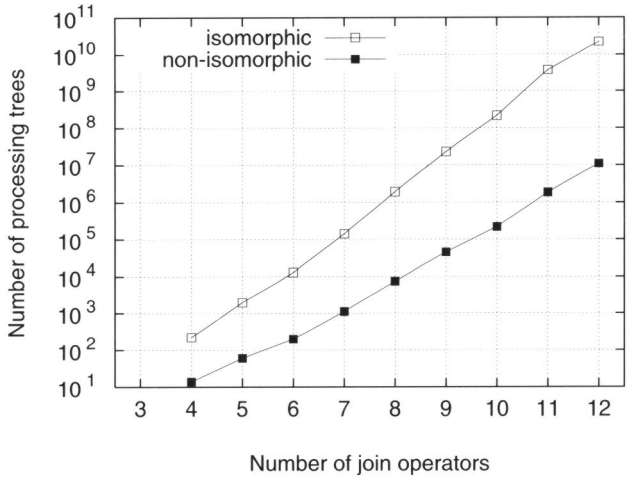


Figure 3.10. Size of Search spaces with and without isomorphic trees

is $N_{clique}(n) = \frac{(2n-2)!}{(n-1)!}2^{1-n}$ if the query graph forms a clique of size n, and $N_{chain}(n) = \frac{(2n-2)!}{n!(n-1)!}$ in case the query graph is a chain.

P r o o f : Independent of the query graph's shape, all processing trees have n leaves and $n-1$ nodes inner nodes. Every isomorphic class has 2^{n-1} elements. The numbers for spaces *including* isomorphic trees [LVZ93] are known to be $N^{iso}_{clique}(n) = \frac{(2n-2)!}{(n-1)!}$ and $N^{iso}_{chain}(n) = \frac{(2n-2)!}{n!(n-1)!}2^{n-1}$. Dividing those figures by 2^{n-1} yields the proposition. □

Finally, with a simple modification we can restrict the algorithm to enumerate only linear processing trees—the most prominent group of trees since the early days of query optimization [SAC+79, IK91]. When dropping the condition $\eta(e) > \eta_{\max}$, we append only edges that are adjacent to $G_{\eta_{max}}$, so at least one of its nodes is already part of the tree, i.e., we add either the bare predicate or a subtree that consists of a leaf only. Thus, the result is a processing tree of height $n-1$.

Corollary 3.2.13

The number of non-isomorphic *linear* trees enumerated by RAPIDENUMERATION is $N_{clique}(n) = \frac{n!}{2}$ if the query graph forms a clique of size n. In case the query graph is a chain we enumerate $N_{chain}(n) = 2^{n-2}$ trees.

P r o o f : In either case we need to focus on the non-redundant edges only. For the clique, every edge that connects to a node which is not part of the

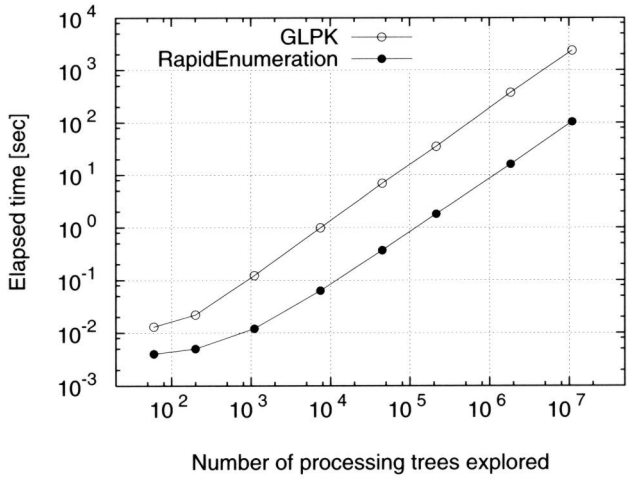


Figure 3.11. Running time of RAPIDENUMERATION compared to GLPK

prefix yet, is non-redundant. Thus, for every prefix of length 1, there are $(n-2)!$ completions. The very first edge can be chosen in $\binom{n}{2} = \frac{n!}{2(n-2)!}$ ways. Therefore, the total number is $\frac{n!}{2(n-2)!} \cdot (n-2)! = \frac{n!}{2}$.

The situation for a chain is as follows. Selecting a prefix of length 1 splits the query graph into two sub-chains that are to be merged by RAPIDENUMERATION. This can be done in $\binom{l_r+l_l}{l_l}$ ways, with l_l and l_r is length of the left and right sub-chain, respectively. Applying this to all $n-1$ prefixes of length 1 yields $N_{chain}(n) = \sum_{k=0}^{n-2} \binom{n-2}{k}$ which can be rewritten to 2^{n-2}. □

3.2.6 Quantitative Assessment

To assess the efficiency of RAPIDENUMERATION is in so far difficult as there is no other method capable of enumerating the same spaces. However, in a different context Galindo-Legaria *et al.* developed counting, ranking, and unranking methods for non-isomorphic processing trees belonging to *tree-shaped* query graphs [GLPK95]. The techniques can be combined and after counting the trees each plan can be generated by unranking its ordinal number.

In Figure 3.11, the running time of RAPIDENUMERATION is compared to this combined technique—in the following referred to as GLPK—for the non-isomorphic search spaces belonging to tree-shaped join graphs (cf.

3.10). All experiments were carried out on a MIPS 10K/250MHz. The queries considered were the same as used for Fig. 3.10. As the graph shows, RAPIDENUMERATION is up to an order of magnitude quicker than GLPK—a gap that widens with increasing query size. For the largest queries run, GLPK is 20 times slower than RAPIDENUMERATION. As the reduction by factor 2^k applies to every query, we found almost the same ratio of running times for other query suites.

3.3 MEMO-based Enumeration

In an analytical analysis, abstract and simplified models as described in the previous sections of this chapter are definitely preferable for the reasons sketched at the very beginning. However, an analysis of a simplified model can only be conclusive and useful in practice if it stands the comparison with its real-world counterpart. In other words, we need to be able to examine cost distributions of a real, fully fletched database system in order to verify our results.

The method developed in this section is based on the MEMO framework as detailed in Chapter 1, but can be transferred to any other optimizer that deploys dynamic programming techniques.

Like with the simplified models, we will devise a ranking algorithm. However, since the term *ranking* has been used in a different notion in [IK84] or [KBZ86], it is important to point out, that our ranking technique is completely in the line of the previous chapter, i.e., with respect to the shape of execution plans rather than in terms of the above cited work.

The previously developed techniques do not extend to the more general problem as the space of alternatives considered by industrial query optimizers is not restricted to an abstract combinatorial problem, such as join reordering. Multiple execution algorithms, index utilization, reordering of grouping operators, special-purpose physical operators, and heuristics to control the time spent on searching, all make up for an *actual* space that is hard to describe succinctly using abstract, regular structures.

3.3.1 Preparatory Steps

Once alternatives are generated, the MEMO structure contains all operators but does not keep track of how many combinations of operators there are, and only the best possible plan is completely assembled. To illustrate the counting framework, let us assume a final state of the MEMO—after generation of alternatives is complete—as given in Figure 3.12.

In order to facilitate later operations we extract all physical operators and materialize the links between operators and their possible children. In Figure 3.13, the materialized links for all children of the previous example's root (operator 7.7) are shown. The resulting structure describes all possible execution plans that can be rooted in this operator. Due to the differences

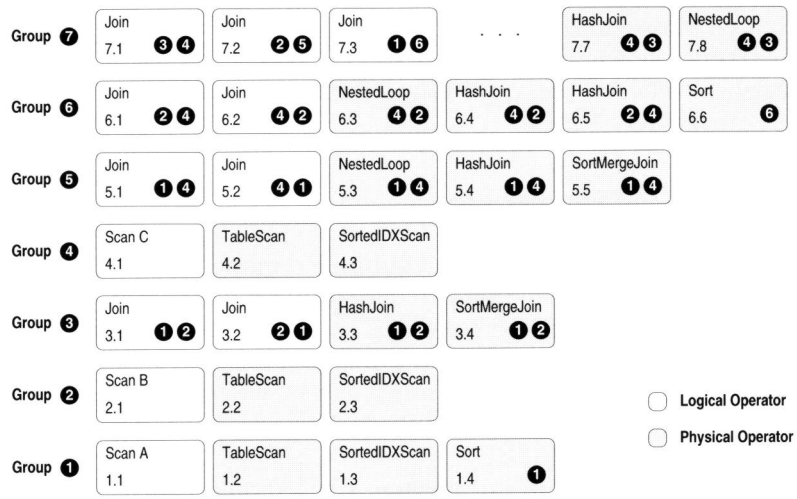


Figure 3.12. Populated MEMO Structure at the end of the actual optimization process

in physical properties some operators of a group may qualify as potential children while others do not.

For instance operator 3.3 in Figure 3.13, can have any operator from group 1 and 2 as left and right child, respectively. Operator 3.4 however can use only the darkened operators 2.3 and 1.3 or 1.4.

3.3.2 Counting Query Plans

We compute the total number of possible plans bottom-up by computing the individual numbers of possible plans that can be extracted from each operator. We denote the number of children of operator v by $|v|$ and the j-th alternative for the i-th child of v by $w_{i,j}^{(v)}$. For example, in Figure 3.13, take $v = 7.7$, then $w_{1,1}^{(v)} = 4.2$, and $w_{2,2}^{(v)} = 3.4$.

To compute the number of plans $N(v)$ rooted in an operator v, we first determine the number of possible alternatives for each child i as

$$b_v(i) = \sum_j N(w_{i,j}^{(v)}).$$

Operator v will take any of the available alternatives on each children, independently, so the number of combined choices is given by a product.

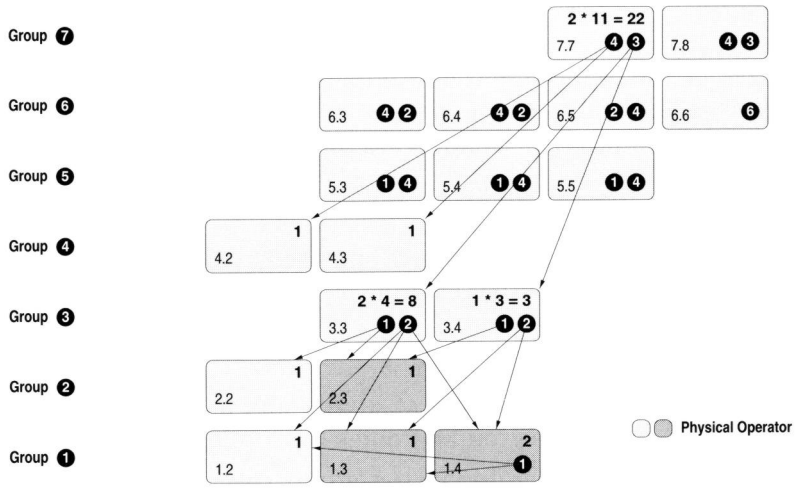


Figure 3.13. MEMO Structure with materialized links between operators and children, for possible plans rooted in operator 7.7

The numbers of plans we can generate using only the first k children is

$$B_v(k) = \prod_{i=1}^{k} b_v(i).$$

Hence, the number of plans rooted in v is

$$N(v) = \begin{cases} 1, & \text{if } |v| = 0 \\ B_v(|v|), & \text{otherwise} \end{cases}$$

In Figure 3.13, this process is illustrated for operator 7.7. The upper right corner of operators has the computation of the number of alternatives that can be extracted using it as a root.

The total number of plans is the sum of possible plans rooted in any of the root group's operators:

$$N = \sum_i N(v_i), \qquad v_i \in G_{root}$$

where G_{root} denotes the root group.

Computing the counts for operators takes linear time, as each operator has to be visited exactly once.[3]

[3]For the number of logical operators for the problem of join reordering, see [OL90,

3.3.3 Unranking Plans

Before we describe the unranking mechanism in detail, it might be helpful to give a short outline of the idea:

Starting with the root group and the rank r, we choose an operator of the group to be the root of the tree. We then compute a *local rank* for this operator. This local rank for an operator v is in the interval $0,\ldots,N(v)$. Now, assume operator v has children alternatives

$$\{w_{1,1}^{(v)},\ldots,w_{1,j_1}^{(v)}\},\ldots,\{w_{n,1}^{(v)},\ldots,w_{n,j_n}^{(v)}\},$$

with $n = |v|$. *n subranks* are computed, and used in each child choice to recursively unrank a subplan. The resulting tree is assembled from unranked suplans, using v as the root.

Detailed steps are described next.

1. Given a pair (r,G) consisting of a rank and a group we determine which operator of this group becomes the root of the sub-plan.

   The first physical operator in the group covers the plan numbers $0,1,\ldots,N(v_1)-1$, the second $N(v_1),N(v_1)+1,\ldots,N(v_1)+N(v_2)-1$ and so on. Thus, the sought operator has index

   $$k = \max\{i|\sum_i N(v_i) \leq r\}.$$

   v_k becomes the root of the (sub-)plan. The local rank r_l of v_k is

   $$r_l = r - \sum_{i=1}^{k-1} N(v_i)$$

   The local rank is necessary to determine the subranks for the children in the next step. ◇

2. Using the concepts introduced in the previous section, we can write the subrank for the i-th child as

   $$s_v(i) = \begin{cases} R_v(i), & \text{if } i = 1 \\ \left\lfloor \frac{R_v(i)}{B_v(i-1)} \right\rfloor, & \text{else} \end{cases}$$

   with

   $$R_v(i) = \begin{cases} r_l, & \text{if } i = |v| \\ R_v(i+1) \bmod B_v(i), & \text{otherwise} \end{cases}$$

   ◇

PGLK97b]. There are a few physical operators for each logical joins, implementing different alternatives of hash join, merge join, and index lookups, so the number of physical joins is usually a small multiple of the count of logical joins.

We add the operator v_k to our plan and repeat this step for each child, i.e., for the i-th child we unrank $(s_v(i), G_i)$ where G_i is the group for this child. We repeat the steps recursively until we reach the terminal operators.

Unranking is in $O(m)$, m being the number of operators in the tree, which is limited by the number of groups in the MEMO.

Example.

This example describes the steps necessary to unrank a plan in detail for the MEMO structure as shown in Figure 2.5. We unrank plan number 13 in group 7, i.e., we unrank the pair $(13, 7)$. First, we determine the operator which becomes the root (operators that become part of the plan are underlined):

$$k = 1, v_k = \underline{7.7}$$

since v_1 covers the plans $1, \ldots, 22$. The local rank computes to

$$r_l = 13.$$

For the first operator in a group like 7.7, the local rank is always equal to the global rank within the group. With

$$R_{7.7}(2) = 13, R_{7.7}(1) = 1$$

the subranks for the children compute to

$$s_{7.7}(2) = 6, s_{7.7}(1) = 1,$$

i.e., we have to unrank the sixth possible subtree of the right child, and the first of the left. We unrank the subranks in the children's groups, i.e., $(s_{7.7}(1), 4)$ and $(s_{7.7}(2), 3)$. Unranking $(s_{7.7}(1), 4)$ gives

$$k = 1, v_k = \underline{4.3}$$

since 4.3 covers the first subplan, and 4.4 the second. As there are no further children, no subranks need to be computed and unranked. For the right son we have to unrank $(s_{7.7}2(,)3)$, which delivers

$$k = 1, v_k = \underline{3.4}$$

Here, the local rank computes to

$$r_l = 0.$$

With

$$R_{3.4}(2) = 0, R_{3.4}(1) = 0.$$

The subranks for the children compute to

$$s_{3.4}(2) = 0, s_{3.4}(1) = 0.$$

Finally, unranking $(s_{3.4}(2), 0)$ yields

$$k = 1, v_k = \underline{2.3}$$

and for $(s_{3.4}(1), 0)$ we obtain

$$k = 1, v_k = \underline{1.3}$$

In total we unranked the operators 7.7, 4.3, 3.4, 2.3, and 1.3. They span the tree shown by darkened operators in Figure 2.5.

3.3.4 Verifying Query Processors

Besides its use to analyze cost distributions be it by sampling or complete enumeration (see next chapter), the counting and un-ranking mechanism presented is also of very practical relevance to the ongoing development process of Microsoft SQL Server: The verification of both query optimization and query execution [5].

The choice of an execution plan is the result of various, interacting factors, such as database and system state, current table statistics, calibration of costing formulas, algorithms to generate alternatives of interest, and heuristics to cope with the combinatorial explosion of the search space. Normally, experimental validation and testing of the query processor is limited to consider the one plan that was chosen by the optimizer for execution. This is a severe limitation, as this plan is only a minuscule fraction of the space of alternatives. In fact, during regular development and maintenance of a query processor, it has been our experience that some code defects can remain undetected for a long time, until the right combination of factors steer the optimizer to chose a plan that exposes the problem.

In [Slu98], Slutz presents a tool to generate SQL statements probabilistically, to increase the test coverage of the database engine. One simple advantage of this approach is the sheer speed at which new, different tests are generated, making it a very effective testing tool. The same claim can be made for the selection and execution of multiple plans given a single query, which increases even further the coverage of the optimizer logic.

In the current implementation in Microsoft SQL Server, we extended the SQL syntax with an option to specify what plan to use for the execution.

```
SELECT  *
FROM    Professors P, Students S, Enrolled E, Courses C
WHERE   S.Name = "Sam White" AND
        S.SID = E.SID AND
        E.Title = C.Title AND
        C.By = P.PID
OPTION  (USEPLAN 8)
```

Figure 3.14. Extended SQL syntax to specify what plan to use for execution

The SQL statement in Figure 3.14 causes the optimizer to build the MEMO structure, count the possible plans, and select plan number 8 for execution.

Using scripting primitives, any given query can be extended easily with the OPTION clause and a loop construct that iterates over a deterministically or randomly selected set of possible plans. This way developers are for instance able to generate test cases for specific queries.

Some advantages of using these techniques in testing are:

1. It is easy to generate large test sets for the engine to scrutinize both correctness of the query execution and its performance.

2. The results are simple to verify since all plans should deliver the same outcome. The probability that an incorrect result is overlooked is rather small as opposed to external testing where each result requires manual verification.

3. It is possible to test operator implementations that the optimizer would not chose with the catalog data in the test database.

4. Optimizer decisions and correct assembling of plans by the optimizer can be easily verified. This point is of particular importance when extending the set of both operators and their implementations.

5. The verification and calibration of cost formulas is no longer restricted to one single plan per query but can also check cost values of sub-optimal plans.

6. The enumeration of complete search spaces for small queries helps check and analyze optimizer principles like cost-bound pruning and search strategies.

The features described are part of the routine testing in the development of Microsoft SQL Server.

3.4 Summary

The query optimization problem can be abstracted in several ways. The most common approach is to focus on join ordering only, since joins are in most cases the cost dominating operators. Another important model is the further simplification using only cross products. Like its richer brother, it is still NP-hard indicating already that cartesian products play a particularly important role.

In this chapter, we presented techniques how to generate all possible solutions for the XOPT, and JOPT, and for cost based query optimization in general. The first as well as the last are based on counting and unranking of n-ary trees , whereas the second one, for the JOPT, utilizes an algorithmic schema of sequences.

The techniques presented can now be used to derive complete or partial cost distributions by enumerating or sampling the search spaces of different problem instances.

4

Cost Distributions

The term *cost distribution* refers to the distribution function of the occurring values of the objective function in the entire search space of a combinatorial optimization problem. This distribution in turn is defined by its *density*, i.e. the frequency of the single values; we will use both terms synonymously, context permitting.

Let us first motivate why cost distributions are an important characteristic of search spaces, and in particular preferable to a characterization by other means. Optimization problems are in general abstracted as finding the minimum, or maximum respectively, of an objective function over a multi-dimensional space. Notions such as "local-minimum", "landscape" or "hill-climbing" implicitly refer to the idea of "neighborhood" among solutions, i.e., a topology of the search space. However, such a topology is not intrinsic to the problem, but defined—intently or not—by the encoding of solutions, or by transformation mechanisms used in search algorithms, e.g. Simulated Annealing. In contrast, the cost distribution in the search space is natural to the problem and has no arbitrary component.

Our principal goal within this section is to identify characteristic features of cost distributions to obtain deeper insight into the mode of action of different optimization techniques. Features of interest include total range of costs, mean and deviation of cost values, concentrations of solutions within the search space and the parameters responsible for the particular effect.

So far, cost distributions received little attention within the database community when it comes to analyze combinatorial optimization problems. Instead, topological structures have been devised and used, mainly because they offer a very intuitive connection to a number of optimization algorithms.

The concept is apparently not restricted to query optimization or a certain class of optimization problems, but extends generally to any combinatorial optimization problem. And, if the principle is generally applicable we can also expect the conclusions drawn from it to hold with similar generality. To this end, we will scrutinize cost distributions of several well-known

and well-understood optimization problems. Indeed, they are classics in this field: Number Partitioning, the Traveling Salesman Problem, and the Knapsack Problem. Modeling the occurring cost distributions with analytical means will also give us a good impression of the limitations of such techniques indicating what assumptions are justified and which need further attention.

An analysis based on cost distributions is specifically desirable as it offers the possibilities to predict the success of optimization strategies without having to analyze the problem and the algorithm at a level of detail that is hard to cope with. Rather, it provides a black box approach for this assessment.

Before we study the abovementioned optimization problems in detail, we need to clarify what to expect from such an analysis and what not. The prospect of such a powerful and general tool for the assessment of optimization algorithms needs to be carefully contrasted with its limitations. Let us first have a look at the means we are going to deploy.

1. We model average cases by using random variables that are often normally distributed and discuss in what way other instances may differ.

2. We approximate distributions by normal distributions where appropriate, substituting originally discrete distributions with continuous ones.

The results are approximations for an increasing size of the problem instances with respect to the deviation of the underlying random variables. Moreover, it characterizes the distribution of a problem *on average*. As we will see, the mathematical model is usually very accurate for large problem sizes and reflects important trends, as far as the classical optimization problems are concerned. For the cross product optimization and join ordering problem we will not be able to provide as accurate a model as for the traditional problems.

In any case, these models do not map particular instances to distribution functions directly. For some of the problems it is not too difficult to come up with special instances that fall beyond our analysis leading to unpredictable distributions. The existence of such cases is of interest for two reasons. First, we need to discuss whether real world instances of a certain problem are usually closer to the average model or often include degenerated cases. Secondly, the susceptibility to such degeneration differs considerably from problem to problem.

This chapter is completed with cost distribution of fully fledged TPC-H queries extracted from Microsoft SQL Server, confirming our findings for the simplified models of cross product and join order optimization.

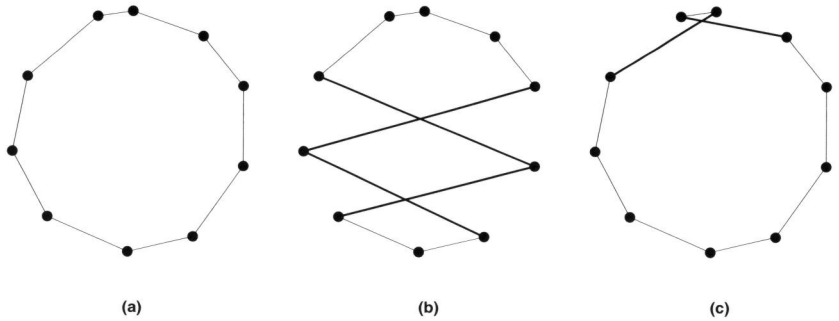


Figure 4.1. Alternative tours for a Traveling Salesman Problem; (a) optimal tour, (b) and (c) tours neighbored to optimal tour under different topologies

4.1 Topologies and Landscapes

Ever since the introduction of blind search algorithms, relentless effort has been devoted to characterizing the search space and its influence on the search algorithms. Usually, the terms *topology* and *landscape* are used to describe certain properties and relations among solutions. Though an important means for the interpretation of certain effects occurring in some optimization algorithms these structures are not intrinsic to the problem.

Given an optimization problem we are usually able to develop a certain notion of distance between solutions that expresses the degree of similarity. Consider for example the Traveling Salesman Problem where the shortest tour via a number of cities is sought. Two tours that differ only in so far that two cities are exchanged while keeping the order of the remaining could be considered similar. More general, the distance between tours could be expressed by the minimal number of pairwise exchanges needed to transform one tour into the other. A pair of tours with distance 1 could be referred to as neighbors. Ergo, we defined a topology on the search space.

This particular distance measure may appear somewhat coarse as it does not take into account any locality. A natural demand for a similarity measure could also involve similarity in terms of the value of the objective function for both neighbors. Exchanging arbitrary cities, however, may result in a neighborhood between the optimal tour and a tour which is nowhere near to be optimal with respect to its length. Thus, it seems reasonable to restrict our distance measure to pairs of cities that are neighbors within the tour, i.e., they visited subsequently without further vias in between. Consequently, we defined a new, fundamentally different topol-

ogy on exactly the same search space. Only for a very small set of tours the distances are the same under both distance measures. In Figure 4.1a, the optimal tour of a simple Traveling Salesman Problem is shown. Possible neighbors under exchange of subsequent and arbitrary cities are depicted in 4.1b and c, respectively.

Our definitions of distances can be used in a rather straight forward way to generate neighboring tours given one initial tour by swapping pairs of cities—either a pair of neighbored or non-neighbored cities. A large variety of different schemas for a guided exploration of neighborhoods has been suggested in the literature [FJMO95]. These search mechanisms proceed transitively in the sense that they investigate also neighbor of neighbors etc. if they are promising.

In order to conduct such a guided exploration of neighborhoods in the quest for the best tour, all tours have to be seen with respect to the objective function, the length of a tour. Combining both those components, a structure which is often referred to as landscape—specifically in the context of genetic algorithms as *fitness landscape*—is obtained. In case the topology defines a planar graph, the landscape can be visualized as a two-dimensional manifold in R^3.

Viewing our landscapes for the Traveling Salesman Problem from this perspective we obtained one rugged and one relatively smooth landscape for the same problem. Kaufman, one of the pioneers in fitness landscapes developed several models and notions of landscapes along the line of different degrees of ruggedness including techniques how to transform or modify them [Kau93].

However, neither of the two landscapes sketched above is intrinsic or natural to the current problem. The problem is well-defined without any such component. So, the only characteristics of the search space are the size of the space and the costs of all its elements. To avoid any misunderstanding at this point, certain optimization algorithms need and define indeed such landscapes, and we will also devise some models for them in one of the next chapters. However, for the analysis of the search space independent of a search algorithm, these models are only of limited use.

Furthermore, we will avoid the notion of fitness but prefer the term costs as fitness can be interpreted—and is indeed often used this way—as relative fitness. In Section 4.2.3 we will give an example for such a relative fitness measure. As opposed to this, costs refer to the absolute value of the objective function.

4.2 An Excursion: Some NP-Complete Problems

In order to get an impression of the concept of cost distributions and its parameters we make a short excursion and analyze and model three fundamental problems. These examples will also be also be useful for further considerations of the difficulty of problems in the following chapters.

4.2.1 Number Partitioning

We define the problem POPT as the associated optimization problem of the PARTITIONING problem [GJ79]:

Let S be a set, and w a weight function that assigns every element of S a certain value. Find a partitioning of S into $\mathbf{S}_1$ and $\mathbf{S}_2$ such that the expression

$$\left| \sum_{s \in \mathbf{S}_1} w(s) - \sum_{s \in \mathbf{S}_2} w(s) \right|$$

is minimal. PARTITIONING in NP-hard and so is POPT.

We model the sizes of the partitions with random variables S_1 and S_2 denoting the size of $\mathbf{S}_1$ and $\mathbf{S}_2$respectively. It is sufficient to focus on S_1 since $S_2 = |S| - S_1$. The set sizes are binomially distributed with equal probability, i.e., for every element, the probability to belong to either $\mathbf{S}_1$ or $\mathbf{S}_2$is $\frac{1}{2}$. For the expected sizes of the sets

$$E(S_1) = \frac{1}{2}|S|$$

holds. We model w, the individual weights with a random variable ω. Let μ and σ be mean and deviation of ω and

$$X := \sum_{i=1}^{S_1} \omega$$

be the weight of a set. According to the central limit theorem of statistics we can approximate the distribution of the weight of a set by a normal distribution $N(\hat{\mu}, \hat{\sigma})$ with $\hat{\mu} = \frac{N}{2}\mu$ and $\hat{\sigma} = \sigma\sqrt{\frac{N}{2}}$. Without loss of generality, we assume $\mu = 0$. We denote the density of $N(0, \hat{\sigma})$ with ϕ.

The density of the approximated cost distribution is then

$$\psi(x) = \int_{-\infty}^{\infty} \phi(t) \cdot \phi(x + t)dt$$

for $x \in R_0^+$.

In Figure 4.2 an experimentally obtained cost distribution—i.e., its density—is contrasted with the approximated one. S is implemented as normally distributed set of random numbers with mean $\mu = 0$ and deviation $\sigma = 10$. The size of S, $|S|$ is 100. The experimental data was obtained by a sample of size 10000. The frequency of occurring differences and the analytically determined probability are shown as functions over the weight differences. The figure shows strong coincidence between the analytical and the experimental results.

The most important feature of the cost distribution is its monotonic decrease which implies that the difference zero, the optimal result, occurs with the highest probability of all solutions.

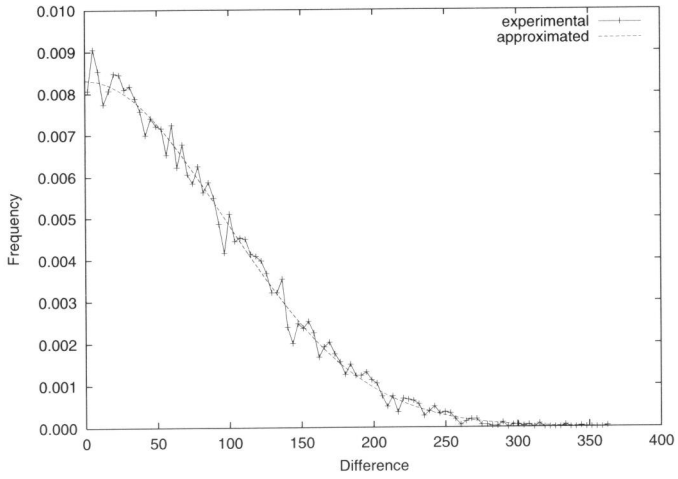


Figure 4.2. Comparision of analytically and experimentally determined cost distributions for $|S| = 100, \sigma = 10$. The experimental distribution is obtained from a sample of size 10000

The accuracy of our analytical assessment relies on the abstraction according to the central limit theorem. The less resemblance the distribution of the sum of weights of the partitions bears with the normal distribution the less accurate our results are, or are to be expected. Apparently, some pathological cases are easy to construct. Consider for example an instance, where all elements have the same weight, or only very few different values. The resulting distribution consists of only one or a few points respectively. Consequently, the approximation by a continuous function is not very meaningful.

On the other hand, if the contribution of the single weights to the sum is sufficiently small, the abstraction by normal distributions is justified independently of the particular distribution of the weights. To assess the susceptibility of this assumption, we varied the distribution and deviation of the random numbers in the original set and also varied the number of elements.

As a measure of difference between experimentally and analytically obtained distribution, we use the *Kulback-Leibler Divergence*—also known as *relative entropy*—, which is defined as

$$D(P,Q) = \sum_x P(x) \cdot ln \frac{P(x)}{Q(x)}$$

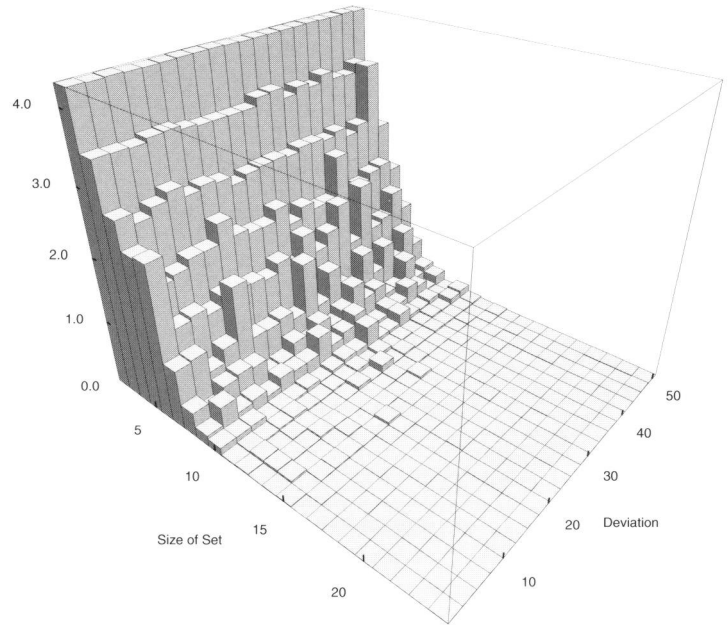


Figure 4.3. Kulback-Leibler Divergence of approximated and actual cost distribution as function of set size and deviation. Elements of original set normally distributed

for two distributions P and Q. The divergence is always non-negative and the smaller the value of this expression the stronger the distributions resemble each other. For instance, the divergence of the distributions in Figure 4.2 is less than 0.006.

In the following experiments we used randomly numbers generated according to Normal and Gamma distributions.

For the experiment with normally distributed numbers, we varied the deviation from 1 to 50 and the size of the original set between 2 and 25. In Figure 4.3, the divergence as function of size and deviation is shown. Very small set sizes, result in the highest divergence, which does not come as a surprise since the deviation for 2 elements for example can hardly compare to a continuous function. Distributions for very small sets contain mostly noise. However, the distributions become quickly stabilized with increasing size. For sets with 10 or more elements, the approximated distribution virtually coincides with the experimental one. As the plots show, this process is independent from the deviation of the underlying distribution of

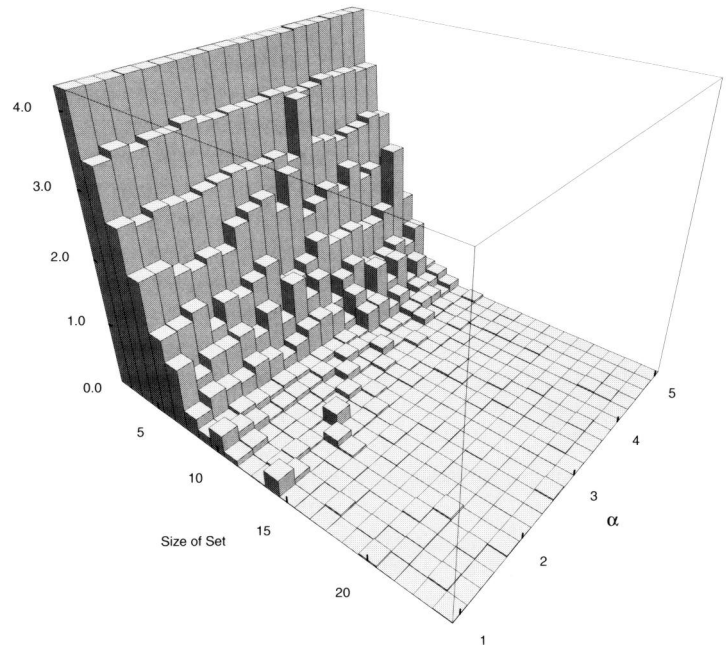


Figure 4.4. Kulback-Leibler Divergence between approximated and actual cost distribution as function of set size and α. Elements of original set distributed according to Gamma distribution

the numbers.

One might argue, that using a symmetric distribution for the numbers is the reason for the latter effect. In the next experiment, we used numbers distributed according to a Gamma distribution with shape parameters α between 1 and 5. The Gamma distribution is asymmetric coinciding with an exponential distribution if α equals 1. In Figure 4.4, the divergence is plotted as a function of the set size and α. As the figure shows, the influence of the underlying distribution is negligible. Again, the set size is the dominating factor: For a problem instance with more than 10 elements, the analytical approximation is highly accurate.

4.2.2 Traveling Salesman Problem

The next problem we analyze is the Traveling Salesman Problem, one of the classic problems in combinatorial optimization. Given a graph with nodes,

Figure 4.5. Map for Traveling Salesman Problem *usa13509* of TSPLIB

connected by edges of certain lengths, we are interested in the shortest possible Hamiltonian Circuit, i.e., a complete tour where every node is visited exactly once. The length of a tour is the sum of the lengths of its edges [GJ79].

The variant we consider here is known as *Symmetric Euclidean Traveling Salesman Problem* and is given by the co-ordinates of the nodes. Every node is connected to every other node by an edge the length of the Euclidean distance of the two nodes. In Figure 4.5, an example is shown. The nodes represent the 13509 cities of the United States with more than 500 inhabitants. This particular problem ranks among the largest Traveling Salesman Problems that have been solved to optimality to date [ABCC98].

In more formal terms, we can state the problem as finding a permutation $v_1, \ldots, v_n$ of cities given as two-dimensional vectors such that

$$l = |v_n - v_1| + \sum_{i=0}^{n} |v_i - v_{i-1}|$$

is minimal. The first part of the sum, results from the condition to connect first and last point of the tour.

In order to approximate the cost distribution of a problem instance we first determine mean μ_c and deviation σ_c of the pairwise distances of the cities. For our example, the histogram of the associated distribution is given in Figure 4.6.

We model

$Y :=$ *"Length of random tour"*

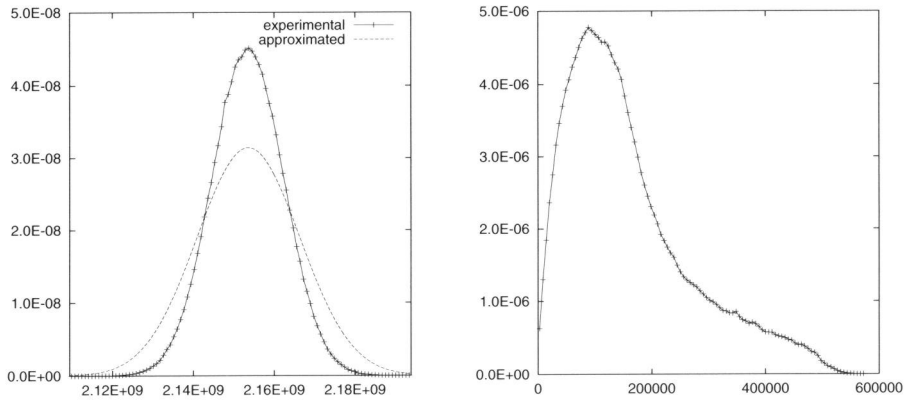


Figure 4.6. Comparison of analytical approximation and actual cost distribution (left); Distribution of pairwise distances (right)

by

$$Y := \sum_{n} X,$$

where X is a normally distributed random variable with parameters μ_c and σ_c.

According to the central limit theorem of statistics, we can approximate the cost distribution of all tours through n cities by a normal distribution with mean

$$\mu_a = n \cdot \mu_c$$

and deviation

$$\sigma_a = \sqrt{n} \cdot \sigma_c.$$

Figure 4.6 shows both cost distributions obtained from a sample of size 10^7 and the approximated one for the example problem. The approximated distribution resembles the actual, obtained by sampling, setting off the symmetry as the major feature of the distribution. Despite the length of the sum and the seemingly coarse approximation by μ_c, μ_a appears highly accurate. Only the deviation σ_a is slightly too large.

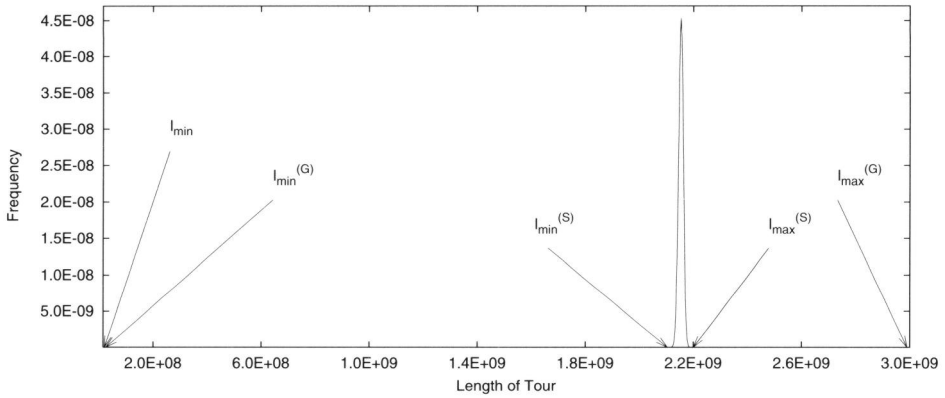


Figure 4.7. Comparison of approximated and actual cost distribution including optimum and longest tour found with greedy heuristic. Distributions coincide almost completely. l_{min} denotes the global optimum, $l_{min}^{(G)}$ and $l_{max}^{(G)}$ shortest and longest tour found by greedy algorithm, $l_{min}^{(S)}$ and $l_{max}^{(S)}$ shortest and longest tour found by sampling

Observation.

As with all minimization problems, the left most quantile of the cost distribution contains the optimal solution. However, the sample can be somewhat misleading since it does not unveil how far to the left the optimal tour is. Theoretically, the optimal tour could have length 199000 as well as 1990 without influencing the shape of the remaining distribution. In this particular example, the optimal tour is of length 57201.

As a consequence, the probability to find the optimum or any tour of length close to it in a random sample is almost zero, i.e., sampling limits us to an analysis of the majority of the plans, which have length close to the *average* tour. To correct the histogram in this respect we need to adjust the range to cover minimal and maximal solution. For nearly all TSP problems in TSPLIB, the optimum is known. For the few, very large instances that have not yet been solved to optimality, at least tight bounds for the optimum are known. For the longest possible tour, we use an approximation generated with a greedy heuristic. In Figure 4.7 the rectified histogram is shown with characteristic points marked up. We will discuss the effect of these characteristics on optimization approaches in the next chapter and focus on the shape only for the moment.

As it became clear from the abovementioned example, a comparison of the actual and the approximated distribution has to take heavy tails into ac-

count, that is, we need an appropriate measure to compute a distance. The integral over the difference is not suitable as the distributions may for example differ by this means by a value close to 2—the maximum distance of two densities under this metric—although both distributions have almost all their weight in the same quantile of one mere per cent. To overcome this problem we take two separate measures, the overlap and the similarity of shape. The first one is expressed by the difference of the means, the second by the Kullback-Leibler Divergence after centering the distributions on their means. In all experiments, the sampled tours were generated with uniform probability. All instances show distributions that are symmetric and largely coinciding with the normal distributions used for approximation as the small values of κ indicate.

Another effect deserving special attention is the fact that the means sampled as well as approximated are in the right half of the total range. At first glance, one might attribute this skew to the fact that the left edge of the distributions is the absolute minimum whereas the right is only approximated with a greedy heuristic. However also the minimum obtained by greedy optimization is distinctly further away from the mean than its counterpart, the maximum. The explanation requires a look at the distribution of the pairwise distances between cities. This distribution is skewed as shown in Figure 4.6. Though not important for the Central Limit Theorem and the shape of the approximation, the skew determines the length of the distribution's tails. For very small problem sizes, this effect is almost negligible but gains importance with increasing problem size.

To test for the generality of our observation, we sampled and compared cost distributions for all 77 instances of the euclidean symmetric TSP in the TSPLIB [Waa99a]. The results are, without any exception perfectly in line with the findings above. In the appendix (page 177), a short overview of these extensive experiments is given in form of tables containing all characteristic values such as total range, sampled range, experimentally and analytically determined mean and deviation, as well as the divergence of approximated and actual distribution.

4.2.3 Knapsack Problem

The last problem in this excursion is the *0/1 Knapsack Problem.* It is described by a set of items which have profit and weight associated with them. The optimization task is to find a subset with the highest possible accumulated value so that its total weight does not exceed a certain capacity [GJ79]. The problem owes its name to the analogy of packing a knapsack.

In formal terms:

$$\text{maximize} \quad \sum_i p_i x_i$$

$$\text{subject to} \quad \sum_i w_i x_i \leq c$$
$$x_i \in \{0, 1\}$$

where x_i is a binary value that indicates whether item i is to be included in the subset or not.

Like with the Traveling Salesman Problem, branch and bound algorithms in various forms became the standard techniques to solve this problem and its derivatives. For a survey on developments in this area see e.g. [Pis95].

The Knapsack problem offers further interesting insights as it can be interpreted as a *multi-objective* optimization problem with the additional condition

$$\text{minimize} \quad \sum_i w_i x_i,$$

i.e., while maximizing the profit, at the same time the weight is to be minimized.

The valid solutions are to be interpreted in a two-dimensional space which is not ordered, i.e., two elements (W_1, P_1), (W_2, P_2) are incomparable if the pairwise comparisons of weight and profit are contradictory. For example, neither of the solutions $(1000, 1000)$ and $(2000, 2000)$ is superior. We say the solution (W_1, P_1) *dominates* the solution (W_2, P_2) if $W_1 < W_2$ and $P_1 > P_2$. In general there is no single optimal solution that dominates all the others.

When discussing fitness functions in Section 4.1 we already mentioned the term relative fitness, however, without giving a description yet. We can close this gap with the example at hand. An immediate definition of fitness is apparently its two dimensional cost value. However, many optimization algorithm need a one dimensional cost value. One way to obtain such a value is to define the fitness of a solution s *relative* to a set of S of solutions as the number of solutions in S that dominate s. A solution with fitness 0 is then preferable to a solution with higher values.

The cost distribution is independent of whether or not we analyze the single- or the multi-objective variant.

To model the problem, we first consider the case

$$\sum_i x_i = k, \qquad \text{with } k > 0$$

where a knapsack has to contain exactly k items. We can approximate both dimensions with normal distributions. If we assume the weight independent of the profit, we obtain the density:

$$\phi_k(w, p) = N(k \cdot \mu_w, \sqrt{k} \cdot \sigma_w) \cdot N(k \cdot \mu_p, \sqrt{k} \cdot \sigma_p).$$

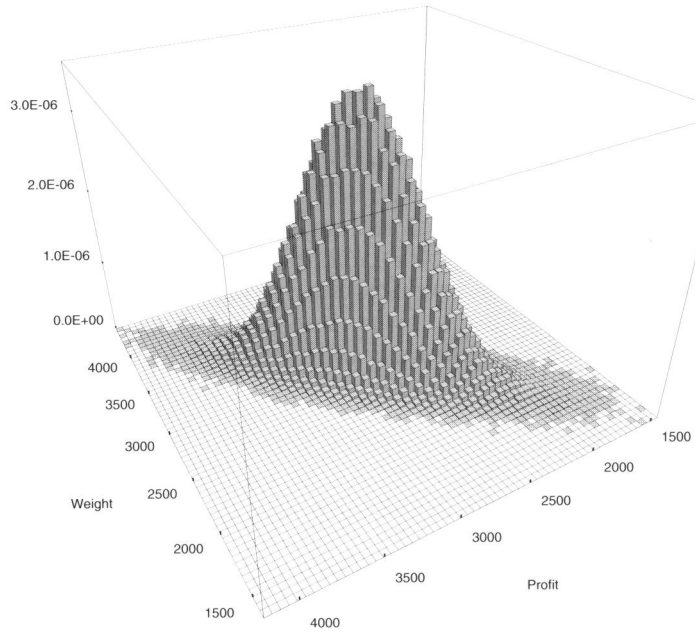


Figure 4.8. Cost distribution of Knapsack Problem obtained from a sample of size 10^6. Region of optima in the foreground

To determine the density for the general case, we need to sum over all possible values of k and multiply with the probability to have k items in the knapsack:

$$\phi(w,p) = \frac{1}{2^n} \sum_{k=1}^{n} \binom{n}{k} \cdot \phi_k(w,p)$$

In Figure 4.8 and 4.9, the cost distribution obtained by sampling and its analytical approximation are shown. The plots exhibit the weak spot of our modeling. The assumption that weight and profit are completely uncorrelated is hardly ever fulfilled but the effects of the correlation are significant. In case that weight and profit are *strongly* correlated, e.g. weight equals profit, the distribution reduces in width the extreme of which is a one dimensional distribution along the diagonal. For decreasing correlation, the distribution grows increasingly wider until it reaches the shape depicted in Figure 4.9. This behavior could be modeled by also taking the covariance of

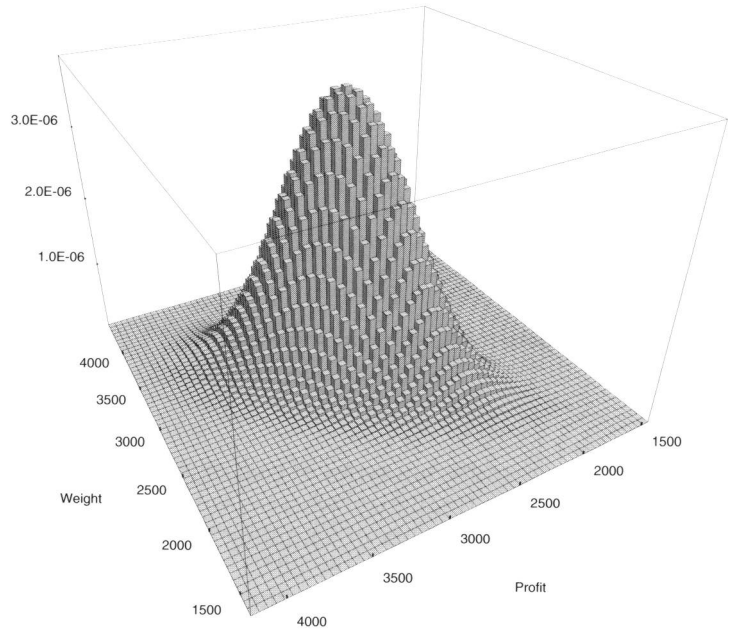


Figure 4.9. Analytical approximation for cost distribution of Figure 4.8

both random variables, weight and profit, into account. However, for our purposes, it is sufficient to understand the parameters and the quality of the shape.

4.2.4 Discussion

The lessons learned from this excursion can be summarized as

- Cost distributions are characteristic for optimization problems, as they reflect the basic properties of the cost function.
- For instances of non-trivial size, the distributions appear often very stable; Cost distribution of small instances may contain a considerable ratio of noise though.
- For most problems, pathological cases can be found, where the cost distribution collapses. For instance, all weights and values in the Knapsack Problem are equal or all cities in the Traveling Salesman Problem have the same coordinates.

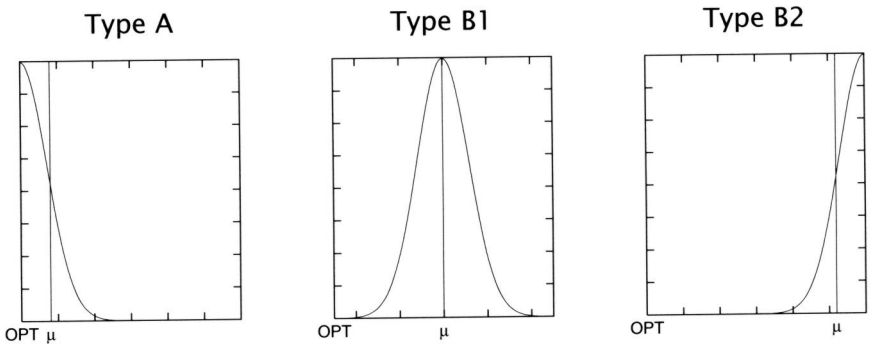


Figure 4.10. Basic types of distributions

We have seen two distinctly different types of cost distributions: With POPT, the optimum had the highest frequency, whereas with napsack and Traveling Salesman Problem the probability of sampling the optimum or any solution near it was practically zero. Yet, the latter two can be distinguished further by the distance of the optimum to the average solution. In Figure 4.10, the three types are sketched qualitatively. We have to be aware, that such a classification can cover real cost distributions only in a qualitative but not a quantitative way, i.e., cost distributions of other problems may appear to be of a shape that does not fit any of the three but is rather between two types. This differences need to be taken into account when discussing the effects on optimization algorithms and may require an "interpolation". We will get back to this point in Chapter 6 when discussing the effects of cost distributions on evolutionary algorithm.

Finally, a point that is important for a transfer of the results to query optimization is the nature of the cost functions scrutinized. In all three cases, the cost functions are *additive* in the sense that the total cost of a solution is the sum of certain input parameters. Accordingly the central limit theorem applies allowing a fairly accurate approximation.

4.3 Cross Product Optimization

The problem XOPT of cross product optimization is a close relative of JOPT—in fact it can be expressed as a special case of the latter where all occurring selectivity parameters equal 1. Although it might seem easier than JOPT on first sight, it is also NP-hard as Scheufele and Moerkotte showed in [SM97].

The problem is defined as follows: Let M_n be the set of labeled, binary trees with n leaves as defined in 3.1. For a tree $t \in M_n$, $K(t)$ denotes the set

of all subtrees, including the complete tree t. A function γ assigns every leaf a rational number. Find a tree t such that

$$c(t) = \sum_{k \in K(t)} \prod_{v \in k} \gamma(v)$$

is minimal, where $v \in k$ means v is a leaf of k.

A model for the cost distributions of non-isomorphic cross product trees is far more difficult than the ones presented in previous sections. Especially, the lack of a meaningful analytical description of the product of two normal distributed random variables confines us to strong simplifications already at an early stage. Nevertheless, we can derive trends and corroborate them experimentally.

4.3.1 Constant Relation Sizes

We represent the leaves of a tree by *independent identically distributed (iid)* random variables X_i, i.e., we assume the leaves normally distributed with the same parameters. For the moment, let us assume the deviation of X_i is zero, that is, all leaves are constant and have the same value. Thus, the cost is

$$\sum_{k \in K(t)} X^{|k|}.$$

In M_n, trees with the following ranks are distinguished. For $j \leq \lfloor \frac{n}{2} \rfloor$

$$l_{n,j} = \left(\sum_{1 \leq i \leq j} B(n-i) \cdot B(i) \right) - 1$$

and

$$b_{n,j} = l_{n,j} + B(n-j) \cdot B(j) - 1$$

denote the lowest and the highest rank of trees with n leaves in total and j in the right subtree.

Lemma 4.3.1
For all trees with rank $r > l_{n,j}$, the following holds

$$h(t(r)) < h(l_{n,j}),$$

where $h(t)$ denotes the height of a tree.

$t(l_{n,j})$ consists of a linear tree with j leaves and one with $n - j$ leaves; $t(b_{n,j})$ consists of two balanced trees (see Figure 4.11).

Proposition 4.3.2
Let $X_i = X$ be iid random variables with deviation zero, then the following holds:

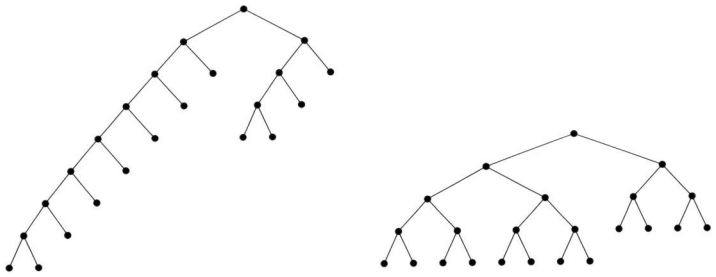


Figure 4.11. Trees $t(l_{n,j})$, $t(b_{n,j})$ for $n = 12$ and $j = 4$

a) $c(t(l_{n,j})) \geq c(t(r)) \geq c(t(b_{n,j}))$, with $l_{n,j} \leq r \leq b_{n,j}$,

b) $c(t(l_{n,j})) \geq c(t(l_{n,j-1}))$

c) $c(t(b_{n,j})) \geq c(t(b_{n,j-1}))$

Proof:
Consider the following algorithm. Given the tree $t(l_{n,j})$

1. Remove a leaf at depth d from t.
2. Insert the removed leaf anywhere in the tree at a depth less than d.

Applying this algorithm repeatedly we can generate any tree t from $l_{n,1}$. The cost associated with a tree is

$$\sum_i a_i X^i.$$

For every removal and insertion of a leaf we identify the closest common ancestor v_c for the removed and the inserted leaf. Costs outside the subtree below v_c are not changed by the maneuver. Assume this subtree has $k+1$ leaves. The costs of the root, i.e., $a_{k+1}X^{k+1}$ can be ignored as they appear in both trees. The costs for the new tree are bound by

$$bX^j + \sum_{\substack{i=1 \\ i \neq j}}^{k-1} a_i X^i \qquad \text{with } j \leq k, b = a_j + a_k$$

Thus, the cost difference between the original and the restructured tree is less than

$$\sum_{i=1}^{k} a_i X^i - \left(bX^j + \sum_{\substack{i=1 \\ i \neq j}}^{k-1} a_i X^i \right)$$

which can be rewritten into

$$a_j X^j + a_k X^k - (a_j + a_k)X^j =$$
$$\frac{1}{X^j}\Big(a_j + a_k X^{k-j} - (a_j + a_k)\Big) = \frac{1}{X^j}\Big(a_k X^{k-j} - a_k\Big),$$

which is greater than zero for $X > 1$.

Notice, this does not imply that all trees are monotonic in costs, rather there is a sequence of trees $l_{n,1}, t', t'', \ldots, t$ such that $c(l_{n,1}) \leq c(t') \leq c(t'') \leq \cdots \leq c(t)$. All three propositions follow immediately from application of the above algorithm. □

The proposition provides upper and lower bounds for sets of trees in M_n with the same number of leaves in their left subtree. Since our ranking function was developed on a similar criteria (see Sec. 3.1), we derive implicitly tight upper and lower bounds for ranges of ranks. Both upper and lower bounds are monotonic across the groups.

Figure 4.12 shows upper and lower bounds for trees with 20 leaves and $\mu = 2, 5, 10, 100$ respectively. The curves show a steep descend narrowing the cost range of plans quickly. For larger values of μ, this effects is intensified. Plots for larger values are, besides the scale, indistinguishable from the last one.

4.3.2 Variable Relation Sizes

In a next step we loosen the restriction of constant relation sizes which were due to a zero deviation. As a consequence a tree is no longer associated with one single cost value but rather a distribution. We use ϕ_r to denote the distribution belonging to tree r. The distributions of different trees typically overlap in range.

The results obtained in the previous paragraph extend to the general case as the bounds of the total costs in the previous setting are now the bounds of the means of the distributions. The shift of the bounds implies here a shift of the complete distribution.

In Figure 4.13, these distributions and their shift is illustrated for a tree with 10 leaves and $\mu = 2$, $\sigma = 0.1$. We use this small relation size to keep the whole distribution narrow so that details are clearly visible. We will discuss larger ranges later.

The diagram is to be read column-wise, that is, the ranks of the trees are given on the one axis (0 is the linear tree, 97 is the bushy most) and the according distribution is plotted along the other axis. For illustration, the distribution of tree 10 is highlighted. Because of the small relation size, the single distributions are almost symmetric which makes the means of each distribution intuitively accessible as it coincides approximately with the maximum of the distribution.

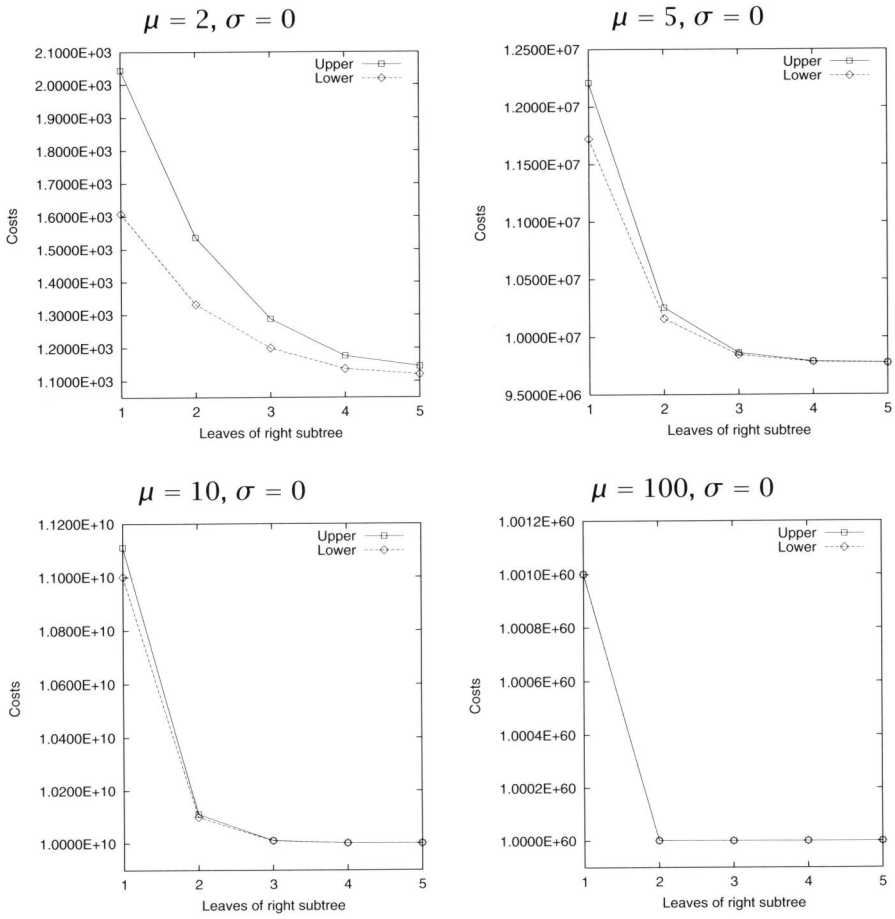


Figure 4.12. Upper and lower bounds in M_{10} as function of number of leaves in right subtree

The figure clearly shows the trends analogous to the previous section: The means shift from the middle of the cost range for tree 0 to the left side with increasing rank, i.e. bushyness. As pointed out before, this shift is not strictly monotonic. Using a larger mean for X has two consequences: The distributions (1) extend to a broader range and (2) show stronger asymmetry. This means for the shape of the result distribution that the "ridge" is further shifted to the left. The quality of the shape stays the same but is scaled in its extent. A larger deviation in X also causes stronger asymmetries. The mean of the distributions per tree is independent of the deviation of the distribution used for X.

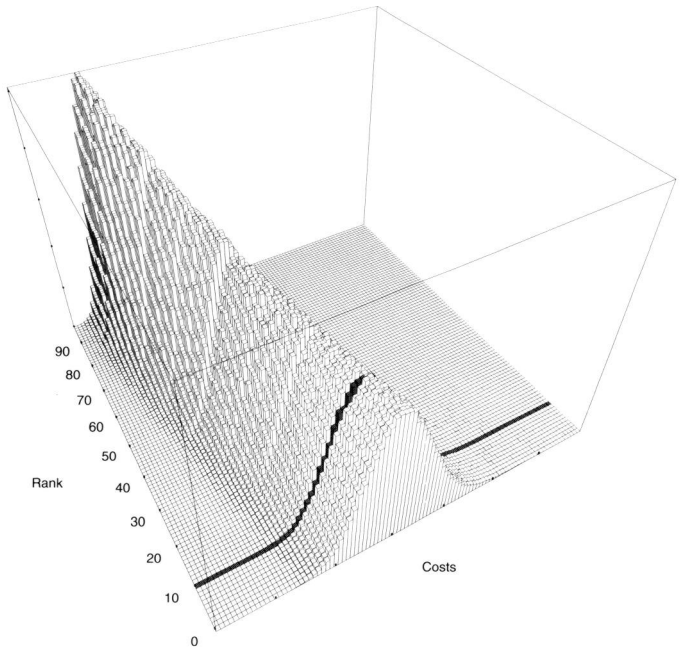


Figure 4.13. Cost distributions $\phi_0, \ldots, \phi_{97}$ of M_{10}; distribution of tree 10 highlighted

Fitting the puzzle pieces of the last two section together, we can now construct cost distribution for XOPT

$$\phi = \frac{1}{|M_n|} \sum_{i=1}^{|M_n|} \phi_i$$

The normalizing factor in front of the sum is necessary as all ϕ_i are distributions by themselves, i.e. $\int \phi_i = 1$. Multiplying with $\frac{1}{|M_n|}$ ensures $\int \phi = 1$.

In Figure 4.14 the dependency of ϕ from mean and deviation of X is shown. The plots show distributions for M_{10} with $\mu = 2, 5, 10, 100$. The deviation is given as fraction of μ, varying between 0.05 and 0.5. We cut the cost range down to $[0; 2\mu^n]$ to exclude outlying but uninteresting data points—beyond the right edge of the diagram, the curves converge further to zero but stretch an enormous interval. Including them in a linear scale would shift all interesting details close to the left fringe of the cost range scrambling all features beyond recognizability; using a logarithmic scale on the other hand causes significant distortions on the shapes of the curves.

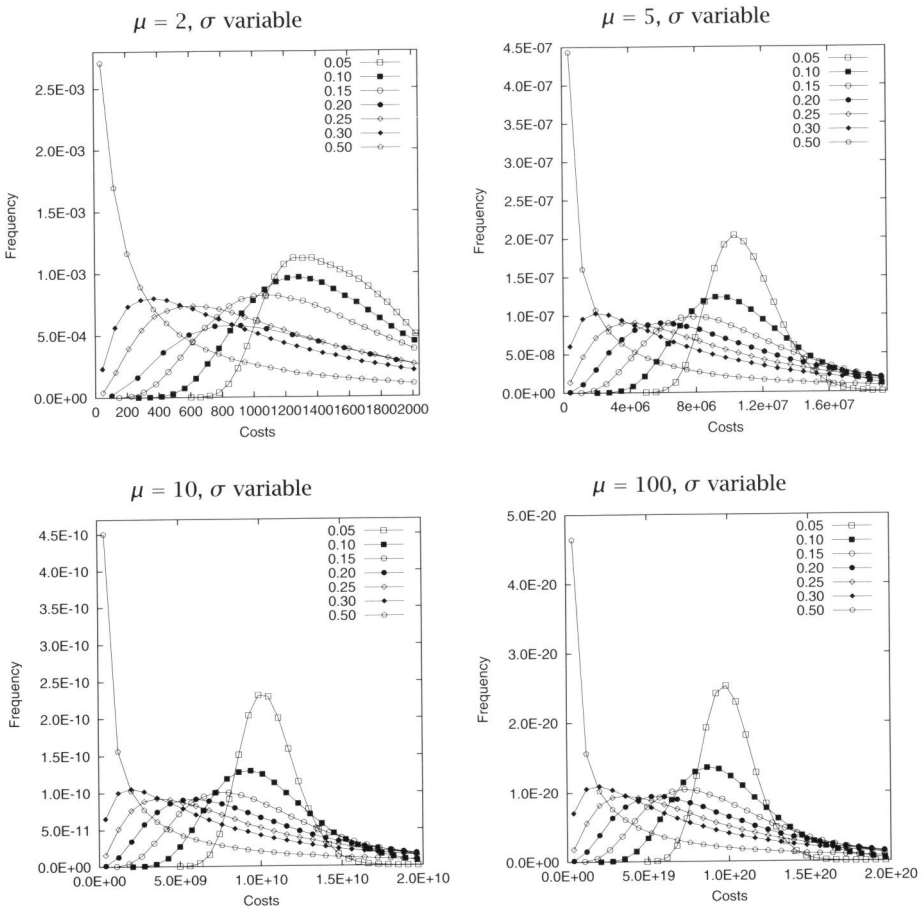


Figure 4.14. Cost distributions of M_{10} for $\mu = 2, 5, 10, 100$ and σ as fraction of μ

All plots show strong similarity: for small deviation of the underlying relation sizes, the distributions are rather symmetric resembling a normal distribution for large values of μ, but shifts quickly to the left, resembling an exponential distribution, with increasing deviation.

As opposed to the cost distributions analyzed in Section 4.2, XOPT displays a wider range of variety, i.e., it is more sensitive to its parameters. This sensitivity is due to the multiplicative character of the cost function where little differences are magnified by subsequent multiplications.

Finally, let us compute the mean of the resulting cost distribution.

Proposition 4.3.3
The mean $\mu(\phi)$ depends only on the mean of X_i but not on the deviation.

P r o o f: Let Y_k be random variable

$$Y_k := \prod_{i=1}^{k} X_i$$

modeling the product of k base tables. Moreover, consider

$$f_{n,i}(t) = |\{t' \in K(t) : |t'| = i\}|$$

which determines for a tree t with n leaves the number of subtrees with i leaves. The costs of a tree can be formulated as

$$c(t) = \sum_{i=1}^{n} Y_i \cdot f_{n,i}(t).$$

The expected value of Y_n computes to

$$E[Y_n] = \prod_{i=1}^{n} E[X_i] = \mu^n.$$

With

$$v_n(i) = \sum_{t \in M_n} f_{n,i}(t),$$

the number of subtrees in M_n of size i, we determine

$$\begin{aligned}\mu(\phi) &= E[\sum_{i=1}^{n} Y_i \cdot f_{n,i}(B)] = \sum_{i=1}^{n} E[Y_i] \cdot E[f_{n,i}(B)] \\ &= \sum_{i=1}^{n} \mu^i \cdot E[f_{n,i}(B)] = \sum_{i=1}^{n} \mu^i \cdot \frac{v_n(i)}{|M_n|}.\end{aligned}$$

Notice, $v_n(i)$ refers to M_n and not to a *single* tree alone. $v_n(i)$ computes to $v_n(i) = 0$ if $n < i$, $v_n(n) = t(n)$ and

$$v_n(i) = \begin{cases} \sum_{j=i}^{n-1} t(n-j) \cdot v_j(i), & n \text{ is odd} \\ v_{n/2}(i) + \sum_{j=i}^{n-1} t(n-j) \cdot v_j(i), & n \text{ is even} \end{cases}$$

otherwise.

□

On the other hand, we have seen the influence of the deviation of the X_i above. The higher the deviation, the more distinct the skew. As a consequence of the above proposition, the cost range expands with increasing deviation, i.e., the cost distribution contains an increasing number of outliers with high costs. In Figure 4.14 we concealed this effect by cutting the right tail off to avoid the strong distortion caused by the very long tails otherwise.

4.4 Join Order Optimization

We already gave a detailed outline of the join order problem in Section 2.2. However, at this point we derive a definition on the basis of the previous tree models in order to transfer previous results.

The join order problem JOPT extends XOPT by predicates which define selectivity between pairs of relations. This selectivity is determined by statistic data about the involved relations R_1, R_2 and is given as a numerical value $p(R_1, R_2)$.

If R_1 and R_2 are connected nodes of the join graph, $p(R_1, R_2)$ is a value between 0 and 1 including both, otherwise $p(R_1, R_2) = 1$. This definition extends to selectivities between sets of relations with

$$p(\mathcal{R}, S) = \prod_{i,j} p(R_i, S_j), \quad R_i \in \mathcal{R}, S_j \in S.$$

Applying this definition recursively computes the selectivity of a tree t as

$$p(t) = p(\{R_1, \ldots, R_n\}, \{S_1, \ldots, S_m\})$$

where $R_i \in t_l$ and $S_i \in t_r$. In case the join graph is tree-shaped, there is at most one predicate per subtree less than 1, in a clique graph there can be up to $|t|$.

Using this notion of selectivity, we can define JOPT to find a tree t such that

$$c(t) = \sum_{k \in K(t)} c(k_l) \cdot c(k_r) \cdot p(k)$$

is minimal. k_l and k_r are left and right subtrees of k, and $K(t)$ as defined as above.

This definition also allows for the inclusion of cartesian products. Normally, one wants to exclude cartesian products as they—usually but not always—lead to more expensive plans and, which is especially important for exhaustive search algorithms, extend the search space by a large factor. Depending on the join graph the up-scale can be exponential.

A variant excluding cartesian products can be defined by the additional condition

$$p(t) = 1 \quad \Leftrightarrow \quad \exists (R, S) \in G, R \in t_l, S \in t_r$$

where G denotes the join graph. This definition permits cartesian products only if they are forced by the user's query.

In order to understand which of the previous results offer the possibility to transfer to the more general case of JOPT we first want to point out the differences and common grounds.

Essentially, JOPT differs in three points: possible shapes of trees are limited to certain sets reflecting the join graph, the selectivities at each node in form of an additional factor have to be accounted for, and lastly, different algorithm may be used to implement the join operators.

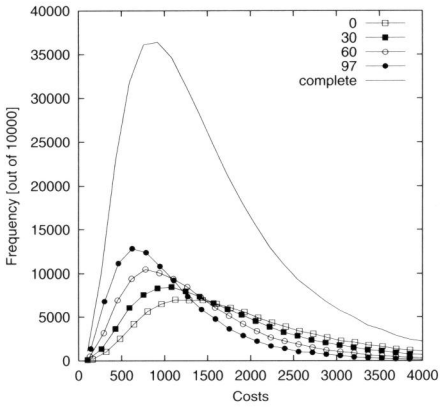


Figure 4.15. Frequencies of costs in a sample of 10000, using only trees $0, 30, 60$ and 97

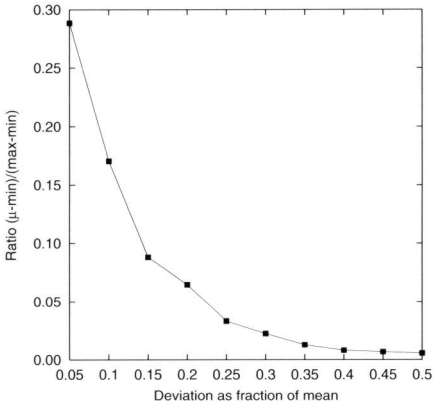


Figure 4.16. Distance of mean and minimum as ratio of total cost range

Join Graphs

According to the join graph only a subset of M_n can be chosen. The limitations in this respect vary drastically. The extremes of which are the *complete graph* and the *star graph*. For the complete graph, the n-clique, all trees are feasible, for the star graph, where all relations are connected to *one* central relation, only one single shape, the linear tree, is possible. Unfortunately, no concise way of describing the remaining graphs is possible.

For the complete cost distribution of the JOPT, this means, that it consists, compared to the XOPT, only of a subsequence of $(\phi_i)_i$. Rather than the slowly drifting sequence we studied above, we obtain only a few parts of the sum. Figure 4.15 shows an example where only 4 out of 98 possible tree shapes are used. Notice, the linear tree is always a feasible shape.

In the previous section we were mainly interested in the quality of the shapes and cut off the distribution at $2\mu^n$ to avoid distortion in the display and retain enough details of the distribution. However, for the further discussion it is also interesting to see the means—particularly the mean of ϕ_0—with respect to the complete cost range. In Figure 4.16, $\mu(\phi_0)$ is plotted as function of the deviation of the underlying relation sizes. In our model, the linear tree formed an upper bound. Accordingly, using less than $|M_n|$ trees makes for a complete cost distribution similar to ϕ_0, and exactly ϕ_0 if only one tree is used. However, as all possibly omitted ϕ_i are left of ϕ_0, ϕ cannot be further to the right than ϕ_0 is.

Selectivities

The role of selectivities is more difficult to assess. If we consider them, for the moment, independent of the particular relation, their effect can be largely disregarded—it is merely an additional multiplication with a random variable in a larger product of random variables.

However, in combination with dependencies, their influence is much stronger. They make the complete cost distribution appear more rugged in general. Tree shaped join graphs imply that all joins use only one predicate. Join graphs with cyclic edges can have several predicates on one join, and more notable can have different predicate combination on a join, if the shape or labeling of the tree is changed. The consequences are, on one hand that the selectivity on the nodes is stronger, i.e., closer to zero, which also causes a stronger left shift of the complete cost distribution. On the other hand the combination of predicates on one join provides higher stability since a join then does not only rely on one single predicate, thus the effect of single predicates is reduced. The latter argument, however, applies only to join graphs with a sizeable number of cycles.

Join Implementations

The discussion of the role of join implementations has been enlivened lately by Graefe [Gra99], pointing out that improvements of sophisticated join algorithms are measurable but not significant in general. Especially in small queries, the right choice of join implementations can have severe impact—in larger queries only if they are dominated by one or very few large tables. However, in most practical cases differences are rather small.

Another point reducing the influence of join implementation on the complete cost distribution is the fact that implementations are optional but not excluding, i.e., nested-loop-join is always a possible implementation; in the event of an equality predicate additionally the implementation of a hash-join becomes also feasible. In total, different join implementations intensify or smoothen dents in the complete distribution.

Before presenting own results corroborating this line of argumentation we also include further operators and consider the case of holistic query optimization in the next section. At this point however, a look at related work confirms our observations: Ioannidis and Kang [IK90, IK91] conducted several experiments where they determined the cost distribution for particular instances of JOPT by random walks in the search space. The number of solutions inspected was chosen large enough to obtain a fair sample. In this experiments they observed distributions similar to what they describe as Gamma distributions with shape parameter α between 1 and 2—the previous corresponds to exponential distributions, the latter is a strongly asymmetric distribution. In Figure 4.17, several instances of this distribution are shown. The statement of Ioannidis and Kang also applies to

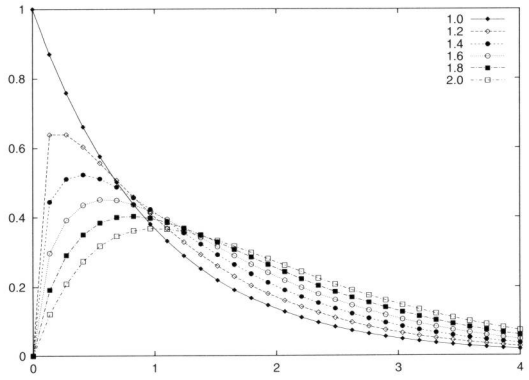


Figure 4.17. Gamma distribution with shape parameter $\alpha = 1, 1.2, 1.4, 1.6, 1.8, 2$

the distributions we found in Section 4.3: for large deviation of the underlying random variables the distributions found resembled an exponential distribution having a significant fatter tail though; for smaller deviations, the complete distributions shifted to the right (see Figure 4.14). The same causality was reported by Ioannidis and Kang in terms of catalog variance. However, they do not specify the term catalog variance in detail. Particularly, this lack of specification lends strong support that the multitude of parameters they included additionally into their cost models are only of marginal influence to the shape of the distribution.

Further observations similar in trend have been reported on by Swami [Swa91], König-Ries *et al.* [KRHM95], Stillger and Spiliopoulou [SS96], as well as Steinbrunn *et al.* [SMK97]. However, the methods used for sampling were usually not fair, in the sense that the sample was not generated with uniform probability.

4.5 Meeting Reality

As sketched in Chapter 2, the relational algebra contains besides joins a large number of additional operators. In a practical implementation this number is further extended to allow for implementation details of both operators and underlying storage model.

Most of these operators are rather inexpensive compared to the join or cartesian product. Specifically unary operators like filters are of almost negligible cost in traditional I/O dominated cost models, though gaining higher importance in main-memory settings [BMK99]. The two most notable exceptions are the sort operator and its close relative aggregation.

		In a sample of 10000				
Query	#Plans	Min°	Mean°	Max°	costs° ≤ 2	costs° ≤ 10
Q5	68572049	1.14	17098	4034135	0.47%	12.15%
Q7	228107572	1.15	3318	178720	0.11%	44.55%
Q8	20112521035	1.01	111	609	1.11%	14.70%
Q9	67503460	1.10	4107	109825	0.11%	4.08%
Q5*	455348910	1.23	105418	1287700	0.29%	5.70%
Q7*	3907373772	1.48	1793052	1523086611	0.03%	2.79%
Q8*	4432829940185	1.31	28159718	32595091399	0.06%	1.85%
Q9*	250657568	1.30	38363213	35866936219	0.02%	7.00%

°as factor of the optimum; *including Cartesian products

Table 4.1. Parameters of search spaces of TPC-H join queries.

Both can contribute severely to the total execution cost of a query however, without the variety of possible combination as seen with join and cartesian products. That is, the contribution of those operators is a significant additional cost with small deviation.

How does this multitude of operators—together with a larger number of additional parameters in an industry quality cost model—now influence the cost distribution?

We can expect a similar effect as seen with the three optimization problem in Section 4.2. Increasing complexity in form of larger instances stabilized the basic shape distribution. In query optimization a similar effect is to expect as a larger number of operators means also a larger number of very small changes—besides a few possibly more severe.

In the following we determine cost distributions of real-world examples as they are encountered in commercial query processing, i.e., using a full-blown set of relational algebra operators.

Using our framework we are able to perform a fair random sampling of costs in the search spaces that are not limited to join ordering only but may include arbitrary relational operators, various kinds of indexes and aggregates, and even cover parallel processing. We carried out numerous experiments with both standard benchmark queries like TPC-H and customer queries taken from various applications [Tra99]. Under the precondition that the queries were of sufficiently large size the distributions obtained were characterized by a relatively strong concentration of costs close to the optimum.

Figures presented here are the result of experiments with TPC-H queries 5, 7, 8, 9, which are the join-intensive queries of the benchmark, and have a larger search space. Table 1 summarizes some of the relevant values obtained. The first four rows consider a space of alternatives that does not allow cross products; while the last four rows allow cross products. Each experiment consists of a random sample of 10,000 plans from the space. The measure of a very large number of plans in the space does not imply

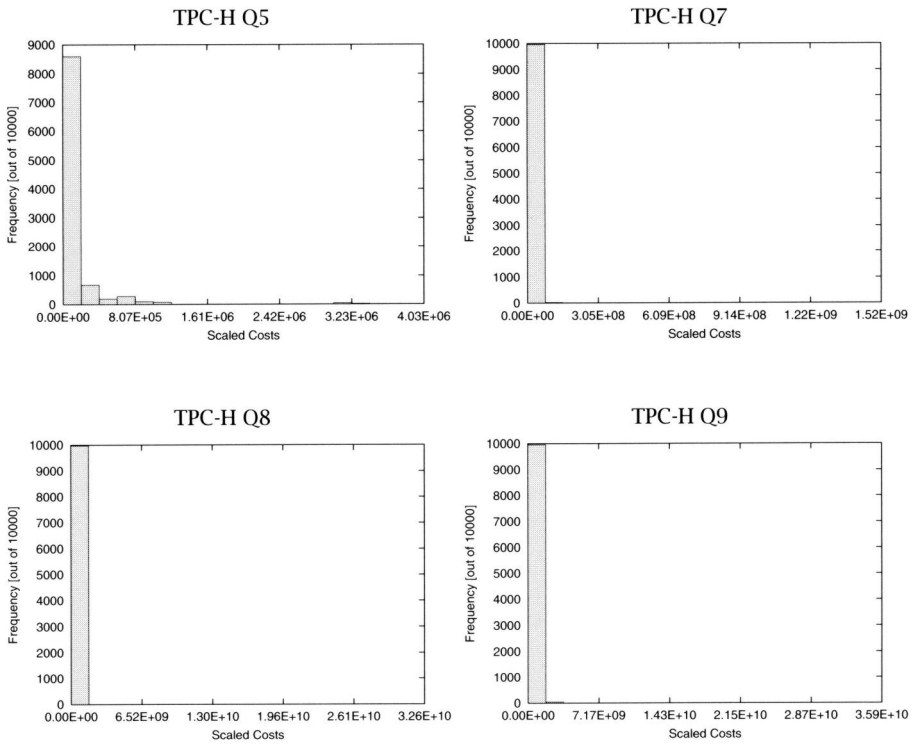


Figure 4.18. Cost distributions for TPC-H Query 5,7,8, and 9

that a structure requires as many bytes—recall that the plans are obtained through composition and reuse of operators from the compact encoding of the MEMO structure. All costs are normalized to the optimum plan found by the optimizer, which has cost 1.0.[1] The "min" column shows that with a relatively small sample, we are able to find plans that are pretty close to the optimum. In fact, the percentage of plans that are within twice the optimum cost is non-trivial. Also, it should be noted that the results are slightly different for the different queries, which vary in their selectivity and other properties. But there are no dramatic differences, and the same trends can be seen in all the experiments.

Figures 4.18, 4.19, and 4.20 provide additional information about the shape of the distributions. The space with cartesian products is shown,

[1] The actual values are proprietary information not published yet.

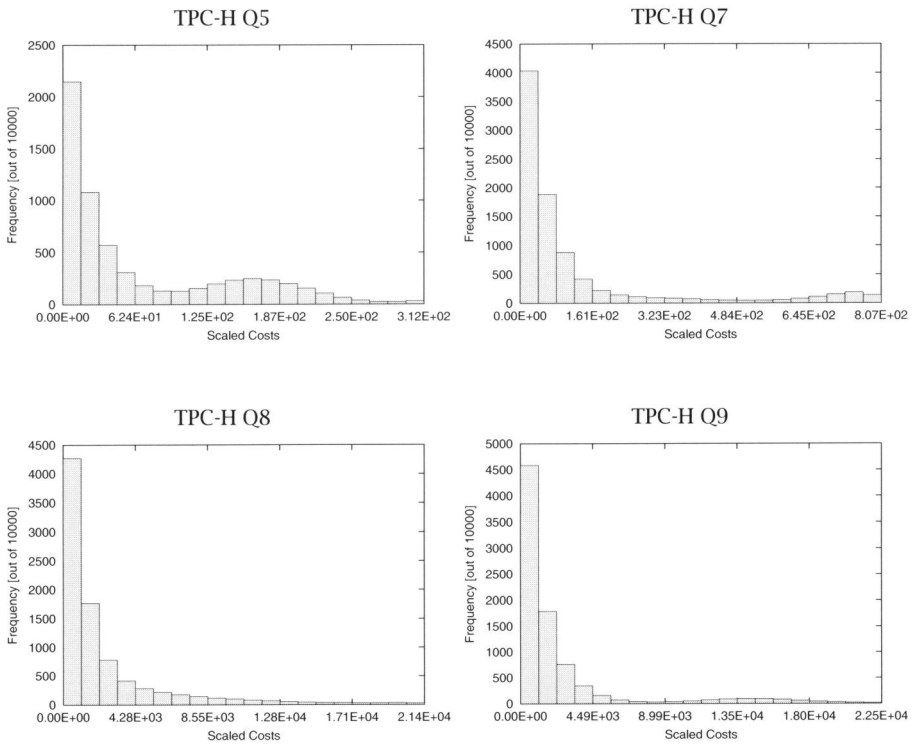


Figure 4.19. Cost distributions for TPC-H Query 5,7,8, and 9; lower 50% sampled costs

but the other is similar. Figure 4.18 shows the histogram of the complete sample, and it shows that most of the plans concentrate on the left, close to the cheap plans.

Figure 4.19 zooms in to the lower 50% sampled costs; that is, the part of the distribution that makes up for 50% of the space. Finally, Figure 4.20 shows a further zoom, to the points that are up to 50 times the cost of the optimum. In the "macro" view, we find that plans tend to be clustered to the left, close to the optimum solution. As we zoom in to the dense area, the histograms get less smooth, but they are still in the line of previous experiments.

The distributions of queries that contained few tables were of no particular shape but consisted only of random noise (e.g. TPC-H 6). Although it is hypothetically possible to devise queries of arbitrary size where the cost

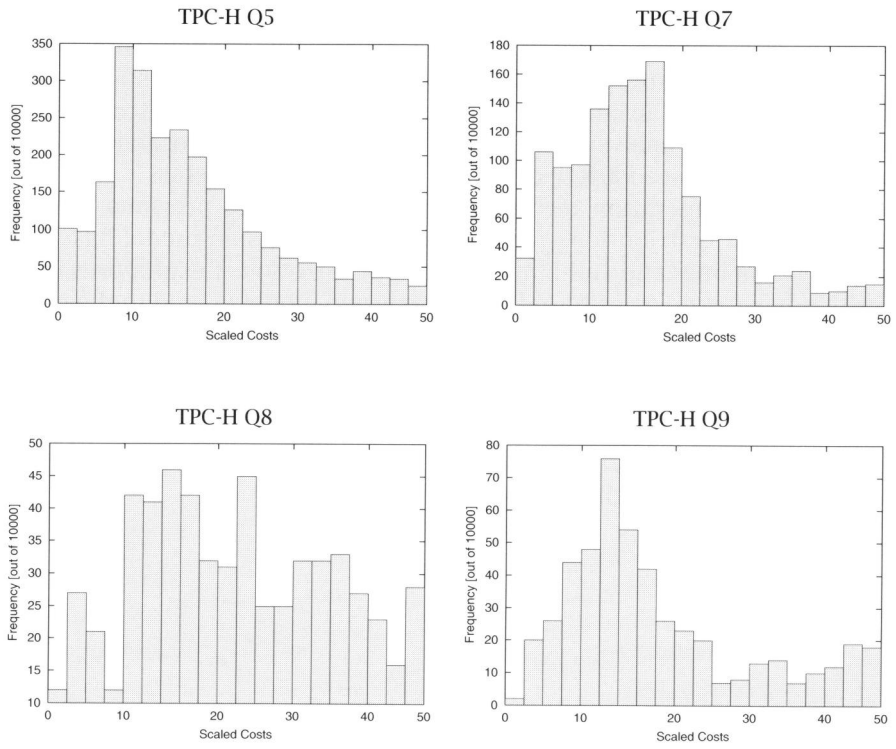


Figure 4.20. Cost distributions for TPC-H Query 5,7,8, and 9; blow-up of the interval $[0, 50]$

distribution degenerates to a single point—e.g. the cross product of several instances of the same table, with a space restricted to be linear joins—we never observed any such tendency in practical instances or customer queries.

4.6 Summary

Cost distributions characterize a combinatorial optimization problem as they are solely reflecting the objective function, uninfluenced of any element that is part of the solution rather than the problem. To view cost distributions without any connection to topologies and particular optimization algorithms is a novel approach toward an analysis that is not limited to certain algorithmic elements used to tackle the problem.

Typically, similar analyzes in previous work superimposed a kind of topology in form of transformation rules, crossover operators or heuristics first and restricted itself to a particular perspective on the problem.

In this chapter we analyzed cost distributions independent of such limitations, for several archetypal NP-hard problems: Partitioning, Traveling Salesman, and Knapsack Problem. These problems display three basic types of cost distributions which we will resort to at a later stage again. This excursion gave an impression of the fundamental means needed to undertake such an analysis but also to assess stability and generality of cost distributions on a variety of distinctly different problems.

Using the enumeration techniques developed in the previous chapter we derived cost distributions for both a simple cartesian model based on cross products only and full-blown query optimization. Results presented show that basic properties found in the cartesian model extend to the much richer problem of query optimization as they have the fundamental property of multiplication as a dominant principle of cost computation in common.

In the further analysis we therefore use approximations based on Gamma distributions to model the query optimization problem and to study various effects in the optimization process.

5

Assessing Difficulty

Since even the XOPT problem is NP-hard, query optimization as a whole cannot be expected to be any easier. As we noted at the very beginning, the theoretical complexity is in contrast with practitioners observation that most optimization decisions along the optimization of a query are straight forward. Roughly speaking, often only the technical details as to which implementation of a join should be used, where to deploy bit filters etc. are the more difficult ones to figure out. Of course, there is no clear division between those optimization decisions and counter examples dominated by this kind of technicalities can be easily made up.

In this Chapter we present fundamental considerations about the seeming discrepancy of theoretical intractability and the practical complexity of the query optimization problem.

To measure a problem's computational difficulty belongs to the very core of theoretical computer science. The well-known concept of NP-completeness which is based on verifiability in polynomial time and reducibility among all problems of this class emerged as a standard to assess the worst case complexity. Since its introduction, literally thousands of problems have been proven NP-complete acknowledging their computational intractability under the common assumption $P \neq NP$ [GJ79]. This concept owes its fame to both its simplicity as well as its generality. Though originally defined for decision problems that allow only a binary answer, the gap to optimization problems is easily bridged by giving a numerical bound B and reformulating the optimization problem as a decision problem as to whether there is a solution having lower costs than B, in event of a minimization problem. Maximization problems are analogous.

Much as NP-completeness has revolutionized computing since then, it also displayed a significant drawback: it describes "only" the worst case complexity, i.e., the potential difficulty of a type of problems without taking a given instance of this problem into account. However, it seems rather natural that not all instances of an NP-complete problem are hard to solve—and it is not only the pathological cases that are an exception, like a Knap-

sack Problem with equal weights and values for all items. Consider for example a Traveling Salesman Problem where all cities are situated on a circle having equal distance from their immediate neighbors; a greedy heuristic apparently delivers the optimal tour. Still, one could argue these cases are exceptions rather than average cases. A more striking example in this field is k-colorability of graphs which also belongs to the class of NP-complete problems. Turner observed that a simple backtracking algorithm is able to color almost all graphs efficiently and that hard instances are seldom encountered [Tur88].

These examples suggest two important further directions to investigate: (a) hard problems may have a large number easy to solve instances, and (b) the ratio of hard to easy instances may differ significantly from problem to problem area but may yet be characteristic for a certain problem.

In the following, we briefly outline the concept of *phase transition* which, still an unproven phenomenon, caused a major shake-up among the artificial intelligence and machine learning communities [FA85, HH87, CKT91]; for a brief survey see [Hay97]. According to this theory, hard cases in decision problems are concentrated in a small interval of a critical *order parameter*. It is, however, unclear how this theory extends to optimization problems.

In a case study of the *Asymmetric Traveling Salesman Problem*, we contrast phase transition elements with cost distributions and analyze the influence of parameters on the difficulty of the problem. Based on these observations we then introduce the concept of the *probabilistic difficulty* of a problem, which assess the chances of finding high quality solutions by random sampling. We discuss the transfer of our results to query optimization and lay the foundations for an analysis of randomized optimization algorithms.

5.1 Phase Transitions

The original term phase transitions refers to the observation that matter commonly undergoes changes of states depending on its temperature. Exceeding a certain temperature, solid matters may liquefy, liquids may evaporate, and reverse. The important detail, however, is that those phase transitions appear within a very small range of temperature compared to the ranges of stable states.

In the field of artificial intelligence the concept of phase transitions has first been observed by Huberman and Hogg [HH87], though a deeper connection between intractability and mechanical statistics has been conjectured already earlier by Fu and Anderson [FA85]. It gained enormous popularity over the last decade and in a ground breaking paper by Cheeseman *et al.* [CKT91] a wider range of applications is demonstrated. They suggest that decision problems have a single, characteristic parameter that determines the difficulty of the problem. Instances where the value of

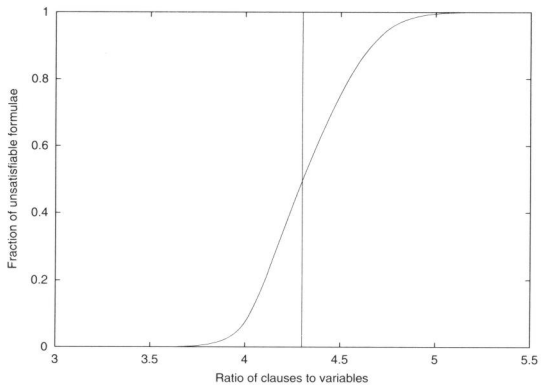


Figure 5.1. Phase transition in k-Satisfiability problem. Fraction of unsatisfiable formulae as function of the ratio of clauses to variables

this parameter happens to be within a small *critical* range appear difficult, whereas instances, where this parameter is off the critical range are easy. Easy means, the problem is still of the same complexity as the hard cases but algorithms like backtracking which potentially enumerate the whole space, can solve this instances with only few corrections, running practically in polynomial time [Pur83, CKT91].

The most prominent example, receiving multi-fold attention recently, is *k-Satisfiability* which is given by a number of disjunctive clauses each of which consisting of a conjunction of k variables. Each variable may also be negated. A typical instance of 3-Satisfiability is

$$(x_1 \wedge x_3 \wedge \overline{x_7}) \vee (\overline{x_4} \wedge \overline{x_5} \wedge \overline{x_7}) \vee (x_3 \wedge x_6 \wedge x_7) \vee (\overline{x_2} \wedge x_4 \wedge \overline{x_5})$$

where a bar above a variable denotes negation. For $k \geq 3$, the problem is NP-complete.

In Figure 5.1, the fraction of unsatisfiable formulae out of a sample of 10000 cases is shown as a function of the ratio of clauses to variables (after [MZK+99]). Up to a value of 4 there are hardly any unsatisfiable formulae—compared to the number of satisfiable ones—encountered. Within a small interval between 4 and 4.5, this ratio suddenly rises from almost zero to almost 1. For a higher ratio almost all formulae are unsatisfiable. Note, the curve sharpens with increasing size of the sample resulting in an abrupt change from 0 to 1 at the threshold. In several further publications following the paper of Cheeseman *et al.*, models for this threshold have been devised. However, the theoretically derived models do not match the ob-

servation, i.e., actual and expected threshold do not coincide. Finding the exact value for the phase transition is still an open problem.

The reducibility among all NP-complete problems suggests that phase transitions could be also transferred between problems. Particularly, the high stability and accuracy of this phenomenon with k-Satisfiability seem appealing to carry this concept over to other problems, separating areas of simple and difficult cases in other problems too. Moreover, using the link between decision and optimization problems, we could hope to find a similar order parameter in optimization problems.

However, the transfer to optimization problems raises a number of fundamental questions:

1. In k-Satisfiability, the order parameter determined the probability of the *existence* of a solution. Optimization problems do not have instances without solutions at all. What would different ranges of a phase transition parameter actually describe?

2. Difficulty in the context of a decision problem always means the difficulty to *solve*—i.e. obtaining a binary answer. In optimization commonly approximation are used in order to achieve reasonable tradeoff between time spent on the optimization and quality of the result. What would be a phase transition with respect to approximation?

3. Assuming we can derive a measure of difficulty, how can this measure be verified? Is there a difference depending on the algorithm used, or can a phase transition be verified with *any* algorithm?

To answer these questions we undertake a case study with the Asymmetric Traveling Salesman Problem. In doing so, we take care to keep close contact to our original problem of query optimization by mimicking the cost distributions found there.

The Traveling Salesman Problem is a highly suitable candidate to study effects regarding the difficulty of an optimization problem for several reasons. It is easy to describe and, as we have seen above, to model, it is known to be difficult not only in the sense of NP-hard but also in terms of approximability [Aro98] therefore often serving as a test bed for novel algorithmic principles, and finally, the large number of papers published about it provide a wealth of knowledge and observations that can be used for our analysis. Specifically, in context of the analysis of phase transitions, the Asymmetric Traveling Salesman Problem has been used as an experimentation platform (see e.g. [ZP94]).

This case study was also inspired by an example taken from TSPLIB—named BR17 in the collection. Though only of size 17 it poses a major difficulty as its number of optimal solutions is very large thus exact algorithms need a considerable running time *after* having found an optimal solution to rule out further improvements. Evidently, this behavior is tightly coupled with the underlying cost distribution. In this case study we gradually

transform an Asymmetric Traveling Salesman Problem from the average case into this very special extreme by manipulating its cost distributions.

5.1.1 Cost Distribution

In Section 4.2.2, we analyzed the cost distribution of the euclidean symmetric variant of the Traveling Salesman Problem, given only by the coordinates of the single cities, referred to as nodes in the following. The resulting cost distribution is, even for small instances, very stable and shows only little variation from what is basically a normal distribution. The problem as is, does not offer a possibility to change this distribution gradually to one that is similar to an exponential distribution, for instance. For an experimentation where we want to investigate the role of the cost distributions we therefore use the Asymmetric Traveling Salesman Problem, which is a generalization as all distances are given explicitly by e.g. a distance matrix. The cost distributions of the Asymmetric Traveling Salesman Problem can be influenced to a certain degree as we will show.

Like in the symmetric case, the cost distribution of the Asymmetric Traveling Salesman Problem converges toward a normal distribution as n approaches infinity due to the central limit theorem. However, the velocity of convergence is significantly lower than that observed in the symmetric case, which enables us to generate skewed cost distributions in the range from normal to exponential distributions.

The key to the desired skew is the *edge distribution*, i.e.,, the distribution of distances between cities (cf. 4.2.2). In contrast to the symmetric variant where these lengths were defined by the coordinates, we can now access them directly. This allows us to generate edge distributions that would be impossible when defined by coordinates in R^2.

Shortest and longest tour are bound by

$$l_{min} = \sum_{n} \min\{d_{ij} \text{ distance between nodes } i \text{ and } j\}$$

and

$$l_{max} = \sum_{n} \max\{d_{ij} \text{ distance between nodes } i \text{ and } j\}$$

where n denotes the size of the instance. We use the previously introduced approximation for the mean of the cost distribution

$$\mu = n \cdot \mu_c,$$

with μ_c being the mean of the edge distribution. We can achieve a skewed cost distribution by reducing the distance between l_{min} and μ_c with respect to the total cost range. To that end, we use Gamma distributions with shape parameter $\alpha \leq 1$, distributions that are left of the exponential distribution (see Figure 5.2).

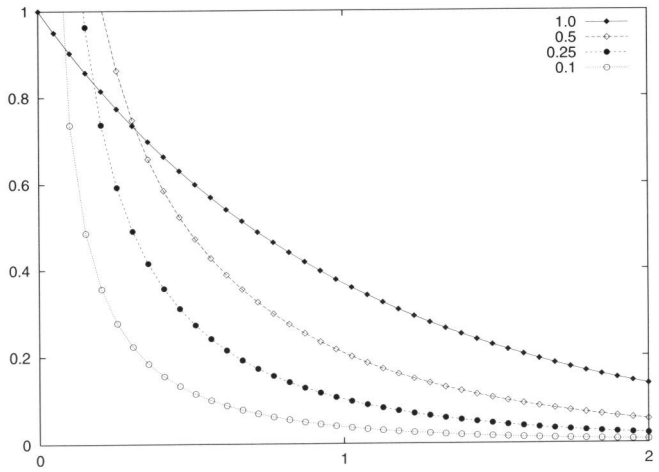


Figure 5.2. Gamma distributions with shape parameter $\alpha \leq 1$

The quality of the bounds depends on the number of edges that are of a length close to l_{min} and l_{max}. Using Gamma distributions with their very strong skew to the left, the lower bound is very tight, the upper bound rather loose, since we focus on left skews only, the loss of accuracy on the upper bound does no harm to our further considerations. However the fact that the lower bound is very tight is favorable as it reliefs us of the problem to determine the distance between l_{min} and μ_a. This is the more important, as the greedy algorithms we used in Section 4.2.2 perform poor on average in the asymmetric case.

The shape parameter α used in the Gamma distribution for the edge distribution serves as a measure of skew. In Figure 5.3 three pairs of edge distributions and resulting cost distributions are shown for a problem size of 50 cities. The tendency of increasing skew with decreasing α is clearly visible. For large α, the cost distribution becomes more and more symmetric, for small values, it shifts increasingly to the left covering the whole range of cost distributions we found for the query optimization problem.

To avoid rounding problems when adding two numbers that are close to zero, we cut the Gamma distribution on the left edge and use only values greater than 10^{-5} and multiply by $\frac{1}{\mu}$. Thus, the lower bound is

$$l_{min} \geq \frac{n \cdot 10^{-5}}{\mu},$$

whereas the mean can be simply approximated by n.

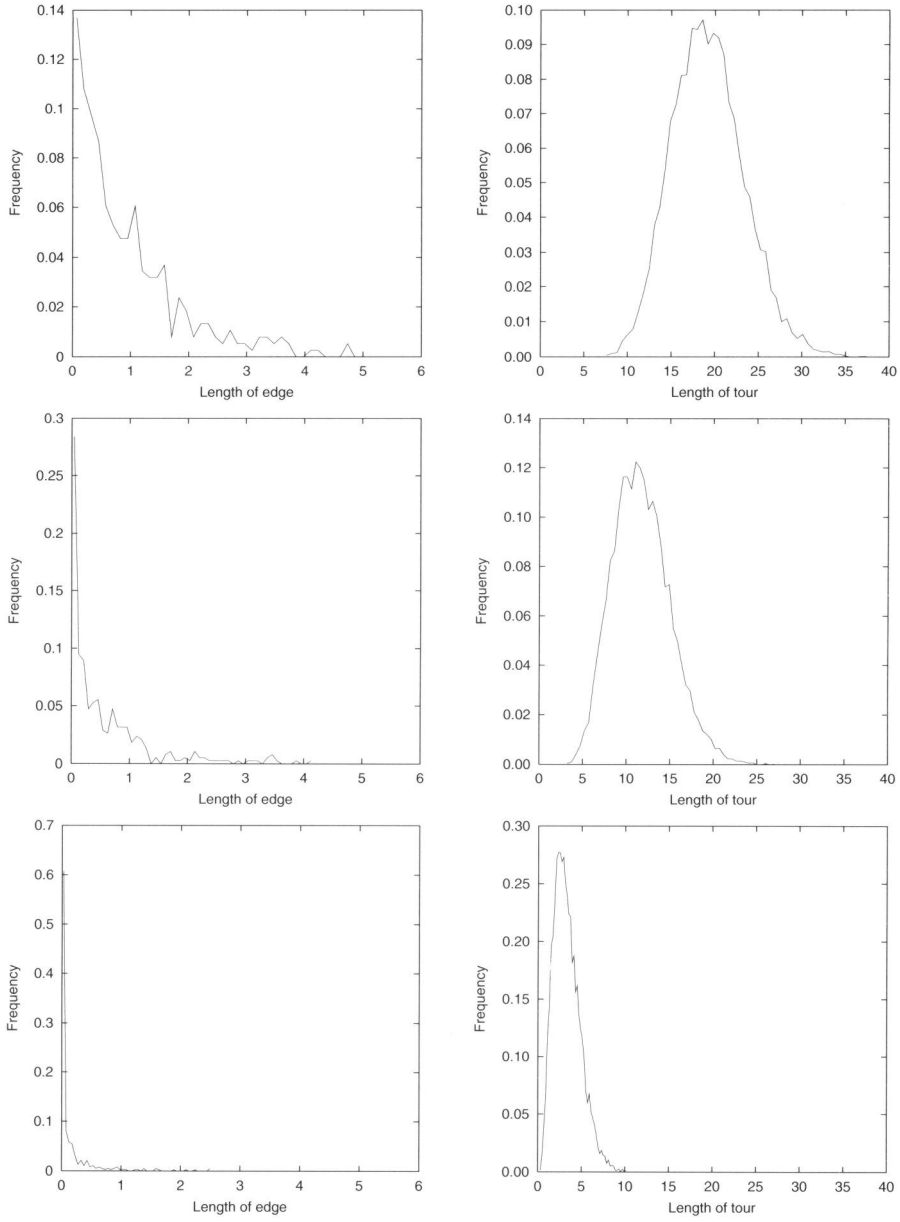


Figure 5.3. Skewed cost distributions for Asymmetric Traveling Salesman Problem of size 20. Left, distributions of edge lengths; resulting cost distribution right

5.1.2 Branch and Bound

Branch and bound algorithms emerged as the standard technique for solving both Symmetric and Asymmetric Traveling Salesman Problem to optimality. Specifically for the symmetric problem, a series of advancements which make use of the geometric properties have led to more sophisticated variants like branch and cuts algorithms [GJR84, PR87, PR91]. Using these algorithms, the problem sizes which could be solved to optimality were pushed from a few hundred in the mid 1980s up to 13509 in 1998 [ABCC98], the problem we analyzed in Section 4.2.2.

However, these algorithms are not applicable to the asymmetric problem so that the largest solved problems in this case are still only the size of a few hundred nodes.

In this Section we want to determine the effects of several parameters on the run time of a branch and bound algorithm. Let us first outline the principles of this technique in some detail to provide the background necessary to understand the later results.

The Asymmetric Traveling Salesman Problem can be written as a linear program of the form

$$\text{minimize} \sum_{i,j} d_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \sum_{j} x_{ij} = 1, \quad \forall i \tag{2}$$

$$\sum_{i} x_{ij} = 1, \quad \forall j \tag{3}$$

$$x_{ij} \text{ describe connected component} \tag{4}$$

$$\text{with } x_{ij} \in \{0,1\}, \quad d_{ij} \geq 0$$

(1) is the optimization objective with d_{ij} being the distance of nodes i and j, and x_{ij} a boolean variable that indicates whether i and j are connected in the tour. Condition (2) and (3) demand that there is only one incoming and one outgoing edge per node, and (4) is the requirement that all selected edges form a connected component. If (2) and (3) hold, (4) implies that we have found a tour.

A *relaxation* of the problem is a linear program where one or more of the conditions (2)-(4) are dropped in exchange of being able to solve the problem in polynomial time. A solution of the relaxation is consequently not necessarily a valid solution to the original problem, conversely however, every solution of the original is also a solution to the relaxed problem. Furthermore, a solution to a relaxation is also a lower bound to the original problem.

A branch and bound algorithm performs a conditional enumeration using a relaxation. In case of L_0 we start with the set of shortest outgoing

edges per node. Clearly L_0 is a lower bound of the shortest tour. If L_0 is a tour, it must be the optimal tour and we are done. Otherwise, L_0 contains at least one edge which does not belong to the optimal solution. We select one of the edges in L_0 and generate two subproblems: one where this edge is contained and one where it is excluded for the rest of the search, respectively. For the restricted subproblem, we compute a new relaxation L_0 and proceed recursively. Whenever a subproblem forms a tour, we found a global upper bound and all subproblems whose total length exceeds the global upper bound can be discarded because further restrictions can only increase their lengths.

In Figure 5.4, an outline of the branch and bound algorithms is given. It is based on a list Q that holds all the subproblems resulting from restrictions and relaxations. The function LOCALLOWERBOUND computes the length of the subproblem p which is the lower bound of all problems, which can be derived form p. The procedure FIXEDGE fixates a given edge all derived subproblems will include, whereas RESTRICTEDGE causes the opposite, the exclusion of the edge. After restricting, we have to compute a new relaxation, denoted by RELAX. The algorithm terminates as soon as there are no further subproblems in the list. For simplicity we omit additional, technical termination criteria needed if for instance no further restrictions are possible.

The solution to relaxation L_0 usually violates conditions (2)—(4) in that a considerable number of restrictions has to be carried out before a valid solution is accomplished. Better relaxations can be achieved by dropping only conditions (2) and (3), or condition (4). Dropping (4) immediately changes the problem into a two-dimensional assignment problem which can be solved in $O(n^3)$. We refer to this relaxation as L_1 in the following. We outline only the basic ideas of this technique and refer the interested reader to [Kuh55, JV86, EM92] for the details on how to solve the associated assignment problem. Condition (2) and (3) demand that any node has one incoming and one outgoing edge. We can remodel the original problem as bipartite graph $G(V, W, E)$ with one node in V and one in W for every original node. For every directed edge between two cities in the original problem, we add an edge between the corresponding nodes in V and W and assign the distance in the original graph as a weight to the edge. In Figure 5.5, an example with four nodes is shown. The left picture is the original Traveling Salesman Problem, the right is the assignment problem associated with L_1.

The perfect matching with minimal weight is the solution to the relaxation. In Figure 5.5, the edges belonging to the solution are indicated by thick lines. Transferring the result back to the Traveling Salesman Problem gives two unconnected subtours between nodes A,B and C,D with total length 4.

```
Algorithm BRANCHANDBOUND

g_u ← ∞
Q ← ⟨⟩
p ← solution to the initial relaxation
append p to Q

while(Q not empty) do
        choose p ∈ Q
        l_l ← LOCALLOWERBOUND(p)
        if (l_l < g_u) then
                if (p is tour) then
                        g_u ← l_l
                else
                        choose edge e in p

                        p' ← FIXEDGE(e, p)
                        append p' to Q

                        p'' ← RELAX(RESTRICTEDGE(e, p))
                        append p'' to Q
                endif
        endif
done
```

Figure 5.4. Outline of BRANCHANDBOUND algorithm

5.1.3 Experiments

Before we describe the actual experiments and their results, we want to outline the particulars of the setup.

The aforementioned problem instance BR17 illustrates that the histogram of lengths of edges alone does not sufficiently describe the hardness of the problem. Rather the location of the single edges within the problem is an important component which is unfortunately hard if at all possible to describe. Specifically, we lack a concise description which was available with k-Satisfiability in the form of the clause-to-variable ratio.

Another point that needs to be taken care of in the design of experiments is the randomness of a test case, that is even if a problem's probability to be easy to solve is close to zero there is still the chance that the initial relaxation—specifically when using more sophisticated relaxation techniques like L_1 or 1-trees—solves the problem. The generation of problems that anticipate solvability by initial relaxation is possible, but would mean an enormous adaption to the algorithm used—thus, probably not being

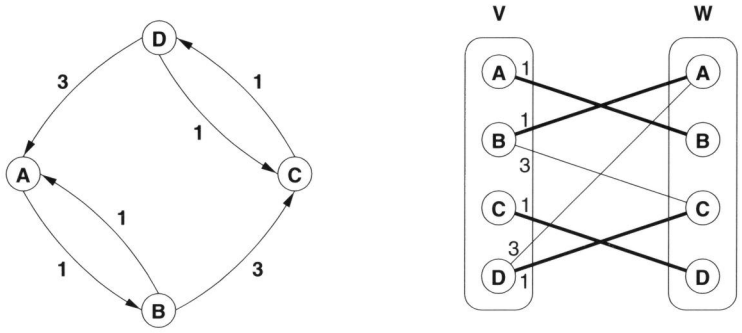


Figure 5.5. Relaxation L_1 as Assignment Problem

general enough any more to allow universal conclusion on the experiment.

To overcome both these problems we use very large numbers of problems per configuration and eliminate outliers by using the median rather than the arithmetic mean. The downside is an vastly increased running time in general, but also the limitation to smaller instances. Problems which require several hours to solve are out of the question in order to maintain practicability of the experiments.

We chose 1000 problems per configuration together with a limitation of the total running time, a setting that proved sufficient stability in numerous preliminary trials. The limitation of the running time has to be tuned carefully for each configuration to prevent untimely termination on average, but filter out outliers only. Since we take the median instead of the arithmetic mean, the truncation does not cause negative side effects. Similar techniques are commonly used in related work; see e.g. [CKT91]. Furthermore, experiments using L_0 relaxations are also limited by the memory requirements which quickly exceed several GigaByte for problems of size greater than 30 nodes. These memory limitations could be overcome by using depth-first-search instead of breadth-first-search, however, the latter proved significantly more effective in that a substantially smaller number of subproblems has to be solved.

As a measure of running time we use the number of subproblems, i.e., relaxations, solved instead of the actual elapsed time. This is a further concession to practicability to run large numbers of experiments in parallel on various platforms. For both relaxations L_0 and L_1, this measure turned out to be very practicable as the times needed for single relaxations differ only marginally.

In Figure 5.6, the number of relaxations solved is plotted as a function of the shape parameter. The smaller this parameter, the stronger the skew,

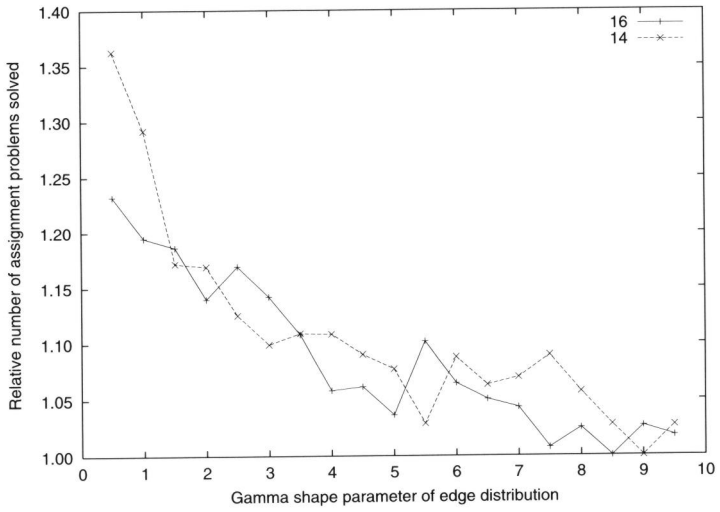


Figure 5.6. Number of subproblems solved as function of skew. Lower χ value indicate higher skew

i.e., the sharper the left flank of the resulting distribution.

The graphs uniformly show gradually increasing running time with decreasing shape parameter. This effect has a simple explanation by the upper and lower bounds used during the optimization. With the gradual shift of the distribution's bulk, the quality of the bounds remains largely unchanged, i.e., the number of subproblems between bounds increases. Since branch and bound is an enumeration technique—capable of exploring the whole search space in the worst case—the number of subproblems that need to be checked depends on the portion of the search space between the bounds. Accordingly, this portion and thus the running time increase with increasing skew. The graphs show scaled figures where 1 corresponds to the minimal number of relaxations solved per experiment.

5.1.4 Discussion

Let us now analyze the results from two different perspectives: in comparison with k-Satisfiability on the one hand and against the background of related work on the other.

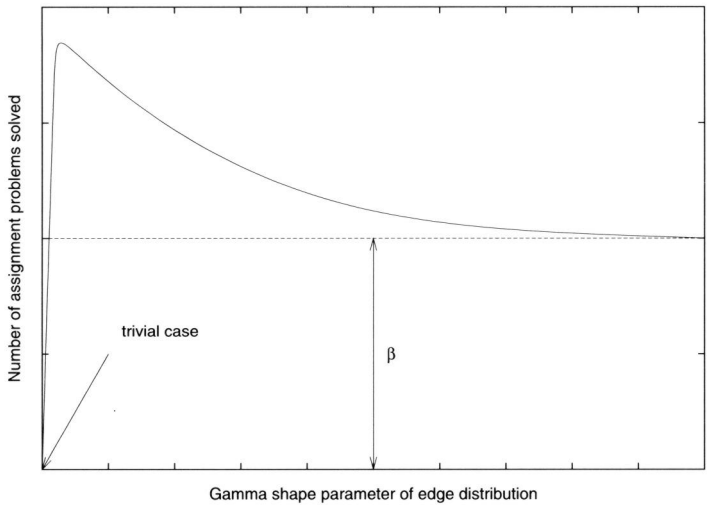


Figure 5.7. Qualitative model of the number of subproblems solved

k-Satisfiability.

In contrast to *k*-Satisfiability our experiments did not show as sharp an increase in running time as was reported there [CKT91]. Their results, suggested an area where difficult cases are concentrated and on *both* sides of this parameter range, the expected running times for the problem are very short compared to the ones needed in the critical range.

The number of solved problems is qualitatively shown in Figure 5.7. We can identify a region of harder problems framed by two areas of apparently easier problems in the Traveling Salesman Problem. The extreme of the cost distribution skew results in a cost distribution that consists only of one single data point. Then the first relaxation is always the optimal solution and thus, the problem is easy to solve as a whole. The other area of easy problems is the area with the least skew. However, the running time in this area, indicated by β in Figure 5.7, increases exponentially with the problem size. The running time for the case $\alpha = 0$ on the other hand is always constant no matter the problem size. The two areas, though separated by a region of higher difficulty, do not correspond in the sense seen in *k*-Satisfiability. Rather, the case $\alpha = 0$ appears to be a pathological extreme.

Moreover, compared to *k*-Satisfiability, the Traveling Salesman Problem does not display the sharp changes from one area to the other which is characteristic for phase transitions.

Previous Work.

Another point problematic with the concept of phase transitions is the observation that different algorithms encounter different kinds of difficulty. Deploying random sampling, as used in our assessment of cost distributions in Chapter 4, now as an optimization algorithm displays this discrepancy: *The stronger the skew the higher the probability to find near optimal tours in a sample of given size.* In contrast to the experiments above, random sampling additionally provides far higher stability as it relies only on the cost distribution whereas branch and bound techniques rely to a considerable degree also on the structure of the graph. Also, the changes in running time are gradually developing.

The difficulty appears fully reversed when using random sampling as optimization technique instead of branch and bound. Consequently, we can identify different kinds of what could be interpreted as a phase transition depending on the algorithm used and whether we solve to optimality or whether we are content with an approximation.

Contrasting these two basic directions is also important when comparing to related work. Cheeseman *et al.* conducted a similar experiment in [CKT91] however using Little's algorithm [Lit63] as optimization strategy, i.e. an approximation technique. Their observation is in line with what we found for bare random sampling: the more solutions there are close to the optimum the easier it becomes to find one thus the shorter the running time. They report on a precipitous decrease when they approach a level of maximal skew. However, different from the results for random sampling or branch and bound algorithms, the authors claim to have found two areas of similar difficulty separated by one of higher difficulty. They interpret their findings as phase transition in the process of reducing the deviation in the edge distribution.

Further related work concentrated on exact solutions of the Asymmetric Traveling Salesman Problem usually with variants of branch and bound techniques like Truncated and Epsilon Branch and Bound [ZP94], however without determining a threshold in the original problem, but rather devising techniques how to overcome difficult cases in enumeration trees in general.

Finding a distinct phase transition in query optimization can be expected to be as difficult as in the Traveling Salesman Problem. On the one hand, like with the Asymmetric Traveling Salesman Problem, there is no concise way of capturing the problem like with k-Satisfiability. On the other hand the multiplicative character of the cost function further compounds the distinction of influential parameters. König-Ries *et al.* undertook an attempt to identify a phase transition of the JOPT only based on the connectivity of the join graph [KRHM95]. However, this approach is questionable as authors used Simulated Annealing to test for the difficulty of an instance. Simulated Annealing is known as an algorithm which heavily depends on a set of parameters thus the experiments must be strictly inter-

preted against the background of a particular parameter setting. And furthermore, the case of connectivity greater 1 or even up to n, the complete clique graph (9 in their experiments) are rarely encountered in practice. Hence, these experiments fall short of explaining easier and more difficult cases in query optimization as known from practical examples.

Our results do not rule out the possibility of the existence of a phase transition in optimization and specifically query optimization. However, the concept as found in k-Satisfiability is not as evident in optimization problems for several reasons:

1. problems like Traveling Salesman Problem or query optimization lack a concise description similar to the one found in k-Satisfiability. Rather these problems constitute several largely independent dimensions of parameters. The resulting compound problems deny the boiling down to one single parameter of difficulty.

2. There is no obvious correspondence between decision problems and approximative optimization. Assume we found a phase transition for the case where solving to optimality is concerned, it is not clear what the consequences are for an approximation. As seen above, approximation can be the easier the more difficult the exact solution is to find.

3. In optimization difficulty appears to be difficulty with respect to a certain optimization algorithm particularly in case of approximation.

Having shown the problems of transferring the concept of phase transition to optimization we abandon this area and devise an own new measure of difficulty. Instead of focusing on a possible separation of areas of difficult and easy instances we discuss difficulty from a probabilistic optimization point of view using cost distributions.

5.2 Quality Measures

In Section 2.2 we pointed out the uncertainties of cost computation due to the underlying statistical data about the database and further inaccuracies arising from error propagation as show in [IC91].

Solving to optimality as seen in Traveling Salesman Problem and other NP-hard problems is thus not desirable and the result optimal by cost value is not necessarily the optimal one in the actual execution. Rather there is a range of solutions that are *good enough*. Further differentiation appears not useful.

In the following, the first discuss scaling-based quality measures and their drawbacks. We develop a new notion of quality base on ranges after that.

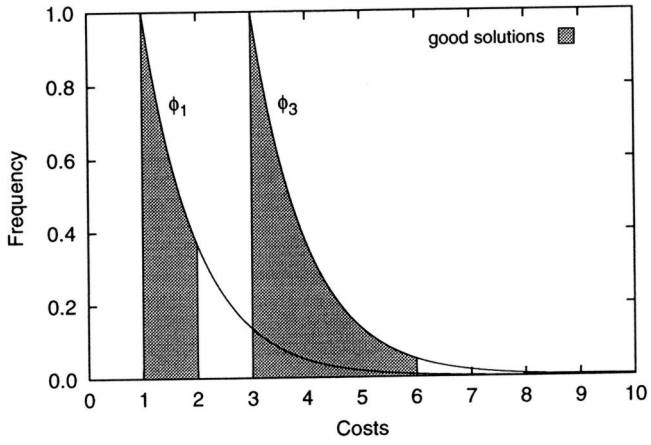


Figure 5.8. c_{min}- Classification

5.2.1 Scaling-based Classification

We need a quality measure based on the actual costs value that is abstracting yet meaningful. In [Swa91], Swami proposed a possible classification. Solutions are divided into three groups: *good*, *acceptable*, and *bad* query plans. Plans are considered *good* if they have costs below twice the minimal costs c_{min}, *acceptable* if they are no more expensive than 10 times c_{min}, and *bad* otherwise.

However, this schema suffers from the severe drawback to be not invariant under additive translation. Consider the two cases shown in Figure 5.8 where two queries have very similar shaped cost distributions. In this example we assume the distribution to be an exponential distribution for both queries, i.e. $\phi(t) = e^{-t}$. The only difference between the two is a shift along the x-axis.

Let us assume that for the original distribution the cost of the cheapest plan is $c_{min} = 1$ and that the average cost of a plan is $c_\mu = 2c_{min}$. The ratio of good plans in the search space, i.e. plans with costs below $c_{good} = 2c_{min} = c_\mu$ is

$$\int_{c_{min}}^{c_\mu} \phi(t)dt$$

which normalizes to

$$\int_0^1 e^{-t}dt \approx 0.63$$

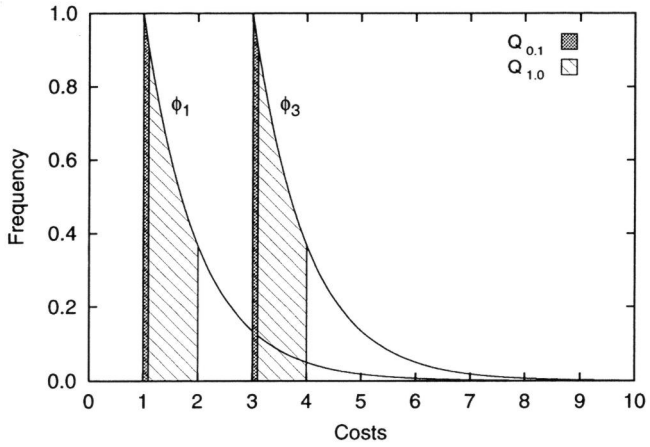


Figure 5.9. T- Classification

(see Figure 5.8, shaded area). We expect the ratio to keep being this way as long as the distributions are shaped similarly no matter what the actual cost values are. The previous classification, however, falls short of this invariance.

Translating the original distribution by adding a factor $2c_{min}$ to the costs, gives $c'_{min} = 3c_{min}$ and $c'_{good} = 2c'_{min} = 6c_{min}$ for the shifted cost distribution

$$\int_{c'_{min}}^{c'_{good}} \phi(t)dt$$

which is, after normalization,

$$\int_{3c_{min}}^{6c_{min}} \phi(t)dt = \int_0^3 e^{-t}dt \approx 0.95$$

(see Figure 5.8 dashed area). The ratio of good plans increased by 50% to 0.95 although the distribution stayed the same—the whole range of costs did not change either.

5.2.2 Range-based Classification

The cause for the insufficient valuing seen above is that only one single reference point, namely c_{min}, is taken into account. To overcome this drawback, we classify plans with respect to two parameters, namely c_{min} and c_μ.

We denote the quality of a plan q by its normalized costs

$$T(q) = \frac{C(q) - c_{min}}{c_\mu - c_{min}}.$$

The new measure is translation invariant. For the optimum, the normalized costs always equal 0 while $T(q)$ is 1 for plans of average costs. Plans above c_μ have normalized costs greater than 1, accordingly. In principle, the maximal cost value could also serve as a reference point, however, incorporating c_μ into the quality measure links it automatically to the particular distribution—c_μ is characteristic for a given distribution. In Figure 5.9, the areas of plans with $T(q) \leq 0.1$ and $T(q) \leq 1.0$ are shown for the same distribution as before.

In our experience high quality plans show a T of less than 0.1, although greater values are justified with respect to large join queries. Hence, the optimization goal we are aiming at is to find a plan with T below 0.1. In Figure 5.9 this target cost range is shaded.

5.3 Probabilistic Difficulty

Also with respect to the analysis of randomized optimization techniques in the following chapter, we develop a model of difficulty based on the nucleus of probabilistic algorithms, namely the randomly selecting of a single solution.

Randomly choosing an element from the unrestricted search space or a restricted subset of it is common to all these algorithms, be it in from of selecting an initial starting point or a larger set as it is the case with multi-start or genetic algorithms. In the further process, often subsets of the search space are used as candidates for additional random choices usually limited by means of a certain locality as in navigation-based algorithms like Hill Climbing or Simulated Annealing. The simplest of randomized algorithms using only the selection primitive without further restrictions is random sampling. For the moment we focus on uniform sampling only. In Chapter 8 we will also consider non-uniform selection schemes and discuss the differences.

In the following we develop a measure of difficulty, called *probabilistic difficulty* as it reflects the difficulty to optimize by random sampling. With the probability

$$Q(x) = \sum_{c=c_{min}}^{x} \phi(c),$$

we could simply use the probability to hit the range of costs below a given numerical threshold x

$$F(x) = 1 - Q(x).$$

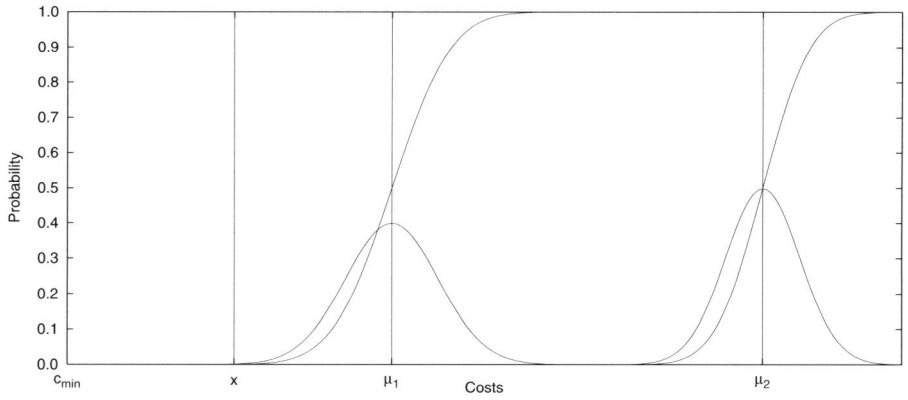


Figure 5.10. Cost distribution of types B1 and B2 with naive measure of difficulty; results of costs x are classified to be equally difficult to achieve under both cost distributions

The resulting measure has values between zero and one, where zero is reached in case of $x = c_{max}$. Though useful when computing the probability of success for repeated sampling, the measure displays a distinct disadvantage: outliers in the distribution are hard to distinguish. Consider the two distributions given in Figure 5.10 of type B1 and B2. $F(x)$ has a value of almost zero for both distributions, thus, they would appear to be of similar difficulty. We overcome this problem by first defining the target quantile and then determining the size of the interval we can expect to hit with this probability. The size of the interval directly translates to the distance from the optimum. Given for instance a distribution like in Figure 5.11, and a target quantile of 0.5, the resulting interval is $[0, \mu]$ as shown by the shaded area in the plot since the probability to select a solution in this interval is 0.5.

Similar to the quality measures presented in the previous section we rely on reference points. In the example, we took the whole cost range but for the reasons outlined above, we use optimum and mean again. Thus, we can define a problem's probabilistic difficulty as

$$D(x) = \frac{Q^{-1} - c_{min}}{\mu - c_{min}}$$

where Q^{-1} is the inverse of the previously defined Q. D does not map a problem to a single value of difficulty but provides a mapping of target quantiles to the ratio of intervals.

As the size of the target quantile depends strongly on the particular problem, the size of the given instance, and factors like the uncertainty in

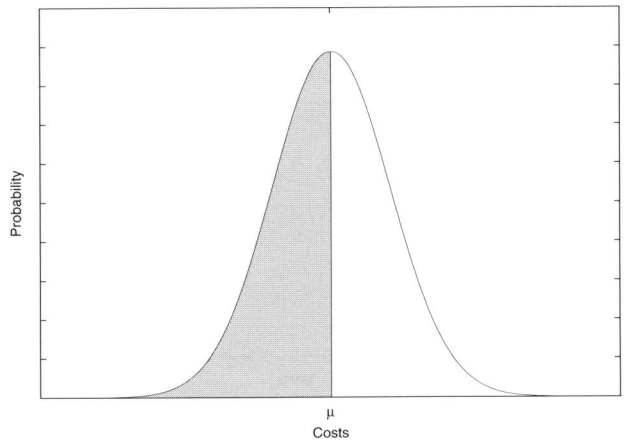


Figure 5.11. Difficulty by quantiles

the cost computation. For some problems a 15% quantile can be a rather slack target, e.g. Traveling Salesman Problem, for others like query optimization it can be a narrow objective.

To give an impression how the three major problem categories compare, we show the difficulty D in Figure 5.12 as a function of the size of the target quantile.

For Type-A, the plot reads as follows: e.g. the quantile of 0.1 next to the optimum spans the interval $[c_{min}; 0.1 \cdot (\mu - c_{min})]$, the quantile 0.5 spans $[c_{min}; 0.7 \cdot (\mu - c_{min})]$. The curve is clipped for values larger than 1 as larger quantiles are no of interest for an optimization anyway. The other curves read analogously. In contrast to Type-A, however, the others soar up for even small intervals meaning that a large portion of the interval $[c_{min}; \mu]$ is needed to cover the target quantile, i.e., there is no cost concentration around the optimum.

The curves reflect our earlier experiences. The partitioning problem is "easy" even for large quantiles. The Traveling Salesman Problem instance in contrast appears difficult even for small quantiles.

5.4 Summary

Query optimization is known to be NP-hard and so are restricted subproblems like JOPT and XOPT. However, the intractability attested is "only" the worst case complexity.

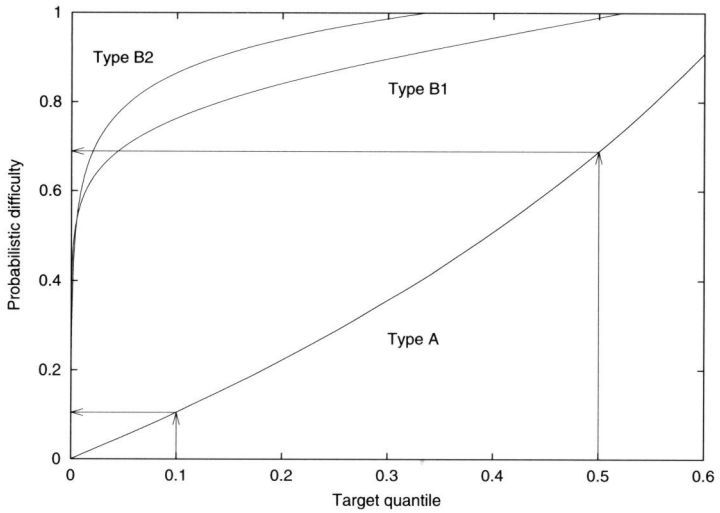


Figure 5.12. Difficulty by quantiles for the basic distribution types

In decision problems like k-Satisfiability phase transitions have been discovered where areas of easy instances are separated by a small area of hard instances. The link between optimization and decision problems might now suggest that similar phenomena occur in optimization problems and thus in query optimization too.

Using the Asymmetric Traveling Salesman Problem as an experimentation platform, we investigated possibilities of phase transitions. Our findings show that an optimization with exact techniques encounters difficulty that has its explanation in the associated cost distribution rather than in a phase transition. The difficulty increases gradually until it collapses into the pathological trivial case where all edges, and so all tours too, are of the same length. A separation as seen in k-Satisfiability cannot be identified.

For approximation algorithms the situation appears even reversed: what was increasingly difficult for branch and bound techniques is growing easier when using for example random sampling. Other heuristics may display further, different regions of difficulty. Thus difficulty in optimization has be viewed with respect to the algorithm used rather than by the problem itself. While a phase transition is not apparent, cost distribution can very well interpret the effect observed. Providing also a tool to assess the chances of successful deployment of branch and bound techniques in other fields. Concerning query optimization our findings thwart the hopes for a successful deployment: On the one hand, the cost distribution is in general

not favorable to these algorithms, on the other hand, the lack of a non-trivial relaxation technique allows only weak bounds. In Section 5.1.3 we have seen the importance of sophisticated relaxations. In preliminary tests in SQL Server with cost bound pruning techniques we observed qualitatively only little gains—often outweighed by the overhead added.

In order to use cost distributions for further analysis of the query optimization problem we re-formulated the optimization goal by defining a measure of quality that overcomes the drawback of invariance under translation. Finally, we developed an alternative measure of difficulty solely based on a problem's cost distribution.

In the following, we will use the concepts we devised here to analyze randomized algorithms as well as evolutionary techniques.

6

Evolutionary Optimization

Evolutionary optimization techniques like Genetic Algorithms or Genetic Programming have been applied to a wide range of combinatorial optimization problems—yet, their popularity remains mainly limited to the Artificial Intelligence and Machine Learning communities.

Evolutionary Algorithms mimic the natural evolutionary process. First, a problem representation is encoded as genes and chromosomes. Starting with an initial, randomly generated gene pool, recombination and selection operators are applied to pairs or sets of chromosomes and individuals. In this field of research, the objective function is called *fitness*. We will stick to the term costs to emphasize the absolute value, as fitness can also be defined as *relative fitness*, seen with respect to other individuals in the population. Over the time the quality of the population converges leading to an increase of high-quality individuals, i.e. solutions.

Though an appealing concept—hardly any information about the problem seems necessary yet evolution guarantees optimal or near-optimal results—it lately came under fire from critics even within the Evolutionary Computing community (see e.g. [Lan95]). Their major concern is that other apparently less sophisticated optimization algorithm match and often exceed the performance of evolutionary algorithms as they find results of similar or better quality in shorter running time.

However more profound criticism has been postulated aiming at the self adaptability of evolutionary techniques. In a truly ground breaking work Wolpert and Macready introduced and proved a set of so-called No-Free-Lunch Theorems, which state that no blind search optimization algorithm can be expected to outperform any other algorithm, i.e. to every optimization algorithm we can find a problem where this algorithm gets outperformed by some other algorithm [MW96, WM97]. These results distinctly thwarted the high hopes that evolutionary computing could serve as a kind of general problem solver due to its abilities of self-adaption. Following the initial paper of Wolpert and Macready further criticism has been formulated doubting the original promises of evolutionary computing (see e.g. [Cul98]).

However, the No-Free-Lunch theorems should not only be viewed as a setback but also from a more pragmatic perspective: If there are classes of problems that pose particular difficulties for evolutionary algorithms and other where evolutionary computing is a suitable optimization strategy it is an exciting challenge to identify the criteria that determine the class a problem belongs to. So far, knowledge as to what problem is hard and which is easy—from the evolutionary point of view—is only available in terms of experience.

In this section, we devise a classification based on cost distributions leading to a recommendation when to use evolutionary techniques and when not.

An analysis of evolutionary algorithm is specifically difficult, because the notion of evolutionary computing is fairly flexible, comprising a large variety of algorithms and techniques. Frameworks as presented in [Gol89, Mit96, Eib96] are capable of simulating algorithms commonly not considered evolutionary, like Random Sampling or Simulated Annealing [Bäc96]. In our analysis, we first sketch the different generic components of evolutionary algorithm, then scrutinize the impact of cost distributions on those components. In particular, we investigate to what degree the single components use randomly selected solutions. Random sampling, uniform or biased, on—possibly restricted-sets of solutions is the very nucleus of all randomized optimization algorithms including evolutionary techniques.

Our analysis is based on the classification presented in Chapter 4. This classification reflects three basic categories of difficulty from the evolutionary computing point of view: problems that are too *easy* in the sense that less sophisticated optimization algorithms achieve results of similar quality in shorter time; problems that are too *difficult* so that evolutionary algorithm get outperformed on average; and finally problems where evolutionary techniques appear most suitable [1].

An intuitive notion of these three categories of performance is not only known since Wolpert's and Macready's No-Free-Lunch Theorems, though their theorems laid a first formal foundation, but has been observed in earlier studies (see e.g. [Mit96]).

6.1 Principles of Evolutionary Algorithms

Figure 6.1 shows an outline of a evolutionary algorithm in pseudo code (cf. e.g. [Eib96]). Starting with a randomly generated initial population, generations are repeatedly derived by selecting a set of parents, generating the offspring by *recombination*, introducing a certain random distortion in form of *mutation*, and subjecting all individuals to a *selection* process. The algorithm terminates as soon as a certain stopping criterion—e.g. timeout, maximum number of individuals reached, or no improvement over a certain number of generations—is fulfilled. In every generation, all individuals

are checked for their fitness, i.e., their costs, not only for the selection of the next generation but also to keep track of the best individual found so far. Simulating the natural evolutionary process the algorithm achieves a gradual improvement concentrating on well suited individuals by selection and the production of closely related offspring.

Let us now scrutinize the single components, using the classification given in Chapter 4, and how they are influenced by the different kinds of cost distributions. We will corroborate the results of our analysis with different optimization problems in the next section.

Initialization.

The influence of the cost distribution on the initial phase is significant as *initializing* means literally *sampling*.

For a type-A distribution the probability to find already near-optimal solutions in the initial sample is high. The subsequent optimization phase cannot improve the initially found solutions substantially. The chances that high quality solutions are included in the initial solution depend further on the size of the population: *very* small populations may differ enormously in quality.

In case of a type-B distribution, the initialization's role is less important, depending on the distance of the cost of the optimal solution from the average cost. The sampled initial individuals are of comparable but distinctly sub-optimal quality. As opposed to the previous case, the size of the population does not affect its quality—the probability to sample a near-optimal solution is virtually zero.

Recombination.

During this phase a *recombination* or *crossover* operator is applied to sets of individuals. It implements the actual evolutionary mechanism that mates two (or more [ERR94]) individuals and derives a new one. Strictly speaking, the result is a *random* solution consisting of parts of its ancestor.

In the case of the type-A distribution, sophistication is usually of limited use as there are plenty of solutions in the close vicinity. However, if there are too many close relatives, it may become also more difficult to guide the recombination process.

The less solutions with similar costs to their ancestors there are, the more astray—i.e., in direction of the average cost—the recombination may lead. Sophisticated algorithms are necessary to avoid a fall back to the bulk of solutions in case of a type-B2 distribution.

Mutation.

The role of mutation is disputed. It is an obvious element of natural evolution. However, it is not clear whether it is vital for evolutionary opti-

```
proc EvolutionaryAlgorithm
    t = 0
    P_t = ∅
    Init(P_t)
    repeat
        P'_t = SelectParents(P_t)
        Recombination(P'_t)
        Mutation(P'_t)
        P_{t+1} = Selection(P_t ∪ P'_t)
        t = t + 1
    until(done)
end
```

Figure 6.1. Outline of evolutionary algorithm

mization techniques. For instance, Kosza suggested a mutation rate of zero [Koz91].

In case of a type-A distribution, mutation can be most fruitful as the odds to improve by random alteration are high.

For a type-B distribution, the probability to achieve an improvement by mutation is rather small and mutations is only useful to avoid undue concentration of certain properties among the individuals.

Restarts.

Evolutionary algorithms, mimicking the natural evolutionary process are characterized by convergence, i.e. the overall fitness of the consecutive generations increases—although not necessarily monotonic. For simplified models of those algorithms, the convergence of the optimum as a limit, provided an infinite running time, has been proven (see e.g. [Rud92], [Bäc96]). Similar facts are known for algorithms like Simulated Annealing. However, depending on the cost distribution, evolutionary algorithm can very well profit from restarting, simply because of the cost distributions influence on the initialization (see above).

In case of a type-A distribution, the impact of re-runs may greatly improve the results, whereas in a type-B2 scenario, re-starts do not make much of a difference.

In Table 6.1, the basic tendencies of influence we identified are summarized. The three types of cost distributions directly suggest three classes of difficulty—from an evolutionary algorithm point of view:

	Type-A	Type-B1	Type-B2
Initialization	$\oplus\oplus$	$\odot$	$\odot$
Recombination	$\ominus$	$\oplus$	$\ominus$
Mutation	$\oplus\oplus$	$\oplus$	$\ominus$
Restarts	$\oplus\oplus$	$\odot$	$\odot$

$(\oplus)\oplus$ = *(strong) positive influence,* $\odot$ = *no influence,* $\ominus$ = *negative influence*

Table 6.1. Influence of cost distribution on individual components of evolutionary algorithm

Type-A is the easiest, where all components but recombination are positively influenced by the distribution. However, problems of this class turn usually out to be *too* simple, rendering evolutionary algorithms an overkill. Especially hill climbing algorithms achieve a much better performance, i.e., results of comparable quality but in significantly shorter running time.

Type-B1 is slightly positive influenced; this is the kind of problem evolutionary algorithm are highly suitable to optimize.

Type-B2 is the most difficult as the cost distribution has mainly negative influence. Problems of this class appear to be difficult for evolutionary algorithms to optimize.

As we pointed out before, a large number of parameters determines success and failure of evolutionary search algorithms, and negative influences of the cost distribution may be leveled—to a certain degree—by more sophisticated design of recombination operators etc. However, cost distributions indicate where advantages as well as problems are to be expected.

We also stressed that the basic framework does not restrict the choice of arbitrary crossover operators, or mutation rates. Even simulation of other optimization algorithms are possible, however, the further we swerve from the simplest of evolutionary algorithm the more likely it is that the overhead induced by the evolutionary framework becomes more and more a hindrance rather than a performance improvement and other, non-evolutionary algorithm may perform better, i.e., they find results of similar quality within shorter running time.

6.2 Examples

To verify our previous analysis we resort to the examples of NP-hard optimization problems we discussed in Chapter 4. We have to be aware, that the given classification is not defined in an exact way. Problems may have a cost distribution that cannot be distinctly assigned to one of the categories but appear to be in between two categories. However, in our experience,

it is not useful to devise a finer system for the classification—e.g. based on the statistically characteristic values like mean, deviation, etc.—without taking further problem specific properties as well as implementation details of the search algorithm into account. On the other hand, the classification with only three categories proved surprisingly well-suited and to be of sufficient generality. Problems with distributions in between can be assessed by interpolation.

6.2.1 Type-A

A typical representative of this class is the number partitioning problem, where a set S of numbers is to be partitioned into 2 subsets S_1 and S_2 such that $S_1 \cup S_2 = S$, and the difference of the total sums

$$|\sum_{s \in S_1} s - \sum_{s \in S_2} s|$$

is minimal.

Optimal and near-optimal costs appear with the highest possible frequency (cf. Fig. 4.2). The distribution clearly is of the type-A kind. Even pure random sampling algorithms are very likely to find good solutions, hill climber and other multi-start algorithms that do not deploy highly sophisticated techniques, achieve excellent results within extremely short running time [KKLO86].

Evolutionary algorithms find very well results of similar quality but require longer running times. With this kind of distribution, the size of the initial population is critical to the stability of the optimization, i.e., a population of size 100 almost certainly contains an optimal or near-optimal solution; the quality of small populations may differ significantly, so that using a tight time limit and re-starting the algorithm a couple of times may improve the results significantly in case of small populations.

6.2.2 Type-B1

The class of type-B1 distributions comprises a large variety of scheduling, timetabling and assignment problems. We chose the *Knapsack Problem* as one of the representatives as its structure is easily accessible and can be well illustrated due to its 2-dimensional nature.

The problems definition is as follows: Given a number of items—each has a profit and a weight associated with it—, a (sub-)set of items is sought such that the total weight does not exceed a given bound but the sum of profits is maximal (see e.g. [GJ79, MT87]). The variant we defined is referred to as single-objective, as there is only one optimization goal, the maximizing of the profit. Branch and bound algorithms as well as dynamic programming are usually the algorithms of choice [MT87, Pis95] (cf. Chap. 4).

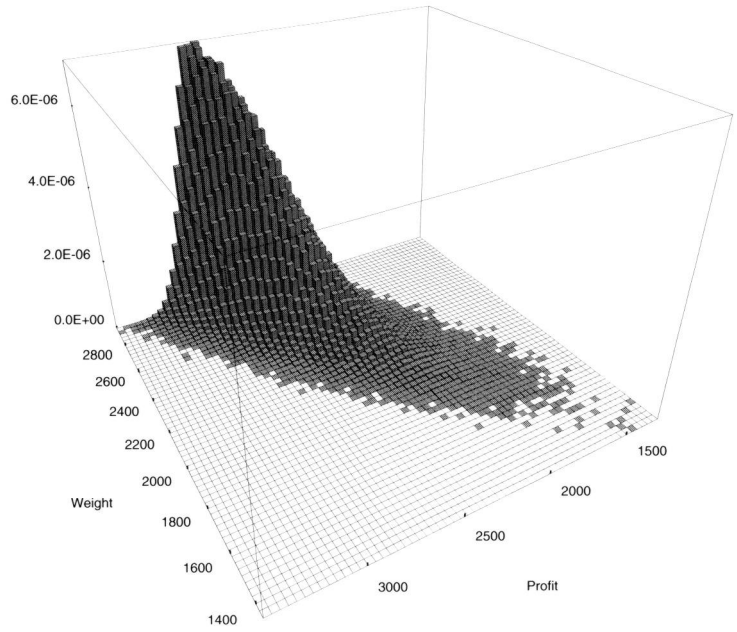


Figure 6.2. Cost distribution of multi-objective 0/1 Knapsack Problem, obtained from a sample of 10^6. Region of optimal solutions in the foreground

Let us now again consider the multi-objective variant of the problem, where not only the profit is to be maximized but at the same time, the weight is to be minimized. As opposed to the single-objective version, there is not only one single optimum but rather one optimum for each different total weight. Genetic algorithms are known to perform very well in the multi-objective case.

In Figure 6.2, the cost distribution of an instance consisting of 100 items is shown. The distribution was obtained by a sample of 20000 packings generated with uniform probability. The values of both weight and profit of the single items were chosen as random numbers between 10 and 100. The capacity of the knapsack was chosen as half the total weight of all items. Such assumptions are common in the literature [ZT99]—in particular, this configuration follows the example of [MT90].

The resulting distribution is characterized by a marginal distribution along each single weight configuration that strongly resembles normal distributions. This does not come as a surprise as both weight and profit of a

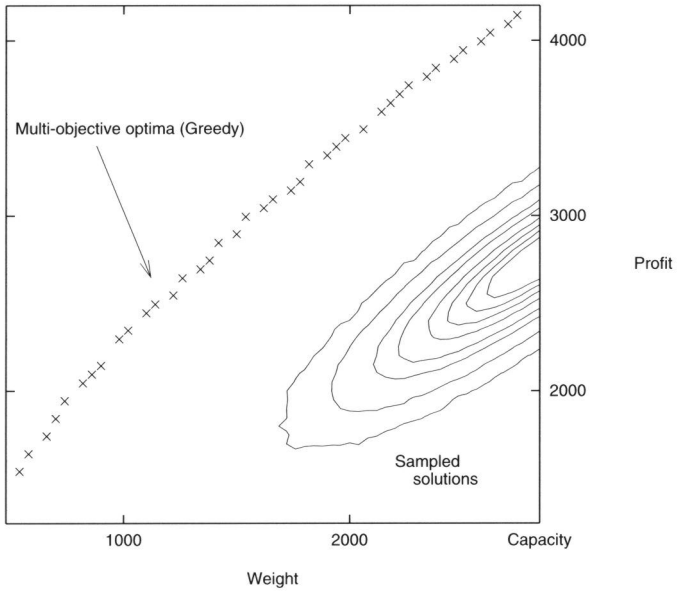


Figure 6.3. Cost distribution of multi-objective 0/1 Knapsack Problem (cf. Fig. 6.2), including best multi-objective solutions found with greedy algorithm

particular packing of a knapsack are sums of random numbers. The shapes of the distribution appear very stable and insensitive to the problem's parameters. We conducted extensive experiments with large varieties of parameter combinations, resulting in distributions with very similar shape, the extent may differ though.

In order to give an indication of whether the distribution is of type B1 or B2, it is necessary to determine the distance of the bulk of solutions from the optima. To assess this distance, we used a greedy algorithm which provides good approximations of the optima.

In Figure 6.3, the distribution is shown as a contour plot. The isolines connect solutions with equal frequency. The theoretic region of optima is in direction of the left upper half, stretching from the left lower corner to the right upper corner. The optimization results found by the greedy algorithm form a lower bound of this region (see Fig. 6.3). The actual optima are in the close vicinity of those solutions.

The effect of this distribution on genetic search is twofold (see above): The sampling of an initial population does not contain high quality so-

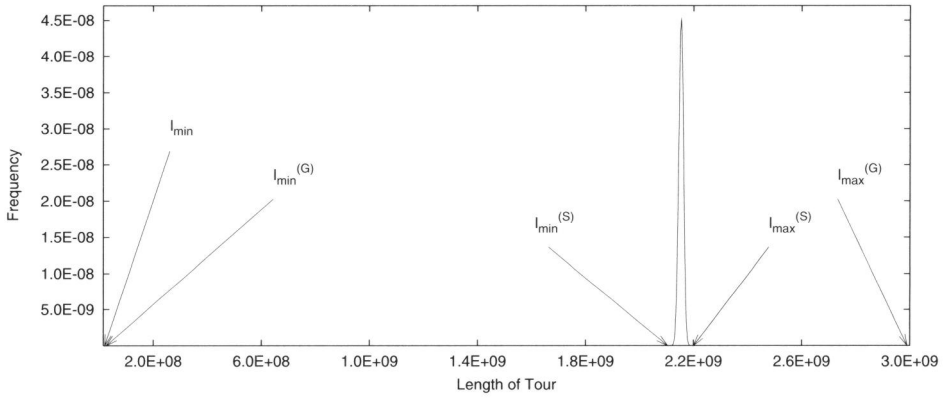


Figure 6.4. Map for Traveling Salesman Problem. Labels as in Figure 4.7

lutions. Also, the random sampling component in form of cross over and mutation is limited—the probability to sample a near optimal solution practically zero. On the other hand, the optima are not too far away from the bulk of solutions. Specifically, for the multi-objective variant, genetic algorithms are known as a suitable and very successful optimization technique.

6.2.3 Type-B2

Given the previous analysis, we should expect a cost distribution of a difficult problem (a) to show a strong concentration of the bulk of solutions and (b) the optimum to be far off the bulk. In statistical terms, we expect a distribution with a particularly heavy tail.

We study the cost distributions of the symmetric Traveling Salesman Problem where instances are given only by the coordinates of the cities. Recall the example used in Chapter 4 that consists of all 13509 cities in the United States with more than 500 inhabitants. The optimal tour is known to be a tour of length 19982859 [ABCC98]. Figure 6.4 depicts the problem's cost distribution obtained from 10^6 uniformly sampled tours.

The cost distribution shows the expected features: Almost all solutions are concentrated—even in the upper half of the total cost range. Moreover, they are concentrated in a very small interval. All sampled tours are longer than 2.1×10^9 and shorter than 2.2×10^9. Consequently, neither when randomly selecting tours for a initial population nor when adding randomly chosen tours during the optimization a tour shorter than 2.1×10^9 is likely to be chosen. The best sampled tour is more than 11 times the length of

the one found by a simple greedy algorithm.

The TSP is know to be a difficult problem for evolutionary algorithm. At the same time, it constitutes a perpetual challenge for geneticists who developed various techniques to make the problem more accessible to evolutionary computing. Nagata and Kobayashi for example report on optimal solutions for problem sizes up to 3038 and modes computation times by using a technique called *edge assembly crossover* [NK97b]. Watson *et al.*, however, could not completely reproduce these results when conducting a comparative case study with other—specifically for the Traveling Salesman Problem developed—extensions of genetic algorithms but achieve results of high quality comparable to standard heuristics [WRE+98]. As Watson *et al.*. attest, the edge assembly crossover cannot possibly be called blind search operator as it comprises very problem specific components including a greedy heuristic applied locally to connect subtours. For a survey and quantitative assessment of other operators and extensions that have been deployed in order to tackle the Traveling Salesman Problem, we refer the reader to [MW92, MdWS91].

6.3 Evolutionary Algorithms in Query Optimization

In the following, we discuss three approaches where query optimization has been tackled with evolutionary algorithm. Bennet *et al.* take the credit for establishing the first transfer of genetic algorithms to query optimization. They propose a framework which uses two different crossover strategies one of which is restricted to small, local changes whereas the other may cause far reaching alterations of the plan [BFI91]. We refer to this approach as BFI91. The setting used by Steinbrunn *et al.* in [SMK93, SMK97] is similar. However, their study focuses on a comparison of a large number of various heuristics and navigational randomized algorithms rather than on tuning these techniques. We refer to their approach as SMK97. Stillger and Spiliopoulou as well as Nafjan and Kerridge propose extension for query optimization for parallel databases. In both cases, all features relevant for parallel processing are incorporated into the fitness evaluation so that crossover operators from previous work can be applied [SS96, NK97a]. Also do the results found for query optimization in parallel database systems add only little new aspects. We include Stillger and Spiliopoulou in our comparison since they devise a powerful and sophisticated crossover operator that tries to take the tree structure of the plans into account and exchanges contiguous parts of the trees rather than unconnected components. This approach is an advancement[1] of SMK97 and a considerable part of the work deals with the tuning of the strategy. In the sequel, this approach is referred to as SS96. We shall scrutinize all three approaches with respect to what extend they use randomly chosen plans and to what extend

[1]Though [SMK97] was published only 1997, major parts of the work were already available as preprint as early as 1993.

	BFI91	SMK97	SS96
Query size	5-16	5-30	10-100^{+}
Population size	64	128	100-1600°
Cross over	—*	65%	95%
Mutation	—*	5%	5%
Termination criterion	—*	30/50 generations without improvement$^{\bullet}$	2000-25000 plans in total
Minimal # of plans	—*	3840	2000
Maximal # of plans	—*	—$^{\diamond}$	25000

*no data available, $^{+}$performance comparison only for range 30-70, $^{\circ}$multiple of the query size, $^{\bullet}$restricted/unrestricted search space, $^{\diamond}$only bound by size of search space

Table 6.2. Parameters used in previous work

they rely on evolutionary principles. In the wider context of database technology, evolutionary algorithm have gained more popularity lately. However, these fields of application are unrelated to the probe of query optimization, see e.g. [NP98, IF98].

In Table 6.2, the parameter settings for all three approaches are shown. It leaps out that BFI91 uses only 64 elements per generation whereas both competitors use more than 100 individuals. As a result their experiments show strong instability indicating that the technique depends heavily on the quality of the initial population. They overcome this deficiency by re-running the optimization 5 times and taking the best of 5 runs as result. This re-run provides a total initial sample of 320 plans which explains why the best-of-five achieves way better results than a single run. No data is available about the number of plans explored in total.

The population size of 128 used in SMK97 proofs to be large enough to cover good plans on average. In fact, authors observed

> *[e]ven the initial population consisted of at least one member with an evaluation cost that is as low as [that of Simulated Annealing] after running more than 100 times as long.* [SMK97], page 207.

It is pointed out that the genetic algorithm converges very slowly—often two orders of magnitude slower than Iterative Improvement by achieving results of only similar quality. Compared to BFI91, the results are of higher stability, but still do not match the stability of navigational algorithms [SMK97]. Furthermore a modest crossover rate of 65% is used in order to preserve a large portion of the gene pool, the mutation ratio of 5% is fairly high, however. SMK97 makes the most use of the evolutionary principles which is reflected in the running time.

In SS96 the sizes of the population are varied which causes some distortion as the sampling of the initial population is independent of the search space's size. They report on finding the best plan already among the very first generations. In most experiments they use less than twenty generations which is in stark contrast to SMK97 as the actual evolutionary process is largely bypassed. Moreover, the high crossover helps cover as large a portion of the search space as possible.

Finally, both SMK97 and SS97 explore a much larger number of plans as is needed to achieve the same quality by random sampling. For BFI91, no data about the minimal or maximal number of plans is available, however the fact that their algorithm takes longer than a dynamic programming algorithm which determines the absolute optimum for the linear subproblem suggests that they also explore a very large number of plans.

Summing up, all three approaches depend more on the ability of exploring a large, randomly chosen portion of the search space than on the actual evolutionary process. Be it that they re-run the actual optimization with different initial populations, increase the initial population size or use very high crossover and mutation rates. The particular crossover implementation appears to be secondary.

6.4 Summary

The original ideas and intuition driving the design of evolutionary algorithms aimed at a framework that is general enough, on one hand, to adapt itself to arbitrary problems, and—according to the principle of survival of the fittest—on the other hand converges to high quality results.

Strictly speaking evolutionary algorithm implement a limited random sampling on the "surroundings" of the individuals within a population. However, the notion of "surrounding" is only loosely defined as the recombination of two individuals leads to a large variety of possible individuals for the offspring multiplied by a vast number of different combinations to mate parent individuals. The offspring may be rather defined as somehow *similar* to its parents. This constellation gave rise to speculative theories about landscapes. However, the view that a landscape is intrinsic to a problem has to be revised with respect to the arguments we brought forward in Chapter 4. Kauffman, a pioneer in this field devised a popular model, called *NK model*, to describe different phenomena in terms of ruggedness of landscapes [Kau93]. To date, the link between these models and reality is still to be establish in order to help these models become more than merely an intuitive illustration. In case a tight and direct link could be achieved, landscapes may proof a helpful tool in the design of evolutionary optimization techniques.

In this chapter, we presented a classification for optimization problems based on cost distributions. Our classification identifies three classes of

problems according to the difficulty they pose to evolutionary algorithm. The first class embodies the kind of problems that is generally too easy rendering evolutionary algorithm an overkill. The second contains problems evolutionary techniques are most suitable to tackle whereas the third represents the hard cases.

Query optimization appears to belong to the class of problems that are better optimized with less sophisticated algorithms. Recall also the discussion of the cost models resolution which we addressed in the introductory chapter. The lower the resolution—i.e. the larger the threshold—the easier the problem.

The review of related work lends strong support to our claims. In all three probes the tuning and improving of the algorithms directly led to an emphasis of the random sampling component, be it by means of larger population sizes or re-runs of single experiments. The actual evolutionary process, however, got more and more bypassed.

Given our analysis one might be tempted to improve the evolutionary algorithm by modifying or even excluding certain phases. Clearly, the result of any such modification could still be viewed as genetic algorithm or program and it is difficult if not impossible to draw the line. As we pointed out at the beginning of the analysis of the framework and its components, evolutionary algorithm are basically capable of simulating randomized algorithm that are commonly not referred to as evolutionary. However, all these improvements simplify and alter the algorithm in a way that they sacrifice the self-adaptability, the major advantage of evolutionary techniques.

Seen from a more pragmatic perspective, the more crucial the modification of the framework, the more likely it becomes that a direct implementation of a simpler schema outperforms the simulation by genetic algorithms in terms of both running time and complexity of implementation.

7

Probabilistic Query Optimization

Probabilistic or *randomized* optimization algorithms are a field of research which attracted enormous attention during the last decade in all fields of combinatorial optimization. Especially with complex optimization tasks that involve a large number of parameters and where heuristics achieved only mediocre results, these algorithms helped tackling problem sizes that defy being solved to optimality due to combinatorial explosion. Major application areas include scheduling problems and graph partitioning as for instance in standard-cell placement in VLSI layout [VK83, LD88, SR91].

These algorithms are distinguished by their black-box approach, that is, no knowledge of the problem's particular structure and properties, but only a set of *manipulators* is required. Manipulators alter a given solution and transform it into a different, but semantically equivalent—i.e., correct—new solution. Manipulators are often also referred to as *rules* or *transformations*; we will use these notions synonymously in the following. The changes are usually of local nature, that is, the new solution is "similar" to the original one. Manipulators imply a topology on the search space as they define neighborhoods. Given a single solution, the transitive application of manipulators defines a subspace of the original search space. An important prerequisite is the irreducibility of this subspace, i.e., the transformations should be defined in such a way that any two solutions can be transformed into each other by a sequence of transformations.

Based on this setup various optimization strategies have been proposed: starting with and arbitrary solution, an optimization algorithm applies the manipulators according to a strategy to generate a set of alternative solutions. One of the alternative solutions is accepted—again according to a strategy—and the neighborhood of this solution is explored in the same way implementing an transitive exploration of neighborhoods. The algorithm records the best solution found during the process. In Section 7.3 we develop a general framework and present the most important representatives of this class of algorithms.

Given the success in other fields of combinatorial optimization and the simple way to adapt these techniques they have been first applied to query optimization by Ioannidis and Wong [IW87] and specifically to join ordering by Swami and Gupta [SG88]. Following these initial studies, Ioannidis and Kang [IK90, IK91] established an in-depth analysis of some of the algorithms for JOPT and presented first theoretical results concerning the basic properties of the search space. In particular they investigated differences between the general search space containing processing trees of arbitrary shape and the restricted one constituted by linear trees only.

Despite the efforts to provide a more comprehensive understanding a commonly accepted result could not be reached owing chiefly to the fact that some of the results were contradicting, i.e., different parties proved different search strategies to be superior [SG88, IK90, IK91, Pel97, SMK97]. The differences have been attributed mainly to the different search spaces, different cost models, and different parameter settings used. Oddly enough, the transformation rules which are responsible for the topology have not been subject to discussion, but were treated as a given rather than a variable. The only concerns with the design of transformations were the irriducibility of the subspace and that all results of a transformation are again valid solutions.

In this chapter, we pursue an analysis based on the techniques and insights developed in the previous parts. First we discuss the question of linear and bushy search spaces from the cost distribution point of view on the lines of Chapter 4. In order to investigate the potential of randomized algorithms we have to liberate the algorithms from the strong interplay of different transformation rules. To that end we devise an *abstract search space model* consisting only of the two invariable components: solutions and their costs. The solutions become annotated nodes in a graph where edges—representing neighborhood relations—are no longer a result imposed by transformation rules, but can be freely chosen to test for various topologies. More general, the abstract search space allows us to experiment with arbitrary parameter combinations that would be hard to achieve in an actual query optimization setting. This way, we can fathom the actual potential of an optimization technique. For the analysis of different probabilistic optimization techniques we break the algorithms into the single algorithmic principles, as seen with genetic algorithms, and analyze these building blocks and subsequently discuss the complete, compound algorithm.

7.1 Linear vs. Bushy

In their first approach to randomized join ordering, Swami and Gupta confined their algorithms to linear processing trees only. The space of linear trees is in so far prominent as it has been used in previous work with exhaustive search techniques as the linear space is considerably smaller than

its bushy counterpart. Commercial database systems like Microsoft SQL Server use linear search spaces enriched with nested queries, i.e., each operator can have a linear tree and a base table, or a linear tree and a subquery—which in turn is a linear tree—as inputs, multiple nesting allowed for.

The question we are interested in here is whether the obvious differences in shapes of trees have immediate consequences on the optimization.

As we have pointed out in Section 2.2, the possible shapes of a processing tree are limited by the join graph, i.e., the query itself. There always is a valid linear processing tree, no matter the shape of the join graph. For star shaped join graphs, linear trees are even the only possible processing trees. The other extreme is the clique graph where any shape of tree is possible. Unfortunately there exists no concise meaningful measure to classify the wealth of join graphs in between star and cliques. Notions like chain, grid, or more general, acyclic graphs etc. are too broad and without further additional parameter do not define a specific subset of shapes.

However, we can apply a similar analysis as in Chapter 4 and differentiate linear and bushy trees in XOPT. Let us define the sub-problem L-XOPT where only linear trees are considered a valid solution. Note, that L-XOPT is a real subset of XOPT. We denote the cost distributions as ϕ and ϕ_L for XOPT and L-XOPT respectively. In Figure 7.1 the cost distribution of L-XOPT is contrasted with the one of XOPT for low and high variance of the relation sizes. The distribution of L-XOPT is highlighted. The plots of Figure 7.1a depict the situation for relation sizes of low variance, 7.1b for a high variance scenario. The left plot shows the total number of solutions as function of the costs, the right one displays the resulting cost distributions, i.e., relative instead of absolute frequencies.

In case of low variances, the distributions differ significantly. Linear trees span a part of the search space with substantially more costly minimum, whereas the maxima of both spaces are comparable. In the event of higher variances, the characteristic of L-XOPT's space changes and reaches eventually those of the general space (see Fig. 7.1 a) and b) right plots).

To capture this shift of the distributions more formally, we use a characterization introduced by Ioannidis and Kang [IK91]: Given two densities ψ_1 and ψ_2, the ratio

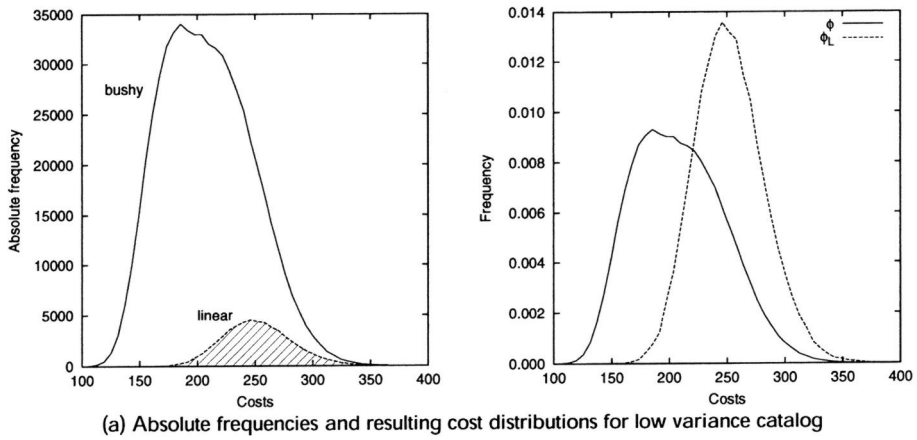
$$s(x) = \frac{\int \psi_1(x)}{\int \psi_2(x)}$$

is called *relative shift*. If $s \geq 1$, we say ψ_1 *is shifted to the left relative to* ψ_2.

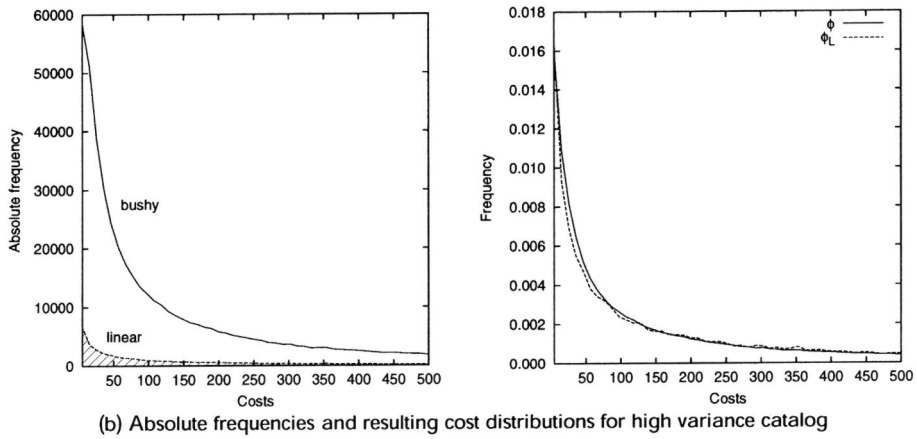
The major observations with our experiment are:

1. the mean of ϕ_L is higher than that of ϕ, and ϕ is relatively shifted to the left;

2. with larger variance, the number of high-quality solutions in ϕ_L increases, i.e., for large variances, the best solution in both distributions are of comparable quality;



(a) Absolute frequencies and resulting cost distributions for low variance catalog



(b) Absolute frequencies and resulting cost distributions for high variance catalog

Figure 7.1. Cost distributions for linear and bushy search space. Left, frequency as absolute figure, right corresponding distribution

The immediate consequence of fact (1) is a different probabilistic difficulty as Figure 7.2 shows. For the low variance case the bushy space is significantly easier, the linear more difficult—for higher variance the linear space becomes more and more less difficult matching the bushy eventually. The analogous observation for jopt has been reported by Bennet *et al.* in [BFI91] who found better results in bushy spaces using a genetic algorithm

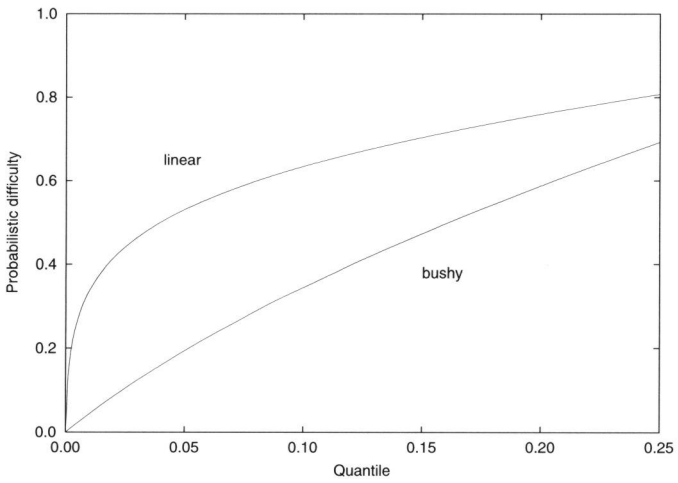


Figure 7.2. Probabilistic difficulty for low variance catalog

than exact methods could find in the linear space for the same query.

Together with fact (2) that both spaces may contain comparable optima the space of bushy trees appears preferable.

Ioannidis *et al.* reached to the same conclusion, however, on totally different grounds [IK91]: They argue that bushy spaces offer a better topology for transformation based search algorithms which puts their entire analysis in direct dependency to the implementation of manipulators. Hence, their analysis would be of very limited generality only. Moreover their investigation applies *only* to transformation based optimization algorithms, but not to multi-start algorithms like Transformation-free [GLPK94]. As opposed to that, our analysis is not restricted to any optimization technique but reflects solely the search space and its character.

In the discussion of optimization algorithms later this chapter, we will come back to this point when analyzing the direct effects of the choice of a search space on the algorithms.

7.2 Abstract Search Space Model

Repeatedly, we pointed out that transformation-based optimization is often abstracted by a guided graph traversal. In order to discuss the effects reported in previous work and to contrast them with our own findings we use a graph model that allows us to study the single parameters and their

n	size of the graph
d	degree, i.e., number of outgoing edges, equal for all nodes
α	shape parameter for gamma distributions used for cost values
σ	deviation of cost between neighbors

Table 7.1. Parameters of abstract search space model

impact. This is of particular interest as authors of previous work discussed effects observed often with respect to this abstraction, i.e., they explained behavior of optimization algorithms with properties the underlying graph was assumed to have. To check for these properties we generate fully synthetic graphs called *abstract search spaces* described by the parameters given in table 7.1. The graphs are distinguished by a number of features. All edges are directed. The degree describes only the number of outgoing edges; the number of incoming edges may be arbitrary. A directed graph matches reality better than an undirected as transformations used in query optimization are usually not symmetric.

In Figure 7.3, the algorithm to construct the graph is outlined. First, the set of nodes is generated and each node is assigned costs according to a gamma distribution as discussed before. To facilitate and steer the generation of edges we sort the nodes according to their costs. This enables us to perform efficient binary search for nodes of given costs. In the next step we construct neighborhoods by first generating a random number from a normal distribution with deviation σ and the costs of the current node as mean, and search for a partner node with the resulting costs. At the fringes of the cost range, we truncate the normal distribution so that only valid cost values are generated. We denote the resulting graph by

$$G_{\sigma}^{(\alpha)}(n, d).$$

We omit indexes if they are apparent from the context. To match the practical case, the degree should be chosen greater than 10 (see also next section). Though there is a positive probability to construct a disconnected graph in principle, with the parameter settings we use for our experiments this case can be ruled out almost certainly. Moreover we modified the binary search for a partner in such a way that not an exact match of the cost value is required but rather the nearest neighbor to that cost value is chosen. Furthermore, we exclude loops.

7.2.1 Experimental Setup

To simulate real world scenarios the algorithm must be able to handle very large graphs. Hence, a parallelization of the graph generation is necessary to maintain practicable running times. All steps of the algorithm offer simple yet enormously effective means for parallelization on a CC-NUMA

architecture. For both distributing costs and constructing the edges the graph can be divided in k independent parts, k being the number of available CPUs. The operations can be carried out on those parts in isolation. In the actual implementation the granularity is chosen as 1000 nodes per sub-graph instead of n/k, ($n > k \cdot 1000$) to achieve better load balancing as the target architecture consisted of 32 CPUs of 2 different types, 8 MIPS R12K300MHz and 24 MIPS R10K250MHz. With k parts, some of the threads would become idle earlier than others due to different CPU speeds. Having smaller parts, however, reduces this effect substantially.

The critical element is the sorting, which is done in a hierarchical decomposition where first each thread sorts the range of nodes it is assigned to with e.g. quick-sort. In the next phase, threads with even id merge the segment of their predecessor with their own. Threads with odd id are terminated. After renumbering the threads, this step is repeated until no thread is active anymore. By then the sorting is complete. Since sorting is only of little running time compared to the other steps mainly because of the substantial share of floating point arithmetic of the random number generators we achieve almost ideal speedup for sufficiently large graphs making graphs the size of several millions of nodes a feasible target.

7.2.2 Topologies

The topology of an abstract search space of given size is essentially determined by the degree and the distribution of costs among neighbors, i.e., the "length" of the edges. While the impact of the degree is evident, the impact of the distribution of costs among neighbors is somewhat more subtle. As a simple example consider a deviation of $5 \cdot 10^{-5}$ with arbitrary but sufficiently large degree, say 50. The resulting graph is the linear graph. Recall, that we do not allow loops but do allow multiple edges. The very small deviation is only used to suggest a direction, i.e., since we forbid loops we are forced to take the next possible node in the given direction. The high degree in this example ensures probabilistically that every node has some neighbors of higher and some of lower costs. Moreover, as seen in this example, σ is only used to search for a neighbor, but the resulting deviation among neighbors may differ.

Another detail deserving attention is the fact that we use a constant degree for all nodes. However this poses no significant restriction. For the transformation sets proposed for the JOPT, this is always true; for the general case of query optimization the degree can differ. However, these differences are usually small compared to the total number of neighbors.

For a comparison with previous work we extend the above schema and allow also to choose neighbors completely at random without the demand to obey to any distribution of neighbor costs other than the global cost distribution. We use the notation

$$G_{\infty}^{(\alpha)}(n, d)$$

Algorithm Generate Abstract Search Space
Input n size, d degree,
α shape parameter, σ neighbor deviation
Output $G(V,E)$ abstract search space

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
generate set of nodes V
foreach $v \in V$
    $c \leftarrow$ random number from Gamma(α)
    annotate v with c
done
sort V according to costs
foreach $v \in V$
    for $i = 1$ **to** d **do**
        $c \leftarrow$ random number from $N(0,\sigma)$
        $v' \leftarrow$ search(V,c)
        $E \leftarrow E \cup (v,v')$
    done
done
return $G(V,E)$

Figure 7.3. Generate Abstract Search Space

to refer to this variant.

7.2.3 Local Minima

Local minima are nodes that have only edges to nodes of higher costs. The fact that we consider directed edges is of special importance. A node can be a local minimum although a node of lower costs has an edge pointing to it. This situation has its equivalent in query optimization, as pointed out above, where transformations are not symmetric in general.

In an optimization based on the traversal of the graph as we will discuss in the next section, the notion of local minima seems not only intuitive but also of strong influence on the behavior of the optimization algorithm. At least it might seem so on first sight.

Let us first analyze the distribution of local minima in the search space. For the case $G_{\infty}^{(\alpha)}(n,d)$ where arbitrary directed edges—i.e., without respecting a cost distributions among neighbors are allowed—the distribu-

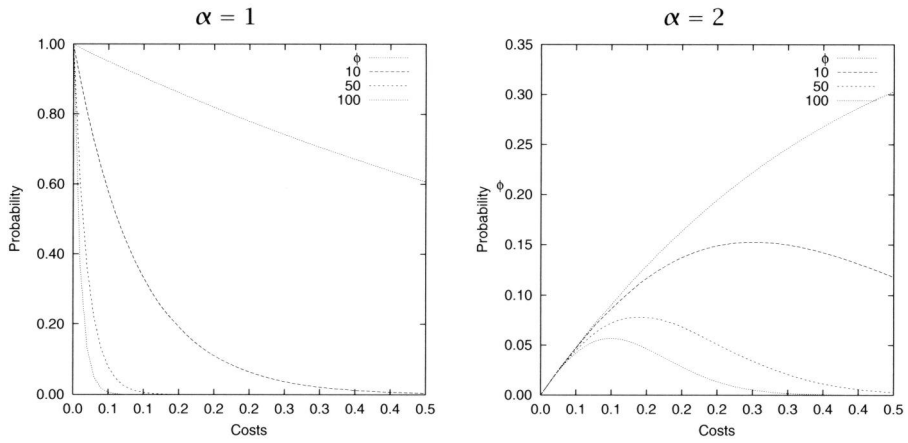


Figure 7.4. Cost distribution together with distribution of local minima as function of cost; shape parameter $\alpha = 1$ in left plot, $\alpha = 2$ in right plot

tion of local minima can be approximated by

$$F(x) = 1 - (1 - \int_0^x \phi(t)dt)^{d+1}$$

(see e.g. [IK91, Kan91]) where ϕ is the overall cost distribution. Clearly, the distribution of local minima in $G_\infty^{(\alpha)}(n, d)$ depends on both α and d. Since we consider directed edges, the size of the graph does not play any role provided $n > d$. Figure 7.4 shows F for an α of 1 and 2, respectively. The degree is varied between 10 and 100.

The plots exhibit a clear trend: local minima are of high incidence only close to the minimum. With higher degree, the distribution of local minima shifts further to the left.

The situation for $G_\sigma^{(\alpha)}$ differs substantially—particularly for very small values of σ. As we choose mainly nodes from a small symmetric range of costs the overall cost distribution is of little influence when choosing neighbors for nodes of costs greater than 2σ. In this case, we can approximate the probability for a local minimum by $(\frac{1}{2})^d$. This changes for costs less than 2σ as the normal distribution used to compute cost values for neighbors gets more and more truncated, up to the point where it degenerates to the right half of the original bell only. In order to maintain the property of being a distribution we multiply the truncated density with the appropriate factor, i.e., we norm it.

In Figure 7.5 experimentally obtained densities are shown. The size of the space was 20×10^6, the degree is varied between 10 and 100, and σ is of values from 0.05 through 0.5. Above 2σ the distribution of local minima is of the quality of the shape of the overall cost distribution as expected. For costs less than 2σ the truncation of the distribution used for selecting neighbors grows more and more asymmetric which results in a larger number of local minima. For higher degrees, the probability for a local minimum above 2σ converges to 0. Local minima occur only close to the global minimum.

In Figure 7.6, an overview over the number of local minima is given for the case where $\alpha = 2$ and both degree, i.e., size of the neighborhood, and deviation of costs among the neighbors is varied.

Figure 7.7 shows the absolute frequency of local minima as close-up for a cost distribution with $\alpha = 2$ and $\sigma = 0.1$. For small degrees the aforementioned effect that local minima are scattered over the whole cost range is clearly visible. With increasing degree, this effect vanishes.

Summing up, we identify a distinct skew of the distribution of local minima in both models: local minima occur mainly at costs close to the optimum. With increasing degrees the concentration of local minima shifts further to left. Specifically, for high degree, the differences in costs between local and global minima are not significant anymore.

7.3 Probabilistic Optimization Strategies

In the introduction to this chapter, we already gave a flavor as to how probabilistic optimization algorithms work. Here, we now present a classification and detailed description of the main representatives.

Blind search optimization algorithms can be divided into several groups according to the optimization principles used, the categories are not exclusive though. We distinguish three major groups: transformation-based, randomized and multi-start algorithms (cf. Fig. 7.8).

The major representative of the first class are classical exhaustive search algorithms as used for example in commercial database systems (cf. Chapter 3). Besides exact methods, also heuristics belong to this class including simple rule-based approaches where a set of transformation is applied until no further application is possible. A typical example is the push-down-selection heuristic, where selection operators are moved down as far as possible in the processing tree. Tabu search is a further member of this class though more sophisticated variants of it periodically also use what is known as *randomized phases* [Glo89, MMS94].

The class of randomized algorithms overlaps with the previous but also contains the class of multi-start algorithms. It includes most prominently *Simulated Annealing* and its close relative *Threshold Accepting*, which are strictly *navigating* algorithms. *Iterative Improvement*—also known as *It-*

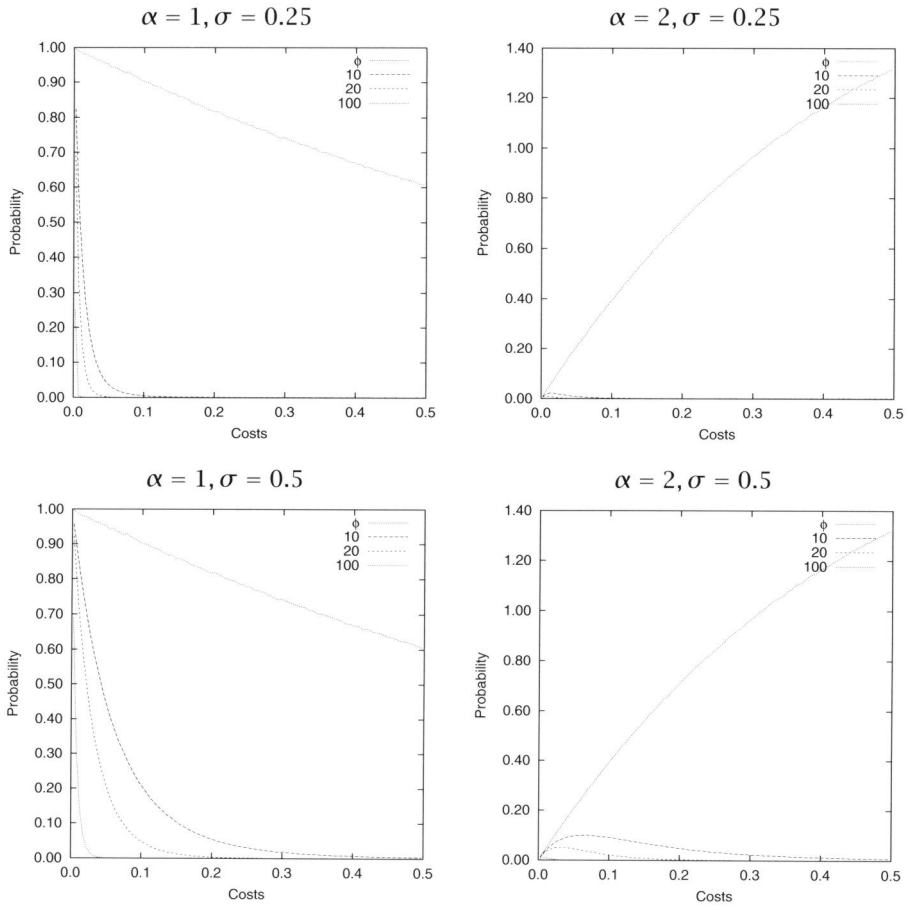
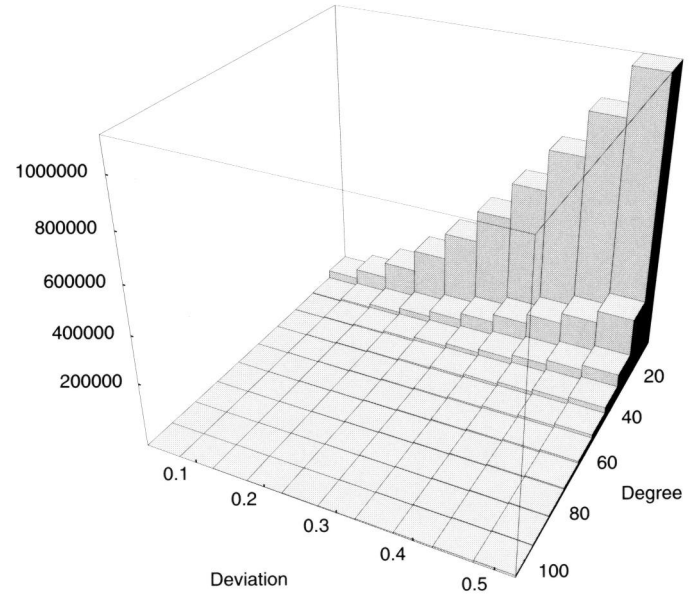


Figure 7.5. Cost distribution together with distribution of local minima as function of cost; shape parameter $\alpha = 1$ in left plot, $\alpha = 2$ in right plot, $\sigma = 0.10$ above, $\sigma = 0.50$ below

erative Local Optimization—conducts simple navigations repeated with a new starting point whenever the local termination criterion is fulfilled. *Transformation-free Optimization* can be viewed as the special case of Iterative Improvement where the navigation is completely omitted. We will devote special attention to the relation of these two later.

Generally, this classification identifies the major building blocks but further combinations are of course conceivable. For example iteratively repeated Simulated Annealing as proposed by Lanzelotte *et al.* [LVZ93]. Like



	Deviation									
Degree	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	
10	32.7	70.9	132.5	217.1	320.2	440.7	577.0	728.8	891.7	1(
20	2.7	10.3	23.0	40.7	63.2	89.9	120.9	157.6	197.4	
30	1.1	4.3	9.5	17.0	26.6	37.8	51.3	66.8	84.3	
40	0.6	2.3	5.2	9.3	14.6	20.9	28.3	36.9	46.7	
50	0.4	1.5	3.3	5.9	9.3	13.1	17.9	23.4	29.6	
60	0.3	1.1	2.3	4.0	6.4	9.0	12.4	16.1	20.6	
70	0.2	0.8	1.7	2.9	4.7	6.6	9.1	11.7	15.1	
80	0.2	0.6	1.3	2.2	3.5	5.1	6.9	8.9	11.4	
90	0.1	0.5	1.0	1.8	2.8	4.0	5.4	7.0	9.0	
100	0.1	0.4	0.8	1.5	2.3	3.3	4.4	5.6	7.3	

Figure 7.6. Number of local minima (in thousands). $n = 20 \cdot 10^6$, $\alpha = 2$, degree and deviation for neighborhood distribution varied

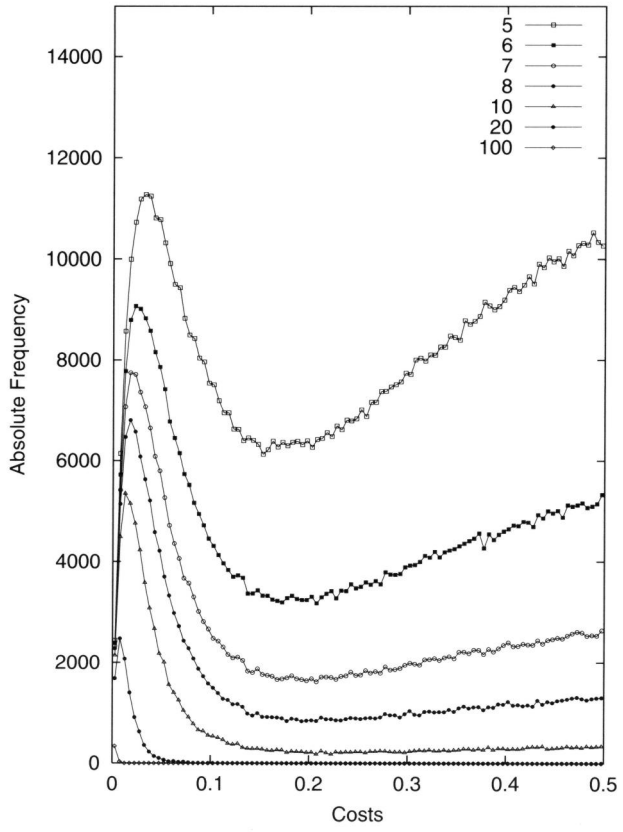


Figure 7.7. Absolute frequency of local minima as function of cost; shape parameter $\alpha = 2$, degree varied

with genetic algorithms, it is not our goal to scrutinize any technical aspect of these algorithm in the sense of tuning, enhancing, or combining single techniques. Rather, we want to check for the generality of the observations made in previous work. All authors of related work tried to explain their results and different models—often built mainly on intuition—have been proposed. We try to view them in a different light to identify strengths and weaknesses of these approaches.

After introducing the algorithms in more detail we will present experiments using the abstract search space model developed above and discuss the effects observed.

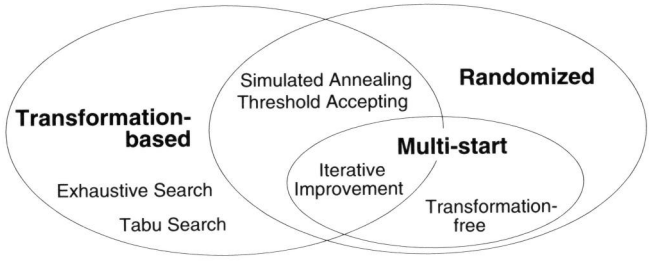


Figure 7.8. Classification of randomized optimization algorithms

7.3.1 Transformation-based Algorithms

First of all, we give an outline of a transformation-based randomized optimization algorithm that serves as a template for all algorithms we discuss later on (cf. Fig. 7.9).

For the *current solution* $s_i \in S$—the initial solution s_0 is an input parameter to the algorithm—an alternative solution $s' \in N(S)$ is chosen randomly, i.e., a transformation is applied to s_i. The new solution is costed and either accepted and becomes the new current solution, i.e., $s_{i+1} \leftarrow s'$, or rejected otherwise, i.e., $s_{i+1} \leftarrow s_i$. The acceptance is controlled by a strategy based on a set of parameters that may include the cost difference between s_i and s' as well as the time elapsed so far. The parameters and their importance vary from algorithm to algorithm. A transition from s_i to s_{i+1} is called *down-hill step* in case $c(s_i) \geq c(s_{i+1})$, and *up-hill step* otherwise. If s' has been accepted we also check for a new record: if $c(s')$ is less than the lowest cost found so far, s' is also stored in s_{best}.

After each such step the parameters are updated and the proceeding is repeated for the new, current solution until certain stopping criteria are fulfilled. Typically, stopping criteria suggested in previous work included elapsed time, number of solutions visited, number of steps without further improvement.

Intuitively, one of the aims in controlling the navigation should be to make up for bad initial solutions and to be as independent of the start solution as possible.

Simple Improvement

What we call Simple Improvement here actually is the "inner loop" of Iterative Improvement. We split Iterative Improvement up into two parts that implement very different principles of randomized optimization.

```
Algorithm  Probabilistic Optimization
Input      s_0 initial solution, A_0 set of initial parameters to accept f
Output     s_best best solution found

i ← 0
s_best ← s_0
c_best ← c(s_0)
repeat
        choose s' ∈ N(s_i)
        if acceptable(c(s'), A_i) then
                s_{i+1} ← s'
                if c(s') < c_best then
                        s_best ← s'
                        c_best ← c(s')
                endif
        else
                s_{i+1} ← s_i
        endif
        A_{i+1} ← update(A_i)
        i ← i + 1
until stopping criteria fulfilled
```

Figure 7.9. Randomized optimization algorithm - Outline

In Figure 7.10 an outline of Simple Improvement is provided using the notation introduced above: A randomly chosen neighbor of the current solution is accepted only if its cost value is less than the current solution's costs. Evidently, this algorithm gets immediately trapped as soon as a local minimum is reached. In this case, any further processing is a waste of time as no improvement can be achieved. Thus, the crucial point of this algorithm is to detect whether the current solution is a local minimum. Usually, to inspect all the current solution's neighbors is too expensive [SG88]. As an approximation, a solution can be judged by a sample of its neighbors, i.e., it is considered to be a local minimum if no cheaper neighbor can be chosen within r consecutive attempts. A local minimum classified this way is referred to as *r-local-minimum* (cf. [IK90]). Consequently, local minima can be identified only with a probability $p < 1$. After r unsuccessful steps the algorithm terminates and the last current solution—which is also the best found so far—is returned. In [IK90] and [SG88] r is either chosen as the number of neighbors or the number of edges in the corresponding join graph.

Our previous experiments concerning local minima however showed

Algorithm Simple Improvement
Input r r-local-minimum size, s_0 initial solution
Output s_i last solution (=best solution found)

$i \leftarrow 0$
$t \leftarrow 1$
repeat
  choose $s' \in N(s_i)$
  if $c(s') < c(s_i)$ **then**
    $s_{i+1} \leftarrow s'$
    $t \leftarrow 1$
  else
    $s_{i+1} \leftarrow s_i$
    $t \leftarrow t + 1$
  endif
  $i \leftarrow i + 1$;
until $t > r$

Figure 7.10. Simple Improvement

that getting trapped in a local minimum is actually not as disadvantageous as may seem simply because almost all local minima in our search space are close to the global optimum and the difference in costs is not significant. Thus we will direct our attention in the next section to the more difficult question whether local minima recognized with the proposed techniques are really local minima.

Simulated Annealing

Simulated Annealing is probably the most prominent of randomized optimization techniques. It has been deployed and studied in almost all fields of research concerned with combinatorial optimization and besides experimental assessment also mathematical models to capture its behavior have been developed.

Simulated Annealing is a dynamic variant of the *Metropolis Algorithm* derived from statistical mechanics [MRR+53]. It also starts with a randomly chosen solution s and a neighbor s' is accepted with the probability

$$e^{\min\{\frac{1}{t}(c(s)-c(s')),0\}}.$$

In contrast to the Metropolis Algorithm, the temperature t decreases gradually and the algorithm terminates after t falls below a given threshold. The output of the algorithm is the least costly solution visited so far [KGV83].

Algorithm Simulated Annealing
Input s_0 initial solution, T_0 initial temperature
Output s_{best} best solution found

$i \leftarrow 0$
$s_{best} \leftarrow s_0$
$c_{best} \leftarrow c(s_0)$
repeat
  repeat
    choose $s' \in N(s_i)$
    choose $p \in [0,1]$
    if $p \leq e^{\min\{\frac{1}{T_i}(c(s_i)-c(s')),0\}}$ **then**
      $s_{i+1} \leftarrow s'$
      if $c(s') < c_{best}$ **then**
        $s_{best} \leftarrow s'$
        $c_{best} \leftarrow c(s')$
      endif
    else
      $s_{i+1} \leftarrow s$
    endif
    $i \leftarrow i + 1$
  until equilibrium reached
  $T_{i+1} \leftarrow$lower(T_i)
until frozen

Figure 7.11. Simulated Annealing

Simulated Annealing is superior to Simple Improvement as it can escape local minima with a time dependent probability.

As previous work pointed out, Simulated Annealing depends heavily on the parameter t and the cooling schedule that determines its decrease. If the initial temperature is too low the process terminates early without finding low costly solutions—if it is too high, time is wasted since many expensive solutions are considered. Experiments show that mathematical parameter estimation is only of very limited use what makes finding appropriate parameters a matter of experience [Haj88]. In [Swa89b] and [IK90] different simple heuristics for a computation of the parameters are presented.

Moreover, additional termination criteria like time limits or a maximum number of generated solutions were introduced. Unfortunately, combination of different termination criteria may veil the impact of single parameters as they may impact each other negatively. For example, if too short a

Algorithm Iterative Improvement
Input r r-local-minimum size
Output s_{best} best solution found

$i \leftarrow 0$
$c_{best} \leftarrow \infty$
repeat
    choose $s \in S$
    $s_{i+1} \leftarrow$ Simple Improvement(s, r)
    if $c(s') < c_{best}$ **then**
        $s_{best} \leftarrow s'$
        $c_{best} \leftarrow c(s')$
    endif
until stopping criteria fulfilled

Figure 7.12. Iterative Improvement

running time limit is imposed, the effects of the coding cooling parameter may vanish etc.

7.3.2 Multi-start Strategies

Multi-start algorithm are repetitive applications of any randomized optimization algorithm starting each run of the particular algorithm with a new initial solution—different from the previous ones if possible.

Essentially, every randomized optimization algorithm can be used for multi-start optimization, however, navigating and multi-start technique are contradicting principles as one of the aims pursued in the previous is independence from the start solution, i.e., in navigating algorithms some sophisticated acceptance strategy is deployed so that algorithms can escape unfavorable conditions such a s local minima. The higher this level of sophistication, the less we should expect the impact of multi-starts. We have seen a similar effect with genetic algorithms (cf. Chapter 6) and re-starts of the optimization.

Iterative Improvement

Iterative Improvement is the multi-start variant of Simple Improvement. It has been investigated in a broad variety of configurations (cf. [SG88, IK90, INSS92, SHC96]). As shown in Figure 7.12, Simple Improvement is applied repeatedly on different start solutions and s_{best}, the best result found so far, is returned. As a termination criterion often a time limit is used be

```
Algorithm  Transformation-free Optimization
Output     s_best best solution found

c_best ← ∞
repeat
        choose s ∈ S
        if c(s) < c_best then
                s_best ← s
                c_best ← c(s)
        endif
until stopping criteria fulfilled
```

Figure 7.13. Transformation-free Optimization

it either explicitly given or implicitly specified by a maximum number of repetitions.

Though neglected already a long time ago because of its simplicity and all too obvious shortcomings it has been re-discovered lately in various application areas as more sophisticated algorithms including genetic algorithms could not achieve performance gains that would justify the increased running time [Boe96].

Transformation-free Optimization

In Transformation-free Optimization the whole optimization is narrowed down to the multi-start principle: Solutions are chosen randomly and costed [GLPK94]. The best solution found is returned as soon as the stopping criterion is fulfilled (cf. Fig. 7.13). In other words Transformation-free Optimization implements uniform random sampling. this approach is distinguished by both the techniques deployed and its results: The algorithm is in so far interesting as the authors develop uniform generation of join orders which is all but a trivial task [GLPK95]. More notably, however, is the quantitative assessment presented. Sampling apparently outperforms other optimization algorithms including Simulated Annealing and Iterative Improvement in that it finds high quality solutions quicker. The differences to solutions found with other strategies are not significant.

In our setting here, Transformation-free Optimization appears rather trivial as the uniform generation of join orders is substituted by a uniform random choice of a node in the graph.

Algorithm Two-Phase Optimization
Input r r-local-minimum size,
T_0 initial temperature, n number of runs
Output s_{best} best solution found

$i \leftarrow 0$
$c_{init} \leftarrow \infty$
while $i < n$ **do**
    $s \leftarrow$Iterative Improvement(r)
    if $c(s) < c_{init}$ **then**
        $s_{init} \leftarrow s$
        $c_{init} \leftarrow c(s)$
    endif
    $i \leftarrow i + 1$
done
$s_{best} \leftarrow$Simulated Annealing(s_{init}, T_0)

Figure 7.14. Two-Phase Optimization

7.3.3 Hybrid Strategies

Recognizing the shortcomings of a particular algorithm, one might be tempted to find improvements. Especially, combinations of algorithms attracted repeatedly attention. In order to speedup Simulated Annealing which can deliver results of high quality but is extremely time consuming, variants that apply a pre-processing first have been introduced [IK90, LVZ93]. The pre-processing tries to exclude very unfavorable initial solutions: The best solution found in the first phase is being explored in detail with Simulated Annealing. Therefore, the parameter setting can be tightened and less running time is needed to obtain low costly result solutions.

Among others the most prominent algorithm of this class is Two-Phase Optimization which applies several Iterative Improvement runs before the Simulated Annealing [IK90] (see Fig. 7.14).

7.4 Discussion

An assessment of randomized optimization algorithms is a difficult task because of the various degrees of freedom in setting up a test bed. In the previous chapters we have seen some fundamental properties common to all cost models and query optimization scenarios which can be used—and

in fact have been used—as a rough orientation; most notably the catalog variance. However all algorithms presented rely on a couple of parameters which are difficult to tune in general. In [SG88] and [IK90], the parameters for Simulated Annealing e.g. have been chosen differently yet both parties claim that their choice was optimal with respect to elaborate test series. This in turn suggests that their test cases must have been of some significant structural difference. Another question totally ignored so far is whether the parameters should be adapted to the particular query type, i.e., its cost distribution. Up to date all heuristics that have been proposed to determine appropriate parameters are constant in the sense that they do not take the differences between given queries into account, e.g. one heuristic suggests to choose the initial temperature in Simulated Annealing so that a certain percentage of neighbors of the initial solution would get accepted. However, depending on the quality of the initial solution this choice may be anything from very slack to very restrictive

As parameter ranges are unbound, at least covering large areas of "reasonable" values, an exhaustive tuning is infeasible. Moreover, the experiments in related work have been conducted with cost models of different levels of sophistication. Though we could expect basic tendencies to be found in all of them as motivated in Chapter 4, the numerical values differ in general. Most of the experiments were also presented as scaled costs, i.e., as factor of the best solution found by any algorithm. This can be a conclusive assessment when using industrial quality cost models, however, it is reflecting merely the trend with less sophisticated cost models. By this we mean that the order may very well be preserved—algorithm A is better then algorithm B—but e.g. twice the number of I/Os does not necessarily imply that B needs twice the running time. This becomes increasingly important if the cost values differ only by a few percent.

Conducting a series of experiments on the lines or related work, i.e., using a fixed set of parameters and an own cost model, would simply add yet another data set to the anyhow large collection of results available. Rather, we use the wealth of results as a basis for a discussion in order to explain some of the effects observed which could not be clarified completely yet.

To that end we will focus on two so far largely neglected aspects: The role of local minima and their classification on the one hand, and the evaluation of the influence of the multi-start principle.

7.4.1 Classification of Local Minima

As we have seen in the previous section *navigation* in the search space is the key element of most probabilistic optimization methods. Landscape models i.e., cost function of the kind $f : R^2 \rightarrow R$, serve the intuition very well when it comes to discussing the behavior of these optimization algorithms. Landscapes subliminally suggest a continuous cost function and, what is synonymous, that the search space can be embedded sensibly into R^2. A reasonable embedding would however require the graph of the search

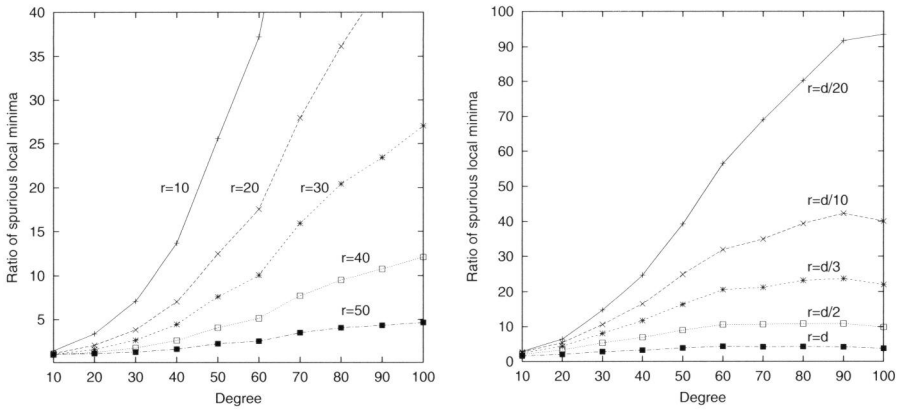
space to be planar, which is not the case for instances of non-trivial size. Let us however cling to this model for a moment. The navigation of an algorithm is then motivated as search along structures that have been described as "slopes", "valleys", "cups" etc. The notion of a local minimum then comes in very intuitively, almost compelling. Based on these considerations local minima have been suspected to have significant impact on the performance of an optimization strategy.

On the other hand, we found there are hardly any local minima of significant distance from the minimum, i.e., finding a local minimum is almost always as good as finding the optimum.

Here, we will investigate the accuracy of the classification of local minima. Judging a local minimum correctly eventually requires to evaluate the costs of all neighbors. However, the generation of all neighbors, i.e., applying all possible rules to a given solution, is a time consuming task. Therefore, all implementations proposed so far use sampling. For large parts of the cost range ($c > 2 \cdot \sigma$), this technique finds a better neighbor within a very small number of retries. Recall that the distributions of neighbor costs is almost symmetric. Thus in the upper cost range, the classification by sampling is sufficiently accurate. In the lower cost range where most local minima are located, the situation is very different. Assume a solution has 99 more costly and 1 less costly neighbor, i.e., it is not a local minimum. To classify the solution correctly for not being a local minimum and therefore finding the only neighbor with lower costs with an error of less than 5% requires almost 300 retries. Keeping the error as low as possible is crucial—just assume the only better neighbor in the example above was the global optimum.

Figures 7.15 show the ratio of spurious local minima, i.e., solutions that were incorrectly classified as a local minimum, to real local minima as function of the degree. The size of the sample, i.e., the number of retries was varied between 10 and 100. In Figure 7.15a the sample is constant throughout the experiment, in Figure 7.15b, the size of the sample is a fraction of the degree. Swami *et al.* suggested what comes down to $\frac{d}{3}$, Ioannidis *et al.* recommend d, d being the degree. Even for large sample sizes, the number of spurious minima is distinctly above twice the number of real minima. Remarkably, as the experiments showed, the spurious minima are in significantly higher distance of the global minimum and scattered over a larger range of costs than real minima.

In the next experiment, we investigate the trade-off between navigation and multi-start principle. We run Simple Improvement with a limit of n retries and n steps in total and compare it to Transformation-free Optimization. The sample size of Transformation-free Optimization is also n. Note, we re-start Simple Improvement if the limit is not yet exhausted. For both techniques we take the average of 1000 runs. Moreover Simple Improvement is always started in a high costly area, i.e., in a solution with roughly 2μ costs. This way we can exclude the effect of randomly choos-



(a) Fixed number of retries (b) Number of retries as fraction of degree

Figure 7.15. Effectiveness of classification of local minima by sampling

ing a very good solution as a starting point. Figure 7.16 shows the average result quality of both optimizers in absolute figures. Transformation-free Optimization achieves a relatively high result quality with only a few retries already. Improvements for more than 70–100 retries are only gradual yet visible.

In contrast to that, Simple Improvement needs a longer time just to navigate to the areas of interest wasting its energy along the way. This does not come as a surprise in principle. However what *is* notable is the extent to which Transformation-free Optimization is able to outrun Simple Improvement. Specifically for small numbers of n. The plot also suggests that in order to make effective use of multiple start a number of more than 20 better 50 restarts is advisable. In the following section we use the results obtained so far to interpret related work, i.e., setup and results and discuss their finding.

7.4.2 Analysis of Related Work

The four most prominent contributions to the debate are [SG88], [IK90, IK91], [SMK97] and [GLPK94]. Their results can be summarized as

- [SG88]: Iterative Improvement outperforms Simulated Annealing
- [IK90, IK91]:Simulated Annealing outperforms Iterative Improvement, Two-Phase Optimization performs best

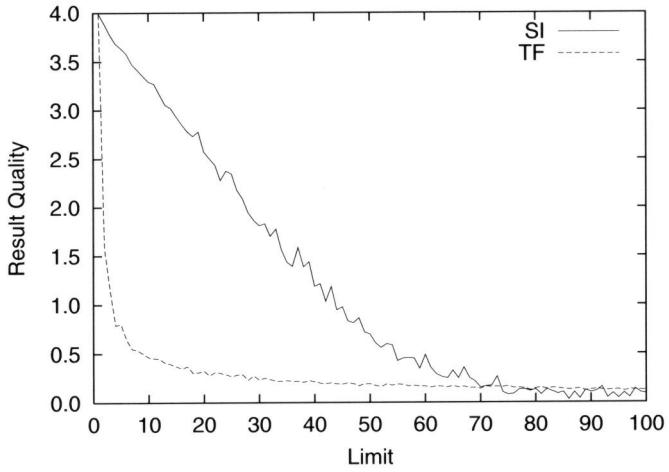


Figure 7.16. Multi-start principle compared to navigation

- [SMK97]: Iterative Improvement (often) outperforms Simulated Annealing, Two-Phase Optimization is better but not significantly
- [GLPK94]: Transformation-free Optimization converges fastest. Iterative Improvement outperforms Simulated Annealing

Further related work like [SI93] or [LVZ93], suggest combinations of algorithms which we will not discuss here in detail.

Though most of the results independently found are coinciding largely, there is a distinct discrepancy concerning Iterative Improvement: In [IK91] authors elaborate on this fact presenting an analysis of the search space based on local minima. However, to understand the differences in result quality it is helpful to analyze the different test beds first.

Swami and Gupta use a linear search space where initial solution are not chosen with uniform probability, however, all solutions can be chosen with a probability that is distinctly greater than zero. Furthermore due to the set of rules, the probability to generate a neighbor solution with exactly the same costs is marginal.

In [IK91] things differ as authors use a bushy search space. However, they use only linear trees as initial solutions. Particularly, in the light of our findings about linear and bushy spaces together with the authors' findings that in most cases the optimal plan was a bushy plan, this suggests that Iterative Improvement is made wasting a large portion of running time just to "escape" into the bushy space. In other words, only a subspace is used

to choose initial solutions.

To prune the search space their cost model applies commutative exchanges when necessary, i.e., the costing decides whether to use a join as given in the plan or to commutatively flip the inputs, whatever is cheaper. As a result the effect of plateaus where many neighbors can be of the same costs has higher incidence that in the previous setup. This leads to a skew of the classification of local minima as neighbors of equal costs are unacceptable in Iterative Improvement. Consequently more time is spent on classification of local minima. Also the number of spurious local minima is substantially higher. In contrast to that [SMK97] used initial solutions generated with non-uniform probability from the general space of bushy plans the same holds for the set up used in [GLPK94]. As an immediate consequence, Iterative Improvement proves a competitive algorithm in their work.

Given our analysis of Chapter 4 and the considerations about restart in Chapter 6 the excellent performance of multi-start algorithm is evident. This is further supported by Ioannidis' findings concerning Two-Phase Optimization as well as Lanzelotte, Valduriez and Zaït's Toured Simulated Annealing; also Swami and Iyer use restarts showing that a very small number starts—less than 10 in their *AB* algorithm—makes already a difference.

Using a fair setup for the algorithms the differences in result quality are below any significance. This fact has been ignore in most studies and even for join orders of up to 100 joins, differences of less than 10% have been recorded and taken for an indication what algorithm to use.

7.5 Summary

Probabilistic algorithms can find acceptable solutions for even very large queries within very short running times. They are distinctly superior to deterministic heuristics, which also require only short running times, but as Steinbrunn *et al.* pointed out, produce increasingly low result quality for larger queries [SMK93].

In this section we first investigated the basic feature of linear and bushy search spaces concluding that the bushy space includes equally good and often better solutions than its linear counterpart. To asses number and location of local minima we devised the concept of abstract search spaces, which also served as a work bench to test algorithms. We discussed incidence and location of local minima with respect to the influence they have on optimization algorithm, in particular on local optimization also known as Iterative Improvement.

Our findings essentially suggest that there are only very few local minima that are of significant difference to the global optimum, i.e., for most of the local minima, the costs are comparable to the optimum in the sense they are acceptable optimization results.

Thus, getting trapped in a local minimum is not the primary reason for inferior performance of these algorithms as conjectured in related work. Rather the classification of local minima is a very time consuming task and too many solutions are wrongly classified as local minima. Algorithms that do not need to classify local minima like Simulated Annealing can put the saved time in further navigation. The effect can be further aggravated by the choice of a search space where the distribution of costs in neighborhoods is unduly skewed by plateaus, caused by a disadvantageous combination of rules and cost model. Another important factor is the choice of initial solutions. If they are chosen—not necessarily uniformly though—from the target space Simple Improvement or Iterative Improvement as well as sampling can find very good solutions. If the choice of initial solution is however limited to an unfavorable sub space, the multi-start principle is largely ineffective. Otherwise simple restart of an optimization can greatly improve the result quality because of the advantageous cost distribution.

8

Good Enough is Easy

In the previous chapter we have explained why it is generally very difficult to distinguish randomized algorithms in performance. This phenomenon has accompanied research in this field from the begin on and led Swami to conclude:

> *These results lead us to speculate that until significant new insights are obtained into the characteristics of the search space it will not be profitable to experiment with very complex methods for optimization [referring to Simulated Annealing].* [Swa89b], page 376.

Now, with knowledge about the structure and characteristics of the search space at hand it seems in order to turn this statement around: Given our assessment of cost distributions and randomized algorithms we can expect even lesser sophisticated algorithms to produce more than just acceptable results.

Galindo-Legaria *et al.* made out a good case for using uniform sampling of plans instead of transformations (see Chap. 7) [GLPK94]. The algorithm devised is a complex construction whose deployment is, however, limited to acyclic graphs. This limitation—though popular with related work—is a distinct restriction. Queries as for instance in the standard data warehouse benchmark suite of TPC-H contain indeed cyclic queries. But this algorithm shows the way how to exploit the shape of cost distributions successfully. The question we tackle here is whether we can overcome the limitations without performances losses, and furthermore, whether uniformity of the sampling is a necessary prerequisite.

Using the principles discussed in Section 3.2 we devise a simple technique that performs biased rather than uniform sampling, but is distinguished by its low complexity and applicability to arbitrary join graphs. It further gives room to a cost bound pruning component that discards partial query plans which cannot lead to a better plan than the currently best, as early as possible [3].

```
Algorithm  QUICKPICK
Input      G(V, E) join graph
Output     s_best best query plan found

r ← ∞                                  // initialize lowest costs so far
E' ← E
q ← G'(V, ∅)                           // initialize query plan
repeat
    choose e ∈ E'                      // random edge selection
    E' ← E' \ {e}
    ADDJOIN(q, e)
    if E' = ∅ or c(q) > r do           // either plan complete or costs exceeded
        if c(q) < r do                 // check for new best plan
            s_best ← q
            r ← c(q)
        done
        E' ← E
        q ← G'(V, ∅)                   // reset query plan
    done
until stopping criterion fulfilled
return s_best
```

Figure 8.1. Algorithm QUICKPICK

8.1 Biased Sampling

To implement a biased sampling we utilize the techniques introduced in the context of the enumeration of non-isomorphic processing trees in Section 3.2. There we presented MERGETREES a simple algorithm to turn sequences of join predicates into processing trees. We then concentrated on the enumeration of non-isomorphic sequences. Instead of *enumerating* sequences of edges, we generate *random* sequences and turn them subsequently into processing trees.

In Figure 8.1, the algorithm, called QUICKPICK, is outlined in pseudo code. After initializing the variable r that records the cheapest plan found, the candidate set E' is initialized with the set of edges of the join graph, and q with the base relations. Throughout the random bottom-up construction of a tree q holds all partial trees, i.e. q is actually a forest. Generally, only at the very end—earlier only for cyclic join graphs—, q is completed to a single processing tree.

Until the stopping criterion, say a time limit, is fulfilled q is incremen-

tally built-up by choosing and removing an edge e from the candidate set and adding the corresponding join to the tree. In doing so, the subtrees that contain the two endpoints of e, i.e. the base relations joined by this edges, are connected with a join operator (see Fig. 3.5). If both relations are already leaves to the same sub-plan, only the predicate of e is added to the tree at the deepest possible point. After each such insertion, the costs of the subtrees are computed and summed up. Recall, that q is generally a forest consisting of several disjoint processing trees. If the costs exceed r, the costs of the best plan found so far, we discard q and initialize E' and q again and start assembling a new tree. If the set of candidate edges is empty—i.e. we have completed the processing tree—we check for a new record and in this case copy the plan to s_{best}. After initializing E' and q we start building a new tree.[1]

Essential for the cost bound pruning is the cost computation along the structure in the making. We assume a monotonic cost formula where operators do not influence the costs of their predecessors other than monotonically increasing, i.e. adding an operator later cannot *reduce* the costs of any subtree.

The algorithm performs a non-uniform, or biased sampling as different sequences of edges may lead to the same result (see also Section 3.2.3). But even computing and excluding redundant edges does not restore uniformity as different prefixes entail different number of possible completions, in general. For example assume there are 1000 non-redundant sequences with e_1 as first element and 2000 for e_2. Since we do not know the numbers in advance, we cannot adjust the probabilities with which they are chosen. Selecting e_1 or e_2 with equal probability clearly leads to a non-uniform sampling.

8.2 Assessment

For an assessment of QUICKPICK the abstract search space model is not useful since the only characteristic exploited by the algorithm is the cost distribution which differs now from the original due to the bias of the sampling. Instead, In order to determine the cost distribution under QUICKPICK, we implemented a cost model comparable to those proposed in [EN94, KS91, Ste96].

[1] The basic principles of QUICKPICK—without cost-bound pruning—have been described already by Pellenkoft [Pel97]. There, this algorithm is called Random Edge Selection and proved to be incapable of achieving uniform sampling. However, no further performance analysis is conducted. Others might have probably used similar algorithms to generate initial solutions. However, they also did not evaluate the potential of this elementary technique.

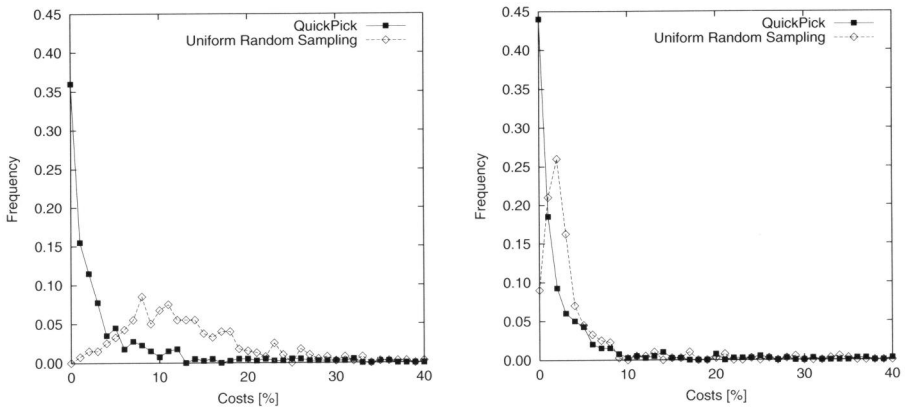


Figure 8.2. Comparison of cost distributions obtained with biased and uniform sampling

8.2.1 Cost Distribution

Clearly, to be successful, the cost distribution ϕ_B under QUICKPICK must be at least as favorable as the original, i.e. shifted to the left relative to ϕ.

In the following we compare ϕ_B and ϕ under three aspects: (1) selective samples, (2) the correlation coefficient between a larger set of cost distributions, and (3) the shift of ϕ_B relative to ϕ.

In Figure 8.2, two pairs of cost distributions for high and low variance catalogs are shown. Both samples are of size 5000, the query used is of size 50. To obtain cost distributions with QUICKPICK we disabled the cost bound pruning so that complete trees were constructed. In a larger series of test cases ϕ_B was without exception always left of ϕ. Moreover, ϕ_B bore in all cases strong resemblance with exponential distributions.

To test for a connection of ϕ_B and ϕ we compute the correlation coefficient. For two random variables, this coefficient computes to

$$k = \frac{E[(X - E[X])(Y - E[Y])]}{\sigma_X \sigma_Y},$$

where $E[X]$ denotes the mean of X and σ_X is the deviation. For fully correlated distributions, k approaches 1. The more the distributions differ, the lower k gets. In Figure 8.3, the correlation coefficient is plotted as a function of the query size. Each point comprises 50 pairs of randomly generated queries. The plot shows a clear trend of decreasing correlation with increasing query size.

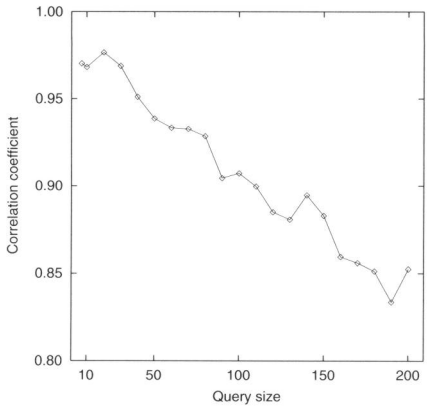


Figure 8.3. Correlation of uniform and biased cost distribution

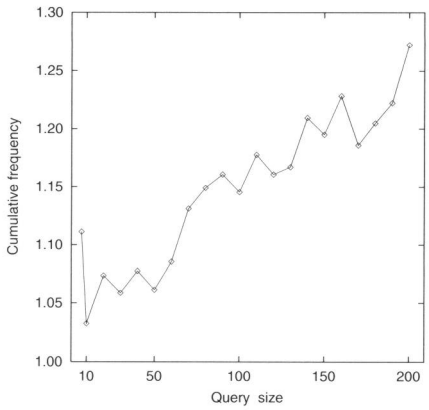


Figure 8.4. Cumulative distributions

Finally, we determine the relative shift of ϕ_B. To that end, we compute the cumulative distributions for ϕ_B and ϕ on the interval $[0, \mu_\phi]$, recall the shift is defined with respect to single reference points (see Section 7.1). In Figure 8.4 the shift $s(\mu(\phi))$ is plotted as function of the query size. Again, each data point represents the average of 50 queries.

Our results pin-point a clear trend that the biased cost distribution is even more favorable to sampling than the original one. With increasing query size, the difference between the two distributions becomes more distinct, showing the biased one stronger to the left of the original.

8.2.2 Quantitative Assessment

According to our analysis of the cost distribution, the results reported on by Galindo-Legaria *et al.* in [GLPK94] can immediately be transferred and serve, so to speak, as an upper bound for the result quality.

Like uniform sampling, QUICKPICK is unlikely to find the optimum as sampling works on the premise that all solutions in the top quantile—the size is parameter to the problem—are equally good. Thus hitting this quantile in the course of the sampling is good enough.

Result Quality

Figure 8.5 shows the quality of the results in terms of quantile-based quality (cf. Chap. 5). For the experiments we differentiated the following shapes

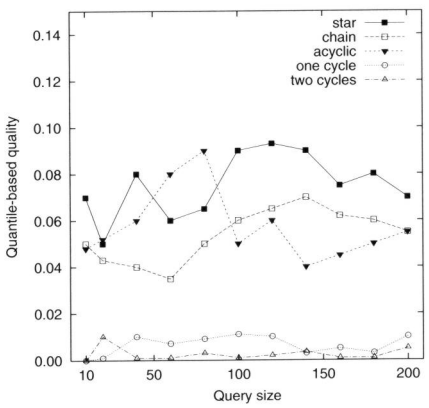


Figure 8.5. Performance

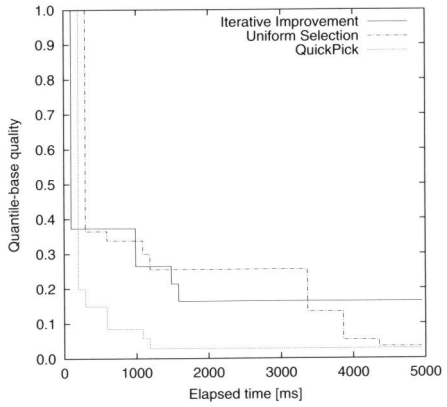


Figure 8.6. Convergence behavior

of query graphs: *stars*, *chains*, and *tree-shaped* on the one hand, and a type which we call *n-cycle* on the other hand. The first group comprises queries that can also be optimized with uniform sampling; The second group exceed these limitations. A graph of type n-cycle contains exactly n cycles, as the name suggests, but the remainder of the graph is unspecified, i.e. we use randomly generated tree-shaped graph and insert n additional edges. Our notion of cyclic graphs reflects real queries better than highly connected graph structures, such as grids or cliques. Also the graph theoretic notion of connectivity is lesser suitable as almost all queries in actual applications are of a connectivity no higher than one.

For acyclic graphs, QUICKPICK delivers results of a quality comparable to that of uniform sampling—for star graphs, QUICKPICK actually implements even uniform sampling. In case of cyclic query graphs, QUICKPICK finds clearly better, near-optimal solutions (see Fig. 8.5).

Convergence Behavior

Like with uniform sampling, QUICKPICK's strong point is its quick convergence. Figure 8.6 shows the costs of the best plan found as function of the elapsed time in comparison with Iterative Improvement and uniform sampling. Due to its biased cost distribution, QUICKPICK converges significantly quicker. With longer running time the competitors catch up. Iterative Improvement often beats QUICKPICK, not significantly though.

To underline the differences between uniform sampling and QUICKPICK, we compute the probability to hit the quantile $Q_{0.1}$ for both algorithms.

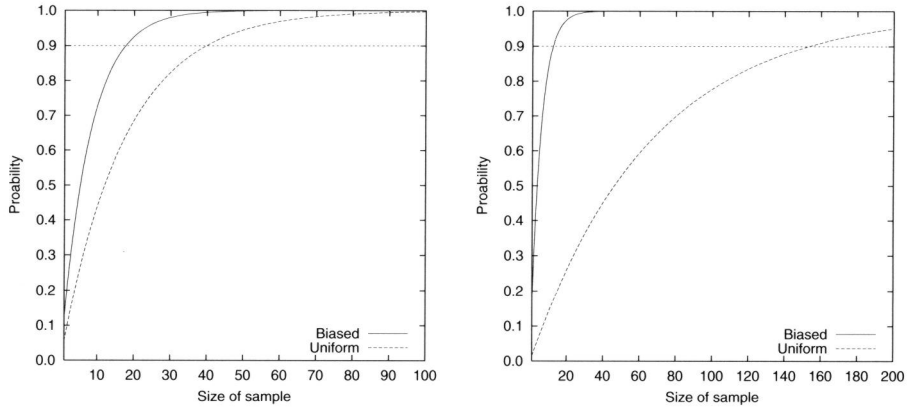


Figure 8.7. Probability to hit quantile $Q_{0.1}$ with QUICKPICK and uniform sampling. Left, high variance, right low variance catalog

$Q_{0.1}$ refers to the respective quantile of the original distribution. In Figure 8.7, these probabilities are plotted as function of the size of the sample. The left plot shows the situation for a high, the right for a low variance catalog. To hit the quantile with more than 90% probability in the high variance case requires a sample size of 18 and 40 for QUICKPICK and uniform sampling respectively. In case of low variance catalog, the numbers differ even more significantly: 13 and 154.

Cost-bound Pruning

Let us finally investigate the impact of cost-bound pruning within QUICKPICK. We introduced the algorithm in the form that partial trees are discarded as soon as their costs exceed the currently best plan's cost.

According to our general considerations about the cost distributions the effectiveness of the pruning depends heavily on the shape of the distributions. The further to the left the distribution is, the lower the gains, i.e. the trees are built-up almost to completeness. In Figure 8.8 this effect is demonstrated with low and high variance catalogs for a query of size 100. The left plot in 8.8a, shows the number of join predicates inserted with ADDJOIN—referred to as size of tree in the figure. As a stopping criterion we used 100000 insertions, which made in this example for 1286 explored trees in total. For each (partial) tree we indicate the size when it was discarded (see Fig. 8.8a left), 100 being the maximum. Note, not every tree

completed is a new record since the last join can still exceed the best costs so far, which happens specifically frequent with high variance catalogs. The plot on the right hand side shows the average tree size as function of the number of trees. Starting at 100 it drops quickly to about 80 (see Fig. 8.8a right).

In Figure 8.8b the same analysis is done for a low variance catalog. Since there is no strong concentration of solutions as opposed to the previous case, pruning kicks in earlier. The average tree size drops to about 40. Consequently, 100000 steps make for a larger number of (partial) trees explored; 2539 in this example.

In the first case savings amount to some 20%, in the second almost 60% on average.

8.3 Summary

For queries of increasing size, the accuracy of the costing techniques drops. Consequently, plans of costs in the top quantile of the distribution are as good as the optimum. This premise formed the basis for the work of Galindo-Legaria *et al.* who developed a mechanism to generate join orders with uniform probability. They proof uniform sampling to be competitive to other randomized algorithms such as Iterative Improvement or Simulated Annealing. Moreover, uniform sampling can be used as a building block for compound algorithms or for generating initial states for other algorithms.

In this chapter we scrutinized the potentials of biased random sampling. Using parts of the algorithms introduced in Section 3.2. The resulting algorithm QUICKPICK is distinguished by its short running time, low complexity with respect to both implementation and run time behavior, and its result quality.

Not only an interesting result in its own right, this analysis also gives an impression of the potential effectiveness and importance of choosing the initial solutions for randomized optimization algorithms as discussed in Chapter 7.

Though QUICKPICK extends the domain of sampling to the general case, achieving better results, the challenge of finding an algorithm for general, *uniform* sampling remains. The algorithm presented in Section 3.3 can well be used for sampling, since counting implies uniform sampling. In fact, we conducted experiments with Microsoft SQL Server backing the claims that sampling is truly a very good alternative to exhaustive strategies. However, this approach is limited by the exponential growth of its MEMO structure.

So far, no method is known for uniform generation of join orders regardless the join graph, nor seem existing techniques to extend to it. Determining the complexity of this problem is still an open problem—i.e. it is

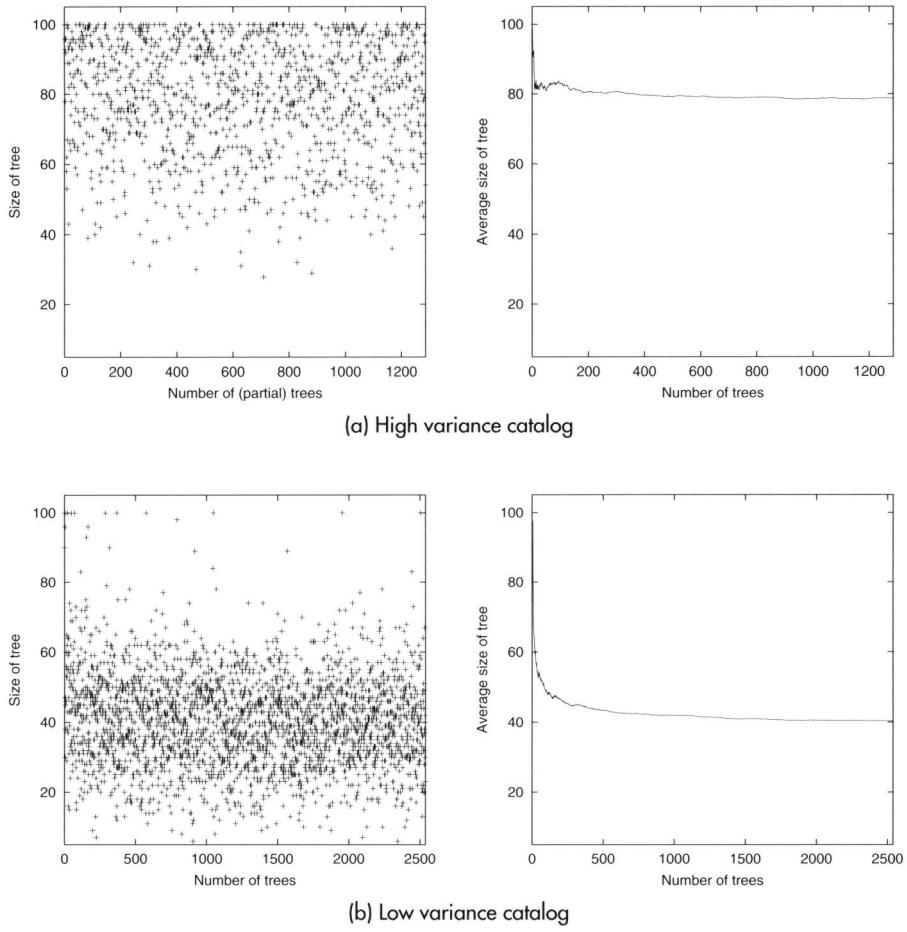


Figure 8.8. Effectiveness of cost-bound pruning in QUICKPICK

unclear whether it is in P or $\#P$. The same holds for counting, which is in general more difficult than uniform generation [JVV86, Sin92].

Another direction of future research is sampling in the context of query optimization beyond join ordering. The abovementioned approach of sampling over the fully expanded MEMO is of course not very useful as the best solution can be extracted immediately once the MEMO is constructed.

However had we a sampling technique for random generation for com-

plete arbitrary SQL92—or even SQL99—statements, the sampling could be used to get very good initial plans that can then be copied in and enumerated with the MEMO. Using only a reduced set of transformation rules ensures that only the "vicinity" of the candidates is searched. In fact, a simpler variant of this idea is for example implemented in SQL Server where a prospective join order is determined before the plan is copied into the MEMO. With a sampling phase and multiple MEMO structures this idea could be pushed way further using say 10 best sampled plans out of a set of 100 and optimizing them simultaneously with very rudimentary rule sets. The sampled plans also serve as fall-back solutions in case time limits are reached.

Additionally, the method sketched offers obvious possibilities for an immediate parallelization of the optimization phase.

9

Conclusion

The problem of query optimization differs from typical combinatorial optimization problems in two important aspects.

Firstly, a problem instance is strictly speaking always a problem instance with respect to a certain system configuration. In contrast to problems like the Traveling Salesman Problem, Knapsack etc. we lack a universal portable problem specification and, more severe, a universal cost model. Hardly any complex query will lead to identical query plans when optimized on two different database systems, yet, for each system they may very well be the best plans. These differences result from a different design and implementation techniques used. From one database system to another basically all components differ in more or less significant ways. For instance, the set of operators usually come in a large variety where each of them reflects some technicalities that are specific to the particular database system. Moreover the sets of operator differ widely; for example constructs like hash teams as an implementation technique for a group of subsequent hash joins is a singular specialty as is bitmap filtering in connection with hash joins [GBC98, CHY93]. Thus given a query there is not just *one* optimal plan but an optimal plan with respect to the cost model used.

Secondly, even within the framework of one database system the question of the optimal solution cannot be answered definitely if the query is of large or very large size. The estimate errors increasingly dominate the cost computation and the costs computed serve only as an approximation of the actual execution costs. Consequently, an optimization beyond the resolution of the cost model, i.e. the capability to distinguish two solutions conclusively in their costs, is not useful. As opposed to this situation, the Traveling Salesman Problem and other classical combinatorial optimization problems have an exact cost formula and are independent of any other background component, thus, these problems are exactly reproducible—the optimal tour of a Traveling Salesman Problem can be determined unambiguously unless there are several tours of the same optimal length.

Both these facts render the query optimization problem subjective and

approximative rather than a problem that could be solved to optimality in isolation. However, the problem also displays general trends that put bounds to the uncertainty, that allow us to postulate a series of basic properties.

In this work we sat out to analyze the problem's underlying structure and addressed specifically the issue of randomized or probabilistic optimization of large queries. In the following, we summarize the results achieved and discuss remaining open problems afterward.

9.1 Summary

Our analysis revolved around the concept of cost distributions and their effects on optimization techniques. Cost distributions determine the frequency of cost values in the complete search space identifying characteristic concentrations of cost values.

First of all, we provided the necessary means to obtain and verify such distributions for three differently sophisticated versions of the problem: cross product optimization, join order optimization with non-isomorphic processing trees, and finally, the unrestricted general case of query optimization. While the first two were developed on simplified models, the latter was devised and implemented in Microsoft SQL Server.

Equipped with this toolkit we extracted and analyzed cost distributions for the different models pointing out their close relationships and similarities. The distributions displayed the same trends—within certain ranges of variation—and are distinguished by their high stability. We contrasted the distributions found with distributions of other NP-hard combinatorial optimization problems including Traveling Salesman, Partitioning and Knapsack Problem. This comparison not only lend strong support to the idea that cost distributions are highly characteristic for a problem but also implied classifications of basic types of cost distributions.

Before discussing the effects of cost distributions on randomized optimization algorithms, we addressed the question of the problem's difficulty. Recent developments in the context of NP-complete decision problems suggest concentrations of difficult cases in a small range of a so-called order parameter. This phenomenon of phase transitions gained enormous popularity in the last decade. However, as our experiments with the Asymmetric Traveling Salesman Problem showed, there is no phase transition of similarly distinct kind in optimization though areas of higher and lower difficulty are clearly to spot. The changes of difficulty—except for trivial cases—are however strongly depending on the algorithm used. These findings put attempts to proof the existence of a phase transition on the lines of the k-Satisfiability problem like undertaken in [KRHM95] in a different light. We concluded our assessment of difficulty with the introduction of probabilistic difficulty, a measure of difficulty that takes a problem's cost distribution into account.

After this, we turned our attention to probabilistic query optimization techniques and evolutionary computing. Firstly, we analyzed evolutionary algorithm not only against the background of query optimization but used the previously introduced classification of cost distributions to give a comprehensive assessment complemented with a case study to verify our results in practice. Our findings explain where these techniques fall short of what is to expect and where they turn out to be well-suited instruments for optimization. Secondly, we scrutinized randomized algorithms like Simulated Annealing, Simple Improvement, Iterative Improvement, Hill Climbing, random sampling, and Two-Phase Optimization. As opposed to earlier studies, we identified the single *principles* a particular techniques is composed of and studied the building blocks in isolation before assessing the compound method. That way, we were able to explain various effects observed previously, which was not fully understood in related work.

Piecing the parts together we finally presented an algorithm performing biased sampling using bottom-up random generation of plans with cost-bound pruning. These findings summarize our analysis best as "Good enough is easy".

9.2 Open Problems

Finally, some thoughts where to go from here. Each of the chapters suggests one or more directions of further research either concerning the practicability of the ideas presented or the transfer to other areas of combinatorial optimization.

We detailed the basic properties of cost distributions found in query optimization. Further differentiation of the extent of the skew—i.e., the concentration around the optimum—could be helpful to determine the difficulty of a query compared to others. Such an assessment could be used to decide on what optimization strategy to use or how to combine several different ones, and how much effort to put into the optimization. It would be particularly challenging to devise means to predict the shape given the declarative query and the usual database statistics only. First steps in this direction could include the investigation of incrementally insertion of additional single joins and analyzing their impact on the distribution's shape; a principle that would easily extend to complete sub-queries. In related work, cost models that take parallelism and main-memory resident base tables into account have been used to promote more sophisticated optimization techniques. While significantly more difficult to optimize with heuristics, these extended models do not appear to differ much from simpler models when addressed with blind search algorithms which raises hopes to unify some of these models.

The neighborhoods defined by transformation rules have a strong influence on the performance. But so far only little research has been devoted to investigate the implied topologies with the aim to generate landscapes

that are favorable to certain algorithms. First results in this field have been reported on by Spiliopoulou [Spi92]. Also the modification of the rule sets at run time in a way that it is able to adopt to the different areas of the search space could be an interesting target for further research.

Finally, the results of Chapter 8 offer the possibility for a combination of randomized and local exhaustive search. Using a framework like the MEMO structure but applying a very restricted set of transformations that fathom only the immediate vicinity of the initial plan together with a sampling phase that provides say 10 potential initial plans, the strengths of both approaches could be combined.

Bibliography

[ABCC98] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the Solution of Traveling Salesman Problems. *Documenta Mathematica, Extra Volume ICM*, pages 645–656, August 1998.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, MA, USA, 1995.

[AKK95] F. Andrés, F. Kwakkel, and M. L. Kersten. Calibration of a DBMS Cost Model with the Software Testpilot. In *Conf. on Information Systems and Management of Data*, pages 58–74, Bombay, India, 1995.

[Aro98] S. Arora. The Approximability of NP-hard Problems. In *Proc. of the Symposium on the Theory of Computing*, pages 337–348, Dallas, TX, USA, 1998.

[Bäc96] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, Oxford, 1996.

[Bat79] D. S. Batory. On Searching Transposed Files. *ACM Trans. on Database Systems*, 4(4):531–544, 1979.

[Bat86] D. S. Batory. Extensible Cost Models and Query Optimization in Genesis. *IEEE Data Engineering Bulletin*, 10(4):30–36, December 1986.

[Ber87] D. P. Bertsekas. *Dynamic Programming.* Prentice Hall, Englewood Cliffs, NJ, USA, 1987.

[BF97] E. Bertino and P. Foscoli. On Modeling Cost Functions for Object-Oriented Databases. *IEEE Trans. on Knowledge and Data Engineering*, 9(3):500–508, September 1997.

[BFI91] K. Bennett, M. C. Ferris, and Y. E. Ioannidis. A Genetic Algorithm for Database Query Optimization. In *Proc. Conf. on Genetic Programming*, pages 400–407, San Diego, CA, USA, July 1991.

[BK99] P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *The VLDB Journal*, 8(2):101–119, 1999.

[BMG93] J. A. Blakeley, W. J. McKenna, and G. Graefe. Experiences Building the Open OODB Query Optimizer. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 287–296, Washington, DC, USA, May 1993.

[BMK99] P. A. Boncz, S. Manegold, and M. L. Kersten. Database Architecture Optimized for the new Bottleneck: Memory Access. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 54–65, Edinburgh, UK, 1999.

[BMK00] P. A. Boncz, S. Manegold, and M. L. Kersten. What Happens During a Join. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, Cairo, Egypt, 2000. Accepted for publication.

[Boe96] K. D. Boese. *Models for Iterative Global Optimization.* PhD thesis, Computer Science Department, University of California at Los Angeles, Los Angeles, USA, 1996.

[Cha97] S. Chaudhuri. Query Optimization at the Crossroads. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, page 509, Tucson, AZ, USA, June 1997. Panel.

[CHY93] M.-S. Chen, H.-I. Hsiao, and P. S. Yu. Applying Hash Filters to Improving the Execution of Bushy Trees. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 505–516, Dublin, Ireland, September 1993.

[CK85] G. P. Copeland and S. Khoshafian. A Decomposition Storage Model. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 268–279, Austin, TX, USA, May 1985.

[CKT91] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Int'l. Joint Conference on Artificial Intelligence*, pages 331–337, Sydney, Australia, 1991. Morgan Kaufmann.

[CM95] S. Cluet and G. Moerkotte. On the Complexity of Generating Optimal Left-Deep Processing Trees with Cross Products. In *Proc. of the Int'l. Conf. on Database Theory*, pages 54–67, Prague, Czech Republic, January 1995.

[Cul98] J. C. Culberson. On the Futility of Blind Search: An Algorithmic View of "No Free Lunch". *Evolutionary Computation*, 6(2):109–128, April 1998.

[Eib96] A. E. Eiben. Evolutionary Exploration of Search Spaces. In Z. Ras and M. Michalewicz, editors, *Foundations of Intelligent Systems*, number 1079 in Lecture Notes in Computer Science, pages 178–188. Springer-Verlag, 1996.

[EM92] J. R. Evans and E. Minieka. *Optimization Algorithms for Networks and Graphs.* Marcel Dekker, Inc., New York, USA, 1992.

[EN94] E. Elmasri and S. B. Navathe. *Fundamentals of Database Sytems.* Benjamin/Cummings, Redwood City, CA, USA, 2nd edition, 1994.

[ERR94] A. E. Eiben, P.-E. Raué, and Zs. Ruttkay. Genetic Algorithms with Multi-parent Recombination. In *Parallel Problem Solving from Nature*, number 866 in Lecture Notes in Computer Science, pages 78–87. Springer-Verlag, 1994.

[FA85] Y. T. Fu and P. W. Anderson. Application of Statistical Mechanics to NP-complete Problems in Combinatorial Optimization. *Journal of Physics A*, 19:1605–1620, 1985.

[FJMO95] M. L. Fredman, D. S. Johnson, L. A. McGeoch, and G. Ostheimer. Data Structures for Traveling Salesmen. *Journal of Algorithms*, 18(3):432–479, 1995.

[Fre87] J.-C. Freytag. A Rule-Based View of Query Optimization. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 173–180, San Francisco, CA, USA, May 1987.

[GBC98] G. Graefe, R. Bunker, and S. Cooper. Hash Joins and Hash Teams in Microsoft SQL Server. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 86–97, New York, NY, USA, September 1998.

[GCD+94] G. Graefe, R. L. Cole, D. L. Davison, W. J. McKenna, and R. H. Wolniewicz. Extensible Query Optimization and Parallel Execution in Volcano. In J.-C. Freytag, G. Vossen, and D. Maier, editors, *Query Processing for Advanced Database Applications.* Morgan Kaufmann, San Mateo, CA, USA, 1994.

[GD87] G. Graefe and D. J. DeWitt. The EXODUS Optimizer Generator. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 160–172, San Francisco, CA, USA, May 1987.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman and Co., New York, 1979.

[GJR84] M. Grötschel, M. Jünger, and G. Reinelt. A Cutting Plane Algorithm for the Linear Ordering Problem. *Operations Research*, 32:1195–1220, 1984.

[Glo89] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[GLPK94] C. A. Galindo-Legaria, A. Pellenkoft, and M. L. Kersten. Fast, Randomized Join-Order Selection - Why Use Transformations? In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 85–95, Santiago, Chile, September 1994.

[GLPK95] C. A. Galindo-Legaria, A. Pellenkoft, and M. L. Kersten. Uniformly-distributed Random Generation of Join Orders. In *Proc. of the Int'l. Conf. on Database Theory*, pages 280–293, Prague, Czech Republic, January 1995.

[GM93] G. Graefe and W. J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proc. of the IEEE Int'l. Conf. on Data Engineering*, pages 209–218, Vienna, Austria, April 1993.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, USA, 1989.

[Gra93] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[Gra94] G. Graefe. Volcano, An Extensible and Parallel Dataflow Query Processing System. *IEEE Trans. on Knowledge and Data Engineering*, 6(1):120–135, February 1994.

[Gra99] G. Graefe. The Value of Merge-Join and Hash-Join in SQL Server. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 250–253, Edinburgh, UK, 1999.

[Haj88] B. Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research*, 13(2):311–329, May 1988.

[Hay97] B. Hayes. Can't Get No Satisfaction. *American Scientist*, 85(2):108–112, March 1997.

[HH87] B. A. Huberman and T. Hogg. Phase Transitions in Artificial Intelligence Systems. *Artificial Intelligence*, 33:155–171, 1987.

[IC91] Y. E. Ioannidis and S. Christodoulakis. On the Propagation of Errors in the Size of Join Results. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 268–277, Denver, CO, USA, May 1991.

[IF98] A. C. Ikeji and F. Fotouhi. Optimization of Constrained Queries with a Hybrid Genetic Algorithm. In *Proc. of the Int'l. Workshop on Database and Expert Systems Application*, number 1460 in Lecture Notes in Computer Science, pages 342–352, Vienna, Austria, August 1998. Springer-Verlag.

[IK84] T. Ibaraki and T. Kameda. Optimal Nesting for Computation N-Relational Joins. *ACM Trans. on Database Systems*, 9(3):482–502, September 1984.

[IK90] Y. E. Ioannidis and Y. C. Kang. Randomized Algorithms for Optimizing Large Join Queries. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 312–321, Atlantic City, NJ, USA, May 1990.

[IK91] Y. E. Ioannidis and Y. C. Kang. Left-Deep vs. Bushy Trees: An Analysis of Strategy Spaces and its Implications for Query Optimization. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 168–177, Denver, CO, USA, May 1991.

[INSS92] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric Query Processing. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 103–114, Vancouver, BC, Canada, August 1992.

[IW87] Y. E. Ioannidis and E. Wong. Query Optimization by Simulated Annealing. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 9–22, San Francisco, CA, USA, May 1987.

[JV86] R. Jonker and T. Volgenant. Improving the Hungarian Assignment Algorithm. *Operations Research Letters*, 5:171–175, 1986.

[JVV86] M. R. Jerrum, L. G. Valiant, and U. V. Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, 43(2–3):169–188, 1986.

[Kan91] Y. C. Kang. *Randomized Algorithms for Query Optimization.* PhD thesis, University of Wisconsin-Madison, Computer Science Department, 1991.

[Kau93] S. A. Kauffman. *The Origins of Order.* Oxford University Press, New York, Oxford, 1993.

[KBZ86] R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of Nonrecursive Queries. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 128–137, Kyoto, Japan, August 1986.

[KE96] A. Kemper and A. Eickler. *Datenbanksysteme – Eine Einführung.* R. Oldenbourg Verlag, München, Wien, 1996.

[KGV83] S. Kirkpatrick, J. C. D. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[KKLO86] N. K. Karmarkar, R. M. Karp, G. S. Lueker, and A. M. Odlyzko. Probabilistic Analysis of Optimum Partitioning. *Journal of Applied Probability*, 23:626–645, 1986.

[KM94] A. Kemper and G. Moerkotte. *Object-Oriented Database Management: Applications in Engineering and Computer Science.* Prentice Hall, Englewood Cliffs, NJ, USA, 1994.

[KMPS94] A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. Optimizing Disjunctive Queries and Expensive Predicates. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 336–347, Minneapolis, MN, USA, May 1994.

[Koz91] J. R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, USA, 1991.

[KRHM95] B. König-Ries, S. Helmer, and G. Moerkotte. An Experimental Study on the Complexity of Left-deep Join Ordering Problems for Cyclic Queries. Technical Report 95-4, RWTH Aachen, Aachen, Germany, 1995.

[KS91] H. Korth and A. Silberschatz. *Database Systems Concepts.* McGraw-Hill, Inc., New York, San Francisco, Washington, DC, USA, 1991.

[Kuh55] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quaterly*, 1:83–97, 1955.

[Lan95] K. J. Lang. Hill Climbing Beats Genetic Search on a Boolean Circuit Synthesis Problem of Koza's. In *Proc. of the Int'l. Conf. on Machine Learning*, pages 340–343, Tahoe City, CA, USA, 1995.

[LD88] J. Lam and J. Delosme. Simulated Annealing: A Fast Heuristic for Some Generic Layout Problems. In *Proc. of the Int'l. Conf. on Genetic Algorithms*, pages 495–498, 1988.

[Lit63] J. D. C. Little. An Algorithm for the Traveling Salesman Problem. *ORSA Journal on Computing*, 11:972–989, 1963.

[LN96] S. Listgarten and M.-A. Neimat. Modelling Costs for a MM-DBMS. In *Real-Time Databases*, pages 72–78, Newport Beach, CA, USA, March 1996.

[LVZ93] R. S. G. Lanzelotte, P. Valduriez, and M. Zaït. On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 493–504, Dublin, Ireland, August 1993.

[McK93] W. J. McKenna. *Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer Generator.* PhD thesis, University of Colorado, Boulder, CO, USA, 1993.

[MdWS91] B. Manderick, M. de Weger, and P. Spiessens. The Genetic Algorithm and the Structure of the Fitness Landscape. In *Proc. of the Int'l. Conf. on Genetic Algorithms*, pages 1143–1150, San Diego, CA, USA, 1991.

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1996.

[MMS94] T. Morzy, M. Matysiak, and S. Salza. Tabu Search Optimization of Large Join Queries. In *Proc. of the Int'l. Conf. on Extending Database Technology*, volume 779 of *Lecture Notes in Computer Science*, pages 309–322, Cambridge, UK, 1994.

[MOW97] S. Manegold, J. K. Obermaier, and F. Waas. Load Balanced Query Evaluation in Shared-Everything Environments. In *Proc. of the European Conf. on Parallel Processing*, pages 1117–1124, Passau, Germany, August 1997.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Computational Physics*, 21(6):1087–1092, 1953.

[MT87] S. Martello and P. Toth. Algorithms for Knapsack Problems. *Annals of Discrete Mathematics*, 31:213–257, 1987.

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley and Sons, New York, USA, 1990.

[MW92] K. Mathias and D. Whitley. Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem. In *Parallel Problem Solving from Nature*, pages 219–228. Elsevier Science Publishers, 1992.

[MW96] W. G. Macready and D. H. Wolpert. No Free Lunch Theorems for Search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, NM, USA, February 1996.

[MW98] S. Manegold and F. Waas. Thinking Big in a Small World—Efficient Parallel Query Execution on Small-scale SMPs. In *High Performance Computing Systems and Applications*, pages 133–146. Kluwer Academic Publishers, Dordrecht, The Netherlands, May 1998.

[MW99] S. Manegold and F. Waas. Integrating I/O processing and Transparent Parallelism—Toward Comprehensive Query Execution in Parallel Database Systems. In A. Dogac, M. T. Özsu, and O. Ulusoy, editors, *Current Trends in Database Systems*, pages 130–152. Idea Group Publisher, January 1999.

[MWK98] S. Manegold, F. Waas, and M. L. Kersten. On Optimal Pipeline Processing in Parallel Query Optimization. In *Int'l. Conf. on Management of Data*, pages 217–235, Hyderabad, India, December 1998.

[MZK+99] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining Computational Complexity from Characteristic 'Phase Transitions'. *Nature*, 400:133–137, July 1999.

[NK97a] K. Nafjan and J. M. Kerridge. Large Join Order Optimzation on Parallel Shared-Nothing Database Machines Using Genetic Algorithms. In *Proc. of the European Conf. on Parallel Processing*, pages 1159–1163, Passau, Germany, August 1997.

[NK97b] Y. Nagata and S. Kobayashi. Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem. In *Proc. of the Int'l. Conf. on Genetic Algorithms*, pages 450–457, East Lansing, MI, USA, July 1997.

[NP98] E. Nonas and A. Poulovassilis. Optimization of Active Rule Agents Using a Genetic Algorithm Apporach. In *Proc. of the Int'l. Workshop on Database and Expert Systems Application*, number 1460 in Lecture Notes in Computer Science, pages 332–341, Vienna, Austria, August 1998. Springer-Verlag.

[OL90] K. Ono and G. M. Lohman. Measuring the Complexity of Join Enumaration in Query Optimization. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 314–325, Brisbane, Australia, August 1990.

[OW96] J. K. Obermaier and F. Waas. Dynamic Aspects of Query Processing in Parallel Database Systems. In *Proc. of the Workshop on Information Technologies and Systems*, pages 223–232, Cleveland, OH, USA, December 1996.

[Pel97] A. Pellenkoft. *Probabilistic and Transformation based Query Optimization.* PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 1997.

[PGLK97a] A. Pellenkoft, C. A. Galindo-Legaria, and M. L. Kersten. Duplicate-free Generation of Alternatives in Transformation-based Optimizers. In *Proc. Int'l. Conference on Database Systems for Advanced Applications*, pages 117–123, Melbourne, Australia, April 1997.

[PGLK97b] A. Pellenkoft, C. A. Galindo-Legaria, and M. L. Kersten. The Complexity of Transformation-Based Join Enumeration. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 306–315, Athens, Greece, September 1997.

[Pis95] D. Pisinger. *Algorithms for Knapsack Problems.* PhD thesis, University of Copenhagen, Copenhagen, Denmark, 1995.

[PR87] M. W. Padberg and G. Rinaldi. Optimization of a 532 City Symmetric Traveling Salesman Problem by Branch and Cut. *Operations Research Letters*, 6:1–7, 1987.

[PR91] M. W. Padberg and G. Rinaldi. A Branch and Cut Algorithm for the Resolution of Large-scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33(1):60–100, 1991.

[Pur83] P. W. Purdom. Search Rearrangement Backtracking and Polynomial Average Time. *Artificial Intelligence*, 21(1&2):117–134, 1983.

[Rei91] G. Reinelt. TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[Rud92] G. Rudolph. Parallel Approaches to Stochastic Global Optimization. In *Proc. of the European Workshop on Parallel Computing*, pages 236–247, Barcelona, Spain, March 1992.

[SAC+79] P. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 23–34, Boston, MA, USA, May 1979.

[SG88] A. Swami and A. Gupta. Optimizing Large Join Queries. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 8–17, Chicago, IL, USA, June 1988.

[SHC96] M. Spiliopoulou, M. Hatzopoulos, and Y. Cotronis. Parallel Optimization of Large Join Queries with Set Operators and Aggregates in a Parallel Environment Supporting Pipeline. *IEEE Trans. on Knowledge and Data Engineering*, 8(3):429–445, June 1996.

[SI93] A. Swami and B. R. Iyer. A Polynomial Time Algorithm for Optimizing Join Queries. In *Proc. of the IEEE Int'l. Conf. on Data Engineering*, pages 345–354, Vienna, Austria, April 1993.

[Sin92] A. Sinclair. *Algorithms for Random Generation & Counting.* Birkhäuser, Boston, Basel, Berlin, 1992.

[Slu98] D. Slutz. Massive Stochastic Testing of SQL. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 618–622, New York, NY, USA, September 1998.

[SM97] W. Scheufele and G. Moerkotte. On the Complexity of Generating Optimal Plans with Cross Products. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 238–248, Tucson, AZ, USA, May 1997.

[SMK93] M. Steinbrunn, G. Moerkotte, and A. Kemper. Optimizing Join Orders. Technical Report MIP9307, Universität Passau, Passau, Germany, 1993.

[SMK97] M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and Randomized Optimization for the Join Ordering Problem. *The VLDB Journal*, 6(3):191–208, August 1997.

[Spi92] M. Spiliopoulou. *Parallel Optimisation and Execution of Relational Queries in an Environment Supporting Parallelism and Pipeline.* PhD thesis, University of Athens, Athens, Greece, 1992. (in Greek).

[SPMK95] M. Steinbrunn, K. Peithner, G. Moerkotte, and A. Kemper. Bypassing Joins in Disjunctive Queries. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 228–238, Zurich, Switzerland, September 1995.

[SR91] Y. G. Saab and V. B. Rao. Combinatorial Optimziation by Stochastic Evolution. *IEEE Trans. on Computer-Aided Design*, 10(4):525–535, April 1991.

[SS96] M. Stillger and M. Spiliopoulou. Genetic Programming in Database Query Optimization. In *Proc. Conf. on Genetic Programming*, pages 388–393, Stanford, CA, USA, July 1996.

[Ste96] M. Steinbrunn. *Heuristic and Randomised Optimisation Techniques in Object-Oriented Database.* DISDBIS. infix, Sankt Augustin, Germany, 1996.

[Swa89a] A. Swami. *Optimization of Large Join Queries.* PhD thesis, Stanford University, Computer Science Department, 1989.

[Swa89b] A. Swami. Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 367–376, Portland, OR, USA, June 1989.

[Swa91] A. Swami. Distributions of Query Plan Costs for Large Join Queries. Technical Report RJ7908, IBM Almaden Research Center, San Jose, CA, USA, January 1991.

[Tra99] Transaction Processing Performance Council, San Jose, CA, USA. *TPC Benchmark H (Ad-hoc, Decision Support),* Revision 1.2.1 1999.

[Tur88] J. S. Turner. Almost All k-Colorable Graphs are Easy to Color. *Journal of Algorithms*, 9(1):63–82, March 1988.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, New York, USA, 1989.

[VK83] M. Vecchi and S. Kirkpatrick. Global Wiring by Simulated Annealing. *IEEE Trans. on Computer-Aided Design*, pages 215–222, October 1983.

[VM96] B. Vance and D. Maier. Rapid Bushy Join-order Optimization with Cartesian Products. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 35–46, Montreal, Canada, June 1996.

[Waa99a] F. Waas. Cost Distributions in Symmetric Euclidean Traveling Salesman Problems—A Supplement to TSPLIB. Technical Report INS-R9911, CWI, Amsterdam, The Netherlands, September 1999.

[Waa99b] F. Waas. Handling Non-deterministic Data Availability in Parallel Query Execution. In *Int'l. Workshop on Parallel and Distributed Databases*, pages 61–65, Florence, Italy, September 1999.

[Waa00] F. Waas. Extending Iterators for Advanced Query Execution. In *Australasian Database Conference*, pages 135–139, Canberra, Australia, January 2000. IEEE Computer Society Press.

[WGL00a] F. Waas and C. A. Galindo-Legaria. Counting, Enumerating and Sampling of Execution Plans in a Cost-Based Query Optimizer. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 499–509, Dallas, TX, USA, May 2000.

[WGL00b] F. Waas and C. A. Galindo-Legaria. The Effect of Cost Distributions on Genetic Algorithms. Submitted for publication, October 2000.

[WM97] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Trans. on Evolutionary Computing*, 1(1):67–82, April 1997.

[WP00] F. Waas and A. Pellenkoft. Join Order Selection — Good Enough is Easy. In *Proc. of the British National Conference on Databases*, Lecture Notes in Computer Science, pages 51–67, Exeter, United Kingdom, July 2000.

[WRE+98] P. Watson, C. Ross, V. Eisele, J. Denton, J. Bins, C. Guerra, D. Whitley, and A. Howe. The Traveling Salesrep Problem, Edge Assembly Crossover, and 2-opt. In *Parallel Problem Solving from Nature*, number 1498 in Lecture Notes in Computer Science, pages 78–87. Springer-Verlag, 1998.

[Yao77] S. B. Yao. An Attribute Based Model for Database Access Cost Analysis. *ACM Trans. on Database Systems*, 2(1):45–67, 1977.

[ZL96] Q. Zhu and P.-Å. Larson. Developing Regression Cost Models for Multidatabase Systems. In *Proc. of the Int'l. Conf. on Parallel and Distributed Information Systems*, pages 220–231, Miami, FL, USA, December 1996.

[ZP94] W. Zhang and J. C. Pemberton. Epsilon-Transformation: Exploiting Phase Transitions to Solve Combinatorial Optimization Problems—Initial Results. In *National Conference on Artificial Intelligence*, pages 895–900, Seattle, WA, USA, 1994.

[ZT99] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Trans. on Evolutionary Computing*, pages 257–271, November 1999.

Appendix

The tables 9.2 through 9.4 summarize the characteristics of all instances of symmetric Traveling Salesman Problems, as given in the TSPLIB [Rei91] (see also Section 4.2.2). The table below explains the symbols used.

n	problem size
l_{min}	length of optimal tour
$l_{min}^{(G)}$	length of shortest tour found by greedy heuristic
$l_{max}^{(G)}$	length of longest tour found by greedy heuristic
$l_{min}^{(S)}$	length of shortest tour found by sampling
$l_{max}^{(S)}$	length of longest tour found by sampling
μ_s	mean of sampled distribution
μ_a	mean of approximated distribution
κ	Kulback-Leibler Divergence of sampled and approximated distribution centered on resp. means

Table 9.1. Identifiers used in the experiments

Name	n	l_{min}	$l_{min}^{(G)}$	$l_{min}^{(S)}$	$l_{max}^{(S)}$	$l_{max}^{(G)}$	μ_s	μ_a	$\frac{\lvert\mu_a-\mu_s\rvert}{l_{max}^{(G)}-l_{min}}$	κ
a280	280	2579	3073	29214	38487	49989	34110	33975	0.00375	0.009
berlin52	52	7542	9535	22177	36237	38810	29915	29287	0.02183	0.161
bier127	127	118282	123924	526794	712190	819758	628963	625653	0.00556	0.238
brd14051	14051	469445	580252	41122399	42785106	58252876	41970412	41938841	0.00075	0.028
ch130	130	6110	7103	37685	53831	65933	46317	45928	0.00814	0.026
ch150	150	6528	8115	44939	63102	77463	53895	53459	0.00769	0.019
d1291	1291	50801	61636	1638848	1827620	2478844	1729060	1726317	0.00154	0.038
d15112	15112	1573152	1949461	131634854	136413750	192411817	134001587	133960947	0.00030	0.020
d1655	1655	62128	74144	2069720	2275126	3166964	2173713	2178425	0.00213	0.018
d18512	18512	645300	797063	58487256	60308007	86367860	59440557	59385279	0.00093	0.017
d198	198	15780	18870	153231	219390	256990	190630	190202	0.00210	0.158
d2103	2103	80450	90910	3102263	3396147	4788346	3250935	3255869	0.00149	0.013
d493	493	35002	42619	405985	492097	605848	449546	449000	0.00119	0.104
d657	657	48912	61904	790093	920312	1238686	854986	854343	0.00074	0.024
eil101	101	-	773	2754	4027	4862	3429	3404	0.00620	0.033
eil51	51	426	493	1231	2045	2297	1654	1622	0.01970	0.041
eil76	76	538	676	2007	3034	3532	2525	2489	0.01439	0.042
fl1400	1400	20127	27889	1567938	1809671	2678912	1688494	1683131	0.00299	0.001
fl1577	1577	22249	27313	1275678	1422292	1857672	1355282	1355336	0.00004	0.059
fl3795	3795	28772	38762	3460116	3697017	4997885	3581035	3576493	0.00124	0.036
fl417	417	11861	16417	424702	559274	776919	496074	494603	0.00268	0.002
fnl4461	4461	182566	226480	8072588	8610931	12073014	8329970	8337060	0.00084	0.018
gil262	262	2378	2987	23462	29939	38492	26707	26586	0.00439	0.017
kroA100	100	21282	27989	131103	210574	250472	171073	169526	0.00816	0.013
kroA150	150	26524	33975	204135	305395	376559	257602	256089	0.00542	0.013
kroA200	200	29368	37099	281968	392788	505065	340251	339576	0.00185	0.010

Table 9.2. Characteristics of cost distributions *(continued on next page)*

Name	n	l_{min}	$l_{min}^{(G)}$	$l_{min}^{(S)}$	$l_{max}^{(S)}$	$l_{max}^{(G)}$	μ_s	μ_a	$\frac{\lvert\mu_a-\mu_s\rvert}{l_{max}^{(G)}-l_{min}}$	κ
kroB100	100	22141	27239	128169	205187	242863	168771	166932	0.01003	0.017
kroB150	150	26130	34543	202216	304182	381542	256728	254960	0.00635	0.007
kroB200	200	29437	39987	274642	386744	487956	332843	330884	0.00547	0.014
kroC100	100	20749	25661	128107	205994	251237	170080	168252	0.00985	0.010
kroD100	100	21294	26698	124642	196602	234766	163101	162075	0.00584	0.021
kroE100	100	22068	30031	133731	215369	254996	173214	171603	0.00832	0.013
lin105	105	14379	18916	94174	149640	176524	123636	122709	0.00684	0.023
lin318	318	42029	53473	519483	652452	852712	587981	587219	0.00125	0.017
nrw1379	1379	56638	70757	1342961	1511475	2086720	1423600	1419946	0.00251	0.016
p654	654	34643	45660	1819275	2260466	3256192	2038096	2041650	0.00160	0.001
pcb1173	1173	56892	71992	1326835	1490734	2058498	1410067	1409785	0.00020	0.018
pcb3038	3038	137694	175711	5203088	5607003	7938318	5402855	5397693	0.00094	0.015
pcb442	442	50778	61612	698796	851360	1117968	772598	772480	0.00015	0.016
pr1002	1002	259045	325813	6019471	6862385	9423294	6448609	6445027	0.00054	0.014
pr107	107	44303	58968	429065	727084	931825	578277	571726	0.00957	0.001
pr124	124	59030	78901	544860	829215	997362	697299	690430	0.00890	0.012
pr136	136	96772	121920	668103	967825	1236896	826037	819891	0.00704	0.008
pr144	144	58537	70353	670724	941562	1180602	812094	806760	0.00603	0.012
pr152	152	73682	83684	845406	1218702	1483775	1051061	1040831	0.00892	0.028
pr226	226	80369	99020	1429063	1927231	2518346	1695745	1687439	0.00449	0.006
pr2392	2392	378032	470864	14623016	15923458	22386371	15249052	15229018	0.00129	0.013
pr264	264	49135	59396	925154	1300008	1811672	1121641	1120748	0.00071	0.002
pr299	299	48191	60780	644017	858433	1118841	759629	757736	0.00233	0.015
pr439	439	107217	133172	1709220	2082403	2613077	1904460	1898842	0.00284	0.073
pr76	76	108159	140909	441387	692833	793497	574464	565617	0.01510	0.062
rat195	195	2323	2942	19007	26199	33363	22725	22605	0.00502	0.016

Table 9.3. Characteristics of cost distributions *(continued on next page)*

Name	n	l_{min}	$l_{min}^{(G)}$	$l_{min}^{(S)}$	$l_{max}^{(S)}$	$l_{max}^{(G)}$	μ_s	μ_a	$\frac{\lvert\mu_a-\mu_s\rvert}{l_{max}^{(G)}-l_{min}}$	κ
rat575	575	6773	8627	102328	124133	167079	113683	113818	0.00115	0.016
rat783	783	8806	11245	163806	194090	263754	179486	179189	0.00160	0.015
rat99	99	1211	1683	6499	10215	12267	8417	8348	0.00765	0.019
rd100	100	7910	10088	43190	66740	80216	55568	55089	0.00813	0.016
rd400	400	15281	18590	190061	231491	309171	211545	211845	0.00139	0.015
rl11849	11849	923368	1117193	85311772	88677680	125989180	86975101	86866566	0.00124	0.027
rl1304	1304	252948	331303	8849710	9919445	13597353	9375766	9382345	0.00068	0.027
rl1323	1323	270199	332641	9229210	10404085	14180847	9794996	9806134	0.00110	0.018
rl1889	1889	316536	394917	14035006	15537183	21815113	14799860	14769477	0.00200	0.014
rl5915	5915	565530	703619	41464009	43597251	61553339	42556375	42615177	0.00137	0.022
rl5934	5934	556045	698687	41046185	43310499	61083560	42199303	42297473	0.00230	0.030
st70	70	675	738	2732	4455	5209	3659	3601	0.01531	0.021
ts225	225	126643	160490	1384995	1808915	2319813	1592997	1585149	0.00466	0.017
tsp225	225	3916	4565	35510	46883	58637	41291	41094	0.00458	0.032
u1060	1060	224094	289197	6243878	7296956	9920865	6758750	6745626	0.00185	0.014
u1432	1432	152970	187720	3711202	4146358	5808347	3945957	3943095	0.00072	0.011
u159	159	42080	50995	373532	528962	651442	449604	446575	0.00621	0.010
u1817	1817	57201	69758	2007988	2229073	3104248	2118896	2119209	0.00014	0.014
u2152	2152	64253	79201	2414892	2657317	3720928	2532561	2534222	0.00064	0.014
u2319	2319	234256	277770	5739187	6242399	8776096	5998456	5980928	0.00292	0.016
u574	574	36905	46849	615516	738068	989276	680730	678708	0.00288	0.019
u724	724	41910	53169	796280	947790	1276491	872599	871433	0.00129	0.016
usa13509	13509	19982859	25165110	2104303273	2195633826	2978881603	2153474196	2158344634	0.00224	0.106
vm1084	1084	239297	297389	8001876	9185708	12617221	8571921	8573635	0.00019	0.014
vm1748	1748	336556	420377	14128926	15738254	21802682	14942427	14938880	0.00023	0.013

Table 9.4. Characteristics of cost distributions

List of Publications

[1] F. Waas and C. A. Galindo-Legaria. The Effect of Cost Distributions on Genetic Algorithms. Submitted for publication, October 2000.

[2] F. Waas and M. L. Kersten. Memory Aware Query Scheduling in a Database Cluster. Submitted for publication, June 2000.

[3] F. Waas and A. Pellenkoft. Join Order Selection — Good Enough is Easy. In *Proc. of the British National Conference on Databases*, Lecture Notes in Computer Science, pages 51–67, Exeter, United Kingdom, July 2000.

[4] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *International Workshop on the Web and Databases (In conjunction with ACM SIGMOD)*, pages 47–52, Dallas, TX, USA, May 2000.

[5] F. Waas and C. A. Galindo-Legaria. Counting, Enumerating and Sampling of Execution Plans in a Cost-Based Query Optimizer. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 499–509, Dallas, TX, USA, May 2000.

[6] F. Waas. Extending Iterators for Advanced Query Execution. In *Australasian Database Conference*, pages 135–139, Canberra, Australia, January 2000. IEEE Computer Society Press.

[7] F. Waas and C. A. Galindo-Legaria. A Method for Counting, Enumerating and Sampling of Execution Plans in a Cost-Based Query Optimizer. MS-D 138320.1, SLWK-D 777.309US1, January 2000.

[8] F. Waas. Handling Non-deterministic Data Availability in Parallel Query Execution. In *Int'l. Workshop on Parallel and Distributed Databases*, pages 61–65, Florence, Italy, September 1999.

[9] S. Manegold and F. Waas. Integrating I/O processing and Transparent Parallelism—Toward Comprehensive Query Execution in Parallel Database Systems. In A. Dogac, M. T. Özsu, and O. Ulusoy, editors, *Current Trends in Database Systems*, pages 130–152. Idea Group Publisher, January 1999.

[10] S. Manegold, F. Waas, and M. L. Kersten. On Optimal Pipeline Processing in Parallel Query Optimization. In *Int'l. Conf. on Management of Data*, pages 217–235, Hyderabad, India, December 1998.

[11] S. Manegold and F. Waas. Thinking Big in a Small World—Efficient Parallel Query Execution on Small-scale SMPs. In *High Performance Computing Systems and Applications*, pages 133–146. Kluwer Academic Publishers, Dordrecht, The Netherlands, May 1998.

[12] S. Manegold, F. Waas, and D. Gudlat. *In Quest of the Bottleneck - Monitoring Parallel Database Systems*, volume 1332 of *Lecture Notes in Computer Science*, pages 277–284. Springer-Verlag, Berlin, New York, etc., November 1997.

[13] S. Manegold, J. K. Obermaier, and F. Waas. Load Balanced Query Evaluation in Shared-Everything Environments. In *Proc. of the European Conf. on Parallel Processing*, pages 1117–1124, Passau, Germany, August 1997.

[14] J. K. Obermaier and F. Waas. Dynamic Aspects of Query Processing in Parallel Database Systems. In *Proc. of the Workshop on Information Technologies and Systems*, pages 223–232, Cleveland, OH, USA, December 1996.

About the Author

After graduating from Passau University in 1995, Florian Waas joined the database research group at Humboldt-University Berlin for two years. Since April 1997 he has been with the database research group of CWI Amsterdam, the Dutch national research laboratory for mathematics and computer science. During this time he also spent several months with Microsoft's SQL Server group in Redmond and the database research group at Bologna University.

His research revolves primarily around query optimization and query execution in relational database systems including parallel and distributed systems. Further areas of interest include processing of semi-structured data and multi-media databases.

Index