Database Query Optimisation Based on Measures of Regret

Khaled Hamed Alyoubi

A Dissertation Presented for the Degree of
Doctor of Philosophy

Department of Computer Science and Information Systems
Birkbeck, University of London
London, UK

August 2016

Declaration

This thesis is the result of my own work, except where explicitly acknowledged in the text.

Abstract

The query optimiser in a database management system (DBMS) is responsible for finding a good order in which to execute the operators in a given query. However, in practice the query optimiser does not usually guarantee to find the *best* plan. This is often due to the non-availability of precise statistical data or inaccurate assumptions made by the optimiser. In this thesis we propose a robust approach to logical query optimisation that takes into account the unreliability in database statistics during the optimisation process. In particular, we study the ordering problem for selection operators and for join operators, where selectivities are modelled as *intervals* rather than exact values. As a measure of optimality, we use a concept from decision theory called *minmax regret optimisation* (MRO).

When using interval selectivities, the decision problem for selection operator ordering turns out to be NP-hard. After investigating properties of the problem and identifying special cases which can be solved in polynomial time, we develop a novel heuristic for solving the general selection ordering problem in polynomial time. Experimental evaluation of the heuristic using synthetic data, the Star Schema Benchmark and real-world data sets shows that it outperforms other heuristics (which take an optimistic, pessimistic or midpoint approach) and also produces plans whose regret is on average very close to optimal.

The general join ordering problem is known to be NP-hard, even for exact selectivities. So, for interval selectivities, we restrict our investigation to sets of join operators which form a chain and to plans that correspond to left-deep join trees. We investigate properties of the problem and use these, along with ideas from the selection ordering heuristic and other algorithms in the literature, to develop a polynomial-time heuristic tailored for the join ordering problem. Experimental evaluation of the heuristic shows that, once again, it performs better than the optimistic, pessimistic and midpoint heuristics. In addition, the results show that the heuristic produces plans whose regret is on average even closer to the optimal than for selection ordering.

Dedicated to

my dear parents and grandmother,

my wife and daughter,

and my brothers

Acknowledgements

First and foremost I would like to express my sincere gratitude to my GOD for all blessings and guidance in all aspects of my life.

I would like to take this opportunity to thank all the people who provided me with any kind of support throughout my PhD. Without those special people, it would not have been possible for this work to come to light.

It is a privilege for me to work with a wonderful supervision team. I would like to express my utmost appreciation and gratitude to my supervisor and mentor Prof. Peter Wood. Thanks for all your guidance, support, hard work and patience with me. It was an absolute honour to learn and work with you. I would also like to express my special appreciation to my second supervisor, Dr. Sven Helmer, for all his constructive advise and profound support. You have been a tremendous mentor throughout my PhD.

To my dear parents and grandmother who have been a beacon of light in every stage of my life, I dedicate this work to you. I am grateful for all your heartfelt prayers and unconditional love. This dedication is also extended to my beloved wife for all her support during the past years. You were the reason that gave me strength to complete this work. A special thanks to the joy of my life, my daughter Sarah. You always give me strength with your shining smile. My deep gratitude to my brothers: Ahmad, Ibraheem and Abdulraheem. Thanks for all your supports and help. You have always been there for me all the time.

I am very grateful to King Abdulaziz University for giving me this opportunity and funding my PhD research. Special thanks to the Dean of the Faculty of Computing and Information Technology and Vice Dean for Graduate Studies and Research, as well as the Head of the Information Systems Department, for their rec-

ommendations and full support which have greatly facilitated my scholarship and PhD research.

Moreover, special acknowledgments go to all the members of the Department of Computer Science and Information Systems here at Birkbeck, including all faculty and staff members as well as all colleagues in the London Knowledge Lab (LKL) who make our department an excellent research environment. Among those, I must thank my colleagues Mohammad, Muawya and Godfried for the constructive conversation. Special thanks to Mohammad for providing me with the Enron data set. I would like to thank every member of LKL for making my days in the office enjoyable. Furthermore, I would like to thank my friends Alaa and Adil Khadidos for all the times we spent together in the UK. This gratitude extends to Abdulrahman Alshareef, Bander Al-Zahrani and Hani Sindi. Your enthusiasm and energy have encouraged me a lot.

Last, but not least, thanks to anyone else who helped me in any way during my PhD research. I will always remember you for the rest of my life.

Publications

Publications relating to the thesis:

- Khaled H. Alyoubi, Sven Helmer, and Peter T. Wood. Ordering selection operators under partial ignorance. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 310–317, New York, NY, USA, 2015. ACM.

- Khaled H. Alyoubi, Sven Helmer, and Peter T. Wood. Ordering selection operators using the minmax regret rule. *CoRR*, abs/1507.08257, 2015.

- Khaled H. Alyoubi, Sven Helmer, and Peter T. Wood. Query optimisation based on measures of regret. In *Proceedings of the 7th Saudi Students Conference in the UK (SSC)*, Edinburgh, United Kingdom, 2014. SSC.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Queries in database systems are usually submitted in a declarative format. In other words, users describe what they want via some query language such as SQL, but do not specify how the database management system (DBMS) should evaluate the query. Therefore, the submitted query needs to go through various steps and transformations before the DBMS evaluates the query. First, the query is parsed and its syntax validated [47]. Then an internal representation (e.g. relational algebra) of the query is generated. After that logical query optimisation is performed. The logical query optimisation phase evaluates many equivalent plans in order to find a suitable one [38, 85]. Data can be accessed in a variety of ways depending on the available access paths and indices for example [28]. Therefore, physical query optimisation is performed next in order to find a physical execution plan which will be used at run-time to retrieve the answer of the query [38, 52]. Our work in this thesis takes place at the logical query optimisation stage.

At logical query optimisation, the optimiser translates a user's query to a logical plan. Usually there are many equivalent logical plans for a given query [106], so the optimiser compares them with the aim to find the best plan. An optimiser typically uses statistical data collected by the DBMS when comparing equivalent plans [64]. However, in practice this statistical information may not be accurate or even available at optimisation time [31]. Examples of these situations are when sys-

tems are confronted with skewed or very unevenly distributed data values, correlated predicates or when working in highly dynamic environments [31, 68, 79].

There are different causes for inaccuracy and many challenges for DBMSs to provide or maintain accurate statistical information. In some cases, the statistical information is out-dated due to infrequent updates [46, 64]. Some DBMSs do not update statistical information automatically and rely on database administrators to trigger an update command [108]. The inaccuracy sometimes results from the source of the information, such as statistics based on user feedback, transmission noise or delay/error in processing transactions [18, 74, 89]. Some statistics are derived based on inaccurate assumptions, such as that the data is uniformly distributed, the values of attributes are independent, or the selectivity of a predicate is always the inverse of the number of distinct values of the attribute involved [12, 46, 47, 64, 76, 113]. Moreover, some information may not be available at optimisation time, such as for user-defined queries or when information is related to the run-time environment [46, 68].

The inaccuracy of statistical data influences the quality of plans chosen by the optimiser [64, 76]. This may lead to suboptimal execution plans which can be orders of magnitude worse than the optimal plan in practice [46, 99]. One of the major challenges for query optimisers is error propagation, where inaccurate estimation of parameters required to evaluate operators in a query has a transitive effect on the subsequent operators, ultimately resulting in inefficient query evaluation [64]. The problem becomes much harder when the optimiser deals with expensive predicates or complex queries (e.g. subqueries) [61]. Consequently, an optimiser should take the unreliability of statistical data into consideration during the optimisation process. Moreover, instead of striving for the optimal plan, an optimiser should try to avoid bad plans based on unreliable statistical data. Numerous studies have been conducted to improve database query optimisation [18, 23, 39, 63, 68], although fewer studies deal with query optimisation in the presence of inaccuracy [31, 51, 55, 76].

In this thesis, we assume that the statistical data used by the optimiser is not known exactly, but instead is known to fall within intervals. Specifically, we assume that the selectivity of each selection and join operator falls within the interval $[0, 1]$.

The use of intervals has been proposed previously in the database literature. For example, a recent study uses intervals to bound error estimation of selectivities in a histogram in order to improve the quality of optimisers [86]. Intervals have also been used to model the level of uncertainty in estimated statistics in a DBMS [18]. These approaches, and others, will be discussed in more detail in Section 2.6.

As an example of a situation in which interval selectivities may arise, let us consider a database in which equi-height histograms of attribute values are maintained (more discussion about histograms will be provided in Section 2.3). In general, histograms approximate the data distribution of attribute values [66, 99]. This can be done by dividing the values for an attribute into k buckets/ranges and counting the frequency of tuples that fall into each range [28]. The aim in equi-height histograms is that each range has same number of tuples [99]. To build an equi-height histogram, the values are sorted in ascending order first [96]. Then $k + 1$ step values are chosen to specify the range of values for each of the k buckets, such that each bucket has the same number of tuples. Typically, $Step(0)$ and $Step(k)$ indicate the smallest and largest value for the attribute respectively. An advantage of this setting is that we know that 100% of tuples satisfy the predicate $attribute \leq Step(k)$, while $(\frac{i}{k} * 100)\%$ of tuples satisfy $attribute < Step(i)$, for $1 \leq i < k$. More importantly, using equi-height histograms, the upper and lower selectivity can be estimated fairly well and quickly [96]. Example 1.1.1 below, adapted from [96], illustrates with concrete values the calculation of selectivity bounds for a given predicate.

Example 1.1.1 Consider a histogram for an `age` attribute, with 10 buckets (i.e. $k = 10$) as shown in Table 1.1. Assume that the attribute's value ranges from 10 to 60. From Table 1.1 we notice that $Step(0) = 10$ and $Step(10) = 60$ since, the values 10 and 60 are the smallest and largest age values stored for the `age` attribute. The selectivity interval for the predicate `age` < 39, for example, can be calculated as follows.

Piatetsky-Shapiro and Connell show that in an equi-height histogram the selectivity bounds for any predicate $attribute < x$, where x falls between $Step(i)$ and

step #	attribute value	total % of tuples
0	10	0 %
1	19	10 %
2	21	20 %
3	22	30 %
4	23	40 %
5	28	50 %
6	32	60 %
7	35	70 %
8	37	80 %
9	44	90 %
10	60	100 %

Table 1.1: Histogram for attribute `age` in Example 1.1.1.

$Step(i+1)$, can be found as follows [96]:

$$\frac{i}{k} < s \leq \frac{i+1}{k}; \text{ where } s \text{ is the selectivity}$$

In this example, the value 39 falls between 37 and 44 represented by $Step(8)$ and $Step(9)$ respectively. Therefore using the above inequality, the selectivity bounds for `age` < 39 are given by: $0.8 < s \leq 0.9$. ◇

In this thesis, we aim to study query optimisation that takes into account the unreliability of database statistics and tries to avoid potentially bad plans. For this purpose we use a decision theory technique known as *minmax regret optimisation* (MRO). When executing a query, the DBMS encounters a particular instance of concrete parameter values, such as selectivities of operators, called a *scenario*. The problem is that, during the optimisation step, the optimiser does not know which scenario the DBMS will face during the execution step. Even at query execution time, the DBMS may not know which scenario it faces because of inaccurate or out-of-date statistics. Moreover, it is highly unlikely that there is a single execution plan that will yield the optimal cost for every potential scenario. For each scenario, there

is a plan with the smallest cost known as the optimal plan for the scenario. The *regret* of a plan under a scenario is the difference between the plan's cost and the cost of optimal plan for the considered scenario. MRO tries to find the plan whose maximum regret is minimum. This means that when the plan is confronted with its worst-case scenario, it will have the best performance among all other plans (when confronted with their worst-case scenarios). Consequently, our goal is to choose a logical query execution plan that performs reasonably well regardless of the scenario it encounters.

Minmax regret optimisation (MRO) is a well-known criterion for decision making in environments with imprecise parameters [3, 69]. It has been used in fields such as economies, statistics, psychology, politics and social science [30, 45, 60, 95], as well as in computer since to deal with problems where parameters may be imprecise such as knapsack, shortest path, spanning tree, assignment and scheduling problems [2, 3, 7, 14, 25, 36]. This criterion is suitable for optimising critical systems which should function well regardless of the scenario encountered (even under a worst case scenario) [3].

When database parameters are estimated by single values, cost-based database query optimisers choose the plan with the smallest estimated cost as the optimal plan [28]. In terms of optimisation under inaccuracy, this case is the same as optimising under a single realisation or scenario. The problem is more complicated if we want to consider the possible range of values for imprecise parameters which renders many different scenarios. One option at optimisation time is to assume that inaccurate parameters take their largest, median (i.e. midpoint) or smallest estimated values. However, this means that the optimisation is performed using a single scenario without taking into consideration all other possible scenarios. One problem with this approach is that a single scenario is given more attention or credit over the others, while in reality there is no knowledge which scenario the system will face at execution time. Another problem of optimising under a single scenario is that an optimal plan for one scenario may be a worst plan for another. Therefore, the query optimiser should be aware of the different scenarios when choosing an optimal plan. By using MRO, the aim is to find the plan that works well regardless of the

Criteria	Plan	Scenario			
		x_1	x_2	x_3	x_4
		$\overline{s}_1$	$\underline{s}_1$	$\underline{s}_1$	$\overline{s}_1$
		$\overline{s}_2$	$\overline{s}_2$	$\overline{s}_2$	$\underline{s}_2$
		$\underline{s}_3$	$\overline{s}_3$	$\overline{s}_3$	$\underline{s}_3$
		$\underline{s}_4$	$\underline{s}_4$	$\overline{s}_4$	$\underline{s}_4$
MRO	$\sigma_2\sigma_1\sigma_3\sigma_4$	**0.225**	0.203	0.2	0.15
Optimistic	$\sigma_1\sigma_2\sigma_3\sigma_4$	0.895	0.003	0.0	**0.92**
Midpoint	$\sigma_2\sigma_3\sigma_1\sigma_4$	0.039	**0.383**	0.38	0.026
Pessimistic	$\sigma_2\sigma_3\sigma_4\sigma_1$	0.0	0.488	**0.527**	0.0

Table 1.2: The regret under some scenarios in Example 1.1.2.

encountered scenario. The following example explains the differences between MRO and the other criteria.

Example 1.1.2 Let $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ be a set of selection operators which are to be applied to a relation R. Suppose the selectivity s_i of each operator σ_i is known to fall within an interval $[\underline{s}_i, \overline{s}_i]$, where $\underline{s}_i$ and $\overline{s}_i$ are the minimum and maximum selectivities respectively, as follows: $s_1 = [0.1, 0.97]$, $s_2 = [0.2, 0.3]$, $s_3 = [0.35, 0.7]$ and $s_4 = [0.6, 0.8]$. So the selectivity s_1 for operator σ_1, for example, can take any value between 0.1 and 0.97 (inclusive). For simplicity, assume that all operators have the same cost of 1 and that the relation R has cardinality 1 (the method used to calculate the cost of a plan under any scenario is given in Section 2.4). The goal is to produce a plan for S, i.e. an order in which to evaluate the four selection operators.

Table 1.2 shows four different criteria for choosing a plan and their performance in terms of regret under a few different scenarios (where a scenario consists of choosing the minimum or maximum selectivity of each operator)[1]. Apart from the MRO criterion, the other criteria are the optimistic, midpoint and pessimistic criteria,

[1]More details will be provided in Chapter 3, including how Table 1.2 can be generated.

which only consider the smallest, median and largest estimated selectivity values. The optimal plan for a particular scenario is one in which the operators appear in increasing order of selectivity. So the optimistic criterion produces a plan that is optimal for the scenario in which each operator has its minimum selectivity, while the pessimistic criterion produces the optimal plan for the one in which each operator has its maximum selectivity. We can see from Table 1.2 that there is no solution which is optimal for all scenarios. The optimistic and pessimistic solutions are optimal (i.e. with a regret value of 0) under one and two scenarios respectively, but their performance in terms of regret varies considerably from one scenario to another. It is true that the MRO solution is not optimal for any scenario in this example, but it has a more stable performance over all scenarios compared to the optimistic, midpoint and pessimistic solutions. Therefore, using the MRO solution is a wise choice because it has good overall performance and its worst regret value of 0.225 is better than the optimistic, midpoint and pessimistic solutions, whose worst-case regret values are 0.92, 0.383 and 0.527 respectively (shown in bold face in Table 1.2).

◇

Our proposed query optimiser approach can add an overhead at the logical optimisation stage. However, the heuristics we develop run in time which is polynomial in the number of operators in the query oppose to other existing approaches dealing with single scenario and run in exponential time (such as those which require access to exponential join enumeration schemes) [61]. The overhead of our approach improved the quality of the result. So this overhead is traded for quality as we will discussed in Sections 5.5 and 8.4.

In the following section, we present the main contributions of this thesis. After that, the structure of the thesis is described.

1.2 Contributions

In this thesis we propose a robust approach to logical query optimisation that takes into account unreliability in the statistical information provided by the DBMS during

the optimisation process. In particular, we study the problems of applying MRO to sets of selection operators and sets of join operators in the presence of interval selectivities. We refer to these two problems as the *selection ordering problem* and the *join ordering problem*. The following are the main contributions of the thesis.

For the selection ordering problem:

- We formalise the selection ordering problem when the selectivities of the selection operator predicates are given as intervals. We assume that predicates are independent of each other. We identify a number of useful properties of the problem. For example, instead of considering the entire selectivity interval, our study shows that it is sufficient to consider only the minimum and maximum selectivity of each selection predicate. We also investigate a number of special cases which can be solved in polynomial time. For example, based on the selectivity intervals of the selection operators, we define the relationship of *domination* between operators. We show that the optimal solution for the selection ordering problem, in which all pairs of operators have a domination relationship between them, can be found in polynomial time by sorting the operators in non-decreasing order according to their minimum or maximum selectivities.

- The associated decision problem for the general selection ordering problem turns out to be NP-hard, so we develop a novel heuristic for the selection ordering problem that runs in polynomial time.

- We evaluate our heuristic for solving the selection ordering problem experimentally using three different data sets: a synthetic data set, the Star Schema Benchmark (SSB) and the Enron email data set. The heuristic was compared to three different baseline heuristics and the experimental results demonstrate the effectiveness of our heuristic over the others. In the SSB data set for example, our heuristic finds the exact, optimal solution in 90% of the cases. More importantly, it avoids bad plans and, considering the experimental evaluation over all data sets, in the worst case the solution found by our heuristic has a regret value 1.27 times the optimal minmax regret.

- We applied our selection ordering heuristic to a related NP-hard problem, namely finding the MRO solution for minimising the total flow time of jobs on a single machine, where the job processing times are given as intervals. Our heuristic was tested experimentally on a synthetic data set and compared with a well-known 2-approximation algorithm which only considers the midpoint of each processing time interval. Our heuristic performs better than the approximation algorithm. For example, the approximation algorithm found the optimal solution in just 56.6% of the tested cases, while our heuristic found the optimal solution in more than 82% of the cases.

For the join ordering problem:

- We formalise the join ordering problem where the selectivity of join predicates are known to fall in intervals and are independent of each other. Since the general join ordering problem (without intervals) is known to be NP-hard, we restrict our attention to chain queries for which join ordering (without intervals) can be solved in polynomial time. More specifically, we limit the search space for optimal plans to the class known as left-deep join trees. We investigate a special property of the problem, called the *precedence adjacency property*, which allows us to identify the relative order of neighbouring relations in so-called precedence graphs relating to a query.

- We develop an effective heuristic for the join ordering problem which runs in polynomial time. The heuristic exploits the precedence adjacency property mentioned above to improve the quality of results and reduce the number of plans it has to consider.

- We evaluate the join ordering heuristic experimentally on a synthetic data set, and compare it with three other heuristics. The experimental evaluation shows that our heuristic outperforms the other heuristics. For example, our heuristic finds the optimal solution in 98% of the tested cases, and in the worst case found plans with a maximum regret no more than 1.23 times the optimal MRO solution.

1.3 Thesis outline

The structure of the thesis is as follows:

Chapter 2 presents an overview of related work. It reviews background research on query languages in general and the relational algebra in particular. Then we discuss query processing as well as query optimisation and evaluation. Moreover, we review related problems faced by query optimisers and their influence on the quality of the resulting execution plans. After that, definitions related to the selection ordering problem and the join ordering problem are given. In addition, the chapter discusses optimisation under inaccuracy, and the approaches used in this field. Minmax regret optimisation (MRO) from decision theory is used in this thesis to deal with inaccuracy in the selection ordering and join ordering problems. We review decision theory in this chapter as well as the MRO approach. Moreover, we review some related problems that use MRO.

Chapter 3 presents the formal definition of the selection ordering problem where the selectivities of selection operator predicates are known to fall within intervals. The minmax regret optimisation approach for the selection ordering problem is defined, along with the associated brute force approach to find an optimal plan. In addition, a number of properties of the problem are identified, as well as some special settings which can be solved in polynomial time.

Chapter 4 describes our novel heuristic, max-min, for solving the selection ordering problem. It can be considered as a template for a number of algorithms based on the chosen parameters of the heuristic. The chapter discusses the possible options for the parameters of the heuristic. Two versions of the heuristic are presented. Both use the same overall method but the second reduces its computational complexity.

Chapter 5 presents an experimental evaluation of the max-min heuristic for selection ordering. The quality of plans produced by the heuristic are compared to optimal plans computed by the brute-force approach as well as to plans produced by three baseline heuristics, namely the midpoint, pessimistic and optimistic heuristics, based on a number of measuring criteria. Three different data sets were used to evaluate the max-min heuristic.

Chapter 6 discusses applying our max-min heuristic for selection ordering to a job scheduling problem where the target is to minimise the total flow time when the processing time of each job is known to fall within an interval of values. It evaluates the heuristic experimentally, comparing its performance with other baseline heuristics, namely the midpoint, pessimistic and optimistic heuristics.

Chapter 7 considers the join ordering problem when the selectivities of join predicates are assumed to fall in intervals. It presents the precedence adjacency property which is used in our novel heuristic for the join ordering problem, which is also described in this chapter.

Chapter 8 presents the experimental evaluation of our heuristic for the join ordering problem. Similar to Chapter 5, the quality of plans produced by the heuristic is compared with the MRO optimal plans and those produced by the midpoint, pessimistic and optimistic heuristics.

Chapter 9 concludes the thesis by summarising its main contributions, and then discussing directions for future research.

Chapter 2

Background and Related Work

This chapter first describes the state of the art in query languages and query processing. Moreover, it discusses query optimisation and evaluation. Then it introduces the formal definitions of the selection ordering problem and the join ordering problem. It also discusses optimisation under uncertainty in the statistical information. After that, decision theory is discussed along with the minmax regret optimisation. In addition, this chapter describes some related problems that we studied in order to improve our understanding of minmax regret optimisation.

2.1 Query languages

The relational model is one of the best known and most popular database models used these days. It was first introduced by Codd in 1970 [34]. Since then many query languages have been designed for it. A *query language* is a language that expresses queries or functions which are then submitted to the database in order to extract some information [27]. Most query languages allow users to describe what they want without specifying how their query is processed. Such query languages are known as declarative query languages [105]. A well known example of a declarative query language is the Structured Query Language (SQL) which was introduced in the nineteen seventies [101].

Other query languages, such as relational calculus and relational algebra, are popular for theoretical investigations, while SQL is considered as a practical query

language [105]. Relational calculus is based on mathematical logic, while relational algebra is based on an algebra of operators equivalent to the calculus [101]. Relational calculus and SQL queries can be represented as relational algebra expressions [105]. In fact, many DBMS translate SQL queries to relational algebra expressions, which are then processed by the query optimiser to generate an execution plan for that query [44].

Originally the relational algebra had five basic operators [101]. These operators were selection, projection, union, cross-product, and difference. Other operators were defined with the use of relational algebra in databases; these included rename and join, which can be viewed as a cross-product followed by a selection [101]. In this thesis, we focus on the selection and join operators. Consider the following relational algebraic expression with selection operator: $\sigma_p(R)$. This expression is defined formally as $\sigma_p(R) = \{t \in R : p(t)\}$ which selects or retrieves all tuples t from relation R that satisfy the predicate p. On the other hand, consider the following relational algebraic expression comprising a join operator: $R_1 \bowtie_p R_2$. This can be expressed formally as: $R_1 \bowtie_p R_2 = \sigma_p(R_1 \times R_2)$, where $\times$ is the Cartesian product of the two relations. The result of this expression is a new relation with tuples from R_1 associated with tuples from R_2 based on satisfying the predicate p. More details about both selection and join operators will be provided in Sections 2.4 and 2.5 respectively.

2.2 Query processing

After considering the query languages in the previous section, let us discuss how a query is processed in general by the database management systems (DBMSs). *Query processing* is a term usually refers to the activities that start with accepting and processing a submitted query in a high-level language, up until retrieving the result of the query [108]. Figure 2.1 presents the main steps of query processing.

Once a query is submitted to the DBMS in a high-level language such as SQL, it is first parsed and validated [47]. This validation includes checking the syntax of the query as well as that of attribute and relation names. After that the system

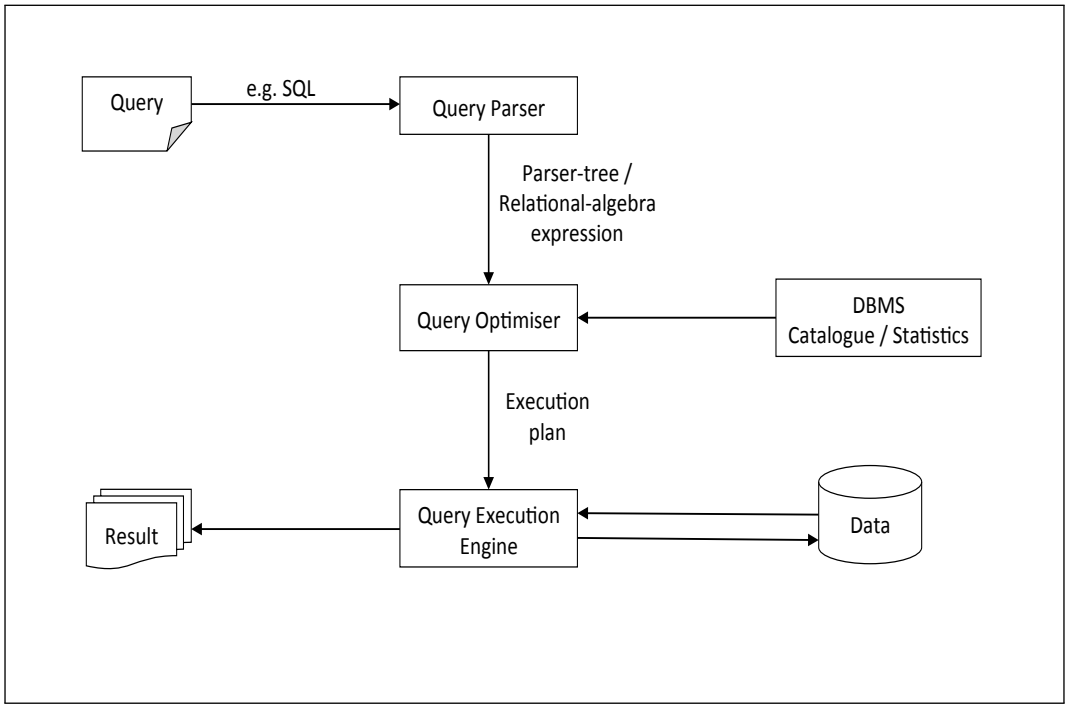


Figure 2.1: Query processing steps.

generates a parse-tree representation of the query called the query tree or query graph. The parse-tree is translated to an internal representation and passed to the query optimiser. In some DBMSs, the internal representation is relational algebra.

A single relational-algebra expression can have many equivalent execution plans [47]. If two plans produce the same result on every database instance, then they are described as equivalent execution plans [108]. Different execution plans have different costs at run-time when retrieving the result of the query [108]. The optimiser is responsible for finding the equivalent execution plans and choosing a suitable one based on statistical information stored in the database-system catalog [59]. This highlights the importance of the optimisation step in choosing a plan that has good execution performance. Sometimes this stage is called logical query optimisation [38, 85]. Our work tries to enhance optimisation at this step. More discussion about query optimisation is provided in Section 2.3.

There are various ways to access the data in a DBMS, such as using an index or performing a scan of a relations to find relevant tuples [85]. Therefore, the query execution engine or the code generator processes the execution plan chosen by the optimiser and generates the code for executing the query at run-time [28]. The

execution code consists of a set of physical operators that produce the result of a query as a data stream. External sort, sequential scan, index scan, nested-loop join and sort-merge join are examples of physical operators [28]. This step sometimes is called physical query optimisation [38, 85]. It is important to mention that different DBMS may have different query processing steps. For example, some DBMSs pass an annotated parse-tree driven by the structure of the SQL query directly to the query optimiser instead of passing a relational-algebra expression [108].

2.3 Query optimisation and evaluation

Although query optimisation in database management systems (DBMSs) has been a topic of research for decades, there are still important unresolved issues. The need for a query optimiser comes from the way the databases are designed as well as their adoption of the data independence principle, which separates the logical data model and how/where the data is stored in the database. Typically, the goal of the query optimiser is to come up with the best plan after computing equivalent plans [106].

There are different approaches to query optimisation. One optimisation approach uses *heuristic rules* to order operations to satisfy an execution strategy such as applying selection operators before join operators [47, 61]. Another well-known optimisation approach is to use a *cost model* which assigns an estimated cost to each equivalent execution plan in order to distinguish between them [47]. The parameters that are used by the cost model can be classified in two main categories: parameters related to system resources and parameters related to workload profile [120]. Examples of system resource parameters include disk accesses, CPU time, buffer size and communication cost (e.g. in distribution databases) [108]. On the other hand, workload profile parameters are concerned with the data and its properties, such as relation cardinality and the number of distinct attribute values, as well as information about queries such as the operators' selectivities and costs [12, 28, 120]. Our work focuses on the cost model optimisation approach dealing with workload profile parameters.

In term of evaluating a query, especially with multiple operators, there are two

main approaches: *materialised evaluation* and *pipelined evaluation* [47]. In materialised evaluation, the system evaluates one expression/operator at a time. After evaluating a single operator in the plan, the result is temporarily stored as a relation to be considered as an input for a subsequent operator. Obviously, the main disadvantage of this approach is the overhead of storing the temporary relations on disk (except for cases where the relation is small and can fit in main memory) and reading these relations again for subsequent operators [108]. On the other hand, in pipelined evaluation, the tuples resulting from evaluation of an operator are directly forwarded to the subsequent operator in the plan. Therefore, there is no need for temporary storage in this approach which tends to make the evaluation process faster than in the materialised approach [108].

Query optimisers usually employ statistics stored in the system catalog to estimate the cost of a query plan [68]. For typical workloads, a DBMS can compile statistical data over time to obtain a reasonable estimate. Relation cardinality, predicate selectivity and cost as well as the number of distinct values of attributes are some of the main statistical parameters used by optimisers [12, 47, 76]. The number of distinct values of an attribute is often used to estimate the selectivity of a predicate which in turns predicts the cardinality of the result of applying the predicate [47]. For example, the selectivity of a predicate which asks for an exact match with a single attribute value can be estimated for an attribute with 6 distinct values as $1/6 = 0.167$. In general, optimisers use selectivities to schedule those operators that reduce the intermediate result before those that do not. This reduces the cost of query evaluation and reduces the consumption of system resources (e.g. I/O, CPU and buffering at run time) [47, 96, 108]. Unfortunately, in practice, statistical information is not precise at all times [64, 108]. A brute-force solution to guarantee up-to-date statistical data is to force the system to update the statistics every time the data is changed (e.g. every update, add or delete on a relation) [108]. However, this is not practical and in real life statistics are only updated occasionally [108]. Therefore, optimisers in practice use an *estimated* cost in the absence of the exact cost. Chaudhuri states that an optimiser is "only as good as its cost estimates" [28].

There are other challenges facing query optimisers when evaluating equivalent

plans, apart from the accuracy of the statistics [64]. Query optimisers sometimes have insufficient information about the run-time environment, such as the available resources, or when they have to handle user-defined queries. The situation also becomes difficult when systems are confronted with very unevenly distributed data values or predicates that are complex or correlated. All these situations influence the effectiveness of the query optimiser [68].

Trying to estimate selectivities in dynamic settings, such as data streams [112], or in non-relational contexts, such as XML databases [98,122], also poses challenges. It may even be impossible to obtain any statistical data, because the query is running on remote servers [121]. Detailed information may also not be available because a user issues an atypical ad-hoc query or utilises parameter markers in a query [61, 68]. Section 2.6 will discuss the main approaches that deal with inaccuracy in the statistical data used by optimisers.

Inaccurate statistics provided to query optimisers lead to them choosing poor execution plans [18,64]. Leis *et al.* show experimentally that even good optimisation algorithms perform poorly in the case of inaccurate statistical information [76]. A number of techniques can be used to improve the accuracy of the statistics in the system catalog. The main examples of these techniques are sampling [74, 93], histograms [63,96,99] and probabilistic methods [55,113]. Histograms can be used for estimating relation size or selectivity of predicates [12, 23]. This technique has been adopted by some well-known DBMSs such as DB2, Oracle and SQL Server [12, 63]. More recent research utilises probabilistic approaches such as probabilistic wavelet synopses, which were originally used in signal and image processing [55].

We now describe histograms in more detail. Having accurate and exact information about data distributions is helpful yet infeasible since it requires huge storage space. Histograms are useful in this case to give good approximations [99]. Moreover, they are used for selectivity estimation and other database operations such as partition-based temporal join execution [63, 96]. In general, histograms are an approach to approximate a data distribution by calculating the frequency of an attribute value [66]. They divide the values of an attribute into k buckets/ranges and count the number of tuples that lie within the range of each bucket [99]. Usually,

the distinct values of an attribute assist in defining the range of histogram's buckets [99, 108]. The value of k can be fixed or changed dynamically [28]. The value of k determines the level of accuracy. The bigger the value of k, the more accurate the estimate is but the more memory that is required [28]. In general, histograms are loaded in the memory and this is why they should not required much memory. A practical example of the use of histograms and how they are generated will be provided in Section 5.3. There are two main types of histograms: *equi-width* and *equi-height* (or *equi-depth*) [28,96,108]. In an equi-width histogram, all buckets have the same size, which means that each bucket covers the same number of distinct values. On the other hand, the aim in an equi-height histogram is that each bucket has the same number of tuples, which may mean that buckets cover different ranges of distinct values. If the cardinality of the considered table is Ω, then each bucket in an equi-height histogram has approximately Ω/k tuples.

The general assumption in histograms is that the values of each bucket are uniformly distributed [28, 99]. Correlations between attributes is one of the biggest challenges for optimisers. An attempt to overcome this problem is the use of multi-dimensional histograms [23,50,63]. However, applying this automatically in practice is not straightforward since the number of possibly correlated attributes is large and finding the right balance between the need for joint information and minimising the size of histograms is not a trivial task [28].

Guy Lohman states that in practice, inaccurate estimation for parameters in a cost model is considered as one of the major causes of poor performance and a common challenge for most query optimisers [79]. One of the practical examples of challenges facing DBMSs is a query with unknown parameters, such as '*Age BETWEEN x AND y*'. Such a query still poses a challenge for the optimiser even with good histograms, since the performance of the selected plan differs from one execution to another based on the values of x and y. Lohman also discusses errors in selectivity estimation, especially for join operators and their implication in real life. For example having a new database with a few months of data in a date attribute with 100 years causes a skewed data which leads to inaccurate selectivity estimate. Indexes can help in this case but to have accurate estimates, the system should

perform frequent maintenance which adds extra cost and overhead for the system. Identifying the number of distinct values for an attribute is used by some optimisers to predicate the selectivity of a predicate, as we discussed earlier. Consider a case where an attribute has 6 distinct values, while 99.99% of rows that satisfy a predicate have the same value. Some optimisers, such as DB2, calculate the selectivity of the predicate as $\frac{1}{|\text{distinct values}|}$ which is 0.167 in this case; however the true selectivity is 0.9999 [79]. This huge difference between the estimated selectivity and the true selectivity leads to the optimiser choosing a suboptimal plan.

Determining the selectivity of correlated attributes remains a hard problem for optimisers to solve since it requires pre-knowledge about the nature of the data. Usually optimisers under-estimate predicate involving correlated attributes which leads to suboptimal solutions. For example, in a cars database with 10 makers and 100 models, the usual selectivity estimate for the predicate: '$Model = Accord$ AND $Make = Honda$' is $1/100 \times 1/10 = 0.001$. However, the actual selectivity is $1/100 = 0.01$ due to the correlation between the $Make$ and $Model$ attributes, since only Honda makes the Accords model. Currently to avoid such problems, database administrators should deliberately update the selectivity of correlated predicates [79]. These kinds of errors result in query execution plans far from optimal. Consequently, an optimiser should try to avoid potentially bad plans rather than strive for an optimal plan based on unreliable information [79]. In this thesis, we propose an optimisation approach that tries to avoid bad plans.

2.4 Selection ordering

In this thesis, one of the problems we are interested in is the selection ordering problem. The selection operator, denoted by σ in the relational algebra, is an operator common to many data querying languages [101]. It is also known as a filter in other contexts such as data stream processing [16, 19, 42] and sensor networks [43, 48, 57, 97], where there is renewed interest in improving the efficiency of processing these operators. A very common setting is determining the order in which to apply a set of commutative filters to a stream or a set of data items, e.g. tuples of a relation,

so as to keep the processing costs to a minimum. In the following, we introduce some basic definitions and discuss the selection ordering problem.

2.4.1 Basic definitions

This section presents a formal definition of the basic selection ordering problem, where both the predicates selectivities and costs of operators are assumed to be known exactly. The selectivity of the predicate p is defined as the probability that a tuple passes through the predicate filter, or alternatively, as the fraction of tuples filtered by the selection predicate [37]. In general, when the selectivity increases, the run-time of the query increases as well [31]. The cost of a selection operator refers to the cost of applying the selection operator to a single tuple [37]. From now on, we will strip out the predicates from the definition of the problem and only refer to their selectivities. The problem can be viewed as ordering a set of selection operators or as considering a single operator with a conjunction of predicates in which we need to order the predicates. The former is what we consider in the following definitions. However, more insight about the two views will be provided in Section 2.4.2.

Definition 2.4.1 Given a set $S = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ of selection operators, for $1 \leq i \leq n$, each selection operator σ_i has a selectivity $s_i \in [0, 1]$ and a cost $c_i \in \mathbf{R}^+$.

Definition 2.4.2 For a selection operator σ_i with selectivity s_i and cost c_i, the *rank* r_i of operator σ_i is defined as [37, 73]:

$$r_i = \frac{s_i - 1}{c_i} \tag{2.4.1}$$

The rank plays an important role in finding the optimal order for a given set of selection operators as we will discuss in the following section. Let π^n be the set of all possible permutations over $1, 2, \ldots, n$. For $\pi_j \in \pi^n$, $\pi_j(i)$ denotes the i-th element of π_j.

Definition 2.4.3 A query execution plan p_j is a permutation $\sigma_{\pi_j(1)}, \sigma_{\pi_j(2)}, \ldots, \sigma_{\pi_j(n)}$ of the n selection operators. The set of all possible query execution plans is given by:

$$P = \{p \mid p = \sigma_{\pi(1)}, \sigma_{\pi(2)}, \ldots, \sigma_{\pi(n)} \text{ such that } \pi \in \pi^n\}.$$

Let Ω be the cardinality of the relation on which we execute the selection operators. The cost of evaluating plan p_j is given by [61]:

$$\begin{aligned} \text{Cost}(p_j) &= \Omega\left(c_{\pi(1)} + s_{\pi(1)}c_{\pi(2)} + s_{\pi(1)}s_{\pi(2)}c_{\pi(3)} + \cdots + \prod_{i=1}^{n-1} s_{\pi(i)}c_{\pi(n)}\right) \\ &= \Omega\left(\sum_{i=1}^{n}\left(\prod_{j=1}^{i-1} s_{\pi(j)}\right)c_{\pi(i)}\right) \end{aligned} \tag{2.4.2}$$

The assumption in the above formula is that the selectivities of the predicates are independent of each other. For the joint selectivity of multiple attributes, much early work and many systems make the *attribute value independence* (AVI) assumption. This assumes that the selectivity of a set of operators $\{\sigma_{i_1}, \sigma_{i_2} \ldots \sigma_{i_m}\}$ is equal to $s_{i_1} \times s_{i_2} \times \cdots \times s_{i_m}$. If instead a system stores (some) joint selectivities (it is infeasible for it to store all of them), we can use the AVI assumption to "fill in the gaps" or use the estimation approach advocated in [81].

2.4.2 Selection ordering problem

There exist techniques for ordering a set of selection predicates at logical optimisation stage to filter out as many tuples as early as possible at the lowest possible cost, by using the concept of ranking [62,73]. Sorting the operators in non-decreasing order of their ranks results in the minimal expected pipelined processing costs [62]. Assuming we have accurate values for both s_i and c_i, we can use Equation (2.4.1) to calculate the *rank* r_i of σ_i. Clearly, the computation of the ranks and the sorting can be done in polynomial time. Basically, ordering selection operators optimally is a solved problem, but only when given exact values for the s_i and c_i.

These techniques that use the ranking concept rely on having accurate values for the operators' selectivities, i.e. the percentage of tuples passing a filter, and their processing costs (per tuple). Getting the estimation of selectivities (and/or costs) wrong can lead to high overall costs for the pipelined execution. As we discussed in Section 2.3, estimating the selectivities of simple predicates on base relations in a relational database is fairly well understood and can be done quite accurately [55,63]. However, the situation changes once we are confronted with very unevenly distributed data values or predicates that are complex or correlated, such

as cases with dynamic settings like data streams [112]. It is also hard to estimate statistical data in non-relational contexts [98, 122]. It may also be hard to obtain some statistical data because the query is running on remote servers or heterogeneous DBMS due to a delay in the network or the availability of the server [121].

Similar optimisation problems have been studied in the context of sequential testing. Here the goal is to find faulty components as quickly as possible by testing them one by one. Each component has a probability of working and a cost for testing it. One of the earliest proposed solutions [67] relies on ranking the components and then ordering them by their ranks, very similar to the selection ordering described above.

There has been a renewed interest in pipelined filter ordering recently. Babu et al. investigate the effect of correlated selection predicates on the adaptive processing of data streams [19], while Neumann et al. avoid the multiple evaluation of common subexpressions in selection predicates [92]. Finally, Condon et al. present algorithms for pipelined filter processing in a distributed setting [37]. The problems discussed in the first two papers are NP-hard, while for the problems discussed in the last paper efficient algorithms have been developed.

Finding the right order of a set of selection operators is vital for both materialised and pipelined evaluations. As we mentioned earlier, an optimal plan should filter out as many tuples as early as possible. If we think about the selection ordering problem as applying one operator at a time using materialised evaluation, then using the optimal plan will minimise the sizes of the intermediate tables that are generated. This in turn reduces the cost and time of storing and reading the intermediate tables from disk and eventually results in faster execution [52]. On the other hand, if we think about the ordering problem in terms of a conjunction of predicates and employ pipelined evaluation, then when processing a tuple it is important to discard it as soon as possible if it does not satisfy the conjunctive condition. Therefore, assuming the cost of testing each predicate is the same, the optimal plan should test predicates with smaller selectivity values before those with larger selectivity values. Such an ordering leads to fewer comparisons and hence faster query execution.

2.5 Join ordering

In this thesis, we also study the join ordering problem. The join operator is a common operator to many querying languages [11, 21]. Usually it is denoted by $\bowtie$ in relational algebra [101]. Studying the join operator is important since it is known to be one of the expensive operators in query optimisation [24]. Finding an efficient order for a set of join operators is still a challenge for query optimisers [61]. The join ordering problem still attracts much attention in the literature. Some papers study the complexity of the problem [33, 87, 94], while others consider heuristics to solve the problem [24, 62]. In order to study the join ordering problem, three parameters should be specified: the allowed query graph structure, the possible join trees to be used as logical query plans and the cost criteria [24, 28]. These parameters play a major role in identifying the complexity of the problem and possible approaches to solve the problem. The following sections define these parameters for the join ordering problem considered in this thesis.

2.5.1 Join ordering problem

This section introduces some basic definitions related to the join ordering problem. Similar to Section 2.4, the assumption here is that the cardinalities of the relations and the selectivities of the join predicates are known exactly.

Definition 2.5.1 A join query Q is a pair (R, P), where $R = \{R_0, R_1, \ldots, R_n\}$ is a set of $n+1$ relations, $n \geq 1$, and P is a set of join *predicates* $p_{i,j} \in P$ representing a join between relations R_i and R_j, $0 \leq i, j \leq n$. Each predicate $p_{i,j}$ has a corresponding *selectivity* $s_{i,j}$ which is defined as the fraction of tuples from the Cartesian product of R_i and R_j which satisfy the predicate $p_{i,j}$. Each relation $R_i \in R$ has a *cardinality* r_i which is defined as the number of tuples in R_i. [73]

The assumption here is that there is at most one predicate between each pair of relations. We also assume that the predicate uses a basic comparison operator (i.e. one of the following comparison operators: $=, \neq, <, >, \leq$ or $\geq$). Therefore, the output cardinality after joining R_i and R_j with a given join predicate $p_{i,j}$ of

selectivity $s_{i,j}$ is [73]:

$$|R_i \bowtie_{p_{i,j}} R_j| = s_{i,j} \times |R_i| \times |R_j| = s_{i,j} \times r_i \times r_j \tag{2.5.1}$$

It can be noticed from the above equation that if $s_{i,j} = 1$, then the join is a Cartesian product. On the other hand, if $s_{i,j} = 0$ then the output cardinality will be zero and no tuples will be produced after joining R_i and R_j. We assume that the selectivities of predicates are independent of each other.

A join query Q can be represented as a graph, known as a *join query graph*.

Definition 2.5.2 Given a join query $Q = (R, P)$, the *join query graph* $G_Q = (V, E)$ for Q is an undirected graph, where each node in V represents a relation from R and there is an edge between R_i and R_j in E if there is a join predicate $p_{i,j}$ in P [49].

Join query graphs, or simply query graphs, can take different forms. The ones that we consider in this thesis are acyclic query graphs and in particular chain query graphs.

Definition 2.5.3 An *acyclic query graph* is a query graph with no cycles, while a *chain query graph* is an acyclic query graph in which each node (i.e. relation) has at most two incident edges (i.e. predicates).

Definition 2.5.4 Given a join query graph $G_Q = (V, E)$ for a join query $Q = (R, P)$ as defined in Definition 2.5.2, where $R = \{R_0, R_1, \ldots, R_n\}$, G_Q is called a *connected* chain query graph if it is a chain query graph and there is a path from R_0 to R_n.

Figure 2.2 shows three different forms of acyclic query graph. Both Figures 2.2(b) and (c) are chain query graphs while Figure 2.2(a) is not. The only connected chain query graph is the one in Figure 2.2(c).

In this study we consider only connected chain query graphs. This means that the join query can be evaluated without having to perform a Cartesian product between any two relations. This implies that if there are $n + 1$ relations in R then there are n predicates in P. Let us now consider the following definitions about *join trees*.

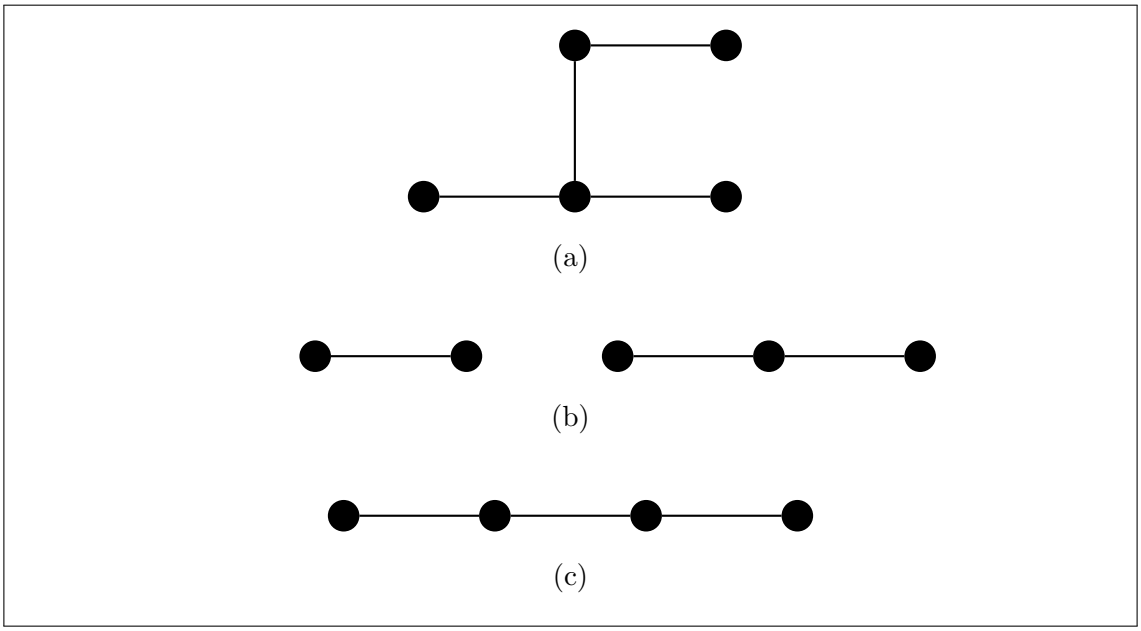


Figure 2.2: Acyclic query graphs.

Definition 2.5.5 A *join tree* is a binary tree that represents the algebraic expression of a join query, where leaf nodes represent relations and non-leaf nodes represent join operators.

There are different types of join tree and generally they are classified into two classes: *linear trees* and *non-linear trees* [38]. In linear trees, each join has at least one base relation. Left-deep and right-deep trees are examples of linear trees, while bushy trees are non-linear trees [38,73]. Each type of join tree may require a different optimisation approach [24]. Therefore, it is important to specify what join trees will be considered before optimising any given join problem. For example some studies consider bushy trees, while others use left-deep trees [87]. Ioannidis and Kang state that sometimes, for example when using probabilistic algorithms, finding optimal bushy trees can be easier than finding optimal left-deep trees [65]. In our study, we consider linear trees and more specificity we use *left-deep join trees* which are defined next [52].

Definition 2.5.6 A *left-deep join tree* (LDJT) is a join tree where the right child of each non-leaf node represents a relation $R_i \in R$.

Linear trees (and therefore LDJTs) have a number of useful properties. One property is that there is only one intermediate result at any stage to be considered by the query optimiser [73,85]. Therefore, the cardinality of the intermediate results can be estimated relatively easily, since each join involves at least one base relation. Another property is that there is one-to-one mapping between a linear tree and a (logical) execution plan [85]. An *execution plan* is the order in which operators are to be performed at execution time. For example, the LDJT of Figure 2.3 corresponds to the execution plan $((((R_0 \bowtie R_1) \bowtie R_2) \dots) \bowtie R_n)$. Moreover, the number of possible linear trees is considerably less than the number of bushy trees [52,85]. For example, the number of left-deep join trees for a query graph with n relations is $n!$ when Cartesian products are allowed. However, if we consider a chain query graph with no cross products, then the number of left-deep join trees is 2^{n-1} [85]. On the other hand, the number of bushy trees is $\frac{(2n-2)!}{(n-1)!}$ [52, 85].

Our aim is to find the left-deep join tree of minimum cost. The following property is very useful in finding the optimal LDJT, as will be shown in the following sections.

Definition 2.5.7 The *optimality principle* states that if tree T is an optimal join tree for a set of relations R, then any sub-tree T' of T is optimal for the relations in T' [85].

The join ordering problem is far from trivial. Therefore, in order to study the problem under inaccurate statistics we decided to start using a basic setting and find the LDJT that minimises the cost of evaluation a chain query. As mentioned in Section 2.2, we are also only considering logical query optimisation. Moreover, as we will discuss in the next section, the cost that we consider is *symmetric* (i.e. $Cost(R_i \bowtie R_j) = Cost(R_j \bowtie R_i)$). Therefore, under these assumptions, it is easy at this stage to transfer any LDJT to a right-deep join tree as they are both linear join trees [85]. The optimal LDJT that is chosen by the optimiser and its cost are passed to the physical query optimiser as inputs [38, 52]. These inputs along with other parameters, such as index availability and access paths, as well as the implementation algorithm of the join (e.g. index scan or nested-loop join) that is adopted during physical query optimisation, influences the choice of query evaluation

approach [38, 47]. For example, pipelined query evaluation cannot be used if the physical optimisation step uses a sort-merge algorithm to perform the joins [38]. Moreover, restricting the space of logical query plans to be generated from LDJTs undoubtedly influences and sometimes limits the choice of approaches to be used at physical query optimisation. For example, using parallelism to build hash tables at the physical optimisation stage is possible with right-deep join trees but not with LDJTs [104].

2.5.2 Cost formalisation

Optimisers usually aim to minimise resource usage such as disk access in order to reduce query execution time [38]. Moreover, there are a number of different measures for calculating the cost of a given join tree T. Some measure the cost based on the number of pages fetched [62]. Another well-known measure is to consider the size of the intermediate results generated by the joins [49]. This measure is denoted by $C_{out}(T)$, representing the summation of the cardinalities of the results in sub-tree T. It is sufficient to use a relatively simple cost measure as long as the statistical information provided is reasonably accurate or the optimiser takes into account the inaccuracy of the statistical information [76]. For a given tree T, $C_{out}(T)$ is defined as follows [49]:

$$C_{out}(T) = 0 \quad \text{if } T \text{ is a single relation}$$
$$C_{out}(T) = |T| + C_{out}(T_1) + C_{out}(T_2) \quad \text{if } T = T_1 \bowtie T_2$$

Next we formulate the cost function for a left-deep join tree which is produced from a given connected chain query graph. Assume that a set $R = \{R_0, R_1, \ldots, R_n\}$ of $n + 1$ relations is given where each relation has a cardinality r_i. Consider a connected chain query graph with a set of predicates. Given the set $S = \{s_1, s_2, \ldots, s_n\}$ of n selectivities, the selectivity s_i represents the predicate selectivity between R_{i-1} and R_i. This simplifies our previous convention in Section 2.5.1 where $s_{i-1,i}$ represents the predicate selectivity between R_{i-1} and R_i. For a given left-deep join tree T (see Figure 2.3), there are n sub-trees. A sub-tree T_i, where $1 \leq i \leq n$, is associated with the join that has relation R_i as the right branch. Considering this left-deep

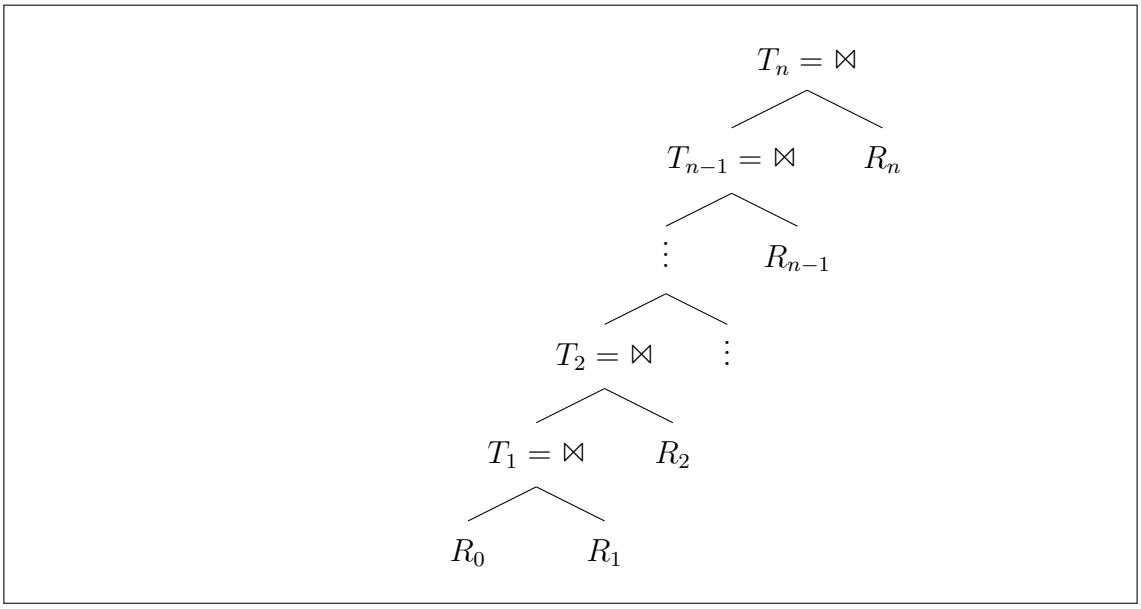


Figure 2.3: Left-deep join tree T

join tree T, the C_{out} cost formula can be reformulated as follows:

$$C_{out}(T) = \sum_{i=1}^{n} |T_i| \tag{2.5.2}$$

Equation (2.5.2) can be used to calculate the cardinality of the sub-trees. Let us define an auxiliary selectivity $s_0 = 1$ which simplifies the cost formulation and does not affect the result. The cardinality calculations for T_1 and T_2 in the left-deep join tree of Figure 2.3 are as follows:

$$\begin{aligned} |T_1| = \quad & |R_0 \bowtie R_1| && = s_0 \times r_0 \times s_1 \times r_1 \\ |T_2| = \quad & |(R_0 \bowtie R_1) \bowtie R_2| && = s_0 \times r_0 \times s_1 \times r_1 \times s_2 \times r_2 \end{aligned}$$

From the above example, we see that the cardinality of any given sub-tree T_i is simply the product of the relation cardinalities and the join selectivities that are involved in the sub-tree. Consequently, the cardinality for any sub-tree T_i can by computed as follows:

$$|T_i| = \prod_{j=0}^{i} (s_j \times r_j)$$

Since C_{out} is the sum of the cardinality of all n sub-trees, Equation (2.5.2) can

be rewritten as follows:

$$C_{out}(T) = \sum_{i=1}^{n} \left(\prod_{j=0}^{i} (s_j \times r_j) \right) \quad (2.5.3)$$

The C_{out} cost measure satisfies the optimality principle, where if tree T is an optimal join tree base on C_{out}, then any sub-tree T' of T is also an optimal tree [85]. Another property of C_{out} is symmetry, where for any relations R_i and R_j, $C_{out}(R_i \bowtie R_j) = C_{out}(R_j \bowtie R_i)$ [85].

The simplicity of a cost formula and its ease of calculation are desirable properties in query optimisation [76]. As seen above, the C_{out} cost formula is relatively simple and this is one of the reason we chose it for our study. Moreover, it has some useful theoretical properties such as symmetry and satisfying the optimality principle, as we will see in the next section. Since we are studying minmax regret in the context of query optimisation, we chose a simple single measure for cost to start with before considering other more complicated measures.

2.5.3 Finding an optimal left-deep join tree

Let $Q = (R, P)$ be a connected chain query, where $R = \{R_0, R_1, \ldots, R_n\}$ is a set of $n + 1$ relations such that $n \geq 2$ and P is a set of join predicates. We present in this section a polynomial time algorithm that finds the optimal left-deep join tree for Q [53, 62, 73, 88]. Before we discuss the proposed algorithm, we introduce some basic definitions and lemmas.

Definition 2.5.8 A *precedence graph* $G_i = (V_i, E_i)$ of $Q = (R, P)$ is a tree where each node in V_i represents a relation in R, each edge in E_i represents a predicate in P, and the root is $R_i \in R$ [85].

Let $G^P = \{G_0, G_1, \ldots, G_n\}$ be the set of all precedence graphs for the connected chain query graph Q. Since Q is a chain query, a precedence graph is a tree with a maximum of two branches such as those shown in Figure 2.4. Precedence graphs G_0 and G_n are always chains, as illustrated in Figures 2.4(a) and 2.4(c) respectively.

Since we have a maximum of two branches in a precedence graph G_i, we use $\pi_{i_L}(j)$ and $\pi_{i_R}(j)$ to identify the index (i.e. position) of any relation in the left and

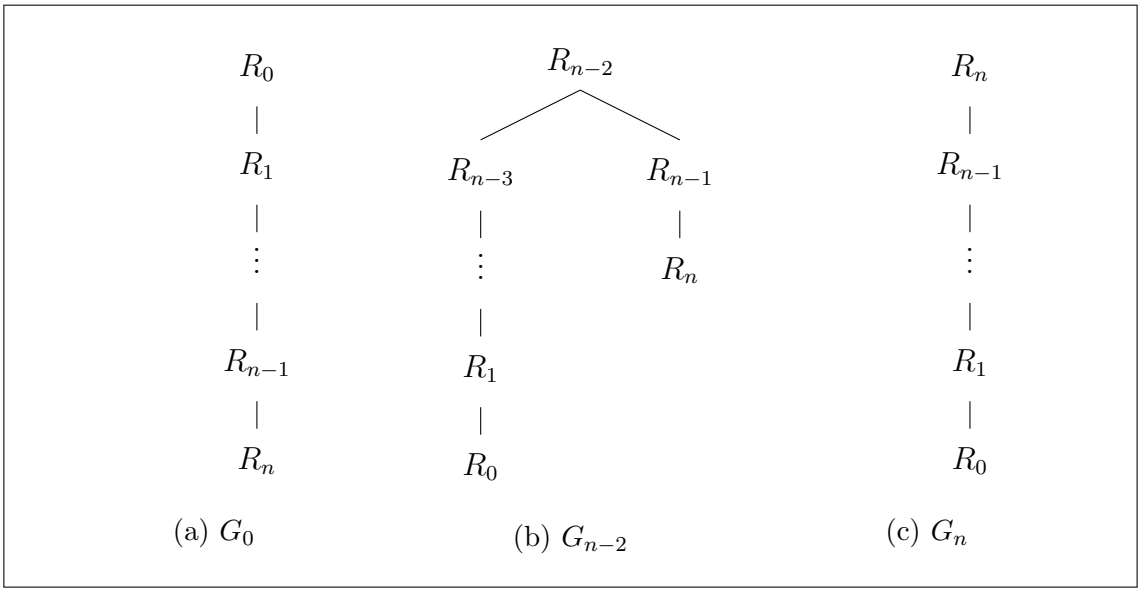


Figure 2.4: Sample precedence graphs.

right branches of G_i respectively. Therefore, the children of R_i in G_i have the index of $\pi_{i_L}(1)$ and $\pi_{i_R}(1)$. For example, for G_{n-2} in Figure 2.4(b), $\pi_{(n-2)_L}(1) = R_{n-3}$ and $\pi_{(n-2)_R}(2) = R_n$. In the case of precedence graphs for G_0 and G_n, we can use either $\pi_{i_L}(j)$ or $\pi_{i_R}(j)$ to indicate the positions of any relations in G_0 and G_n. However, we prefer to use $\pi_{i_R}(j)$ with G_0 and $\pi_{i_L}(j)$ with G_n. We can generally say that $\pi_{i_L}(u)$ and $\pi_{i_R}(v)$ are used to identify the index of any relation in any given precedence graph G_i where $0 \leq i \leq n$ such that $u \in [1, i]$ and $v \in [1, n - i]$.

Based on the given precedence graph G_i, the cost formula C_{out} for any relation or sequence of relations can be defined in a recursive way by introducing the following functions:

$$
\begin{aligned}
C(R_i) &= 0 && \text{if } R_i \text{ is the root} \\
C(R_j) &= r_j \times s_j && \text{if } R_j \text{ is any relation other than the root} \\
C(MN) &= C(M) + \Gamma(M) \times C(N) && \text{where } M \text{ and } N \text{ are sequences of relations} \\
\Gamma(N) &= \textstyle\prod_{R_j \in N} (r_j \times s_j)
\end{aligned}
$$

Definition 2.5.9 The rank for any sequence of relations N is computed as follows:

$$Rank(N) = \frac{\Gamma(N) - 1}{C(N)} \tag{2.5.1}$$

The following lemma defines a property of the cost formula C_{out} that allows us to use the rank formula to decide the relative order of any sequences of relations in the optimal plan.

Lemma 2.5.10 The cost formula C_{out} has the adjacent sequence interchange (ASI) property. Therefore, for any sequences of relations L, M, N and O, where M and N are non-empty sequences, the following is true given that both $(LMNO)$ and $(LNMO)$ are consistent with the given precedence graph:

$$C_{out}(LMNO) \leq C_{out}(LNMO) \quad \Leftrightarrow \quad Rank(M) \leq Rank(N)$$

Proof: This proof is based on [62, 73]. Using the recursive formulation for C_{out}, we break down the cost calculations of $C_{out}(LMNO)$ and $C_{out}(LNMO)$ as follows:

$$\begin{aligned} C_{out}(LMNO) &= C(L) + \Gamma(L) \times C(M) + \Gamma(L) \times \Gamma(M) \times C(N) \\ &+ \Gamma(L) \times \Gamma(M) \times \Gamma(N) \times C(O) \\ C_{out}(LNMO) &= C(L) + \Gamma(L) \times C(N) + \Gamma(L) \times \Gamma(N) \times C(M) \\ &+ \Gamma(L) \times \Gamma(N) \times \Gamma(M) \times C(O) \end{aligned}$$

If we subtract the two costs we get:

$$\begin{aligned} C_{out}(LMNO) - C_{out}(LNMO) &= \Gamma(L)\left(C(M) + \Gamma(M) \times C(N)\right) \\ &- \Gamma(L)\left(C(N) + \Gamma(N) \times C(M)\right) \end{aligned}$$

$$C_{out}(LMNO) - C_{out}(LNMO) = \Gamma(L) \times C(M) \times C(N) \left(\frac{\Gamma(M) - 1}{C(M)} - \frac{\Gamma(N) - 1}{C(N)}\right)$$

$$C_{out}(LMNO) - C_{out}(LNMO) = \Gamma(L) \times C(M) \times C(N)\left(Rank(M) - Rank(N)\right) \tag{2.5.2}$$

The sign of Equation (2.5.2) is determined by the result of the difference in ranks since the other terms are always positive. As a result, the cost C_{out} has the ASI property and we have the following:

$$C_{out}(LMNO) \leq C_{out}(LNMO) \quad \Leftrightarrow \quad Rank(M) \leq Rank(N)$$

□

Now let us introduce some new terminology which will be used next. A *node* in a precedence graph is a single relation or a group of relations. Consider the following lemma which states that if a parent node in any given precedence graph G_i has a rank greater than or equal to the rank of its child, then the child node will appear immediately after the parent node in the optimal plan with no node in between them. The proof for Lemma 2.5.11 can be found in [88].

Lemma 2.5.11 For any given precedence graph G_i, if there exist two nodes M and N such that $\pi_{i_L}(u) = M$ and $\pi_{i_L}(u+1) = N$ where $1 \leq u \leq i-1$ (or $\pi_{i_R}(v) = M$ and $\pi_{i_R}(v+1) = N$ where $1 \leq v \leq n-i-1$) with $Rank(M) \geq Rank(N)$, then in the optimal plan, M will be followed immediately by N with no relation in between. This is called *precedence adjacency* property.

We now present an algorithm that finds the optimal LDJT for a connected chain query graph using an ASI cost formula. This algorithm is based on one that was first introduced by Ibaraki and Kameda [62] and then extended by Krishnamurthy et al. to cover further cost functions which have the ASI property [73]. This algorithm is relevant since some aspects of it are used in our novel heuristic for the join ordering problem with interval selectivities discussed in Chapter 7.

The algorithm is stated formally as Algorithm 1, but let us describe the general idea here. We find the optimal plan for each precedence graph and the overall optimal plan is the one with the smallest cost among them. In other words, we consider each relation as a starting point for building the optimal plan, after which we choose the plan with the smallest cost among them to be the overall optimal plan.

In order to find the optimal plan for a given precedence graph G_i we first *normalise* the two branches of G_i. The purpose of the normalisation step is to sort the nodes on each branch in non-decreasing order according to their ranks. If there exists an index u such that $Rank(\pi_{i_L}(u)) \geq Rank(\pi_{i_L}(u+1))$ where $1 \leq u \leq i-1$ (or an index v such that $Rank(\pi_{i_R}(v)) \geq Rank(\pi_{i_R}(v+1))$ where $1 \leq v \leq n-i-1$ for the right branch), then we combine both $\pi_{i_L}(u)$ and $\pi_{i_L}(u+1)$ into one node

Algorithm 1: *FindingOptimalLDJT*

```
1 getQueryOptimalLDJT(Q)
    Input: Connected chain query graph Q
    Output: Optimal LDJT solution optSol for Q
    // Let G_0, ..., G_n be the precedence graphs for Q
2   optSol = getTreeSolution(G_0);
3   for 1 ≤ i ≤ n do
4       tempSol = getTreeSolution(G_i);
5       if Cost(optSol) > Cost(tempSol) then
6           optSol = tempSol;
7   return optSol;

1 getTreeSolution(G_i)
    Input: precedence graph G_i
    Output: Optimal LDJT solution sol for G_i
2   normLeft = normalise(left branch of G_i);
3   normRight = normalise(right branch of G_i);
4   normalisedSequence = R_i + merge(normLeft, normRight);
5   return denormalise(normalisedSequence);

1 normalise(C)
    Input: A chain C of relations
    Output: A normalised sequence for C
2   while there is a node M with child N such that Rank(M) ≥ Rank(N) do
3       Group M and N in a new node;
4   return C;
```

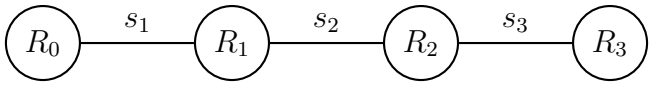


Figure 2.5: The connected chain query graph Q in Example 2.5.1.

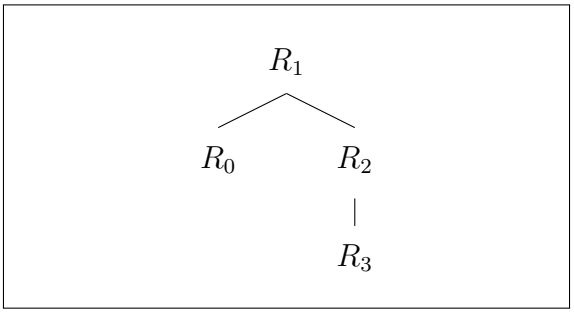


Figure 2.6: The precedence graph G_1 in Example 2.5.1.

and recalculate the rank for the new node. This process continues until the end of the branch. As a result, the nodes of the branch are sorted according to their ranks (normalised). Once the two branches are normalised, we merge the nodes on them in non-decreasing order according to their ranks. The optimal plan for the precedence graph G_i is R_i followed by the unpacked (denormalised) sorted sequence. For precedence graphs G_0 and G_n, there is only one plan to consider which will be their optimal plan. This is because G_0 and G_n are chains of relations.

Example 2.5.1 Let Q be a connected chain query graph as shown in Figure 2.5. Let $R = \{R_0, R_1, R_2, R_3\}$ be set of relations involved in Q with the cardinalities of the relations as follows: $r_0 = 200$, $r_1 = 2000$, $r_2 = 10000$ and $r_3 = 100$. Let $S = \{s_1, s_2, s_3\}$ be a set of selectivities for the join predicates, where $s_1 = 0.01$, $s_2 = 0.2$ and $s_3 = 0.5$.

Algorithm 1 starts by passing the query graph Q to *getQueryOptimalLDJT*. This function considers all precedence graphs for Q and finds the optimal plan for each of them. The plan with the smallest cost is considered as the overall optimal solution. The function *getTreeSolution* is used to find the optimal plan for a specific precedence graph.

In this example, let us consider finding the optimal plan for the precedence graph G_1 (shown in Figure 2.6) using *getTreeSolution*. Two lists are created by *getTreeSolution* to hold the normalised order of each branch using *normalise*. Since

the left branch consists of only one relation, it is already normalised. Therefore, $normLeft = R_0$. In order to normalise the right branch, the rank for each relation should be calculated first. The following are the rank calculations using Equation (2.5.1):

$$\begin{aligned} Rank(R_2) &= \frac{(10000 \times 0.2) - 1}{10000 \times 0.2} = 0.9995 \\ Rank(R_3) &= \frac{(100 \times 0.5) - 1}{100 \times 0.5} = 0.98 \end{aligned}$$

From the above equations, we notice that $Rank(R_2) > Rank(R_3)$ which means that the parent relation has a greater rank than its child. Therefore, R_2 and R_3 should be grouped (i.e. combined) in a new node, say $R_{2,3}$, and the new rank calculated as follows:

$$\begin{aligned} Rank(R_{2,3}) &= \frac{\Gamma(R_{2,3}) - 1}{C(R_{2,3})} = \frac{\Gamma(R_{2,3}) - 1}{C(R_2) + \Gamma(R_2) \times C(R_3)} \\ &= \frac{(r_2 \times s_2 \times r_3 \times s_3) - 1}{r_2 \times s_2 + r_2 \times s_2 \times r_3 \times s_3} = 0.9804 \end{aligned}$$

Now the right branch is normalised, so $normRight = R_{2,3}$. The overall solution for G_1 will be the root R_1 followed by the nodes from lists $normLeft$ and $normRight$ sorted in non-decreasing order of their ranks. Therefore, in this case we need to find the correct order for R_0 and $R_{2,3}$. We already know the rank for $R_{2,3}$; the following is the calculation of the rank for R_0:

$$Rank(R_0) = \frac{(200 \times 0.01) - 1}{200 \times 0.01} = 0.5$$

The temporary solution for G_1 is $normalisedSequence = R_1, R_0, R_{2,3}$, since $Rank(R_0) < Rank(R_{2,3})$. The last step is to denormalise $normalisedSequence$ (i.e. unpack the nodes to a sequence of relations) to find the overall solution for G_1, which is R_1, R_0, R_2, R_3.

The above process is repeated for each precedence graph in order to find its optimal plan. The algorithm then chooses the plan with the smallest cost (among the solutions for all precedence graphs) as the overall optimal solution for the query graph Q. ◇

2.6 Optimisation under inaccuracy

Most database query optimisers use cost models to estimate the costs of query plans and to choose an efficient one [12, 22, 44, 59, 105]. Obviously, in cost models some parameters may be inaccurate or even unknown as we discussed in Section 2.3. Clearly, without any information to go on, it will be difficult to perform query optimisation, so in the following we cover different assumptions on the available data and how to make use of it.

There are number of approaches for dealing with inaccuracy in parameters during the query optimisation process. Some suggest modelling the inaccuracy using a probability distribution over a random variable [4, 5]. However in many real-life problems there is a lack of information about the accuracy of parameter estimate and its probability distribution [32, 99]. In the presence of inaccuracy, some use the mean based on the probability information [68, 119]. However, this is not suitable for cases where a high quality solution is required. Some DBMSs use pre-defined default values for situations where the selectivity of some predicates is unavailable [12]. Alternatively some DBMSs perform what is called *dynamic statistics* which is a technique that runs an SQL query to scan a small random part of a table in order to get a selectivity estimation for some predicates [12].

Using the mean or modal value of the parameters to find the plan with least cost under the assumption that this value remains constant during query execution is called Least Specific Cost (LSC) in [32]. As Chu et al. point out in [32], if the parameters vary significantly, this does not guarantee finding the plan of least expected cost. An alternative is to use probabilistic information about the parameters, an approach known as Least Expected Cost (LEC) [32]. (A discussion regarding the circumstances under which LEC or LSC is best appears in [31].) In decision-theoretic terms, we are making decisions under risk, maximising the expected utility. However, probability distributions for the possible parameter values are needed to make this approach work, whereas in our case we assume that we do not have this information.

In parametric query optimisation several plans can be precompiled and then, depending on the query parameters, one plan can be selected for execution [51].

Moreover, in a dynamic environment, such as stream processing, one can more quickly switch to a better plan when the parameters change. However, if there is a large number of optimal plans, each covering a small region of the parameter space, this becomes problematic. First of all, we have to store all these plans. In addition, constantly switching from one plan to another in a dynamic environment (such as stream processing) just because we have small changes in the parameters introduces a considerable overhead. In order to amend this, researchers have proposed reducing the number of plans at the cost of slightly decreasing the quality of the query execution [39]. The MRO approach we use in this thesis can be seen as an extreme form of parametric query optimisation by finding a single plan that covers the whole parameter space.

Another approach to deal with the lack of reliable statistics is adaptive query processing, in which an execution plan is re-optimised while it is running [16, 18, 68, 82]. It is far from trivial to determine at which point to re-optimise, and adaptive query processing may also involve materialising large intermediate results. More importantly, this means modifying the whole query processing engine; in our approach no modifications of the actual query processing are needed. A gentler approach is the incremental execution of a query plan [91]. In this approach, a query optimiser starts with an initial plan. If the considered plan introduces an error above a certain threshold, the optimiser considers that part of the plan which has a major influence on the overall cost (under the inaccurate environment), re-optimises it and starts build the solution incrementally. However, deciding on how to decompose a plan into fragments and putting them together is a complex task.

Wu et al. state that predicate estimations are mostly provided as single values, but that the inaccuracy should be indicated with estimations modelled as intervals [118]. The use of intervals to estimate parameter values also appears explicitly in [18] and implicitly in [86]. Babu et al. [18] use intervals to model the inaccuracy of a single-point estimate. The uncertainty level L of the estimate E is represented by a value from 0 (none) to 6 (very high) based on the source of estimation. For example, if the estimation is obtained from the catalog, then it is considered as less uncertain and hence $L = 1$. The estimation interval $[E_l, E_u]$, where E_l and

E_u are the lower and upper bounds respectively, is calculated using the the single-point estimate E and the uncertainty value L such that $E_l = E * (1 - 0.1 * L)$ and $E_u = E * (1 + 0.2 * L)$. During optimisation, only three scenarios, those using the low estimates E_l, the single-point estimates E and the high estimates E_u, are considered and a single plan for each of them is stored. The paper uses a proactive re-optimisation approach where multiple plans are chosen at optimisation time and one of them is used at run-time based on the environment. Therefore, these plans from the three scenarios are considered as robust and switchable in order to avoid re-optimisation at run-time. On the other hand, in our approach we consider all scenarios as opposed to the three in this approach and choose only one of them at optimisation time. Babu et al. mention that the choice of scenarios and number of scenarios to be considered still need further study [18].

As mentioned on Section 2.3, histograms can be used to find the range/interval of parameters under inaccurate statistical information. Moerkotte et al. [86] study histograms which provide so-called q-error guarantees. Given an estimate $\hat{s}$ for parameter s, the q-error of $\hat{s}$ is $\max(s/\hat{s}, \hat{s}/s)$. An estimate is q-acceptable if its q-error is at most q. So if an estimate $\hat{s}$ is q-acceptable, the true value s lies in the interval $1/q \times \hat{s} \leq s \leq q \times \hat{s}$, but there is no knowledge about any distribution within the interval. The authors of [86] show that these histograms can be implemented efficiently in real-world systems such as SAP HANA. We should mention that the technique of using intervals can be applied to other approximate or error-tolerant queries as well. All we need is the selectivity for an exact query as the lower bound and the selectivity for a query that determines a candidate set with false positives as the upper bound.

For another situation in which interval selectivities arise, consider estimating the selectivities of string predicates which perform substring matching using SQL `like`, a problem known to be difficult [29]. Let us consider a database in which email messages are stored in a relation `emails`, with attributes such as `sender`, `subject` and `body` (the textual contents of the email). Assume that many queries use selection predicates such as `subject like '%invest%'`, so the database maintains indexes on words and on 2-grams (say) of words which allow it also to provide selectivities

for these.

Although the database maintains an index on words, the selectivity for the word 'invest' will be an underestimate for the selectivity of `subject like '%invest%'` since the strings 'reinvest' and 'investigation' (and many others) also match this predicate. Even if we are able to enumerate all words containing the string 'invest', we do not know how to combine their individual selectivities into a single selectivity. Instead we can use an interval selectivity with the exact match as a lower estimate. As the upper estimate, we can use the minimum selectivity of all the 2-grams of 'invest' since any string containing 'invest' must contain all of its 2-grams as well.

Example 2.6.1 As a concrete example, consider the following query on the Enron email data [35]:

```
select sender
from   emails
where  body like '%action%' and
       body like '%like%' and
       subject like '%use%';
```

Let us abbreviate the three predicates by A, L and U (for 'action', 'like' and 'use'). The interval selectivities for the three predicates, as computed using the method proposed above and explained in more detail in Section 5.4.1, are $[0.03, 0.68]$ for A, $[0.17, 0.27]$ for L and $[0.0008, 0.06]$ for U. Even if we consider only the upper and lower bounds of these intervals, they give rise to 8 possible scenarios. No single plan (order) is optimal for all 8 scenarios, so the best we can do is find the plan which minimises the maximum regret. It turns out that this plan corresponds to the order UAL. The maximum regret for this plan arises in the scenario when U has its maximum selectivity (0.06), while A and L have their minimum selectivities (0.03 and 0.17, respectively). The optimal plan for this scenario is AUL. ◇

Notions of robustness in query optimisation have been considered in [17, 18, 82]. Babcock and Chaudhuri [17] use probability distributions derived from sampling as well as user preferences in order to tune the predictability (or robustness) of query plans versus their performance. For Markl et al. [82], robustness is measured

by checking the estimated cost against the actual cost from query execution in a progressive manner (i.e. oscillating between optimization and execution steps). If the difference is larger than a certain value, then the generated plan is considered as sub-optimal and the system stop execution to perform re-optimisation. On the other hand, Babu et al. [18] consider a plan to be robust only if its cost is within e.g. 20% of the cost of the optimal plan. None of these papers consider robustness in the sense of minmax regret optimisation (MRO), which is the focus of this thesis. Moreover, these techniques need additional statistical information to work. We propose to use techniques from decision theory for making decisions under ignorance (more details will be provided in Section 2.7), meaning that we know what the alternatives and their outcomes are, but we are unable to assign concrete probabilities to them [95]. Our measure of optimality is the minimisation of the maximum regret as we will discuss in the following sections.

2.7 Decision theory

Decision theory is a well-known area that generally studies how decisions can be made using imprecise information. Some argue that decision theory was formally introduced and used by the philosopher Condorcet in 1793, while others argue that it is as old as the existence of humanity, since decision making is part of human activity [60]. In many cases, there is a lack of sufficient information when taking a decision. Moreover, defining the *"best"* decision varies from one person to another and from one circumstance to another [95]. Since the middle of the 20^{th} century, decision theory has been used in different fields such as economies, statistics, psychology, politics and social science [30, 45, 60, 95].

In general each action in the process of decision making leads to different outcomes which affect the following course of action. Decision theory is a well known approach for taking a decision or a sequence of decisions in situations where there is a lack of information or inaccurate information. Therefore, decision theory is used to take a decision under risk, ignorance or uncertainty. When the probabilities of the outcomes are known, then it is called decision under risk [95]. However, if the

probabilities are unknown, then it called decision under ignorance [60, 95], and this is the assumption in our work. Decision under uncertainty is sometimes used as a synonym for decision under ignorance or refers to decision under both risk and ignorance [95]. Peterson surveys different criteria for making decisions under ignorance in decision theory [95], one of the criteria being the minmax regret approach which is what we consider here. The *minmax regret* criterion was first introduced in the statistics field by Savage in the nineteen fifties [103]. It also known as minimax risk, minimax loss or simply minimax [60].

Decision theory also considers the decision maker's view or preference with respect to the problem [45]. Sometimes a decision maker has an optimistic view when taking a decision, while others may have a pessimistic view. As a result, decision theory has both optimistic and pessimistic criteria [30]. In the *optimistic* criterion, the decision is taken assuming that the best situation is most likely to happen. On the other hand, in the *pessimistic* criterion, the assumption is that the worst situation is most likely to occur and the decision is taken under that situation [30, 45]. Both optimistic and pessimistic criteria have been used in our experimental evaluation and compared with the minmax regret criterion in Chapter 5.

2.8 Minmax regret optimisation

Recently there has been renewed interest in studying minmax, minmax regret and minmax relative deviation/regret criteria for various combinatorial optimisation problems, as well as studying their complexity and approximation [2, 3, 13, 40, 69]. Minmax regret optimisation (MRO) has been applied to a number of optimisation problems where some of the parameters are (partially) unknown [3].

The difference between the minmax and minmax regret criteria is discussed in [3, 119] and is as follows. The aim of minmax criterion is to find the solution that performs best in its worst case scenario when compared with other solutions under their worst case scenarios. On the other hand, by considering all scenarios, the aim in the minmax *regret* criterion is to find the solution which minimises the maximum difference between the cost of the plan and the cost of the optimal plan of the

corresponding scenario. The *relative deviation* of a plan p under some scenario s can be calculated by dividing the difference between the cost of p and the cost of the optimal plan for s by the cost of the optimal plan for the s [3,119]. The minmax relative deviation measure considers an optimal plan as a plan that minimises the maximum relative deviation over all scenarios [40]. Yang, Kouvelis and Yu discuss the differences between the three measures and their use in some combinatorial optimisation problems [72, 119].

In general minmax, minmax regret and minmax relative regret criterion are suitable for critical systems which should function well even under a worst case scenario [3]. These criteria aim to minimise the impact of a worst case scenario when it occurs. The minmax criterion is appropriate for non-repetitive decision making, and for precautionary measures in vital systems such as nuclear accidents or public health where the goal must be met under any variation in the parameters. In applications where the decision maker evaluates the performed decision, the minmax regret criterion is useful as an indicator of how the decision could be improved if the imprecision in the parameters are resolved [3]. Moreover, the minmax regret criterion is useful for cases where the objective is to find a solution that can perform as close as possible to the optimal value under all scenarios. This is also true for the minmax relative regret criterion. The minmax relative regret criterion is used to find the ratio of the cost of the chosen plan compared to the cost of the optimal plan, which can be considered as the percentage deviation of the given plan from the optimal one [15]. Therefore, it measures the degree of (sub) optimality of a plan under a scenario. This criterion is more appropriate when the decision maker is interested in the percentage of loss rather than the actual value [15].

An important advantage for minmax, minmax regret and minmax relative regret criteria is that they only need basic information about the imprecise parameters (i.e. the interval of the minimum and maximum expected value of the parameter) unlike other approaches which require more information such as the probability distribution of values [3, 119]. On the other hand, these criteria are not appropriate where the target is to find the best possible plan or when the decision maker is optimistic and willing to take some risk and find a better solution [3]. Moreover, in an environment

where the worst case scenario rarely occurs, then these criteria are not recommended.

The minmax criterion is viewed as a pessimistic approach since it compares plans just under their worst scenarios assuming that parameters taking their worst values. Such approach has been considered in our work among others and compared with MRO as we will see in Chapters 5 and 8. We applied minmax regret criterion in our work because we want to know how good/bad a plan compared to the optimal plan for a scenario. Moreover, in query optimisation the aim is to find a plan that can perform as close as possible to the optimal value under all scenarios and minmax regret criterion is suitable for such cases as mentioned before.

The complexity of the MRO version of a problem is often higher than that of the original problem [75]. Many optimisation problems with polynomial-time solutions turn out to be NP-hard in their MRO versions [3]. It is believed that, in general, optimising the minmax regret version of a problem is harder than the original problem [2]. In addition, if the number of scenarios is not constant, then it is highly likely that the minmax regret version is NP-hard even if the original problem is solvable in polynomial time [2].

Aissi et al. considered the minmax and minmax regret to the shortest path and minimum spanning tree problems [2]. An approximation algorithm is provided for these two problems [2]. The complexity of applying minmax and minmax regret on problems such as shortest path, spanning tree, assignment and knapsack are also considered in [3, 14], as are approximation algorithms for some of the NP-hard problems discussed [3].

In his book, Kasperski studies further problems that use minmax and minmax regret as measuring criteria and associated approximation algorithms [69]. Averbakh studied the minmax regret version of the problem of choosing p elements out of m elements, $p \leq m$, such that the total weight is minimum [13]. In this problem the weights of the elements are uncertain and modelled as intervals. The problem is proved to be NP-hard, and a heuristic algorithm with complexity $O((\min\{p, m - p\})^2 m)$ is provided which uses the means of the interval estimates of the weights.

2.9 Scheduling problems

Machine scheduling problems have attracted considerable attention and study in computer science. These problems are relevant to our study since the goal in machine scheduling is to find the optimal order for a set of jobs, while in our case we want to find the optimal order for a set of operators. In Section 2.9.1 we study a specific scheduling problem, called the Total Flow Time (TFT) problem, that uses MRO to deal with imprecise parameters. Then we discuss some optimisation approaches for the TFT problem in Section 2.9.2. This improves our understanding of MRO for the problems of selection and join ordering.

Some studies have focused on the complexity of the scheduling problems [54, 77, 78, 114] and their applications [72], while others have focused on approximation algorithms [36, 75, 84, 116]. Machine scheduling comes in many versions, such as single-machine job scheduling and multiple-machine scheduling [107]. In some literature, the problem of job scheduling on multiple machines is known as the load balancing problem [7].

One exhaustive study on the complexity of different machine scheduling problems was undertaken by Lenstra et al. [77]. A more recent study is by Allahverdi et al. [6]. Lenstra et al. identify which machine scheduling problems are polynomial-time solvable and which are NP-complete. The paper classifies machine scheduling problems according to their objective functions and to the number of machines and. The NP-completeness proofs for some problems, such as minimising the total weighted completion time in single-machine scheduling and maximising the completion time in a flow-shop on multiple-machines are also provided. In [78], it is proven that scheduling jobs with precedence constraints on a single-machine according to the completion time or due date criteria is NP-complete.

2.9.1 Total flow time scheduling problem

Now let us consider the problem of job scheduling on a single machine where the objective is to find the order which minimises the total flow time (defined below). In the case where the processing time for each job is known, the ordering problem

is a deterministic problem [80]. The optimal solution in this case can be found by sorting the jobs in non-decreasing order according to their processing time. Now let us discuss the case where each job has a processing time that is uncertain [75]. We studied this problem because it is somewhat similar to the selection order problem we consider in this thesis.

Definition 2.9.1 We are given a set $J = \{j_1, j_2, \ldots, j_n\}$ of n jobs where $n \geq 2$. Each job has a processing time $t_i = [\underline{t}_i, \overline{t}_i]$ where $\underline{t}_i \geq 0$ and $\underline{t}_i \leq \overline{t}_i$. The set J is processed on a single machine. Assigning a specific value to each t_i, where $1 \leq i \leq n$, is called a *scenario*. A scenario is called an *extreme scenario* if each $1 \leq t_i \leq n$ is either equal to $\underline{t}_i$ or $\overline{t}_i$.

Definition 2.9.2 Given two jobs j_i, $j_j \in J$, job j_i is *nested* in job j_j if $\underline{t}_j < \underline{t}_i$ and $\overline{t}_i < \overline{t}_j$.

Let $X = \{x \mid x \in [\underline{t}_1, \overline{t}_1] \times [\underline{t}_2, \overline{t}_2] \times \cdots \times [\underline{t}_n, \overline{t}_n]\}$ be the set of all possible scenarios. Let $P = \{u_1, u_2, \ldots, u_m\}$, such that $m = n!$, be the set of all scheduling plans which are formed from the permutations of jobs, where $u_k = (j_{u_k(1)}, j_{u_k(2)}, \ldots, j_{u_k(n)})$.

Definition 2.9.3 The *total flow time* (TFT) of plan u_k under scenario x is [69]:

$$\begin{aligned} f(u_k, x) &= (t_{u_k(1)}) + (t_{u_k(1)} + t_{u_k(2)}) + (t_{u_k(1)} + t_{u_k(2)} + t_{u_k(3)}) + \cdots + \sum_{i=1}^{n} t_{u_k(i)} \\ &= (n)t_{u_k(1)} + (n-1)t_{u_k(2)} + (n-2)t_{u_k(3)} + \cdots + t_{u_k(n)} \\ &= \sum_{i=1}^{n} (n-i+1)t_{u_k(i)} \end{aligned} \tag{2.9.1}$$

For every scenario, there is a scheduling plan where the total flow time is minimum. Let this be represented as: $f^*(x) = min_{u_k \in P} f(u_k, x)$. The absolute regret $\gamma(u_k, x)$ for any plan u_k under scenario x is:

$$\gamma(u_k, x) = f(u_k, x) - f^*(x) \tag{2.9.2}$$

The optimal regret $R(P, X)$ which minimises the maximal regret for the set P of scheduling plans under the set X of all possible scenarios is as follows:

$$R(P, X) = \min_{u \in P}(\max_{x \in X}(\gamma(u, x))) \tag{2.9.3}$$

Given set of J jobs, let $P(J)$ be the set of possible plans for J and $X(J)$ be the set of possible scenarios. The minmax regret optimisation problem for J, which denoted as $MRO(J)$, is to find a plan whose maximum regret matches $R(P(J), X(J))$.

Applying a brute-force approach to find the optimal minmax regret scheduling plan for n jobs requires considering all $n!$ possible scheduling plans in set P under all scenarios in set X. Daniels and Kouvelis [40] proved that it is sufficient to consider only the extreme scenarios (i.e. 2^n scenarios). Therefore, the total flow time $f(u_k, x)$ of every plan u_k under each of 2^n different scenarios x can be computed first. For each scenario x, the optimal plan $f^*(x)$ for scenario x is used to calculate the regret $\gamma(u_k, x)$ of plan u_k under scenario x using Equation 2.9.2. After calculating the regret of each plan under each scenario, the maximum regret of each plan is identified. Finally, the MRO optimal plan is the one that has the smallest maximum regret.

2.9.2 TFT minmax regret optimisation approaches

The total flow time formula in Equation (2.9.1) is simpler than the cost formula for the selection ordering problem as shown in Equation (2.4.2), since it consists of only a summation of simple terms, while the cost formula for the selection ordering problem is a summation of products, each of which depends on prevouse terms in the formaul. Nevertheless, studying the TFT problem enhanced our understanding.

A special case of applying MRO to the TFT problem is studied in [75]. It considers the case of *nested* intervals of uncertain processing times, where all intervals have the same midpoint equal to zero, and it assumes that no two intervals share the same boundary value. Sorting the jobs in any uniform order (where uniform order means that the wider intervals are towards the middle of the permutation and the narrower intervals are toward the ends), solves the problem in $O(n \log n)$ time, if the number of jobs is even, while the problem is NP-hard if the number of jobs is odd. In addition, the paper shows that if the number of jobs is even, then the worst-case scenario for a uniform permutation is the one where the first half of the jobs take their maximum processing times and the last half of the jobs take their minimum processing times [75]. This propery does not hold for the selection ordering problem (although it provided insperations for the use of what we call max-min scenarios in

our heuristic described in Chapter 4).

Using the midpoint of the processing time intervals of the job can be used in an approximation algorithm for the minmax regret optimisation of the total flow time problem [40,69–71]. Using the midpoint leads to a 2-approximation: that is, the cost of the approximated solution is no more than twice that of the optimal solution [70]. The approximate solution can be found in polynomial time by simply sorting the jobs in non-decreasing order according to the midpoints of their interval processing times [69]. We tried using this approach for the selection ordering problem. However, considering only the midpoint selectivities of the selection operators does not guarantee a MRO solution that does not exceed twice the optimal solution in terms of regret (more detail is provided in Section 3.4.4). There are many other approaches to approximate the minmax regret of total flow time job scheduling. Some of these approaches apply the so-called stability approach to the problem [110, 111].

A method known as adjacent pairwise interchange has been used by [88, 109] to solve the problem of the total flow time job scheduling. This method uses a rank to sort the jobs. The results presented in [109] hold for unconstrained sequencing, i.e. there are no precedence or preference orderings among the jobs, while in [88] more general cases are covered. A proven property states that if two adjacent jobs are not in preference order, then swapping these jobs will result in a better scheduling without increasing the cost [109]. Therefore, the main idea of this approach is to compare the cost of a given schedule and the cost of the same schedule but with the positions of two neighbouring jobs being swapped. A better schedule is produced if the cost after the swap is smaller than the cost of the original plan [88]. An enhancement of adjacent pairwise interchange is called adjacent *sequence* interchange, which proposes swapping two sequences of jobs instead of swapping two indeviual jobs. Conditions and limitations are provided in [88]. Adjacent pairwise interchange is useful in the join ordering problem, as we saw in Section 2.5.3.

For minmax regret total flow time scheduling, [40, 75] show that the worst-case scenario for any schedule is an extreme one (i.e. contains only minimum and maximum values). We have proved that the same holds for our selection ordering problem as we show in Section 3.4.2. In the case where one job j_i dominates another job

j_k, [40, 75] show that j_i must precede j_k in the optimal minmax regret solution and this is the case in our problem as well (more details are provided in Section 3.4.3). In total flow time scheduling, the worst-case scenario x of any given schedule u can be easily determined by comparing u to the optimal schedule $f^*(x)$ [40]. If a job appears in u before it appears in $f^*(x)$, then the job will take its maximum processing time in the worst-case scenario x [69]. However, if the job appears in u after or at the same position as it appears in $f^*(x)$, then it will take its minimum processing time in x [69]. This property does not hold for the selection ordering problem. Consider the following counter-example:

Example 2.9.1 Let $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7\}$ be a set of selection operators, with selectivities $s_1 = [0.0, 1.0]$, $s_2 = [0.05, 0.97]$, $s_3 = [0.07, 0.92]$, $s_4 = [0.247, 0.76]$, $s_5 = [0.248, 0.68]$, $s_6 = [0.258, 0.67]$ and $s_7 = [0.37, 0.66]$. All operators have the same cost 1 and the relation has cardinality $\Omega = 1$. If we consider plan $p = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6\sigma_7$, we find that its worst case scenario is $x = (\overline{s}_1, \overline{s}_2, \overline{s}_3, \overline{s}_4, \overline{s}_5, \underline{s}_6, \underline{s}_7)$ and the optimal plan for scenario x is $p_{opt(x)} = \sigma_6\sigma_7\sigma_5\sigma_4\sigma_3\sigma_2\sigma_1$. Operator σ_4 appears in the same position in both p and $p_{opt(x)}$, however it is assigned its maximum selectivity in the worst-case scenario. Also, operator σ_5 appears in p after its position in $p_{opt(x)}$, however it is assigned its maximum selectivity in the worst-case scenario x.

◇

2.10 Conclusion

This chapter has provided a review of relevant related work. We started by reviewing query languages in general and the relational algebra in particular. Then we discussed how query is processed in DBMSs. After that the query optimisation and evaluation are discussed as well as how optimisers work. We considered problems related to the statistical data that is used by query optimisers, such as its accuracy and availability, and how these problems affect the quality of plans produced by the optimiser. We also defined the selection ordering and join ordering problems which are the focus of this thesis.

Moreover, we have reviewed optimisation under inaccurate statistics and the

state of the art approaches in this area. In this thesis, we solve the selection/join ordering problems under impresice statistica using minmax regret optimisation (MRO), an approach used in decision theory. We reviewed decision theory and some related problems, such as the total flow time problem, that use MRO. In the following chapters we will study in more detail the problems of selection ordering and join ordering under impresise statistics, in addition to presenting the heuristics we developed in order to solve the both problems.

Chapter 3

The Selection Ordering Problem

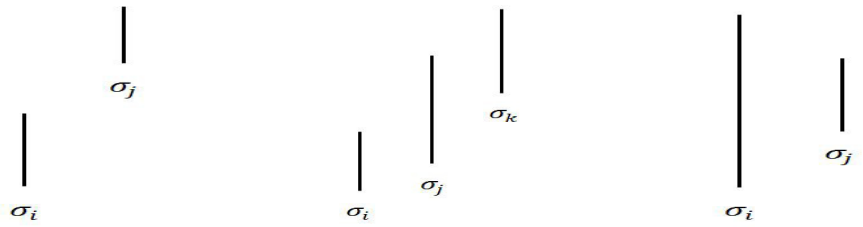
This chapter first provides a formal definition of the ordering problem for selection operators. Then, it defines the minmax regret optimisation problem for selection operator ordering and explains the brute force approach to solve the problem. After that, a number of properties of the problem will be identified, followed by some special cases which are solvable in polynomial time. Some of the work presented in this chapter has been published in papers [9] and [10].

3.1 Basic problem definition

This section presents a formal definition for the general problem of selection ordering where the selectivities are defined partially and fall within some particular interval of values. The costs of operators can also be assumed to be within some interval, but we will restrict ourselves to partially defined selectivities. As mentioned in Sections 2.4.1 and 2.4.2, it is important in logical query optimisation to find the best order in which to evaluate either a conjunction predicates or a set of selection operators. Similar to Section 2.4.1, in the following definitions we consider the problem of ordering a set of selection operators.

Definition 3.1.1 Given a set $S = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ of selection operators, each has a selectivity s_i and a cost c_i. Each selectivity is defined by a closed interval: for $1 \leq i \leq n$, $s_i = [\underline{s}_i, \overline{s}_i]$ with $\underline{s}_i, \overline{s}_i \in [0, 1]$ and $\underline{s}_i \leq \overline{s}_i$. For $1 \leq i \leq n$, $c_i \in \mathbf{R}^+$ represents the cost of σ_i for processing an input tuple.

Due to imprecise statistics in DBMSs, the selectivity s_i can take any value in the closed interval $[\underline{s}_i, \overline{s}_i]$. Similar to other works (e.g. [18,28,37,46,81]), we start solving the selection ordering problem by assuming that the selectivities of operators are independent of each other. It is true that this assumption is a limitation, however it is still used in some modern optimisers [46, 79]. Various relationships between two selection operators can be defined based on their selectivity intervals. In the following we define some relationships that play significant roles, as we will see later in Section 3.5.



(a) Strictly dominant (b) Dominant overlapped (c) Nested

Figure 3.1: Possible relationships between selection operators.

Definition 3.1.2 Given two selection operators $\sigma_i, \sigma_j \in S$, we say that σ_i *dominates* σ_j if $\underline{s}_i \leq \underline{s}_j$ and $\overline{s}_i \leq \overline{s}_j$. A dominant set is a set of selection operators $S = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ where for each pair of operators $\sigma_i, \sigma_j \in S$, either σ_i dominates σ_j or σ_j dominates σ_i.

Figures 3.1(a) and 3.1(b) show different sets of dominant operators, where the vertical dimension indicates selectivity. The domination relationship has properties which are helpful in optimising selection orders. This will be discussed in more detail in the following sections. Now consider the following definition.

Definition 3.1.3 Given two selection operators $\sigma_i, \sigma_j \in S$, we say that σ_i and σ_j are *equal* operators if $\underline{s}_i = \underline{s}_j$ and $\overline{s}_i = \overline{s}_j$. An equal set is a set of selection operators $S = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ where for each pair of operators $\sigma_i, \sigma_j \in S$, σ_i and σ_j are equal.

There are different kinds of domination based on whether a common selectivity value exists between the dominant operators or not.

Definition 3.1.4 Given two selection operators $\sigma_i, \sigma_j \in S$, we say that σ_i *strictly dominates* σ_j if $\overline{s}_i < \underline{s}_j$. A strictly dominant set S is a set of selection operators where for each pair of operators $\sigma_i, \sigma_j \in S$, either σ_i strictly dominates σ_j or σ_j strictly dominates σ_i.

Definition 3.1.4 states that operators σ_i and σ_j have no common selectivity value as shown in Figure 3.1(a). However, if they have a common selectivity value, then we call them *dominant overlapped* operators.

Definition 3.1.5 Given two selection operators $\sigma_i, \sigma_j \in S$, we say that σ_i and σ_j are *dominant overlapped* operators if $\underline{s}_i \leq \underline{s}_j \leq \overline{s}_i \leq \overline{s}_j$ but σ_i and σ_j are not equal.

An dominant overlapped set S is a set of selection operators where each pair of operators has an dominant overlapped relationship. We usually use the term dominant overlapped operators to refer to an ordered set of dominant operators sorted in non-decreasing order according to their maximum/minimum selectivities where each operator can only overlap with its immediate successor and/or immediate predecessor. This means that if σ_i overlaps with σ_j, and σ_j overlaps with σ_k then σ_i does not overlap with σ_k. This case is represented in Figure 3.1(b). However, if the term "dominant" is used alone, then it refers to domination in general.

Given two selection operators $\sigma_i, \sigma_j \in S$, if neither σ_i dominates σ_j nor σ_j dominates σ_i, then σ_i and σ_j form a *nested* pair of operators as shown in Figure 3.1(c) or an equal pair of operators. The *nested* relationship can be defined formally as follows:

Definition 3.1.6 Given two selection operators $\sigma_i, \sigma_j \in S$, we say that σ_j is *nested* in σ_i if $\underline{s}_i \leq \underline{s}_j$ and $\overline{s}_i \geq \overline{s}_j$ but σ_i and σ_j are not equal. A nested set S is a set of selection operators where for each pair of operators $\sigma_i, \sigma_j \in S$, either σ_i is nested in σ_j or vice versa.

In our setting each selectivity is known to be within some interval. However, at query run time each selection operator will have a specific selectivity value from its selectivity interval.

Definition 3.1.7 An assignment of concrete values to all n selectivities is called a *scenario* and is defined by a vector $x = (s_1, s_2, \ldots, s_n)$, with $s_i \in [\underline{s}_i, \overline{s}_i]$.

The set of all possible scenarios can be defined as $X = \{x \mid x \in [\underline{s}_1, \overline{s}_1] \times [\underline{s}_2, \overline{s}_2] \times \cdots \times [\underline{s}_n, \overline{s}_n]\}$. A query evaluator encounters one specific scenario each time it runs a query. However, it is unaware of which scenario it will encounter during the optimisation step. In Section 3.4 we will see that not all scenarios in X are important. Actually there are specific types of scenarios we are interested in which are defined next [8].

Definition 3.1.8 An *extreme scenario* is a scenario $x_{ext} = (s_1, s_2, \ldots, s_n)$ in which, for each $1 \leq i \leq n$, s_i is equal to either $\underline{s}_i$ or $\overline{s}_i$.

Recall from Section 2.4.1 that π^n is the set of all possible permutations over $1, 2, \ldots, n$ and for $\pi_j \in \pi^n$, $\pi_j(i)$ denotes the i-th element of π_j. Recall also a query execution plan p_j which is a permutation $\sigma_{\pi_j(1)}, \sigma_{\pi_j(2)}, \ldots, \sigma_{\pi_j(n)}$ of the n selection operators where the set of all possible query execution plans is given by:

$$P = \{p \mid p = \sigma_{\pi(1)}, \sigma_{\pi(2)}, \ldots, \sigma_{\pi(n)} \text{ such that } \pi \in \pi^n\}.$$

Let us rewrite Equation (2.4.2) to define the cost of evaluating plan p_j under a given scenario x as follows:

$$\begin{aligned} \text{Cost}(p_j, x) &= \Omega\left(c_{\pi(1)} + s_{\pi(1)}c_{\pi(2)} + s_{\pi(1)}s_{\pi(2)}c_{\pi(3)} + \cdots + \prod_{i=1}^{n-1} s_{\pi(i)}c_{\pi(n)}\right) \\ &= \Omega\left(\sum_{i=1}^{n}\left(\prod_{j=1}^{i-1} s_{\pi(j)}\right)c_{\pi(i)}\right) \end{aligned} \tag{3.1.1}$$

where Ω denotes the cardinality of the relation on which we execute the selection operators, and the selection predicates are assumed to be stochastically independent.

For every scenario x there is an execution plan $p_{opt(x)}$ which has the minimal cost and a permutation $\pi_{opt(x)}$ associated with this plan. Since the selectivities of the selection operators are uncertain, a plan p_j may potentially face many different scenarios having an impact on the plan quality. Even though a given plan p_j may be optimal with respect to some scenario x, it may not be optimal when faced with another scenario y [8]. Therefore, the criterion for evaluating the optimality of a

plan p_j is different to the one used in the classical selection ordering problem. To determine the quality of a plan, we utilise minmax regret optimisation.

3.2 Minmax regret optimisation

Having defined the general problem of selection ordering, now we define the minmax regret optimisation problem for selection ordering. The regret of a plan for a given scenario is defined as follows:

Definition 3.2.1 Given a plan p and a scenario x, the absolute *regret* $\gamma(p, x)$ of p for x is:

$$\gamma(p, x) = \text{Cost}(p, x) - \text{Cost}(p_{opt(x)}, x) \tag{3.2.1}$$

The regret of a plan under its worst-case scenario is known as plan's maximal regret which can be defined as: $\max_{x \in X}(\gamma(p, x))$. The plan with the optimal regret, denoted $R(P, X)$, is the one that has the smallest maximal regret. It is formally defined as follows:

Definition 3.2.2 Given the set P of all possible execution plans and the set X of all possible scenarios, minimising the maximal regret is defined as follows:

$$R(P, X) = \min_{p \in P}(\max_{x \in X}(\gamma(p, x))) \tag{3.2.2}$$

Given a set S of selection operators, let $P(S)$ denote the set of possible plans for S and $X(S)$ denote the set of possible scenarios for S. Now consider the following definition.

Definition 3.2.3 The *minmax regret optimisation* problem for S, which we denote $MRO(S)$, is to find a plan whose maximum regret matches $R(P(S), X(S))$.

For simplicity, and when there is no confusion, we also use $MRO(S)$ to denote a plan which minimises $R(P(S), X(S))$.

3.3 Brute force approach

Minmax regret optimisation in general tries to find the plan whose maximum regret is minimum. This means that when the plan is confronted with its worst-case scenario, it will have the best performance among all other plans (when confronted with their worst-case scenarios).

The brute force approach simply works as follows. If there are n operators, there will be $n!$ different execution plans. Then, the cost of each plan can be computed based on every scenario x in the scenario set X. We show later in Theorem 3.4.2 that it is sufficient to consider only the extreme scenarios since the worst-case scenario for any plan is always an extreme one. Hence, if there are n operators, we need to consider 2^n extreme scenarios. For each scenario, the plan with the smallest cost is known as the optimal plan for this scenario and is used to calculate the regret of the plan based on this scenario using Equation (3.2.1) in Section 3.2. After calculating the regret of each plan in each scenario, the maximum regret of each plan is identified. Finally, the optimal plan is the one that has the smallest maximum regret. This is illustrated in the following example.

Example 3.3.1 Let $S = \{\sigma_1, \sigma_2, \sigma_3\}$ be a set of selection operators, with selectivities $s_1 = [0.2, 0.8]$, $s_2 = [0.3, 0.5]$ and $s_3 = [0.1, 0.4]$. For simplicity, assume that all operators have the same cost 1 and that the relation has cardinality $\Omega = 1$ (so to get the real costs, the numbers in Tables 3.1 and 3.2 have to be multiplied by the true cardinality). To find the plan which minimises the maximum regret, we can perform an exhaustive enumeration of all possible execution plans under every possible scenario. For our example, Table 3.1 shows the 48 cost values for the 6 possible plans under each of 8 extreme scenarios.

For example, consider the first plan $p_1 = \sigma_1\sigma_2\sigma_3$ under scenario $x_1 = (\underline{s}_1, \underline{s}_2, \underline{s}_3) = (.2, .3, .1)$. We first calculate $\text{Cost}(p_1, x_1)$ using Equation (3.1.1) (note that Ω and each c_i is set to 1 here):

$$\text{Cost}(p_1, x_1) = (1 + .2 + .2 \times .3) = 1.26$$

The optimal plan $p_{opt(x)}$ for any scenario x is one in which the operators are in non-decreasing order of their selectivities (assuming that all operators have the same

Plan	Scenario							
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
	$\underline{s}_1$ $\underline{s}_2$ $\underline{s}_3$	$\underline{s}_1$ $\underline{s}_2$ $\overline{s}_3$	$\underline{s}_1$ $\overline{s}_2$ $\underline{s}_3$	$\underline{s}_1$ $\overline{s}_2$ $\overline{s}_3$	$\overline{s}_1$ $\underline{s}_2$ $\underline{s}_3$	$\overline{s}_1$ $\underline{s}_2$ $\overline{s}_3$	$\overline{s}_1$ $\overline{s}_2$ $\underline{s}_3$	$\overline{s}_1$ $\overline{s}_2$ $\overline{s}_3$
$p_1 = \sigma_1\sigma_2\sigma_3$	1.26	**1.26**	1.3	1.3	2.04	2.04	2.2	2.2
$p_2 = \sigma_1\sigma_3\sigma_2$	1.22	1.28	1.22	**1.28**	1.88	2.12	1.88	2.12
$p_3 = \sigma_2\sigma_1\sigma_3$	1.36	1.36	1.6	1.6	1.54	1.54	1.9	1.9
$p_4 = \sigma_2\sigma_3\sigma_1$	1.33	1.42	1.55	1.7	1.33	**1.42**	1.55	1.7
$p_5 = \sigma_3\sigma_1\sigma_2$	**1.12**	1.48	**1.12**	1.48	1.18	1.72	1.18	1.72
$p_6 = \sigma_3\sigma_2\sigma_1$	1.13	1.52	1.15	1.6	**1.13**	1.52	**1.15**	**1.6**

Table 3.1: The cost for each plan under each scenario in Example 3.3.1.

cost 1). Alternatively we can visualise the optimal plan for any scenario as the one with the smallest cost in each column in Table 3.1 which is shown in bold face. Therefore, the optimal plan for scenario x_1 is $p_{opt(x_1)} = \sigma_3\sigma_1\sigma_2$ and its cost is:

$$\text{Cost}(p_{opt(x_1)}, x_1) = (1 + .1 + .1 \times .2) = 1.12$$

The regret of plan p_1 under scenario x_1 using Equation (3.2.1) is given by:

$$\begin{aligned} \gamma(p_1, x_1) &= \text{Cost}(p_1, x_1) - \text{Cost}(p_{opt(x_1)}, x_1) \\ &= 1.26 - 1.12 = 0.14 \end{aligned}$$

In order to find the minmax regret solution, the maximum regret of each plan needs to be found. For plan p_1, the maximum regret is 1.05 which occurs in scenario $x_7 = (\overline{s}_1, \overline{s}_2, \underline{s}_3)$, its worst-case scenario. The maximum regret for each plan is shown in bold face in Table 3.2.

Finally, we are looking for the plan with the smallest maximum regret (i.e. the smallest value in the last column of Table 3.2). As a result the minmax regret solution, $MRO(S)$, is plan $p_5 = \sigma_3\sigma_1\sigma_2$, which has the best performance among all plans when confronted with their worst-case scenarios. ◇

Plan	Scenario								Maximum Regret
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
	$\underline{s}_1$ $\underline{s}_2$ $\underline{s}_3$	$\underline{s}_1$ $\underline{s}_2$ $\overline{s}_3$	$\underline{s}_1$ $\overline{s}_2$ $\underline{s}_3$	$\underline{s}_1$ $\overline{s}_2$ $\overline{s}_3$	$\overline{s}_1$ $\underline{s}_2$ $\underline{s}_3$	$\overline{s}_1$ $\underline{s}_2$ $\overline{s}_3$	$\overline{s}_1$ $\overline{s}_2$ $\underline{s}_3$	$\overline{s}_1$ $\overline{s}_2$ $\overline{s}_3$	
$p_1 = \sigma_1\sigma_2\sigma_3$	0.14	0	0.18	0.02	0.91	0.62	**1.05**	0.6	1.05
$p_2 = \sigma_1\sigma_3\sigma_2$	0.1	0.02	0.1	0	**0.75**	0.7	0.73	0.52	0.75
$p_3 = \sigma_2\sigma_1\sigma_3$	0.24	0.1	0.48	0.32	0.41	0.12	**0.75**	0.3	0.75
$p_4 = \sigma_2\sigma_3\sigma_1$	0.21	0.16	**0.43**	0.42	0.2	0	0.4	0.1	0.43
$p_5 = \sigma_3\sigma_1\sigma_2$	0	0.22	0	0.2	0.05	**0.3**	0.03	0.12	**0.3**
$p_6 = \sigma_3\sigma_2\sigma_1$	0.01	0.26	0.03	**0.32**	0	0.1	0	0	0.32

Table 3.2: The regret for each plan under each scenario in Example 3.3.1.

It is clear that applying the brute force approach to find the optimal minmax regret solution is not practical. Even when it considers only the extreme scenarios (using Theorem 3.4.2), it still requires $n! \times 2^n$ calculations for a set of n operators. However, since MRO(S) is an NP-hard problem (as shown in Section 3.4), we implemented the brute force approach in order to evaluate the performance of our heuristic, as will be described in Chapter 5.

Going back to Example 3.3.1, it is interesting to note which scenario gives rise to the maximum regret for each plan. In this case, each worst-case scenario is such that the operators at the beginning of the plan take on their maximum selectivity followed by the remaining operators that take on their minimum selectivity. We call such a scenario a *max-min* scenario.

Definition 3.3.1 Let p be the plan $\sigma_{\pi(1)}, \sigma_{\pi(2)}, \ldots, \sigma_{\pi(n)}$. A scenario for p is called a *max-min* scenario if there is a $0 \leq k \leq n$ such that for all $1 \leq i \leq k$, $s_{\pi(i)} = \overline{s}_{\pi(i)}$, and for all $k+1 \leq i \leq n$, $s_{\pi(i)} = \underline{s}_{\pi(i)}$.

This means that the first k operators in p take on their maximum selectivity, while the rest of the operators take on their minimum selectivity. For a plan p with n operators, there are $n+1$ max-min scenarios. This special class of scenarios

is important in some of the cases which can be solved in polynomial time, as we will see in Section 3.5. Moreover, they play a major role in the heuristic we have designed, as we will discuss in Chapter 4. However, it is important to mention that, in general, a max-min scenario may not be the worst-case scenario for a plan.

3.4 MRO properties for selection ordering

Applying the brute-force approach for solving MRO(S) is not practical. Unfortunately, MRO(S) is NP-hard as we show in the next subsection. Therefore, it is essential to discover some properties of the problem in order to find cases which can be solved in polynomial time. Moreover, considering these properties allows us to develop an efficient heuristic. This section highlights some of the main properties[1].

3.4.1 Hardness of MRO for selection operator ordering

In this subsection, we show that the decision problem for general $MRO(S)$, which we call MINMAX REGRET, is NP-hard. In this version of the problem, we are given a set $S = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ of selection operators, with each operator σ_i assumed to have unit cost. We are also given a set $X = \{X_1, X_2, \ldots, X_m\}$ of scenarios, where each scenario X_j specifies a selectivity s_{ij} for each operator σ_i, $1 \leq j \leq m$ and $1 \leq i \leq n$.

To simplify the notation, let us identify a plan p with the permutation π, and from now on $\pi(i)$ is used to denote the index of the operator appearing in position i in plan π.

Below we define the decision problem MINMAX REGRET as well as the well-known NP-complete problems SET COVER and EXACT COVER BY 3-SETS.

MINMAX REGRET: given a set S of n selection operators, a set X of m scenarios, and a real number R, is there a plan whose maximum regret is less than R?

SET COVER: given a finite set A, a collection T of subsets of A, and a positive

[1]I would like to thank my supervisors for formulating the proofs in this section, which are included for completeness.

integer r, is there a subset $C = \{C_1, \ldots, C_r\}$ of T such that $\bigcup_{C_i \in C} C_i = A$, that is, such that C covers A?

EXACT COVER BY 3-SETS: given a finite set A with $|A| = 3q$ and a collection T of 3-element subsets of A, is there a subset C of T such that each element of A occurs in exactly one member of C?

It is known that a restriction of EXACT COVER BY 3-SETS which requires that each element of the set A appears in exactly three subsets of T is NP-complete [58]. Since SET COVER is a generalisation of EXACT COVER BY 3-SETS, we also have that RESTRICTED SET COVER, defined below, is NP-complete.

RESTRICTED SET COVER: given finite set A, collection T of subsets of A such that each element of A appears in exactly three subsets of T, and positive integer r, is there a subset $C = \{C_1, \ldots, C_r\}$ of T such that C covers A?

We show that MINMAX REGRET is NP-hard by reducing RESTRICTED SET COVER to it.

Theorem 3.4.1 MINMAX REGRET is NP-hard.

Proof: We reduce RESTRICTED SET COVER to MINMAX REGRET. Given an instance of RESTRICTED SET COVER represented by A, T and r, we construct an instance of MINMAX REGRET as follows. Let $|A| = m$ and $|T| = n$. Each subset C_j in T is represented by an operator σ_j in S, and each element $a_i \in A$ is represented by a scenario $X_i \in X$ such that the selectivity for operator σ_j in X_i, that is, s_{ij} is $1/(n+1)$ if $a_i \in C_j$ and 1 if $a_i \notin C_j$. Since each element of A appears in exactly three subsets, each scenario $X_i \in X$ has three selectivities of $1/(n+1)$ and $n-3$ selectivities of 1. Hence the optimal plan for each scenario has the same cost, say, p. We set R to $r - p$ and claim that there is a subset of T of size r which covers A if and only if there is a plan whose maximum regret over all scenarios is less than R.

Assume there is a subset $C = \{C_{k_1}, \ldots, C_{k_r}\}$ of T which covers A. Let π be any plan in which $\pi(i) = \sigma_{k_i}$, $1 \leq i \leq r$, that is, in which the first r operators correspond to subsets in the cover. Since C is a cover, for no scenario X_i can it be the case that the selectivity for each of the first r operators in π is 1. At worst, the first $r-1$ operators have selectivity 1, with the r'th operator having selectivity $1/(n+1)$, and

the remaining $n-r$ operators having selectivity 1. The cost of this plan is therefore $r-1+(n-r+1)/(n+1)$, which is always less than r. Hence the regret is less than $R = r - p$, where p is the cost of the optimal plan.

Now assume no subset of T of size r covers A. In other words, for every subset of size r, at least one element of A is not in any set in the subset. Hence, for every plan π, there must be some scenario in which the first r operators have selectivity 1. Finding a plan which minimises the maximum regret is the same as finding a plan which minimises the maximum cost since the cost of the optimal plan is the same for each scenario. Since every plan in this case has cost at least r, there is no plan whose maximum cost is less than r. Hence there is no plan whose maximum regret is less than $R = r - p$, where p is the cost of the optimal plan. □

Since the decision problem for general MRO(S) of selection operators is NP-hard, we considered two directions. In one direction we tried to find the exact minmax regret solution for some special cases in polynomial time, as we will see in Section 3.5. In the other direction we developed an efficient heuristic for MRO(S), as we will discuss in Chapter 4.

3.4.2 Extreme Scenarios

The following theorem shows that in order to determine the worst-case scenario of a plan, i.e., the scenario for which a plan exhibits its largest regret, we only have to check extreme scenarios.

Theorem 3.4.2 The worst-case scenario for any query plan p is always an extreme scenario.

Proof: We introduce the following notation to show that our cost formulas are piecewise linear functions:

$$L_x^\pi(y) := \sum_{i=1, i\neq x}^{y} \left(\prod_{j=1, j\neq x}^{i} s_{\pi(j)} \right)$$

$$R_x^\pi(y) := \sum_{i=y, i\neq x}^{n-1} \left(\prod_{j=1, j\neq x}^{i} s_{\pi(j)} \right)$$

where $L_x^\pi(y)$ computes the cost of plan p with the operator permutation π up to the operator at position y. We skip the operator at position x, i.e., the summand in which $s_{\pi(x)}$ appears first is left out of the sum and $s_{\pi(x)}$ is omitted in all products. Analogously, we define $R_x^\pi(y)$ which computes the cost to the end of the plan starting from position y. If we do not want to skip any operators, we simply write $L^\pi(y)$ or $R^\pi(y)$.

Consider a selection operator σ_m in p such that its selectivity s_m is not extreme, i.e., $\underline{s_m} < s_m < \overline{s_m}$. Expressing the costs of p and $p_{opt(x)}$ as a function of s_m:

$$\begin{aligned} \text{Cost}(p, x, s_m) &= L^\pi(v-1) + s_m R_v^\pi(v-1) \\ \text{Cost}(p_{opt(x)}, x, s_m) &= L^{\pi_{opt(x)}}(w-1) + s_m R_w^{\pi_{opt(x)}}(w-1) \end{aligned}$$

we see that $\text{Cost}(p, x)$ is a linear function in s_m. $\text{Cost}(p_{opt(x)}, x, s_m)$ is linear as long as $s_{\pi_{opt(x)}}(w-1) \leq s_m \leq s_{\pi_{opt(x)}}(w+1)$. If s_m leaves this range, then $p_{opt(x)}$ will change, as all operators are sorted in ascending order of their selectivity. Nevertheless, $\text{Cost}(p_{opt(x)}, x, s_m)$ is a piecewise linear function. Clearly, we can swap the positions of two operators in an optimal plan without changing its optimality if the operators have exactly the same selectivity. So if $s_m = s_{\pi_{opt(x)}}(w-1) = \cdots = s_{\pi_{opt(x)}}(w-k)$, then we can swap σ_m with σ_{w-k}. Analogously, if $s_m = s_{\pi_{opt(x)}}(w+1) = \cdots = s_{\pi_{opt(x)}}(w+k)$, then we can swap σ_m with σ_{w+k}. For the cost of the optimal plan, this means $\text{Cost}(p_{opt(x)}, x, s_m)$

$$= \begin{cases} \vdots \\ L^{\pi_{opt(x)}}(w-k-1) + s_m R_w^{\pi_{opt(x)}}(w-k-1) \\ \qquad \text{if } s_{\pi_{opt(x)}}(w-k-1) \leq s_m \leq s_{\pi_{opt(x)}}(w-k) \\ L^{\pi_{opt(x)}}(w-1) + s_m R_w^{\pi_{opt(x)}}(w-1) \\ \qquad \text{if } s_{\pi_{opt(x)}}(w-1) \leq s_m \leq s_{\pi_{opt(x)}}(w+1) \\ L_w^{\pi_{opt(x)}}(w+k) + s_m R_w^{\pi_{opt(x)}}(w+k) \\ \qquad \text{if } s_{\pi_{opt(x)}}(w+k) \leq s_m \leq s_{\pi_{opt(x)}}(w+k+1) \\ \vdots \end{cases}$$

Figure 3.2 illustrates the cost functions for p and $p_{opt(x)}$.

We show that $\text{Cost}(p_{opt(x)}, x, s_m)$ is a concave (or convex upwards) function. For our piecewise linear function this means proving that by increasing s_m (moving into

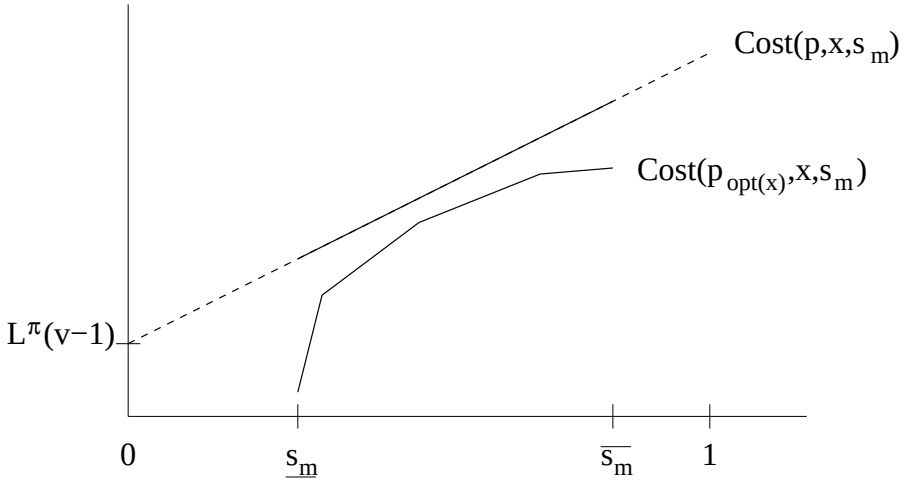


Figure 3.2: Visualisation of $\text{Cost}(p, x, s_m)$ and $\text{Cost}(p_{opt(x)}, x, s_m)$

the next piece) the slope will never increase, while if we decrease s_m, the slope will never decrease.

Increasing s_m will change the slope of $\text{Cost}(p_{opt(x)}, x, s_m)$ from $R_w^{\pi_{opt(x)}}(w-1)$ to $R_w^{\pi_{opt(x)}}(w+k)$. $R_w^{\pi_{opt(x)}}(w+k)$ is less or equal than $R_w^{\pi_{opt(x)}}(w-1)$, as they are identical except for the additional summands $w-1$ to $w+k-1$ in $R_w^{\pi_{opt(x)}}(w-1)$. Analogously, decreasing s_m will change the slope from $R_w^{\pi_{opt(x)}}(w-1)$ to $R_w^{\pi_{opt(x)}}(w-k-1)$ (which is greater or equal).

So $\gamma(p, x) = (\text{Cost}(p, x) - \text{Cost}(p_{opt(x)}, x))$ is a convex function, whose domain is restricted to a polyhedral convex set, defined by the lower and upper bounds of the selectivities. The global maximum of such a function is always found at one of the extreme points of the polyhedral convex set (Corollary 32.3.4 in [102]). □

From the above it is clear that, in finding the optimal MRO solution, it is sufficient to consider only the extreme scenarios since the worst-case scenario for any plan is always an extreme one. That means in the case of n operators, each execution plan will be confronted with only 2^n extreme scenarios.

3.4.3 Domination

Recall Definition 3.1.2, which states that for two operators $\sigma_a, \sigma_b \in S$, σ_a dominates σ_b if $\underline{s}_a \leq \underline{s}_b$ and $\overline{s}_a \leq \overline{s}_b$. In the following we show that σ_a precedes σ_b in the minmax regret solution if σ_a dominates σ_b.

Theorem 3.4.3 If σ_a dominates σ_b, then there exists a plan p minimising the maximal regret in which σ_a precedes σ_b.

Proof: Assume that p is a plan minimising the maximal regret in which σ_b precedes σ_a: $\pi(w) = b$ and $\pi(w+k) = a$. Furthermore, assume that p' is constructed from p by swapping σ_b and σ_a: $\pi'(w) = a$ and $\pi'(w+k) = b$. All the other operators are in exactly the same order as in p. We assume that p' does *not* minimise the maximal regret.

Let us investigate the difference in regret between p' and p for any given scenario x. Since the optimal plan is the same for both regrets

$$\text{Cost}(p', x) - \text{Cost}(p, x) = (s_a - s_b) \sum_{i=w}^{w+k-1} \left(\prod_{j=1, j \neq w}^{i} s_{\pi(j)} \right)$$

we only need to check what happens between positions w and $w+k-1$, as $L^\pi(w-1)$ and $R^\pi(w+k)$ (see the proof of Theorem 3.4.2 for the meaning of this notation) are identical for both p and p'.

Because this holds for every scenario, it also holds for the worst-case scenario y' of p'. Let us assume first that the selectivities for σ_a and σ_b in y' are either $(\underline{s_a}, \underline{s_b})$, $(\underline{s_a}, \overline{s_b})$, or $(\overline{s_a}, \overline{s_b})$. Since σ_a dominates σ_b, we know that $\underline{s_a} \leq \underline{s_b}$, $\underline{s_a} \leq \overline{s_b}$, and $\overline{s_a} \leq \overline{s_b}$. Therefore,

$$(s_a - s_b) \sum_{i=w}^{w+k-1} \left(\prod_{j=1, j \neq w}^{i} s_{\pi(j)} \right) \leq 0$$

which means that the maximal regret of p' cannot be greater than that of p. This also holds for strict domination, i.e., when $\overline{s_a} \leq \underline{s_b}$. However, this is a contradiction to our assumption. Thus, for the worst case scenario y', we must have the selectivities $(\overline{s_a}, \underline{s_b})$ with $\overline{s_a} > \underline{s_b}$.

Let us look at a scenario y'' which is identical to y' except for $s_a = s_b = s_c$ with $\overline{s_a} > s_c > \underline{s_b}$ (see Figure 3.3). From Theorem 3.4.2 we know that the regret can be increased by moving to an extreme scenario. In this case s_a has to be increased from s_c to $\overline{s_a}$ and s_b has to be decreased from s_c to $\underline{s_b}$ to reach the maximal regret $\gamma(p', y')$.

Clearly, $\gamma(p, y'') = \gamma(p', y'')$. The following is illustrated in Figure 3.3. Increasing s_b from s_c to $\overline{s_a}$ and decreasing s_a from s_c to $\underline{s_b}$ for p under scenario y'' (dotted

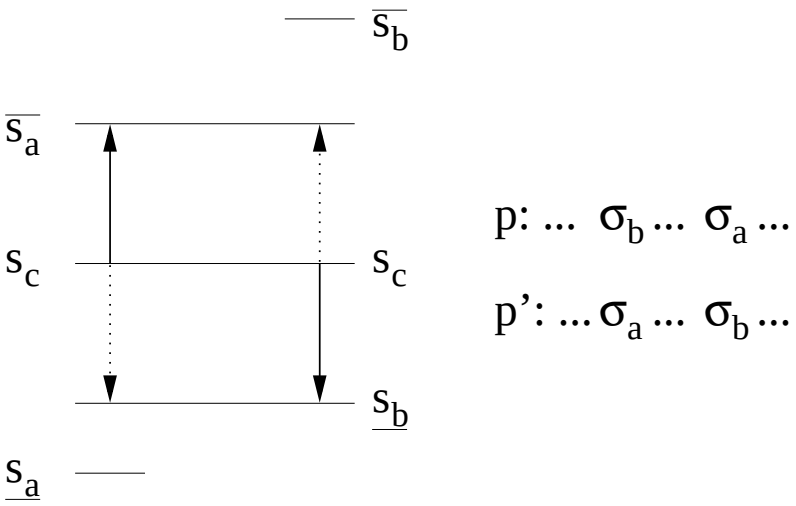


Figure 3.3: Visualisation for scenario y''

arrows) will have exactly the same effect as increasing s_a and decreasing s_b for p' under scenario y'' (solid arrows). However, this may not be an extreme case scenario for p yet. Further increasing s_b to $\overline{s_b}$ and decreasing s_a to $\underline{s_a}$ can never decrease the regret (according to Theorem 3.4.2). But that means we have found a scenario for p which has at least the same regret as the worst-case scenario for p', which contradicts our assumption. □

Applying Theorem 3.4.3 to each pair of dominant operators will place the dominant operators in the correct position relative to each other in the minmax regret solution. Consequently for a problem with unit cost for each operator and only dominant operators, the minmax regret solution is the one where the operators are sorted in non-decreasing order according to their minimum (or maximum) selectivity value. Obviously finding this solution can be done in polynomial time. We will have more discussion on this in Section 3.5.

3.4.4 Midpoints of Intervals

For the TFT problem, Kasperski used the simple heuristic of sorting jobs in non-decreasing order according to the midpoints of their intervals, yielding a 2-approximation [69]. This approach does not guarantee a bound for the MRO(S) selection ordering, as shown below. The quality of the midpoint solution can become arbi-

trarily bad. Before showing this, we need the following lemma.

Lemma 3.4.4 Given a query plan p and a scenario x, we have the following relationship between the summands in $\text{Cost}(p, x)$ and $\text{Cost}(p_{opt(x)}, x)$, where $p_{opt(x)}$ is the optimal plan for scenario x:

$$\prod_{j=1}^{k} s_{\pi(j)} \geq \prod_{j=1}^{k} s_{\pi_{opt(x)}(j)} \text{ for all } k \text{ with } 1 \leq k \leq n-1$$

Proof: If there exists an $s_m = 0$, then $\prod_{j=1}^{k} s_{\pi_{opt(x)}(j)} = 0$ for all k and the above holds, as $s_i \geq 0$ for all i. This is due to $s_{\pi_{opt(x)}(1)} = s_m = 0$ ($p_{opt(x)}$ sorts the selections in non-decreasing order of their selectivities according to the ranking algorithm). Thus, in the following all $s_i > 0$.

We assume there is a k for which $\prod_{j=1}^{k} s_{\pi(j)} < \prod_{j=1}^{k} s_{\pi_{opt(x)}(j)}$ (proof by contradiction). Let $\pi^k = \{\pi(i) \mid 1 \leq i \leq k\}$ be the set of indexes of the first k selection operators in p and $\pi^k_{opt(x)} = \{\pi_{opt(x)}(i) \mid 1 \leq i \leq k\}$ the set of indexes of the first k selection operators in $p_{opt(x)}$. If $\pi^k = \pi^k_{opt(x)}$, then the two products are equal, which is a contradiction to our assumption. So in the following we assume $\pi^k \neq \pi^k_{opt(x)}$.

Nevertheless, the intersection between π^k and $\pi^k_{opt(x)}$ may be non-empty. In this case we can discard all the selectivities common to both products:

$$\prod_{j \in \pi^k \cap \pi^k_{opt(x)}} s_j \prod_{j \in \pi^k \setminus \pi^k_{opt(x)}} s_j < \prod_{j \in \pi^k \cap \pi^k_{opt(x)}} s_j \prod_{j \in \pi^k_{opt(x)} \setminus \pi^k} s_j$$
$$\Leftrightarrow \prod_{j \in \pi^k \setminus \pi^k_{opt(x)}} s_j < \prod_{j \in \pi^k_{opt(x)} \setminus \pi^k} s_j$$

Therefore, there exists $i \in \pi^k \setminus \pi^k_{opt(x)}$ such that $s_i < \max(s_l \mid l \in \pi^k_{opt(x)} \setminus \pi^k)$ (or the inequality would not hold). However, that means σ_i has appeared in p but not yet in $p_{opt(x)}$. This is a contradiction: the selection operators are sorted in non-decreasing order in $p_{opt(x)}$ and σ_i should have appeared in $p_{opt(x)}$ before the selection operator with selectivity $\max(s_l \mid l \in \pi^k_{opt(x)} \setminus \pi^k)$. □

To show that using the midpoint does not guarantee a bound for MRO(S), let us consider the following. Given a set S with $2n+1$ operators, let the first n operators have the selectivities $\underline{s_i} = 0$ and $\overline{s_i} = 1$ $(1 \leq i \leq n)$, while the next n operators have the selectivities $\underline{s_i} = \overline{s_i} = 0.5 + \epsilon$ $(n+1 \leq i \leq 2n)$ for some small ϵ. The final

operator has a constant selectivity of 1 to guarantee that it will always be in last position, meaning that its selectivity will not impact any further steps.

The midpoint heuristic will give a plan p by ordering the operators in exactly this way: $\sigma_1 \cdots \sigma_{2n+1}$ from 1 to $2n+1$. Clearly, the worst-case scenario x for this plan is when s_i is set to 1 for $1 \leq s_i \leq n$. In the optimal plan for this scenario x, the operators σ_i with $n+1 \leq i \leq 2n$ will be executed first.

The regret of this plan is computed as follows:

$$\begin{aligned}\gamma(p,x) = \quad & 1 \quad + \quad 1^2 \quad \ldots \; + \quad 1^n \quad + \; f(n) \\ - \quad & (0.5+\epsilon) \; - \; (0.5+\epsilon)^2 \quad \ldots \; - \; (0.5+\epsilon)^n \; - \; g(n)\end{aligned}$$

where $f(n)$ and $g(n)$ stand for the cost of the remaining operators in the plan. Using Lemma 3.4.4, we know that $f(n) \geq g(n)$. Therefore, a lower bound for this regret expression is the following:

$$\text{Midpoint lower regret bound} = n - n(0.5+\epsilon) = n(0.5-\epsilon) \tag{3.4.1}$$

With increasing n and having small value for ϵ, this expression can get arbitrarily large. Consider the following example.

Example 3.4.1 Using Equation (3.4.1), suppose that $n = 10$ and $\epsilon = 0.001$. Then the lower bound regret for the midpoint solution is as follows:

$$\begin{aligned}\text{Midpoint lower regret bound} &= 10(0.5 - 0.001) \\ &= 4.99\end{aligned}$$

◇

Despite the above result, we do evaluate the midpoint heuristic in Chapter 5.

3.5 Polynomial solvable cases

In this section we show that, for a set of selection operators S that satisfies some particular properties, $MRO(S)$ can be found in polynomial time. In some of these cases we can also characterise the worst-case scenario for the optimal solution. As before, we assume throughout this section that the cost of each operator is one. Before introducing the polynomial-time cases, it is helpful to recall the definitions

of the relationships between the operators from Section 3.1, such as domination (including strictly and overlapped operators), nested and equal operators.

3.5.1 Constant and dominant operators

Let S be a set of selection operators such that the selectivity of each operator can be estimated accurately (i.e., each selectivity is constant). Then, according to Section 2.4, the minmax regret solution can be found in polynomial time by simply sorting the operators in non-decreasing order according to their rank given by Equation (2.4.1). However, if all operators have the same cost as in our assumption, then just sorting the operators in non-decreasing order of their *selectivities* will find the optimal solution $MRO(S)$.

Now consider the general case of domination. Recall from Section 3.1 that a dominant set S of operators is one in which, for each pair $\sigma_i, \sigma_j \in S$ either σ_i dominates σ_j or σ_j dominates σ_i. Also recall Theorem 3.4.3 from Section 3.4.3. This theorem allows us to conclude that the minmax regret solution is one where the operators are sorted in non-decreasing order according to their minimum (or maximum) selectivity values. Note that a set of constant operators is a special case of a dominant set. As a result we can state the following.

Corollary 3.5.1 If S is a dominant set of operators, then $MRO(S)$ can be solved in $O(n \log n)$ time.

The $O(n \log n)$ complexity in Corollary 3.5.1 is basically the complexity of sorting. It is interesting to note that the regret value of the minmax regret solution for a set S of constant or strictly dominant operators is always zero. This is because the operators in $MRO(S)$ under any scenario are in exactly the same order as the corresponding optimal plan.

3.5.2 Strictly dominant operators with constant operators

The minmax regret solution for a set S of *strictly* dominant operators, can be found in the same way as in the general case of domination. However, the problem becomes more difficult when we include nested operators. As a step toward solving the general

problem, here we introduce a single constant operator to be nested within one of the non-constant strictly dominant operators. Next we will define the problem formally. Let $S = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ be a set of dominant selection operators. We say that S is a set of *strictly* dominant operators if for all pair of operators $\sigma_i, \sigma_j \in S$ either σ_i strictly dominates σ_j, or σ_j strictly dominate σ_i (see Definition 3.1.4 and Figure 3.1(a)).

Let S be a strictly dominant set and σ_c be a constant operator nested within one of the non-constant operators in S, say σ_i. In this case, we know how to place the dominant operators relative to each other in $MRO(S)$ but we need to determine the position of σ_c in $MRO(S)$. Since $\underline{s}_i \leq s_c \leq \overline{s}_i$, the constant operator σ_c should be placed either immediately before or immediately after σ_i in $MRO(S)$. It turns out that the position of σ_c is determined only by comparing its selectivity value and the midpoint selectivity value of σ_i, as shown below.

Proposition 3.5.1 Let S be a strictly dominant set of n operators such that $MRO(S) = (\sigma_1, \ldots, \sigma_n)$. Let σ_c be an operator with constant selectivity s_c such that $\underline{s}_i \leq s_c \leq \overline{s}_i$, for some $1 \leq i \leq n$, and $S' = S \cup \{s_c\}$. In $MRO(S')$, σ_c is placed between:

- σ_{i-1} and σ_i if $s_c \leq (\underline{s}_i + \overline{s}_i)/2$, or
- σ_i and σ_{i+1} if $s_c \geq (\underline{s}_i + \overline{s}_i)/2$.

Proof: The operators σ_i and σ_c are always neighbours in an MRO solution (appearing after σ_{i-1} and before σ_{i+1}): any operator σ_h such that $1 \leq h \leq i-1$ dominates σ_c and σ_i, and σ_c and σ_i dominate any operator σ_j such that $i+1 \leq j \leq n$.

Let us assume that σ_i has selectivity $\underline{s}_i$ in scenario x and selectivity $\overline{s}_i$ in scenario x'. Moreover, let us define plan p that is constructed from $MRO(S)$ by placing σ_c before σ_i, $\pi(v) = c$ and $\pi(v+1) = i$. Similarly we define plan p' by placing σ_c after σ_i (i.e. $\pi'(v) = i$ and $\pi'(v+1) = c$). We consider below the two cases generated from the scenarios x and x'.

Case 1: The optimal plan $p_{opt(x)}$ places σ_i before σ_c in scenario x (the operators are sorted in non-decreasing order of their selectivities), i.e., σ_i and σ_c are at position v and $v+1$, respectively. We now compute the maximum regret of p and p' for

scenario x:

$$\begin{aligned}
\text{Cost}(p_{opt(x)}, x) &= L^{\pi}(v-1)(1+\underline{s}_i+\underline{s}_i s_c) + R^{\pi}(v+2) \\
\text{Cost}(p, x) &= L^{\pi}(v-1)(1+s_c+s_c\underline{s}_i) + R^{\pi}(v+2) \\
\text{Cost}(p', x) &= L^{\pi'}(v-1)(1+\underline{s}_i+\underline{s}_i s_c) + R^{\pi'}(v+2)
\end{aligned}$$

(Recall that the above notation was introduced in the proof of Theorem 3.4.2). As $L^{\pi}(v-1) = L^{\pi'}(v-1)$ and $R^{\pi}(v+2) = R^{\pi'}(v+2)$, we can compute the regret of p and p' as follows:

$$\gamma(p, x) = L^{\pi}(v-1)\,(s_c - \underline{s}_i) \tag{3.5.1}$$

$$\gamma(p', x) = 0 \tag{3.5.2}$$

So for scenario x, plan p has a greater regret than p' and it can be calculated by Equation (3.5.1).

Case 2: In scenario x', σ_i follows σ_c in $p_{opt(x')}$. For computing the costs of the different plans, this means:

$$\begin{aligned}
\text{Cost}(p_{opt(x')}, x') &= L^{\pi}(v-1)(1+s_c+s_c\overline{s}_i) + R^{\pi}(v+2) \\
\text{Cost}(p, x') &= L^{\pi}(v-1)(1+s_c+s_c\overline{s}_i) + R^{\pi}(v+2) \\
\text{Cost}(p', x') &= L^{\pi'}(v-1)(1+\overline{s}_i+\overline{s}_i s_c) + R^{\pi'}(v+2)
\end{aligned}$$

Consequently, the regret of p and p' is

$$\gamma(p, x') = 0 \tag{3.5.3}$$

$$\gamma(p', x') = L^{\pi}(v-1)\,(\overline{s}_i - s_c) \tag{3.5.4}$$

In this case (scenario x') plan p' has a greater regret than p and it is calculated using Equation (3.5.4).

Comparing both cases we can see that p has a smaller maximum regret than p' whenever Equation (3.5.1) < Equation (3.5.4). This is the case when $s_c < (\underline{s}_i + \overline{s}_i)/2$ and then we place σ_c before σ_i. Similarly, plan p' has a smaller maximum regret than p' whenever Eq. (3.5.1) > Eq. (3.5.4). We place σ_c after σ_i when $s_c > (\underline{s}_i + \overline{s}_i)/2$. For the breakeven point, i.e. $s_c = (\underline{s}_i + \overline{s}_i)/2$, σ_c can be placed before or after σ_i.

If σ_c comes before σ_i, then x is the worst-case scenario (σ_i having a selectivity of $\underline{s}_i$). Otherwise, when we place σ_c before σ_i, then x' is the worst-case scenario (the selectivity of σ_i being $\overline{s}_i$). □

The following proposition shows that the worst-case scenarios of the $MRO(S)$ solution described in Proposition 3.5.1 are max-min scenarios.

Proposition 3.5.2 Let S be a strictly dominant set of n operators such that $MRO(S) = (\sigma_1, \ldots, \sigma_n)$. Let σ_c be an operator with constant selectivity s_c such that $\underline{s}_i \leq s_c \leq \overline{s}_i$, for some $1 \leq i \leq n$, and $S' = S \cup \{s_c\}$. The scenario $(\overline{s}_1, \ldots, \overline{s}_{j-1}, s_c, \underline{s}_j, \ldots, \underline{s}_n)$, in which either σ_{j-1} or σ_j is equal to σ_i, is a worst-case scenario for $MRO(S')$.

Proof: From Proposition 3.5.1 we know that if $s_c \leq (\underline{s}_i + \overline{s}_i)/2$, then the minmax regret is computed by Equation (3.5.1) for plan p. In plan p, σ_i follows σ_c (so $\sigma_j = \sigma_i$) and the selectivity of σ_i is $\underline{s}_i$. The other selectivities do not influence the regret, so we can set the selectivity of the operators σ_1 to σ_{j-1} to their upper bounds and the selectivity of the operators σ_{j+1} to σ_n to their lower bounds. If $s_c \geq (\underline{s}_i + \overline{s}_i)/2$, then the minmax regret is computed by Equation (3.5.4) for plan p'. In plan p', σ_c follows σ_i (so $\sigma_{j-1} = \sigma_i$) and the selectivity of σ_i is $\overline{s}_i$. Here we choose the upper bounds for the operators σ_1 to σ_{j-2} and the lower bounds for the operators σ_j to σ_n. In both cases this results in a max-min scenario. □

Proposition 3.5.1 can be extended to the case in which each non-constant strictly dominant operator has at most one constant operator nested within it. The minmax regret solution of this case can be formed by grouping each constant operator with the interval operator in which it is nested and solving each group locally as above. Analogically to Proposition 3.5.2, the worst-case scenario of the minmax regret solution of this case is also as the above but with the follow improvements:

- Each interval operator which appears before the first group in the minmax regret solution should be set to its upper bound.

- Each interval operator which appears after the last group in the minmax regret solution can be set to its upper or lower bound and we choose the lower bound.

3.5.3 Dominant overlapped operators

Now let us consider the case of dominant overlapped operators as described earlier in Section 3.1. Here we assume that, when the operators are sorted in non-decreasing order of their minimum (or maximum) selectivities, each operator overlaps only with its immediate predecessor and its immediate successor. So we are looking at cases such as those shown in Figure 3.1(b). For example, if operator σ_i overlaps with operator σ_j and at the same time operator σ_j overlaps with operator σ_k, then σ_i and σ_k do not overlap with each other. It is clear that such setting is a special case of domination. Therefore, for a given set S of dominant overlapped operators, the $MRO(S)$ solution can be found in polynomial time by just sorting the operators in non-decreasing order according to either their maximum or minimum selectivity values.

Finding the worst-case scenario for the $MRO(S)$ solution is not as straightforward as we saw in Section 3.5.2 for the case of strictly dominant operators. This is because any operator $\sigma_i \in S$ can change its position in the corresponding optimal solution compared to its position in the $MRO(S)$ solution. The interesting fact in this setting is that any operator in $MRO(S)$ can move at most one position earlier in its corresponding optimal plan, and this is true under any scenario. Consider the following example:

Example 3.5.1 Let $S = \{\sigma_1, \sigma_2, \sigma_3\}$ be dominant overlapped set of selection operators with selectivities as follows: $s_1 = [0.1, 0.3]$, $s_2 = [0.25, 0.5]$ and $s_3 = [0.4, 0.9]$.

Since S is a dominant overlapped set, the operators are sorted in non-decreasing order according to their minimum/maximum selectivities and $MRO(S) = \sigma_1\sigma_2\sigma_3$. We notice that σ_2 overlaps both σ_1 and σ_3, but there is no overlap between σ_1 and σ_3.

We know that the optimal plan for any given scenario can be easily generated by sorting the operators in non-decreasing order according to their selectivities. Now consider scenario $x_a = (\overline{s}_1, \underline{s}_2, \underline{s}_3)$; its optimal plan is $p_{opt(x_a)} = \sigma_2\sigma_1\sigma_3$. We notice that σ_2 moves only one position earlier in $p_{opt(x_a)}$ compared to its position in $MRO(S)$ and this is the farthest move for any operators in this setting. On the other

hand, there are scenarios such as $x_b = (\underline{s}_1, \overline{s}_2, \overline{s}_3)$ with optimal plan $p_{opt(x_b)} = \sigma_1\sigma_2\sigma_3$, where the operators do not change their position compared to $MRO(S)$. $\diamond$

We have designed a polynomial time algorithm to find the worst-case scenario for the $MRO(S)$ solution. Before introducing the algorithm, we first introduce some terminology and prove some properties. Let S be the set of dominant overlapped operators that consists of $2n+1$ non-constant dominant overlapped operators. We assume that operators in S also do not have equal selectivity intervals or equal selectivity boundaries. Let plan p be the minmax regret solution $MRO(S)$. The first operator in p is represented as σ_F and the rest of the operators are grouped in n groups, each with a pair of operators represented by $\sigma_{A(i)}$ and $\sigma_{B(i)}$, where $1 \leq i \leq n$. We assume $\sigma_{A(i)}$ dominates $\sigma_{B(i)}$, $1 \leq i \leq n$, and $\sigma_{B(i)}$ dominates $\sigma_{A(i+1)}$, $1 \leq i \leq n-1$. Let plan $p^{<i}$ represent the partial plan of the minmax regret solution that only includes operator σ_F and the first $i-1$ groups. Similarly, let $x^{<i}$ denote a partial scenario for operators in plan $p^{<i}$. The regret of plan $p^{<i}$ under scenario $x^{<i}$ is written as $\gamma(p^{<i}, x^{<i})$ which we sometimes abbreviate to $R^{<i}$. The regret after adding group i with its partial scenario say $(\underline{s}_{A(i)} \cdot \overline{s}_{B(i)})$ to the operator σ_F and the first $i-1$ groups with scenario $x^{<i}$ is written as $\gamma(p^{<i+1}, x^{<i} \cdot \underline{s}_{A(i)} \cdot \overline{s}_{B(i)})$. We define $\prod^{\sigma_i}$ to be the product of the selectivites for all operators up to and including operator σ_i under the considered scenario. Now consider the following lemma.

Lemma 3.5.2 For a given set S of $2n+1$ dominant overlapped operators as defined above, a worst-case scenario for the minmax regret solution is one that starts with $\overline{s}_F$ followed by either $\left(\overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ or $\left(\underline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right)$ for each group i, $1 \leq i \leq n$.

Proof: In order to prove this lemma we will use induction on n. For the base case of $n = 1$, S includes the operators σ_F, $\sigma_{A(1)}$ and $\sigma_{B(1)}$.

Base case: We need to prove that there is no scenario with a bigger regret than the regret under scenarios $\left(\overline{s}_F \cdot \underline{s}_{A(1)} \cdot \overline{s}_{B(1)}\right)$ or $\left(\overline{s}_F \cdot \overline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right)$.

This can be verified by looking at the regret of all eight possible extreme scenarios. According to previous results, any scenario for which the operators are sorted

in non-decreasing order according to their selectivities will make the regret of the plan zero. Therefore the following are the zero regret scenarios for the base case:

$$\begin{aligned}\gamma\left(p^{<2}, \overline{s}_F \cdot \overline{s}_{A(1)} \cdot \overline{s}_{B(1)}\right) &= \gamma\left(p^{<2}, \underline{s}_F \cdot \overline{s}_{A(1)} \cdot \overline{s}_{B(1)}\right) \\ &= \gamma\left(p^{<2}, \underline{s}_F \cdot \underline{s}_{A(1)} \cdot \overline{s}_{B(1)}\right) \\ &= \gamma\left(p^{<2}, \underline{s}_F \cdot \underline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right) = 0 \qquad (3.5.5)\end{aligned}$$

The following are the scenarios where the regret is not zero:

$$\begin{aligned}\gamma\left(p^{<2}, \overline{s}_F \cdot \underline{s}_{A(1)} \cdot \overline{s}_{B(1)}\right) &= \gamma\left(p^{<2}, \overline{s}_F \cdot \underline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right) \\ &= \left(\overline{s}_F - \underline{s}_{A(1)}\right) \qquad (3.5.6)\\ \gamma\left(p^{<2}, \overline{s}_F \cdot \overline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right) &= \overline{s}_F\left(\overline{s}_{A(1)} - \underline{s}_{B(1)}\right) \qquad (3.5.7)\\ \gamma\left(p^{<2}, \underline{s}_F \cdot \overline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right) &= \underline{s}_F\left(\overline{s}_{A(1)} - \underline{s}_{B(1)}\right) \qquad (3.5.8)\end{aligned}$$

From Equation (3.5.6) we can see that $\gamma\left(p^{<2}, \overline{s}_F \cdot \underline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right)$ is never larger than $\gamma\left(p^{<2}, \overline{s}_F \cdot \underline{s}_{A(1)} \cdot \overline{s}_{B(1)}\right)$. From Equations (3.5.7) and (3.5.8) we can see that $\gamma\left(p^{<2}, \overline{s}_F \cdot \overline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right) > \gamma\left(p^{<2}, \underline{s}_F \cdot \overline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right)$, since $\overline{s}_F > \underline{s}_F$. As a result, the maximum regret for the base case is under scenario $\left(\overline{s}_F \cdot \underline{s}_{A(1)} \cdot \overline{s}_{B(1)}\right)$ or scenario $\left(\overline{s}_F \cdot \overline{s}_{A(1)} \cdot \underline{s}_{B(1)}\right)$.

Inductive case: Now assume the result holds for a plan with $i-1$ groups, where $2 \leq i \leq n$. We show that when adding group i to plan $p^{<i}$, scenario $\left(\overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ or $\left(\underline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right)$ will produce the maximum regret for group i.

Due to the special setting of this problem, any operator can move at most one position earlier in its corresponding optimal plan compared to its position in the minmax regret solution. So, in order to verify the inductive case, we only need to study operators $\sigma_{B(i-1)}$, $\sigma_{A(i)}$ and $\sigma_{B(i)}$. This is due to the fact that the rest of the operators (i.e. operators σ_F to $\sigma_{A(i-1)}$) do not change their position in the optimal plan, no matter what scenario is chosen for group i. Therefore, if there is any change in the positions of operators in the corresponding optimal plan after adding group i, it will be only by swapping:

- $\sigma_{B(i-1)}$ with $\sigma_{A(i)}$ under the scenario that has $\sigma_{B(i-1)}$ assigned its maximum selectivity and $\sigma_{A(i)}$ assigned its minimum selectivity, or

- $\sigma_{A(i)}$ with $\sigma_{B(i)}$ under the scenario that has $\sigma_{A(i)}$ assigned its maximum selectivity and $\sigma_{B(i)}$ assigned its minimum selectivity.

Now let us study the regret after adding group i and find the partial scenario $x^{<i+1}$ which maximises the overall regret. First, we consider the new scenarios (after adding group i) that are based on scenario $x^{<i}$ which ends with $\underline{s}_{B(i-1)}$. When $\sigma_{B(i-1)}$ has its minimum selectivity, there will be no swap between $\sigma_{B(i-1)}$ and $\sigma_{A(i)}$ in the new optimal plan for scenario $x^{<i+1}$, since $\underline{s}_{A(i)}$ and $\overline{s}_{A(i)}$ are always greater than $\underline{s}_{B(i-1)}$. Consequently, the regret of plan $p^{<i+1}$ in this case will include the value of $\gamma\left(p^{<i}, x^{<i}\right)$ in addition to the contributed regret value generated by adding $\sigma_{A(i)}$ and $\sigma_{A(i)}$. The following are the calculations of $\gamma\left(p^{<i+1}, x^{<i+1}\right)$ for each of the four scenarios when scenario $x^{<i}$ ends with $\underline{s}_{B(i-1)}$:

$$\begin{aligned}
\gamma\left(p^{<i+1}, x^{<i} \cdot \overline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right) &= \gamma\left(p^{<i+1}, x^{<i} \cdot \underline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right) \\
&= \gamma\left(p^{<i+1}, x^{<i} \cdot \underline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right) \\
&= R^{<i} + 0 = R^{<i} \qquad (3.5.9) \\
\gamma\left(p^{<i+1}, x^{<i} \cdot \overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right) &= R^{<i} + \prod^{\sigma_{B(i-1)}} \left(\overline{s}_{A(i)} - \underline{s}_{B(i)}\right) \qquad (3.5.10)
\end{aligned}$$

Next, let us consider the new scenarios that are based on $x^{<i}$ which ends with $\overline{s}_{B(i-1)}$. When $\sigma_{B(i-1)}$ is assigned its maximum selectivity, its position will be at the end of plan $p^{<i}$ as well as at the end of the corresponding optimal plan for scenario $x^{<i}$, since $\overline{s}_{B(i-1)}$ is larger than the selectivity of any preceding operator. Moreover, it can be noticed that the last term in the cost formula (i.e. the product of the selectivities of all operators) of the plan $p^{<i}$ and the corresponding optimal plan for scenario $x^{<i}$ will be exactly the same, hence they will cancel out in the regret formula when the cost of optimal plan is subtracted from the cost of the plan $p^{<i}$. As a result $\overline{s}_{B(i-1)}$ do not appear at the regret of $x^{<i}$ in this case. So $\gamma\left(p^{<i}, x^{<i}\right)$

can be rewritten as follows[2]:

$$\begin{aligned}\gamma\left(p^{<i}, x^{<i}\right) &= \gamma\left(p^{<i}, x^{<i-1} \cdot s_{A(i-1)} \cdot \overline{s}_{B(i-1)}\right) \\ &= \gamma\left(p^{<i}, x^{<i-1} \cdot s_{A(i-1)}\right) \\ &= R^{<i} \end{aligned} \tag{3.5.11}$$

Considering Equation (3.5.11), it is clear that even if there is a swap between $\overline{s}_{B(i-1)}$ and $\underline{s}_{A(i)}$ in the new optimal plan for scenario $x^{<i+1}$ after adding group i, $\gamma\left(p^{<i+1}, x^{<i} \cdot s_{A(i)} \cdot s_{B(i)}\right)$ will still include the regret value of $\gamma\left(p^{<i}, x^{<i}\right)$ as it is. Consequently, the new regret in this case after adding group i will include the regret value of $\gamma\left(p^{<i}, x^{<i}\right)$ in addition to that of $\sigma_{B(i-1)}$ and that generated by attaching group i to plan $p^{<i}$. The following are the calculations of $\gamma\left(p^{<i+1}, x^{<i+1}\right)$ for each of the four scenarios when $x^{<i}$ ends with $\overline{s}_{B(i-1)}$:

$$\begin{aligned}\gamma\left(p^{<i+1}, x^{<i-1} \cdot s_{A_{i-1}} \cdot \overline{s}_{B(i-1)} \cdot \underline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right) &= \gamma\left(p^{<i+1}, x^{<i-1} \cdot s_{A_{i-1}} \cdot \overline{s}_{B(i-1)} \cdot \underline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right) \\ &= R^{<i} + \prod^{\sigma_{A(i-1)}} \left(\overline{s}_{B(i-1)} - \underline{s}_{A(i)}\right) \end{aligned} \tag{3.5.12}$$

$$\gamma\left(p^{<i+1}, x^{<i-1} \cdot s_{A_{i-1}} \cdot \overline{s}_{B(i-1)} \cdot \overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right) = R^{<i} + \prod^{\sigma_{B(i-1)}} \left(\overline{s}_{A(i)} - \underline{s}_{B(i)}\right) \tag{3.5.13}$$

$$\gamma\left(p^{<i+1}, x^{<i-1} \cdot s_{A_{i-1}} \cdot \overline{s}_{B(i-1)} \cdot \overline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right) = R^{<i} + 0 = R^{<i} \tag{3.5.14}$$

Clearly the regrets in Equations (3.5.10), (3.5.12) and (3.5.13) are greater than those in Equations (3.5.9) and (3.5.14). Therefore, partial scenarios $\left(\overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ or $\left(\underline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right)$ maximise the regret for group i; hence the inductive case holds. □

Proposition 3.5.3 For a given set S of $2n+1$ dominant overlapped operators, any group i with partial scenario $\left(\overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ in the worst-case scenario of MRO(S) will be followed by $\left(\overline{s}_{A(j)} \cdot \underline{s}_{B(j)}\right)$ for all $j > i$.

Proof: This proposition can be proved by reconsidering Equations (3.5.9) and (3.5.10) in the proof of Lemma 3.5.2. These equations show the regret of plan $p^{<i+1}$ when scenario $x^{<i}$ ends with $\underline{s}_{B(i-1)}$. They show that the regret $\gamma\left(p^{<i+1}, x^{<i+1}\right)$ is maximised under the partial scenario $\left(\overline{s}_{A(i+1)} \cdot \underline{s}_{B(i+1)}\right)$. As a result, scenario

[2]Note: in the following formulas $s_{A(i-1)}$ has no overline/underline, which means that operator $\sigma_{A(i-1)}$ has whatever selectivity it was assigned in scenario $x^{<i}$.

$x^{<i+1}$ ends with an operator assigned its minimum selectivity ($\underline{s}_{B(i+1)}$) again. Consequently, all the following groups j, where $i < j \leq n$, will have the selectivities $\left(\overline{s}_{A(j)} \cdot \underline{s}_{B(j)}\right)$. □

Proposition 3.5.4 The worst-case scenario for MRO(S), where S is a set of $2n+1$ dominant overlapped operators, can be found in polynomial time.

Proof: From Lemma 3.5.2 we know that for each group i, only two partial scenarios $\left(\overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ and $\left(\underline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right)$ need to be tested in order to find a maximum regret. Once the partial scenario $\left(\overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ is found to produce the larger regret, all subsequent groups will follow the same partial scenario pattern (i.e. $\left(\overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$) as proven in Proposition 3.5.3.

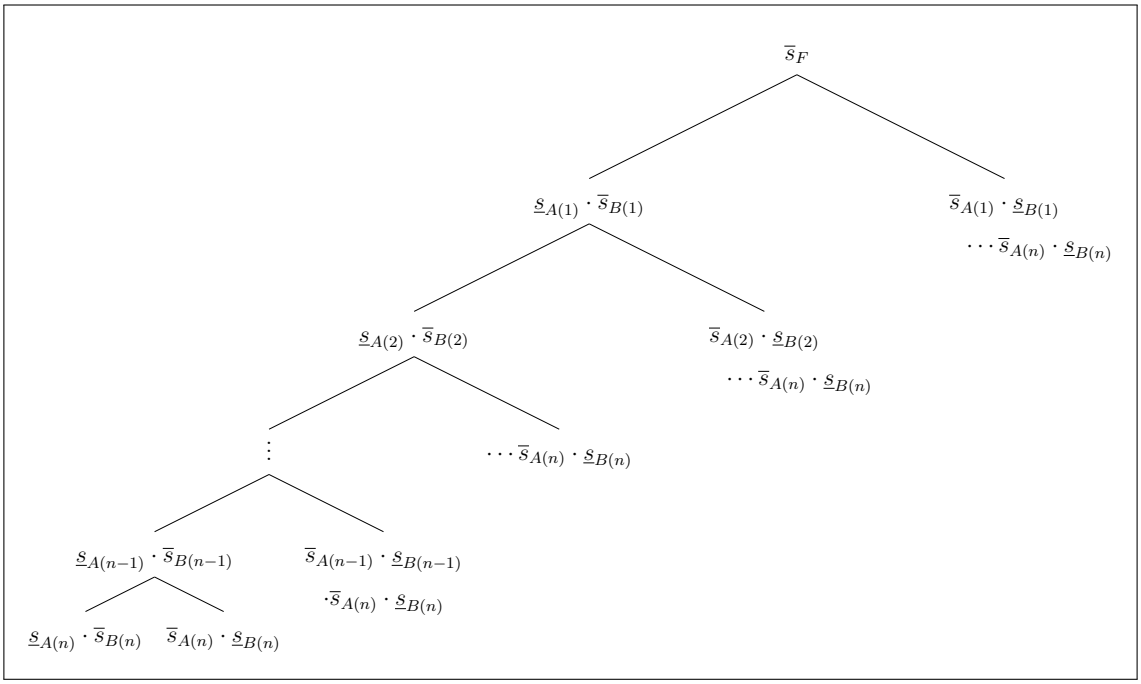


Figure 3.4: The left-deep tree for finding the worst-case scenario for MRO(S) where S comprises $2n + 1$ dominant overlapped operators.

In general, the process of choosing the partial scenario for each group forms a left-deep tree as shown in Figure 3.4. Overall we only need to consider at most $2n$ partial scenarios corresponding to the non-root nodes of the tree. □

Algorithm 2 shows how the worst-case scenario can be found in polynomial time for a set of dominant overlapped operators. It takes as input the set S, which consists

of $2n + 1$ non-constant dominant overlapped operators, and the minmax regret solution p. The algorithm starts by initialising the worst-case scenario $worstSce$ with the maximum selectivity of the first operator σ_F in the minmax regret solution as shown in line 2. Then the algorithm considers each group in turn. For each group i, where $1 \leq i \leq n$, the algorithm compares the regret of the partial plan $p^{<i+1}$ under the partial scenarios $\left(x^{<i} \cdot \overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ and $\left(x^{<i} \cdot \underline{s}_{A(i)} \cdot \overline{s}_{B(i)}\right)$ as seen in line 5. The one that results in the larger regret value for plan $p^{<i}$ will be assigned to $worstSce$. At any stage, if the scenario $\left(x^{<i} \cdot \overline{s}_{A(i)} \cdot \underline{s}_{B(i)}\right)$ causes the larger regret, the algorithm terminates and returns the worst-case scenario obtained by assigning all the following groups with the same pattern (i.e. $\overline{s}_{A(j)} \cdot \underline{s}_{B(j)}$) as proven by Proposition 3.5.3, and shown by the for clause on line 6.

3.5.4 Equal interval operators

Now consider the case when S is a set of n operators with equal selectivity intervals, that is, for each pair of operators $\sigma_i, \sigma_j \in S$, we have that $\underline{s}_i = \underline{s}_j$ and $\overline{s}_i = \overline{s}_j$, $1 \leq i, j \leq n$. In this case it is obvious that MRO(S) will be given by any permutation of $(\sigma_1, \ldots, \sigma_n)$. The following proposition shows that the worst-case scenario for $MRO(S)$ is a max-min scenario.

Proposition 3.5.5 Let S be a set of operators with equal intervals, where $MRO(S)$ is given by $(\sigma_1, \ldots, \sigma_n)$. The worst-case scenario for $MRO(S)$ is given by $(\overline{s}_1, \ldots, \overline{s}_j, \underline{s}_{j+1}, \ldots, \underline{s}_n)$, for some $0 \leq j \leq n$.

Before proving Proposition 3.5.5 we show the following lemma.

Lemma 3.5.3 Let S be a set of n operators with equal intervals. Consider any scenario x and plan p which has operators σ_a and σ_b with selectivities $\underline{s}_a$ and $\overline{s}_b$ such that $\pi(v) = a$ and $\pi(w) = b$ where $1 \leq v < w \leq n$. Then a new plan p' with at least the same regret for scenario x can be generated from p just by swapping the positions of σ_a and σ_b to be $\pi'(v) = b$ and $\pi'(w) = a$.

Proof: As we are calculating the regret of both plans p and p' under the same scenario, the optimal plan does not change. Thus, we can compare the regret of p

Algorithm 2: To find the worst-case scenario for MRO(S) where S comprises $2n+1$ dominant overlapped operators.

```
1  findWorstCaseScenario(S, p)
       Input:
         • A set S of 2n + 1 dominant overlapped operators.
         • A plan p = σ_F σ_{A(1)} σ_{B(1)} ··· σ_{A(n)} σ_{B(n)} which is the optimal minmax regret
           solution.
       Output: The worst-case scenario worstSce for MRO(S).
2      worstSce = s̄_F;
3      for 1 ≤ i ≤ n do
4          x^{<i} = worstSce;
5          if γ(p^{<i+1}, x^{<i} · s̄_{A(i)} · s_{B(i)}) > γ(p^{<i+1}, x^{<i} · s_{A(i)} · s̄_{B(i)}) then
6              for i ≤ j ≤ n do
7                  worstSce = worstSce · s̄_{A(j)} · s_{B(j)};
8              return worstSce;
9          else
10             worstSce = worstSce · s_{A(i)} · s̄_{B(i)};
11     return worstSce;
```

and p' by comparing their costs, which can be computed as follows:

$$\begin{aligned} \text{Cost}(p, x) &= L^{\pi}(v-1) + \underline{s}_a R_v^{\pi}(v-1) \\ \text{Cost}(p', x) &= L^{\pi'}(v-1) + \overline{s}_b R_v^{\pi'}(v-1) \end{aligned}$$

Since $L^{\pi}(v-1) = L^{\pi'}(v-1)$, $R_v^{\pi}(v-1) = R_v^{\pi'}(v-1)$, and $\underline{s}_a \leq \overline{s}_b$, we can immediately see that p' has at least the same regret as p. □

Now let us come back to Proposition 3.5.5 and prove it as follows:

Proof: This proposition is a generalisation of Lemma 3.5.3. Using Lemma 3.5.3 to perform multiple pairwise swaps of any operator σ_i with selectivity $\underline{s}_i$ preceding operator σ_j with selectivity $\overline{s}_j$, where $1 \leq i, j \leq n$, until there are no such pairs of operators left, can only increase the regret and will eventually produce a worst-case scenario of the required form for MRO(S). □

This result provides another case in which max-min scenarios play an important role in finding the worst-case scenario of MRO(S).

3.6 Further investigations

After introducing some properties of the *MRO* problem for selection operator ordering and identifying some polynomial solvable cases, we discuss here some further investigations. During our study of the selection ordering problem, we noticed that the case of nested operators is more difficult than others. Obviously, any good heuristic should cope with this particular case well. Although some of the approaches investigated in this section do not turn out to lead to a good heuristic for our problem, they improved our understanding and helped us in designing our novel heuristic described see in Chapter 4.

3.6.1 Towards an approximation algorithm

We started our investigation by finding a way to represent the selection operators so we can easily identify the relationships of the operators. Once we identified the relationships between the operators, we can find the relative order for the dominant operators. After that the target was to design an approximation algorithm

that can cope well with the nested operators and find the solution with a bounded approximation.

For a given set of selection operators $S = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, two lists are constructed. The first, called the *Lower list* L, has the selection operators sorted in non-decreasing order according to their minimum selectivity. The second list, called the *Upper list* U, has the selection operators sorted in non-decreasing order according to their maximum selectivity. Let $L(i)$ and $U(i)$ denote the i^{th} operator in the list L and U respectively where $1 \leq i \leq n$. Two operators are called a *neighbouring pair*, if they have a consecutive indexes in L or U. The following definition provides an alternative way of defining a pair of nested operators.

Definition 3.6.1 Let S be a set of selection operators with $\sigma_j, \sigma_k \in S$ be any two selection operators with $U(w) = \sigma_j$, $L(x) = \sigma_j$, $U(v) = \sigma_k$ and $L(y) = \sigma_k$. The operators σ_j and σ_k are *entangled* if $v < w$ and $y > x$ (as shown in Figure 3.5) or if $v > w$ and $y < x$.

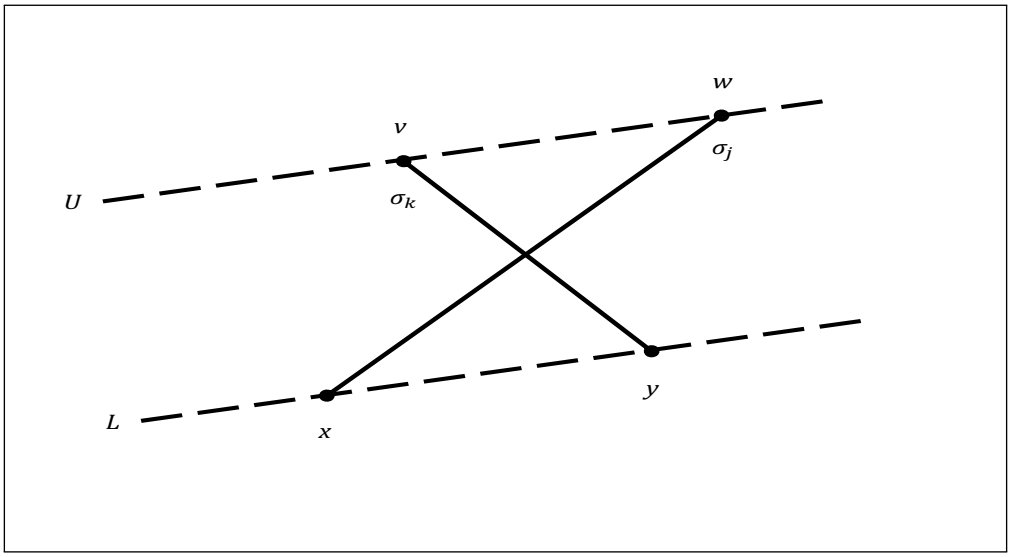


Figure 3.5: Entanglement of operators σ_j and σ_k in Definition 3.6.1.

As seen in previous sections, dealing with dominant operators is relatively easy, since we know their relative order in the optimal plan as, Theorem 3.4.3 states. However, the ordering problem becomes more complicated in the presence of nested operators. Therefore, we developed a *disentangling algorithm* to identify nested

operators and try to find a suitable order for them. The following definition suggests how the disentangling algorithm works.

Definition 3.6.2 Let σ_j, $\sigma_k \in S$ be an entangled pair as defined in Definition 3.6.1 with $U(w) = \sigma_j$, $L(x) = \sigma_j$ and $U(v) = \sigma_k$, $L(y) = \sigma_k$ such that $v > w$ and $y < x$ (as illustrated in Figure 3.5). The operators σ_j, σ_k are *disentangled* by swapping their positions in either L or U. This means that the disentangling can be done by either making $L(x) = \sigma_k$ and $L(y) = \sigma_j$, or making $U(w) = \sigma_k$ and $U(v) = \sigma_j$

If there is an entanglement, then there is at least one neighbouring pair involved in the Upper and Lower list. One of the good aspects of the disentangling algorithm is that disentangling a neighbouring pair only removes the entanglement between these neighbours and does not create any new entanglement.

The disentangling algorithm successively looks for two neighbouring operators which are entangled. In an attempt to bound the approximation, the algorithm starts with entangled neighbours which have the smallest selectivity differences in either the maximum or minimum selectivities. This procedure continues until there are no more entanglements.

In practice the disentanglements can be done by (slightly) changing the upper or lower selectivity values of the entangled operators based on in which list, U or L, they are neighbours. The aim of changing the selectivity value is to convert the nested operators into dominant operators, so that we can find their relative order in the optimal plan. The hope is that these changes in the selectivity values do not affect the quality of the result too much. There are different possible ways to perform the disentanglements. One way of disentangling two operators σ_j and σ_k is by setting their selectivity to be equal to $Max(\overline{s}_j, \overline{s}_k)$ or $Min(\underline{s}_j, \underline{s}_k)$ if the entanglement is in U or L respectively. Another possible way is by setting the selectivities of both operators to the average value of their selectivities, i.e. $\overline{s}_j = \overline{s}_k = (\overline{s}_j + \overline{s}_k)/2$ if the entanglement is in U or $\underline{s}_j = \underline{s}_k = (\underline{s}_j + \underline{s}_k)/2$ if the entanglement is in L. We implemented and tested this approach of disentanglement.

Unfortunately, keeping control of the changes in the selectivity values throughout the disentangling algorithm turns out to be difficult. This is because each time

a disentanglement is performed, a change in the selectivities is produced which in turn introduces an error. However, estimating this error is not an easy task since any changed selectivity introduces a multiplicative effect in the following terms in the cost formula in Equation (3.1.1). Moreover, the changes in the selectivities mean that the original problem is also changed which may have a significant impact on the quality of the solution. In our experimental evaluation, the result of the disentangling algorithm was up to three times worse than the optimal solution. More discussion about this approach, including an example, can be found in Appendix A.1.

3.6.2 Average midpoint heuristic

As discussed in Section 2.9, by considering the midpoint of the uncertain processing time, a 2-approximation algorithm has been proposed for TFT job scheduling problem [75]. However, using the midpoint heuristic alone does not work well for selection ordering problem as shown in Section 3.4.4. The average midpoint heuristic was an attempt to improve the performance of the midpoint heuristic, particularly for nested operators.

The formal algorithm of the average midoint heuristic, Algorithm 8, is presented in Appendix A.2. Here we give a brief overview. Basically, the heuristic first calculates the average midpoint selectivity of all operators in the given set S. This is the sum of the midpoint selectivities of each operator divided by the number of operators in S. Next the algorithm sorts the operators in non-increasing order according to the width of their selectivity intervals. The first operator in this list, which is the operator with the largest selectivity interval, is considered as the base of the solution.

Two lists are created, L and R, which hold the left and right parts of the solution respectively. The final solution will be the concatenation of list L, the operator with the largest selectivity interval, and list R. At each iteration, the heuristic considers the operator with the next biggest selectivity interval and if its midpoint is less than the average midpoint, it is placed in list L, otherwise it is placed in list R. When an operator is placed in a list, the algorithm makes sure that dominant operators are

in the correct order. All nested operators placed in L are sorted in non-decreasing order according to the width of their selectivity intervals, while those in list R are sorted in non-increasing order according to the width of their selectivity intervals.

Overall the average midpoint heuristic performs better than midpoint heuristic in handling a set of nested operators. The results of our experimental evaluation are given in Appendix A.2. Both midpoint and average midpoint heuristics have difficulty with cases in which all operators have (nearly the) same selectivity midpoint. In such cases, the average midpoint heuristic cannot decide whether to place a given operator before or after the operator with the largest selectivity interval. As a result, the solution produced by both heuristics may be arbitrarily bad. However, the average midpoint heuristic helped us in understanding the importance of passing the operators to the heuristic in a particular order. We also learned that we could gradually build a solution by placing one operator at a time, a technique used in our max-min heuristic described next in Chapter 4.

3.7 Conclusion

In this chapter we formulated the selection ordering problem under imprecise parameters. In particular, we considered the problem when the selectivities are known to fall within some interval of values. We defined minmax regret optimisation for the selection ordering problem and described a brute force approach to find a solution. After that, we proved some properties of the selection ordering problem, followed by considering special settings of the problem in which the optimal minmax regret solution can be found in polynomial time. Some further investigations were also described which, along with the identified properties, helped us in designing our novel heuristic which is presented in the following chapter.

Chapter 4

Max-min Heuristic for Selection Ordering

As seen in the previous chapter, finding MRO(S) for a set S of selection operators with interval selectivity using a brute force approach requires exponential computation and so is impractical. In this chapter we introduce our novel heuristic for solving the selection ordering problem in polynomial time.

Computing the regret of every selection ordering for every possible extreme scenario using the brute-force algorithm considers are $n!$ different orderings and 2^n extreme scenarios, given n operators. So in order to find an efficient heuristic, we have to significantly reduce the number of orderings and scenarios. While doing so, we want to leverage the insights gained from our theoretical investigation.

Two versions of our max-min heuristic are discussed in this chapter. Both use the same method but the latter improves the computational complexity of the heuristic. The max-min heuristic can be considered as a template for a number of algorithms by changing its parameters. These parameters are also discussed in the chapter. Some of the work presented in this chapter has been published in papers [9] and [10].

4.1 Basic max-min heuristic algorithm

We set three objectives when we designed the max-min heuristic. The first was to reduce the number of scenarios that the heuristic should consider, the second was

reducing the number of tested plans, and the third was that the maximum regret of the plan produced should be close to that of the MRO optimal plan. The max-min heuristic manages to accomplish these objectives. Reducing the number of scenarios by considering a subset of scenarios is a known technique to deal with hard MRO problems [3, 20, 26, 41].

Let us first look at the number of possible scenarios. As have seen in Sections 3.5.1, 3.5.2 and 3.5.4, max-min scenarios can play a special role when it comes to the maximum regret of a given plan p. Intuitively this makes sense, as in an optimal plan for p an operator σ_i located towards the beginning of p with selectivity $\overline{s_i}$ will tend to trade places with an operator σ_j located towards the end of p with selectivity $\underline{s_j}$. Consequently, there can be a large difference in cost between the plan p and an optimal plan for a max-min scenario, leading to a substantial (if not maximal) regret for p.

Lebedev and Averbakh [75] confirm the importance of max-min scenarios for MRO of the TFT problem. They identify an important class of job ordering plans, called uniform plans, for which the worst-case scenario is a max-min scenario. Such plans are ordered so that the jobs with greater uncertain processing time are in the middle of the plan, while jobs with smaller uncertain processing time are placed towards the beginning and the end of the plan. The worst-case scenario for such a plan is proved to be a max-min scenario, where the processing time for the jobs in the first half of the plan are assigned their maximum processing time, while the remaining ones are assigned their minimum processing time. In our heuristic, we only consider max-min scenarios while aiming to generate plans that perform well in term of maximum regret. In the max-min heuristic, each plan is tested under $n + 1$ max-min scenarios, as opposed to 2^n scenarios in the brute force approach.

In our heuristic we incrementally build the solution instead of considering all $n!$ possible plans. There are two well-known basic methods for doing this (efficiently). The first one is constructing a plan by combining partial plans in a way that leads to a final execution order. Very often putting the partial plans together requires using a heuristic to solve a combinatorial problem. The second method is to quickly create a complete plan (e.g., by using a simple heuristic) and then try to improve

the plan by rewriting it (e.g., by swapping or removing and re-inserting operators). In our approach we wanted to have both options available, so we decided to develop different variants. The complexity of our heuristic shows slight differences depending on the variant we use; however, the algorithms we apply all have polynomial complexity.

We refer to our max-min heuristic algorithm as $H(p, q)$. It is parameterised by two inputs: p, a (possibly empty) starting plan, and q, an order in which to process operators. Clearly, to generate a complete plan the union of p and q has to contain all the operators. If the intersection of p and q is empty, our algorithm is similar to insertion sort: we consider in turn each operator in q and place it into p at the position that minimises the regret over all max-min scenarios. If an operator in q is already present in p, then we remove it from p before re-inserting it. This is equivalent to moving an operator to a different position. Again we determine the position minimising the regret over all max-min scenarios.

Now let us discuss the basic version of the max-min heuristic in more detail. This version is easy to follow and implement, while an improved algorithm for the max-min heuristic is discussed in Section 4.3. Algorithm 3 presents the basic max-min heuristic. The outer **for** loop considers one operator t from q at a time. The algorithm checks if the current operator t already exists in the initial plan p. If so, the operator t will first be removed from the initial plan p. Operator t is then checked at each position in plan p as a result of the **for** loop at line 5.

Each time t is placed at a position in p, a new plan is formed. The regret for such a plan is calculated under all max-min scenarios and this is what the **for** loop on line 7 does. Finally, the solution for the current stage will be the plan with the smallest maximum regret involving t in p. This plan will be the initial plan p for the next stage and ultimately will form the solution for the heuristic.

It is clear that the max-min heuristic runs in time which is polynomial in the number of operators n. The **for** loop on line 1 is executed at most n times. The **if** statement on line 2 runs in $O(n)$ time in the worst case but it get dominated by the complexity of the **for** loop at line 5. The **for** loop at line 5 considers operator t in $i + 1$ possible positions, $i < n$, which requires $O(n)$ time in the worst case.

Algorithm 3: $H(p, q)$

```
1  foreach operator t from the sequence q do
2      if t is in p then
3          remove t from p;
4      Assume p currently comprises i operators;
5      foreach position j, 1 ≤ j ≤ i + 1, in p do
6          Temporarily insert t in position j in p;
7          foreach max-min scenario for p do
8              Calculate the regret of plan p;
9              Store the maximum regret for position j;
10     Choose as the final position for t in p the one that minimises the
       maximum regret;
11 Return p;
```

In each of these positions, the **for** loop at line 7 considers $i + 2$ max-min scenarios which also requires $O(n)$ time in the worst case. Line 8 calculates the regret of each plan under each max-min scenario. The cost of any plan can be calculated in $O(n)$ using Formula 3.1.1. However, we need $O(n \log n)$ to sort the operators in non-decreasing according to their selectivities in order to find the optimal plan for the max-min scenario. Therefore, the complexity for calculating the regret at line 8 is $O(n \log n) + O(n) + O(n) = O(n \log n)$, for finding the optimal plan and calculating its cost as well as calculating the cost of plan p respectively. As a result, the overall complexity is the complexity of the three nested **for** loops, which is $O(n^3)$, times the complexity of the regret calculation, which is $O(n \log n)$. Consequently, we have an $O(n^4 \log n)$ algorithm. However, by computing costs incrementally when an operator moves position and one max-min scenario moves to the next, we can implement the heuristic to run in time $O(n^3)$ as shown in Section 4.3. Figure 4.1 shows the optimisation time when using the basic max-min heuristic. To measure the time, we run the algorithm ten times for each number of operators. Then we report the average time.

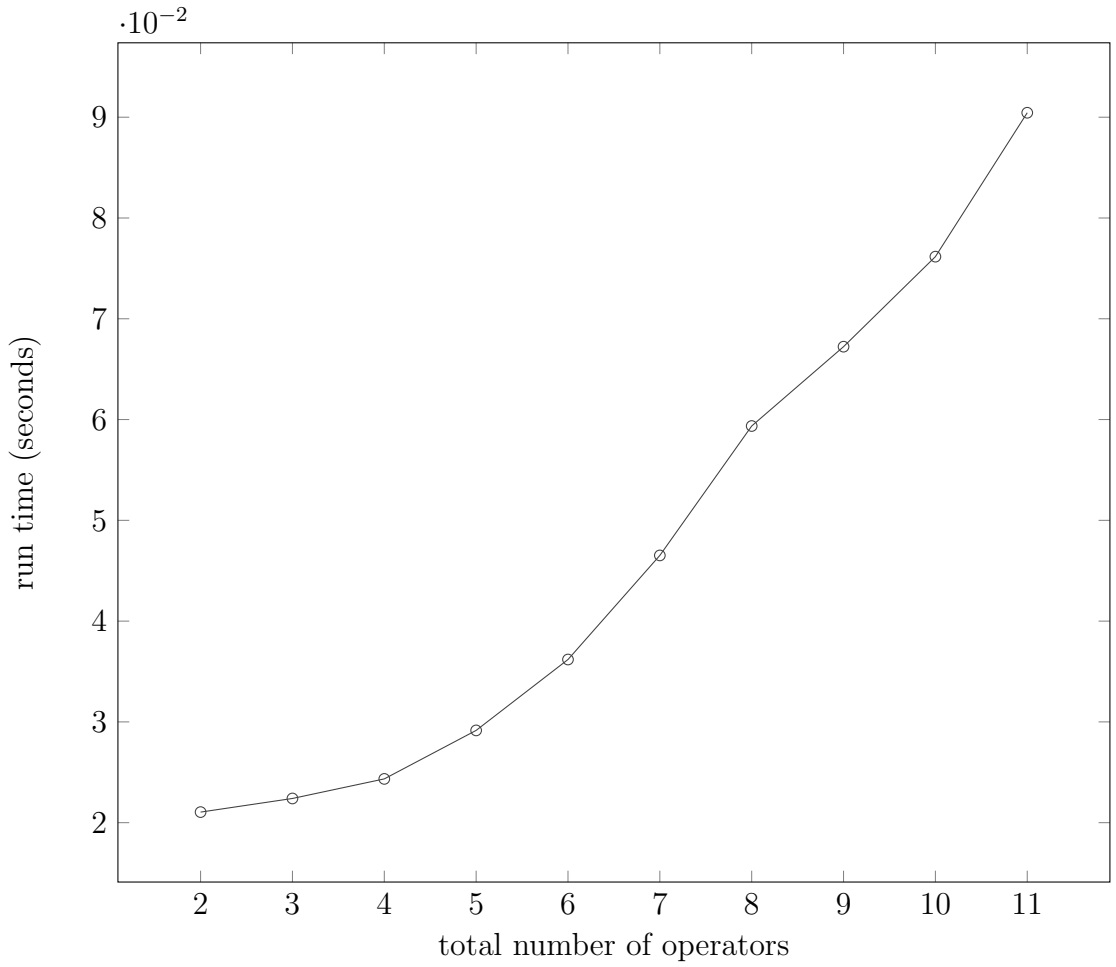


Figure 4.1: Run time of the basic max-min heuristic algorithm.

4.2 Max-min heuristic parameters

As mentioned above, the max-min heuristic $H(p, q)$ has two inputs: p, the initial plan and q, the order of passing (remaining) operators to the heuristic. In this section, we consider various criteria for choosing an initial plan and for ordering the remaining operators.

4.2.1 Choosing an initial plan

Even though we can run our heuristic with an empty initial plan p, i.e., building a solution by inserting all operators one by one, often it makes sense to start with a pre-built partial plan.

One particular and important case is that of dominant operators. Given a set S of operators, if we can identify a subset $S' \subseteq S$ of dominant operators, we know

that we can find an optimal solution p' for S' quickly and that the relative order of the operators in p' will not change in any optimal plan for S (see Theorem 3.4.3). Thus, taking p' as the initial plan when calling $H(p, q)$ makes good sense. However, there may be different ways to choose S' because in general there may be more than one such dominant set. If we have more than one dominant set, we can use one of the following criteria to make a decision:

1. Choose the subset S' with the maximum cardinality. This option is denoted by *D:C* (**D**ominant:**C**ardinality).

2. Choose the subset S' whose operators have the largest total width of their selectivity intervals. This option is denoted by *D:W* (**D**ominant:**W**idth).

3. Choose the subset S' with the maximum cardinality whose total width of the selectivity intervals is greatest. This option is denoted by *D:CW* (first **D**ominant:**C**ardinality, then **W**idth).

The following example demonstrates how the initial plan can be chosen based on the above criteria.

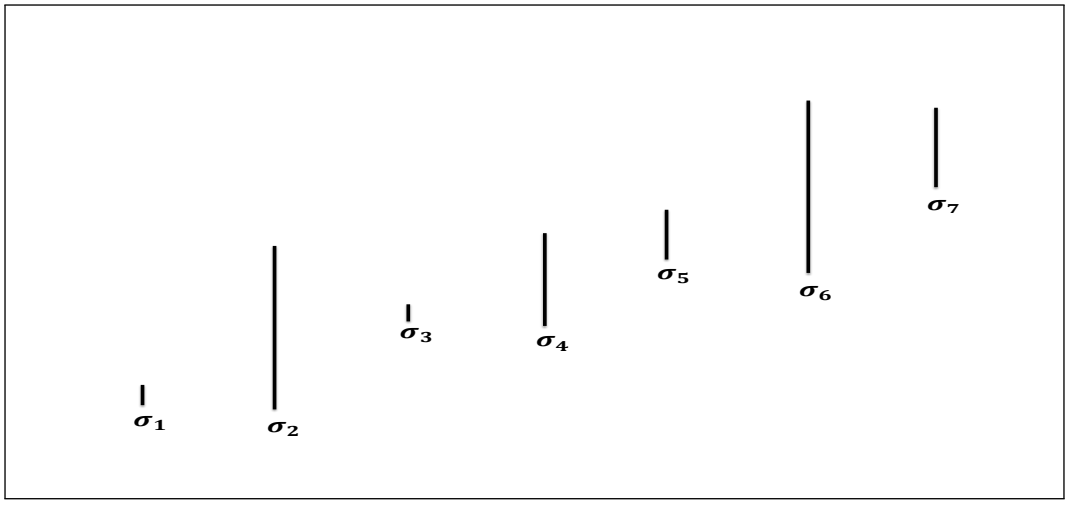


Figure 4.2: Selectivity intervals for selection operators in Example 4.2.1.

Example 4.2.1 Let $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7\}$ be set of selection operators as shown in Figure 4.2. Let the selectivities of these operators be as follows: $s_1 = [0.12, 0.14]$, $s_2 = [0.10, 0.55]$, $s_3 = [0.32, 0.33]$, $s_4 = [0.30, 0.60]$, $s_5 = [0.52, 0.67]$,

Subset	Cardinality	Total width
$S_1 = \{\sigma_2, \sigma_4, \sigma_6\}$	3	1.23
$S_2 = \{\sigma_1, \sigma_3, \sigma_5, \sigma_7\}$	4	0.43
$S_3 = \{\sigma_1, \sigma_4, \sigma_5, \sigma_7\}$	4	0.72
$S_4 = \{\sigma_2, \sigma_4, \sigma_5\, \sigma_7\}$	4	1.15

Table 4.1: Possible dominant subsets in Example 4.2.1.

$s_6 = [0.50, 0.98]$ and $s_7 = [0.70, 0.95]$. Table 4.1 shows the possible dominant subsets for S.

Referring to Table 4.1, there are different dominant subsets which can be used to build the initial plan p for the max-min heuristic $H(p, q)$. S_1 with total width equal to 1.23 will be picked if the criterion is $D : W$, which chooses the set of dominant operators whose selectivity intervals have the largest total width.

On the other hand, if we are looking for the subsets with the maximum cardinality $D : C$, then any one of S_2, S_3 or S_4 can be chosen since they have the largest cardinality of 4. As can be seen, there is more than one option for the $D : C$ criterion so we need to refine our choice. Therefore, we can use the $D : CW$ criterion instead, which distinguishes between S_2, S_3 and S_4 by choosing the subset with the largest total width of selectivity intervals. In this case that is S_4, with a total width of 1.15.

Finally, the chosen subset will be used to build the initial plan p for the max-min heuristic $H(p, q)$. It is interesting that sometimes the subset with the maximum total selectivity interval width is not the subset with the largest cardinality, as for S_1 in this example. ◇

The approach described in Section 3.6.1 can be used to find the subsets of dominant operators. However, in our implementation of the heuristic we used a graph to represent the domination relationships, where vertices represent selection operators and each edge represents a domination relationship between two operators. In this approach, the largest set of dominant operators is identified by finding the longest path using the Bellman-Ford algorithm (more details can be found in Appendix E.2).

We have evaluated the max-min heuristic experimentally using all the above initial plan options. We found that the *D:CW* approach gave the best overall results as can be seen in Chapter 5. Furthermore, we can use the output of $H(p, q)$ as input for another iteration of the heuristic in order to refine this result. We have examined this option experimentally as well and it shows its worth, as reported in the next chapter. Moreover, having an initial plan allows us to combine our algorithm with other heuristics. We can take the output of another algorithm as an initial plan p and then refine this result by running $H(p, q)$ on it. In our experiments, we feed the max-min heuristic a solution obtained by the midpoint, pessimistic and optimistic heuristics as initial plan p. Our heuristic was able to improve the results considerably. More details are provided in Chapter 5.

4.2.2 Ordering criteria

Since the max-min heuristic $H(p, q)$ originally makes only a single pass over all the operators when (re-)inserting them into the plan, the ordering of operators in q may have a significant impact on the final outcome. For example, when inserting selections into an empty initial plan, operators considered earlier in q are tested in fewer positions relative to each other compared to those considered later.

We have considered two different major ordering criteria in our experiments:

1. Passing the operators to the heuristic according to the *midpoint* of their selectivity intervals, denoted by M. Given a selectivity interval $s = [\underline{s}, \overline{s}]$, the midpoint of s is $(\underline{s} + \overline{s})/2$. The midpoint ordering itself can be sorted in:

   (a) non-decreasing order, denoted by $M+$.

   (b) non-increasing order, denoted by $M-$.

2. Passing the operators to the heuristic according to the *width* of their selectivity intervals, denoted by W. Given a selectivity interval $s = [\underline{s}, \overline{s}]$, the width of s is $\overline{s} - \underline{s}$. The W ordering itself also can be sorted in:

   (a) non-decreasing order, denoted by $W+$.

   (b) non-increasing order, denoted by $W-$.

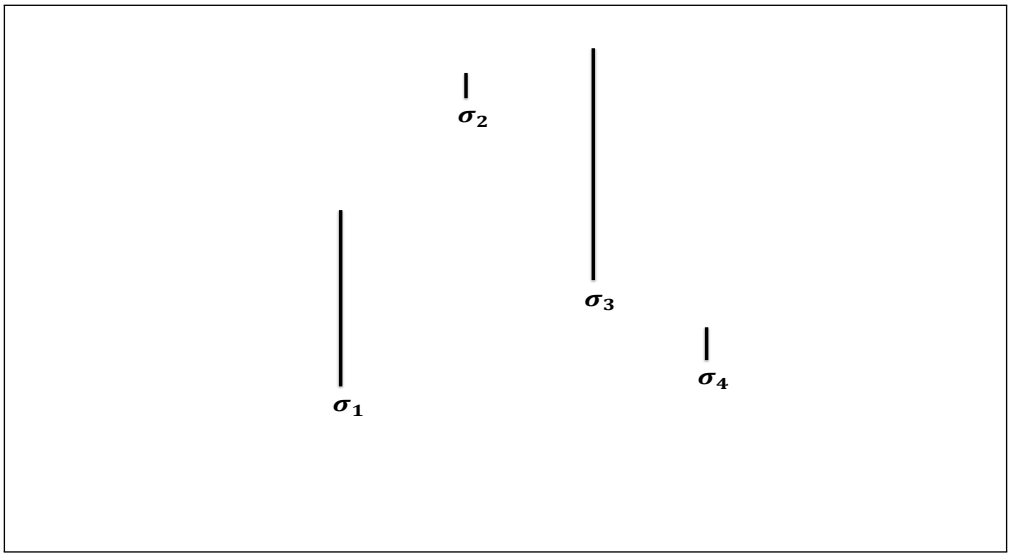


Figure 4.3: Selectivity intervals for selection operators in Example 4.2.2.

Operator	Selectivity interval	Width	Midpoint
σ_1	$s_1 = [0.14, 0.58]$	0.44	0.360
σ_2	$s_2 = [0.76, 0.81]$	0.05	0.785
σ_3	$s_3 = [0.32, 0.90]$	0.58	0.610
σ_4	$s_4 = [0.20, 0.26]$	0.06	0.230

Table 4.2: Width and midpoint for selectivity intervals of operators in Example 4.2.2.

In our experimental evaluation, we ordered the operators in q using all the above four criteria. Chapter 5 evaluates the effectiveness of these ordering criteria in more detail. Example 4.2.2 illustrates the above four ordering criteria using some concrete values.

Example 4.2.2 Consider the set $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ of selection operators with selectivity intervals as follows: $s_1 = [0.14, 0.58]$, $s_2 = [0.76, 0.81]$, $s_3 = [0.32, 0.90]$ and $s_4 = [0.20, 0.26]$ as shown in Figure 4.3.

Table 4.2 gives the width and midpoint selectivity for each operator in set S. Therefore, according to the ordering criteria and assuming an empty initial plan p, q in the max-min heuristic $H(p, q)$ will be as follows:

- If the ordering criterion is $M+$, then $q = (\sigma_4, \sigma_1, \sigma_3, \sigma_2)$.

- If the ordering criterion is $M-$, then $q = (\sigma_2, \sigma_3, \sigma_1, \sigma_4)$.
- If the ordering criterion is $W+$, then $q = (\sigma_2, \sigma_4, \sigma_1, \sigma_3)$.
- If the ordering criterion is $W-$, then $q = (\sigma_3, \sigma_1, \sigma_4, \sigma_2)$.

$\diamond$

For the naming convention of the max-min heuristic, $H(D : W, W+)$ for example means that the initial plan for the heuristic is the set of dominant operators which has the largest total width. Then the remaining operators are fed to the heuristic in non-decreasing order according to the width of their selectivity intervals. The following example describes how the max-min heuristic works.

Example 4.2.3 Recall from Example 4.2.2 the set $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ of selection operators, with selectivities $s_1 = [0.14, 0.58]$, $s_2 = [0.76, 0.81]$, $s_3 = [0.32, 0.90]$ and $s_4 = [0.20, 0.26]$. Suppose that we are using the version $H(D : CW, W+)$ of our max-min heuristic.

Figure 4.3 shows that there is more than one dominant subset. These dominant subsets are: $\{\sigma_1, \sigma_2\}$, $\{\sigma_1, \sigma_3\}$, $\{\sigma_2, \sigma_4\}$ and $\{\sigma_3, \sigma_4\}$. Since $D : CW$ is the chosen criterion for deciding the initial plan, $S' = \{\sigma_1, \sigma_3\}$ is the subset with the largest cardinality and maximum total width (see Table 4.2). The optimal plan for S' will be the value of the parameter p in the heuristic. According to Theorem 3.4.3, sorting the operators of S' in non-decreasing order according to their maximum/minimum selectivity values will form the optimal plan for S', therefore $p = \sigma_1, \sigma_3$.

Once p is specified, the remaining operators are assigned to q according to the specified order. In this case the order is $W+$, so the operators are sorted in non-decreasing order according to the width of their selectivities. Table 4.2 shows the selectivity width for each operator; hence $q = \sigma_2, \sigma_4$.

The heuristic then operates as follows. The first operator in q is checked in each position of p. For each position and resulting plan, the regret is calculated under all max-min scenarios. In this case σ_2 should be checked in three positions: before σ_1, after σ_3 and between them. Consider the plan in which σ_2 is the last operator of p, for instance. The regret will be calculated for the max-min scenarios $(\underline{s}_1, \underline{s}_3, \underline{s}_2)$,

$(\overline{s}_1, \underline{s}_3, \underline{s}_2)$, $(\overline{s}_1, \overline{s}_3, \underline{s}_2)$ and $(\overline{s}_1, \overline{s}_3, \overline{s}_2)$. Assuming that the relation cardinality Ω and each cost c_i is set to 1, the maximum regret for plan $\sigma_1, \sigma_3, \sigma_2$ is 0.26 which occurs in scenario $(\overline{s}_1, \underline{s}_3, \underline{s}_2)$. The plan with the smallest maximum regret after placing σ_2, namely $\sigma_1, \sigma_3, \sigma_2$, will be the intermediate solution.

Now $p = \sigma_1, \sigma_3, \sigma_2$ and the same procedure will be repeated for σ_4 from q, namely σ_4 will be inserted at each position in p, and the regret of the resulting plan calculated under each max-min scenario. The plan that has the smallest maximum regret after placing σ_4 in p is $\sigma_4, \sigma_1, \sigma_3, \sigma_2$ with maximum regret of 0.12 under the scenario $(\overline{s}_4, \underline{s}_1, \underline{s}_3, \underline{s}_2)$. Since there are no more operators in q, this is the final solution. As a matter of fact, the solution returned by the max-min heuristic is the same as the actual minmax regret solution in this case. ◇

As mentioned in Section 3.4.1, the selection ordering problem under imprecise statistics is NP-hard. The max-min heuristic is designed to find a good solution in polynomial time, by reducing the number of tested plans and only considering certain scenarios (i.e. max-min scenarios) as discussed earlier. This obviously does not guarantee to find the optimal MRO plan since the worst case scenario might not be a max-min scenario. It seems to be difficult to measure the improvement of the heuristic theoretically or to guarantee a bound. However, experimental evaluation demonstrates the effectiveness of the max-min heuristic parameters, namely choosing an initial plan and ordering criteria. In our experiments the heuristic in general finds a solution equivalent to or close to the optimal MRO plan, as we will discuss in Chapter 5.

4.3 Improved max-min heuristic

This section describes a more efficient version of the max-min heuristic, which reduces the complexity of the heuristic to $O(n^3)$. It is important to mention that this version of the heuristic follows essentially the same method as Algorithm 3 and produces the same result. However, this version is re-engineered to perform calculations more efficiently by storing various values to be reused in subsequent steps. This improved version stores the cost of the optimal plans for max-min scenarios

and reuses them. It also reuses some pre-calculated values to find the cost of a plan. Therefore, it trades (more) memory for (less) processing time.

Algorithm 4 presents the improved version of the max-min heuristic. Although the algorithm appears long, a number of computations are repeated. For example, lines 19–31 are initially the same as lines 1–15. The algorithm in lines 1–15 starts by considering the first operator σ_0 in p. At this stage, various parameters are initialised, such as *optPlan*, *optCost*, $minMaxPlan$ and $minMaxRegret$, so they can be used in the subsequent steps including the block from line 19 to line 31 when considering the rest of the operators in p.

Algorithm 4 starts with some plan p which could be generated by various methods as discussed in Section 4.2. The plan p is considered under the first max-min scenario (i.e. $j = 0$ which means the scenario with no maximum selectivities). The $leftSum$ array (line 1) is created as described in details later. This will be used in calculating the $cost(p, 0)$, the cost of p under scenario $j = 0$, as well as the cost of p under subsequent max-min scenarios. At the same time, the optimal plan and its cost for each max-min scenario is stored in the arrays *optPlan* and *optCost* respectively, as shown in lines 10 and 11. The regret of plan p is calculated under each max-min scenario and the largest regret for p is used to initialise $minMaxRegret$ as shown in line 14. Then the function *checkOperatorAtEachPosition* is called to move the first operator σ_0 to each position in plan p. Each time the operator σ_0 moves its position one step forward, the optimal plan and cost for only one max-min scenario is updated in the arrays *optPlan* and *optCost* as shown in lines 7 and 8 in function *checkOperatorAtEachPosition*. If a smaller maximum regret is found after considering all max-min scenarios, $minMaxRegret$ and $minMaxPlan$ are updated as shown in line 13. The same process is repeated to test the other operators in each position as shown by the loop starting at line 16 in Algorithm 4. Ultimately, the solution of the algorithm will be $minMaxPlan$ with the regret value stored in $minMaxRegret$.

Now let us consider the algorithm in more details. The following example describes how *optPlan* and *optCost* are initialised.

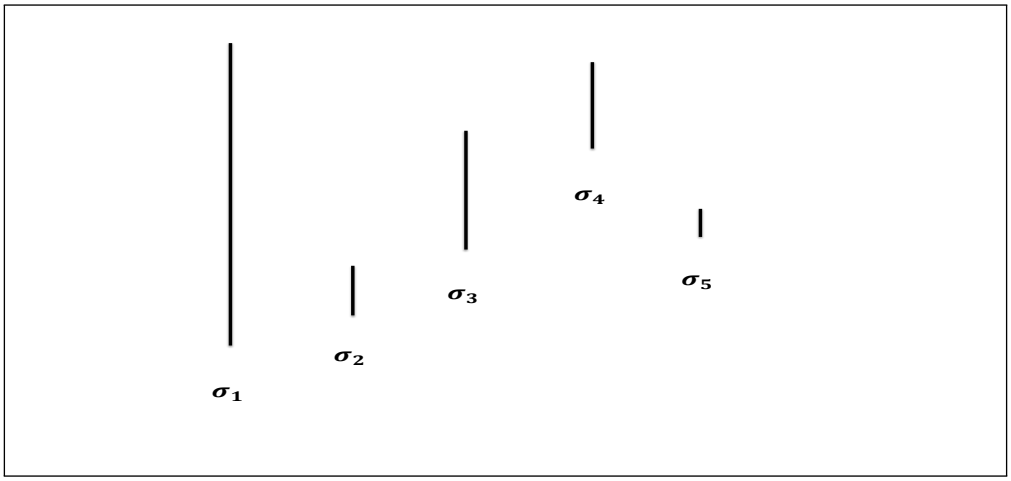


Figure 4.4: Selectivity intervals for selection operators in Example 4.3.1.

Example 4.3.1 Let set $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$ be a set of selection operators with selectivity intervals: $s_1 = [0.1, 0.9]$, $s_2 = [0.2, 0.3]$, $s_3 = [0.35, 0.7]$, $s_4 = [0.6, 0.8]$ and $s_5 = [0.4, 0.45]$ as shown in Figure 4.4. Suppose that we start with the initial plan $p = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$ and we are considering σ_1 at the first position in p. Table 4.3 shows all max-min scenarios for p and the initialisation of the array *optPlan* as in Algorithm 4. Note that in line 10, *optPlan[j]* can be found from *optPlan[j-1]* by moving left the operator whose selectivity changed from maximum to minimum. So in Table 4.3 *optPlan[5]* is obtained from *optPlan[4]* by moving s_1 left until it is located in its correct position (i.e. position one). The array *optCost* will have the cost values for the optimal plans in *optPlan* as shown in line 11. The optimal plan for any scenario can be found by sorting the operators in non-decreasing order according to their minimum or maximum selectivities as in the second column of Table 4.3. ◇

One of the interesting facts about the max-min heuristic is that when an operator is moved in a plan p from one position to an adjacent position to form plan p', only one max-min scenario (out of $n + 1$ scenarios) changes between p and p'. So we only need to update one new optimal plan and calculate its cost. More precisely, let operator σ be in position i and operator σ' be in position $i+1$ in plan p, respectively.

Algorithm 4: *Improved max − min heuristic*

Input: Given a set of n operators $O = \{\sigma_0, \sigma_1, \cdots, \sigma_{n-1}\}$.

Suppose we start with initial plan $p = [\sigma_0, \sigma_1, \cdots, \sigma_{n-1}]$.

// Throughout the algorithm z is the index of an operator in plan p, while j is the number of minimum selectivities in a max-min scenario.

// Create arrays *optPlan* and *optCost*, where *optPlan*[j] will be the optimal plan for the max-min scenario starts with j mins in plan p and *optCost*[j] will be its cost.

// Consider the first scenario (i.e. $j = 0$) with all max and no min selectivities.

1 $leftSum$ = createLeftSumArray($p, 0$);

2 $cost(p, 0) = leftSum[n-1]$;

3 Let *optPlan*[0] be the optimal plan for max-min scenario with 0 min selectivities;

4 Let *optCost*[0] be the cost of *optPlan*[0] calculated using Eq. (3.1.1);

5 $regret(p, 0) = cost(p, 0) - optCost[0]$;

6 $maxRegret(p) = regret(p, 0)$;

7 $minMaxPlan = p$;

8 **for** $1 \leq j \leq n$ **do**

  // i.e. consider the max-min scenarios with j mins.

  // In each step the selectivity for the operator at position $n - j$ in plan p changes from max to min.

9 $cost(p, j)$ = getCost(p, j, *leftSum*);

10 *optPlan*[j] is generated from *optPlan*[$j-1$] by moving left the operator that changed from max to min to the correct position such that the optimal plan is sorted;

11 Let *optCost*[j] be the cost of *optPlan*[j] calculated by Eq. (3.1.1);

12 $regret(p, j) = cost(p, j) - optCost[j]$;

13 $maxRegret(p) = Max(maxRegret(p), regret(p, j))$;

14 $minMaxRegret = maxRegret(p)$;

// Now consider σ_0 in each following position in p.

15 checkOperatorAtEachPosition(σ_0, p);

```
   // Now consider the rest of the operators (other than σ_0) at each
      position in plan p under all max-min scenarios.
16 for 1 ≤ z < n do
17    Remove σ_z from plan p and insert it at position 0 in p;
18    Re-initialise optPlan and optCost to hold the optimal plans and their cost
      respectively for the new plan p under its max-min scenarios;
      // Consider the 1^st scenario with all maxes and 0 min.
19    leftSum = createLeftSumArray(p, 0);
      // Get the cost of plan p under scenario 0.
20    cost(p, 0) = leftSum[n − 1];
21    Retrieve the optimal plan cost optCost[0];
22    regret(p, 0) = cost(p, 0) − optCost[0];
23    maxRegret(p) = regret(p, 0);
24    minMaxPlan = p;
25    for 1 ≤ j ≤ n do
         // Consider the remaining scenarios.
         // In each step the selectivity for the operator at position
            n − j in plan p changes from max to min.
26       cost(p, j) = getCost(p, j, leftSum);
27       Retrieve the optimal plan cost optCost[j];
28       regret(p, j) = cost(p, j) − optCost[j];
29       maxRegret(p) = Max(maxRegret(p), regret(p, j));
30    minMaxRegret = maxRegret(p);
      // Now consider σ_z in each following position in p.
31    checkOperatorAtEachPosition(σ_z, p);
32 return minMaxPlan;
```

Algorithm 5: Functions used by Algorithm 4

```
1 createLeftSumArray(p, j)
    Input: The plan p and the number of mins j in its max-min scenario.
    Output: Array leftSum that has the partial costs for plan p.
    // leftSum[k] is the sum of the products in the cost formula from 0
       up to the k^th term where (0 ≤ k < n).
2   for 0 ≤ k ≤ p.length do
3       leftSum[k] = Σ_{i=1}^{k+1} ( Π_{j=1}^{i-1} s_{π(j)} ) c_{π(i)};
4   return leftSum;

1 getCost(p, j, leftSum)
    Input: Plan p, the number j of minimum selectivities in the max-min scenario,
           and the array leftSum that has the partial costs for plan p.
    Output: The cost of plan p under its max-min scenario with j minimum
            selectivities
2   m = n − j;
3   if m = n − 1 then
4       cost = leftSum[m];
5   else
6       cost = leftSum[m] + (σ_m / σ̄_m) ∗ (leftSum[n − 1] − leftSum[m]);
        // Update leftSum[n-1] to be used in the next scenario.
7       leftSum[n − 1] = cost;
8   return (Ω ∗ cost);
```

```
1  checkOperatorAtEachPosition(σ, p)
       Input: The operator σ and the plan p where σ will be checked. This function
              should have access to: optPlan, optCost, minMaxRegret and
              minMaxPlan.
       Output: Updated values for minMaxRegret and minMaxPlan after checking
               σ at each position in p.
2      for each position 1 ≤ i ≤ n − 1 in plan p do
           // i.e. consider operator σ at each position in plan p other
              than the 1st position under all max-min scenarios.
           // Consider the 1st scenario with all maxes and 0 min.
3          leftSum = createLeftSumArray(p, 0);
           // Get the cost of plan p under scenario 0.
4          cost(p, 0) = leftSum[n − 1];
5          regret(p, 0) = cost(p, 0) − optCost[0];
6          maxRegret(p) = regret(p, 0);
           // Update the scenario j = n − i.
           // Note: Here we assume that operator σz is moving right in
              plan p. However, if operator σz is moving left in plan p
              then scenario j where j = n − i − 1 will be updated instead.
7          optPlan[n − i] is updated by moving left the operator that changed from
           max to min, and moving right the operator that changed from min to max
           such that optPlan[n − i] is sorted;
8          Let optCost[n − i] be the cost of optPlan[n − i] calculated by Eq. (3.1.1);
9          for 1 ≤ j ≤ n do
               // Consider the remaining scenarios.
               // In each step the selectivity for the operator at position
                  n − j in plan p changes from max to min.
10             cost(p, j) = getCost(p, j, leftSum);
11             regret(p, j) = cost(p, j) − optCost[j];
12             maxRegret(p) = Max(maxRegret(p), regret(p, j));
13         if maxRegret(p) < minMaxRegret then
14             minMaxRegret = maxRegret(p);
15             minMaxPlan = p;
```

max-min scenarios	sorted selectivities	*optPlan*	
		index	optimal plan
$(\overline{s}_1, \overline{s}_2, \overline{s}_3, \overline{s}_4, \overline{s}_5)$	$\overline{s}_2, \overline{s}_5, \overline{s}_3, \overline{s}_4, \overline{s}_1$	0	$\sigma_2\sigma_5\sigma_3\sigma_4\sigma_1$
$(\overline{s}_1, \overline{s}_2, \overline{s}_3, \overline{s}_4, \underline{s}_5)$	$\overline{s}_2, \underline{s}_5, \overline{s}_3, \overline{s}_4, \overline{s}_1$	1	$\sigma_2\sigma_5\sigma_3\sigma_4\sigma_1$
$(\overline{s}_1, \overline{s}_2, \overline{s}_3, \underline{s}_4, \underline{s}_5)$	$\overline{s}_2, \underline{s}_5, \underline{s}_4, \overline{s}_3, \overline{s}_1$	2	$\sigma_2\sigma_5\sigma_4\sigma_3\sigma_1$
$(\overline{s}_1, \overline{s}_2, \underline{s}_3, \underline{s}_4, \underline{s}_5)$	$\overline{s}_2, \underline{s}_3, \underline{s}_5, \underline{s}_4, \overline{s}_1$	3	$\sigma_2\sigma_3\sigma_5\sigma_4\sigma_1$
$(\overline{s}_1, \underline{s}_2, \underline{s}_3, \underline{s}_4, \underline{s}_5)$	$\underline{s}_2, \underline{s}_3, \underline{s}_5, \underline{s}_4, \overline{s}_1$	4	$\sigma_2\sigma_3\sigma_5\sigma_4\sigma_1$
$(\underline{s}_1, \underline{s}_2, \underline{s}_3, \underline{s}_4, \underline{s}_5)$	$\underline{s}_1, \underline{s}_2, \underline{s}_3, \underline{s}_5, \underline{s}_4$	5	$\sigma_1\sigma_2\sigma_3\sigma_5\sigma_4$

Table 4.3: Max-min scenarios for $p = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$ and the associated *optPlan* for Example 4.3.1.

If σ moves from position i in p to position $i+1$ to form a new plan p', then only scenario $n-i$ needs to be updated for plan p' by changing the selectivity of σ from its maximum to its minimum and that of σ' from its minimum to its maximum. The remaining elements in *optPlan* and *optCost* stay the same. This update is performed at line 7 in the *checkOperatorAtEachPosition* function. Therefore, this update happens once each time an operator changes its position. Since operators are tested in $O(n^2)$ plans, we need to make sure that for each new plan, only $O(n)$ time is spent in recalculating the optimal plans.

Example 4.3.2 Recall from Example 4.3.1 the set $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$ of selection operators, with selectivities $s_1 = [0.1, 0.9]$, $s_2 = [0.2, 0.3]$, $s_3 = [0.35, 0.7]$, $s_4 = [0.6, 0.8]$ and $s_5 = [0.4, 0.45]$. In Example 4.3.1, operator σ_1 is in the first position in plan $p = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$. Now let us consider moving operator σ_1 to the second position, forming the plan $p' = \sigma_2\sigma_1\sigma_3\sigma_4\sigma_5$.

Table 4.4 shows all the max-min scenarios for p'. By comparing Tables 4.3 and 4.4, it is clear that the max-min scenarios for p and p' are identical except for the scenario at index 4 (the row in bold face in Table 4.4). In particular, when an operator (σ_1 in this case) is checked in the next position, arrays *optPlan* and *optCost* can be reused to calculate the regrets of the new plan p' under its max-min scenarios after updating the content of $optPlan[n-i]$ and $optCost[n-i]$. ◇

max-min scenarios	sorted selectivities	*optPlan*	
		index	optimal plan
$(\overline{s}_2, \overline{s}_1, \overline{s}_3, \overline{s}_4, \overline{s}_5)$	$\overline{s}_2, \overline{s}_5, \overline{s}_3, \overline{s}_4, \overline{s}_1$	0	$\sigma_2\sigma_5\sigma_3\sigma_4\sigma_1$
$(\overline{s}_2, \overline{s}_1, \overline{s}_3, \overline{s}_4, \underline{s}_5)$	$\overline{s}_2, \underline{s}_5, \overline{s}_3, \overline{s}_4, \overline{s}_1$	1	$\sigma_2\sigma_5\sigma_3\sigma_4\sigma_1$
$(\overline{s}_2, \overline{s}_1, \overline{s}_3, \underline{s}_4, \underline{s}_5)$	$\overline{s}_2, \underline{s}_5, \underline{s}_4, \overline{s}_3, \overline{s}_1$	2	$\sigma_2\sigma_5\sigma_4\sigma_3\sigma_1$
$(\overline{s}_2, \overline{s}_1, \underline{s}_3, \underline{s}_4, \underline{s}_5)$	$\overline{s}_2, \underline{s}_3, \underline{s}_5, \underline{s}_4, \overline{s}_1$	3	$\sigma_2\sigma_3\sigma_5\sigma_4\sigma_1$
$\boldsymbol{(\overline{s}_2, \underline{s}_1, \underline{s}_3, \underline{s}_4, \underline{s}_5)}$	$\boldsymbol{\underline{s}_1, \overline{s}_2, \underline{s}_3, \underline{s}_5, \underline{s}_4}$	**4**	$\boldsymbol{\sigma_1\sigma_2\sigma_3\sigma_5\sigma_4}$
$(\underline{s}_2, \underline{s}_1, \underline{s}_3, \underline{s}_4, \underline{s}_5)$	$\underline{s}_1, \underline{s}_2, \underline{s}_3, \underline{s}_5, \underline{s}_4$	5	$\sigma_1\sigma_2\sigma_3\sigma_5\sigma_4$

Table 4.4: Max-min scenarios for $p' = \sigma_2\sigma_1\sigma_3\sigma_4\sigma_5$ and the associated *optPlan* for Example 4.3.2.

Another important feature of the improved algorithm is its efficiency in calculating the cost of a plan. Since we are testing $O(n^2)$ plans and in each plan we consider $n+1$ max-min scenarios, we need to be able to calculate the cost of a current plan under a scenario in $O(1)$ time. Before we discuss how the algorithm does this, let us consider the following definition.

Definition 4.3.1 A *partial sum* represents the cost up to the k^{th} term of a given plan p of n selection operators under a scenario as follows:

$$\sum_{i=1}^{k+1}\left(\prod_{j=1}^{i-1} s_{\pi(j)}\right) c_{\pi(i)}, \text{ where } 0 \leq k < n \tag{4.3.1}$$

The improved algorithm first computes the partial sums for plan p under its first max-min scenario (i.e. the one with zero operators having their minimum selectivity) by calling *createLeftSumArray* function and stores them in $leftSum$ array (lines 1 and 19). The *createLeftSumArray* function uses the cost formula in Equation (3.1.1) to calculate the partial sums which requires $O(n)$ time. So for the initial plan p and the first max-min scenario, the cost is the value of $leftSum[n-1]$ (lines 2 and 20). After that, the cost can be updated incrementally as we move from one scenario to the next. So the partial sums can be reused to calculate the cost of plan p under the next max-min scenario in $O(1)$ time (lines 9 and 26) using function *getCost* in Algorithm 5. When an operator moves to a new position, we can afford

leftSum	
index	partial costs
0	$c_1 = 1$
1	$c_1 + \overline{s}_1 c_2 = 1.9$
2	$c_1 + \overline{s}_1 c_2 + \overline{s}_1 \overline{s}_2 c_3 = 2.17$
3	$c_1 + \overline{s}_1 c_2 + \overline{s}_1 \overline{s}_2 c_3 + \overline{s}_1 \overline{s}_2 \overline{s}_3 c_4 = 2.359$
4	$c_1 + \overline{s}_1 c_2 + \overline{s}_1 \overline{s}_2 c_3 + \overline{s}_1 \overline{s}_2 \overline{s}_3 c_4 + \overline{s}_1 \overline{s}_2 \overline{s}_3 \overline{s}_4 c_5 = 2.5102$

Table 4.5: Initialisation of $leftSum$ array created by $createLeftSumArray(p, 0)$ for Example 4.3.3.

to perform $O(n)$ time work initially, as long as the time taken to compute the regret for each scenario is $O(1)$. Example 4.3.3 demonstrates this idea with some concrete values.

Example 4.3.3 Recall from Example 4.3.1 the set $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$ of selection operators, with selectivities $s_1 = [0.1, 0.9]$, $s_2 = [0.2, 0.3]$, $s_3 = [0.35, 0.7]$, $s_4 = [0.6, 0.8]$ and $s_5 = [0.4, 0.45]$. For simplicity, assume that the cardinality of the relation is 1 and the cost for each operator c_i is also 1. Any time a new plan p is considered by the algorithm, an array called $leftSum$ is created by calling $createLeftSumArray$ as defined in Algorithm 5. Array $leftSum$ represents the partial sums as defined in Definition 4.3.1. Now consider plan $p = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$. In line 1, the algorithm calls $createLeftSumArray(p, 0)$ to create the $leftSum$ array for plan p under the first max-min scenario with index 0 (i.e. all operators are assigned their maximum selectivity). The $createLeftSumArray$ function uses Equation (3.1.1) to calculate the partial costs. For a plan with n operators, there are n partial costs, where $leftSum[k]$ represents the summation of the first k terms in the cost formula of Equation (3.1.1). Table 4.5 shows the values in $leftSum$ array after calling $createLeftSumArray(p, 0)$.

Creating $leftSum$ and calculating its elements clearly can be performed in $O(n)$ time. After creating $leftSum$, the cost of p under its first max-min scenario can be found in $O(1)$ time by retrieving the value of $leftSum[4]$, so $cost(p, 0) =$

content	σ_1	σ_2	σ_3	σ_4	σ_5
index (m)	0	1	2	3	4

Table 4.6: The array content of plan p in Example 4.3.3.

$leftSum[4]$ (as in lines 2 and 20 of Algorithm 4 as well as in line 4 of function *checkOperatorAtEachPosition*). This is because $leftSum[4]$ has the full cost calculation for p under its first max-min scenario.

Recall that j represents the number of minimum selectivities in the max-min scenario for plan p. The cost of plan p under each subsequent max-min scenario can also be found in $O(1)$ time using the *getCost* function in Algorithm 5. Each time $cost(p, j)$ is calculated, the content of $leftSum[n-1]$ is updated with the same value as $cost(p, j)$ to be used in calculating the cost of p under the next max-min scenario as shown in line 7 of *getCost* in Algorithm 5. The following shows the cost calculation for plan p under the remaining max-min scenarios with $1 \leq j \leq n$ using the *getCost* function. Note that n denotes the total number of selection operators, while m is the index of an operator in the plan array as shown in Table 4.6.

- $(\overline{s}_1, \overline{s}_2, \overline{s}_3, \overline{s}_4, \underline{s}_5)$: $j = 1 \Rightarrow m = 4$ (i.e. $m = n - 1$)

$$\begin{aligned} getCost(p, 1, leftSum) &= leftSum[m] \\ &= 2.5102 \end{aligned}$$

- $(\overline{s}_1, \overline{s}_2, \overline{s}_3, \underline{s}_4, \underline{s}_5)$: $j = 2 \Rightarrow m = 3$

$$\begin{aligned} getCost(p, 2, leftSum) &= leftSum[3] + \frac{\underline{s}_m}{\overline{s}_m} * (leftSum[n-1] - leftSum[3]) \\ &= 2.359 + \frac{0.6}{0.8} * (2.5102 - 2.359) \\ &= 2.4724 \\ leftSum[n-1] &= 2.4724 \quad // \text{ Update } leftSum[n-1] \text{ for the next step.} \end{aligned}$$

- $(\overline{s}_1, \overline{s}_2, \underline{s}_3, \underline{s}_4, \underline{s}_5)$: $j = 3 \Rightarrow m = 2$

$$\begin{aligned} getCost(p, 3, leftSum) &= leftSum[2] + \frac{\underline{s}_m}{\overline{s}_m} * (leftSum[n-1] - leftSum[2]) \\ &= 2.17 + \frac{0.35}{0.7} * (2.4724 - 2.17) \\ &= 2.3212 \\ leftSum[n-1] &= 2.3212 \quad // \text{ Update } leftSum[n-1] \text{ for the next step.} \end{aligned}$$

- $(\overline{s}_1, \underline{s}_2, \underline{s}_3, \underline{s}_4, \underline{s}_5)$: $j = 4 \Rightarrow m = 1$

$$\begin{aligned} getCost(p, 4, leftSum) &= leftSum[1] + \frac{\underline{s}_m}{\overline{s}_m} * (leftSum[n-1] - leftSum[1]) \\ &= 1.9 + \frac{0.2}{0.3} * (2.3212 - 1.9) \\ &= 2.1808 \\ leftSum[n-1] &= 2.1808 \quad // \text{ Update } leftSum[n-1] \text{ for the next step.} \end{aligned}$$

- $(\underline{s}_1, \underline{s}_2, \underline{s}_3, \underline{s}_4, \underline{s}_5)$: $j = 5 \Rightarrow m = 0$

$$\begin{aligned} getCost(p, 4, leftSum) &= leftSum[0] + \frac{\underline{s}_m}{\overline{s}_m} * (leftSum[n-1] - leftSum[0]) \\ &= 1 + \frac{0.1}{0.9} * (2.1808 - 1) \\ &= 2.1312 \\ leftSum[n-1] &= 2.1312 \quad // \text{ Update } leftSum[n-1] \text{ for the next step.} \end{aligned}$$

$\diamond$

In summary different features helped in improving the complexity of the improved max-min heuristic. It trades memory for processing time. In this version of the heuristic, the cost of the optimal plans for max-min scenarios are stored and reused in the subsequent steps. Moreover, pre-calculating the partial sums allows the heuristic to calculate the cost of a plan under max-min scenarios in $O(1)$ time. As the result, the improved max-min heuristic reduces the complexity to $O(n^3)$ opposed to $O(n^4 logn)$ for the basic version of the heuristic.

4.4 Conclusion

This chapter has presented our novel heuristic for solving the problem of selection operator ordering when selectivities are given as intervals. We used the insight we gained from studying the problem and its properties as well as some polynomial solvable cases to develop the max-min heuristic $H(p, q)$. The heuristic considers only a polynomial number of plans and examines them under max-min scenarios which play a special role as seen in the previous chapter. This, in turn, reduces the number of scenarios, yet still produces results with good quality as we will see in the next chapter.

The complexity of the basic algorithm for the max-min heuristic turns out to be $O(n^4 logn)$. However, we also presented an improved version of the heuristic which reduces the complexity to $O(n^3)$ and produces the same result as the basic algorithm. The next chapter presents an experimental evaluation for the max-min heuristic using a number of different data sets.

Chapter 5

Experimental Evaluation of the Selection Ordering Heuristic

The previous chapter introduced our novel heuristic, max-min, which solves the problem of selection ordering under imprecise database statistical information. This chapter presents the experimental evaluation of the max-min heuristic and other heuristics, which consider a single point from each selectivity interval to find the optimal solution, namely the midpoint, pessimistic and optimistic heuristics. We used three different data sets to test the max-min heuristic. These data sets are: a synthetic data set, the Star Schema Benchmark (SSB) [100], and the Enron email data set [35]. A description of the data sets is provided in this chapter as well as the results of using the heuristic on these data sets. Some of the work presented in this chapter has been published in papers [9] and [10].

We have implemented the max-min heuristic and tested the impact of different parameters on its performance. In addition we have implemented the brute-force algorithm to find the true optimal solution which is used to measure the quality of the max-min heuristic. For a case with n operators, finding the optimal plan using the brute-force algorithm requires calculating the cost of all possible $n!$ plans under all 2^n extreme scenarios, as has been described in Section 3.3. This process takes a lot of time. Therefore, we were restricted to a maximum of eleven operators in order to be able to compare the results of the heuristic with those of the brute-force algorithm. For instance, the brute-force algorithm took over 20 minutes to find the

minmax regret solution for a single case of nine operators. By contrast, the heuristic took approximately 0.067 seconds for the same case (recall Figure 4.1 in Section 4.1).

Our heuristic works at the logical optimisation level, as mentioned in Section 2.2, and the submitted logical plan may change at the physical optimisation stage [38, 108]. Therefore, in this experimental evaluation, we stress test our heuristic in an isolated environment rather than in a database server with the intricacies of optimisations performed at different levels, known approach used in the literature [16, 49, 83, 115, 117]. Another reason for doing the evaluation in an isolated environment is to study the effectiveness of the max-min heuristic with selection ordering first, before including other operators and different settings. However, we did compare our heuristic with approaches that assume a single value for the imprecise parameters (e.g. using the mean), an assumption made by some optimisers.

A commodity PC, with 8 GB RAM, Intel Core i5 processor running at 3.19 GHz and Windows 7 Enterprise (64-bit), was used to perform the experiments. The minmax regret brute-force algorithm and max-min heuristic were implemented in Java and compiled with the Eclipse IDE (Juno release), which is JDK compliant and uses the JavaSE-1.7 execution environment.

5.1 Measuring criteria

In this section we present the measuring criteria that we used to evaluate the max-min heuristic. Recall from Chapter 4 that the max-min heuristic has two parameters namely p, the choice of the initial plan (possibly empty), and q, the order in which the (remaining) operators are passed to the heuristic. In the experimental evaluation, we studied the impact of choosing different values for these parameters on the quality of the plan generated by the heuristic. Moreover, we also investigated the effects of multiple iterations on the quality of the heuristic. In multiple iterations, the heuristic is run multiple times, with the result of one iteration passed to the next iteration as an initial plan.

Recall from Section 3.2 that $R(P(S), X(S))$ denotes the regret value of the optimal plan that minimises the maximum regret for a given set S of selection op-

erators, where $P(S)$ is the set of possible plans and $X(S)$ is the set of possible scenarios. Similarly let $R(H(p,q),X(S))$ be the regret of the plan returned by the max-min heuristic. The first measuring criterion is the *percentage of exact solutions found*. For this criterion, we count the number of cases where the heuristic generates a plan equivalent to the optimal minmax regret solution, i.e. for which $R(H(p,q),X(S)) = R(P(S),X(S))$. This is then divided by the total number of cases in the experiment. Using this measure, the larger the value the better the performance of the heuristic.

The second measuring criterion is called the *regret ratio* $\lambda(S)$, which is the regret computed by $H(p,q)$ divided by the optimal minmax regret. The regret ratio $\lambda(S)$ can be defined formally as follows:

$$\lambda(S) = \frac{R(H(p,q),X(S))}{R(P(S),X(S))} \tag{5.1.1}$$

We are aware that the denominator $R(P(S),X(S))$ may be equal to zero. However, recall from Section 3.5.2 that the only case where the optimal minmax regret equals zero is when S is a set of strictly dominant selection operators. In such a case we do not use this measure. The max-min heuristic always finds the optimal minmax regret solution for cases where S is a strictly dominating set. Therefore, we define $\lambda(S)$ to be one in such cases.

We arranged the experimental cases in groups based on the number of selection operators, k. Therefore, group k involves all cases which have k selection operators. In view of having a number of test cases, j, for each group k, we calculate the *average regret ratio* and the *worst regret ratio* (which is simply the maximum value of $\lambda(S)$ over the j test cases). Both *average regret ratio* and the *worst regret ratio* are also used as measuring criteria to evaluate the performance of the heuristic. Smaller values for the *average regret ratio* and the *worst regret ratio* indicate better performance of the heuristic.

For a whole set of experiments (i.e., over all groups), we also calculated the overall value for the above three measuring criteria, namely the *percentage of exact solutions found*, the *average regret ratio* and the *worst regret ratio*. For example in the SSB data set, we generated one hundred test cases (i.e. $j = 100$) per group k, where $k \in [2,11]$ selection operators, giving a total of 1000 test cases. We calculated

the *percentage of exact solutions found*, the *average regret ratio* and the *worst regret ratio* for the 100 test cases of each group k for various versions of the max-min heuristic. Then we calculated the overall value for each measuring criteria over the 1000 test cases. To measure the stability of the results, we calculated the variance and confidence interval for the overall percentage of exact solutions found and the overall average regret ratio measuring criteria. Since both the variance and estimated margin of error indicated by the confidence interval were significantly low, we do not include them in the following discussions of the results. However, full details can be found in Appendix B.4.

We used as a base case the version of the max-min heuristic $H(p, q)$ with an empty initial plan p and with the ordered sequence q generated randomly (unlike other versions of the heuristic where the order of q is pre-defined). Each test case can be passed to the max-min heuristic in different permutations. For each test case we run the heuristic ten times with ten different permutations selected randomly. Then we considered the maximum regret of the ten permutations as the worst regret ratio for this test case. For the average regret ratio and the percentage of the exact solutions found, we averaged the ten average regret values and the percentage of the exact solutions found respectively for the ten permutations. For group 2 and group 3 (i.e. test cases with 2 and 3 selection operators), we tested the heuristic with all possible permutations, since the total number of permutations in these cases is less than ten. We denote this version by $H(\emptyset, U)$, with U referring to the fact that the operator order is essentially *unsorted*.

In addition, we implemented the midpoint algorithm as discussed in Section 2.6 and Section 2.9, which takes the midpoint of the selectivity interval for each operator instead of considering the entire selectivity interval. Moreover, we implemented the pessimistic and optimistic approaches from decision theory, as discussed in Section 2.7. The performance of these algorithms is compared with that of the max-min heuristic.

5.2 Synthetic data set

This section describes how the synthetic data set was generated. It also discusses how changing the parameters of the max-min heuristic impacts its performance. Finally this section presents the main results and analysis of the experimental evaluation using the synthetic data set.

5.2.1 Generating test data

We designed a tool to generate random synthetic data sets. This tool was implemented in Java and compiled with the Eclipse IDE (Juno release), which is JDK compliant and uses the JavaSE-1.7 execution environment. The random generator tool allowed us to generate synthetic data sets satisfying various conditions, for example, a synthetic data set with only strictly dominant operators or one with only overlapped dominant operators (recall the definitions from Section 3.1).

Now let us describe the main synthetic data set that we used in this experimental evaluation. This data set used a *mixed* set of operators, that is not restricted to one of the special sets of operators defined in Section 3.1. In this data set, each test case corresponds to a group k of selection operators, with k ranging from 2 to 10. For each group k, we generated one hundred different test cases. While group 2 (i.e. $k = 2$) is not particularly hard to solve, it was mainly included for verification purposes (since any heuristic should be able to find the optimal plan for this simple case). Obviously, the worst case scenarios of all cases in group 2 are max-min scenarios. For each test case in group k, we determined the lower and upper bounds of the selectivity intervals for the selection operators by generating $2k$ uniformly distributed random numbers between 0 and 1. Then each two consecutive values from the $2k$ random numbers were used to form the selectivity interval of one operator, where the smaller number was associated with the lower bound of the interval and the larger number was assigned to the upper bound of the interval. Ten operators was the upper limit for the synthetic data set because, to find the optimal solution for just one test case in group 10, we need to check $10! \cdot 2^{10}$ (≈ 3.7 billion) different costs for each test case.

The same procedure was used to generate a strictly dominant data set and an overlapped dominant data set. However for each set of selection operators in the strictly dominant data set, the $2k$ uniformly distributed random numbers were sorted in non-decreasing order. Then each two consecutive values were assigned as the lower and upper selectivity bounds for each interval. For the overlapped dominant data set, after sorting the $2k$ random numbers, the first and third numbers were repeatedly removed from the list and assigned to the lower and upper selectivity bounds respectively for each operator. This process continued until only two random numbers were left in the list; they were assigned to the last operator as its lower and upper selectivity bounds. Consider the following example which demonstrates how selectivities were generated in the overlapped dominant data set.

Example 5.2.1 Assume that the tool needs to generate a synthetic test case with four overlapped dominant operators, so $k = 4$. The tool generates 8 uniformly distributed random numbers between 0 and 1, and then sorts the numbers in non-decreasing order. Let A be the following sorted list of numbers: $0.0916, 0.3458, 0.3463, 0.4302, 0.7336, 0.8760, 0.9076, 0.9980$. The following steps demonstrate how the selectivity s_i of each operator, $1 \leq i \leq 4$, is chosen:

- Step 0: $A = (0.0916, 0.3458, 0.3463, 0.4302, 0.7336, 0.8760, 0.9076, 0.9980)$.
- Step 1: $s_1 = [0.0916, 0.3463]$, $A = (0.3458, 0.4302, 0.7336, 0.8760, 0.9076, 0.9980)$.
- Step 2: $s_2 = [0.3458, 0.7336]$, $A = (0.4302, 0.8760, 0.9076, 0.9980)$.
- Step 3: $s_3 = [0.4302, 0.9076]$, $A = (0.8760, 0.9980)$.
- Step 4: $s_4 = [0.8760, 0.9980]$, $A = ()$.

Therefore, the selectivies of the four overlapped dominant operators are $s_1 = [0.0916, 0.3463]$, $s_2 = [0.3458, 0.7336]$, $s_s = [0.4302, 0.9076]$, $s_4 = [0.8760, 0.9980]$.

$\diamond$

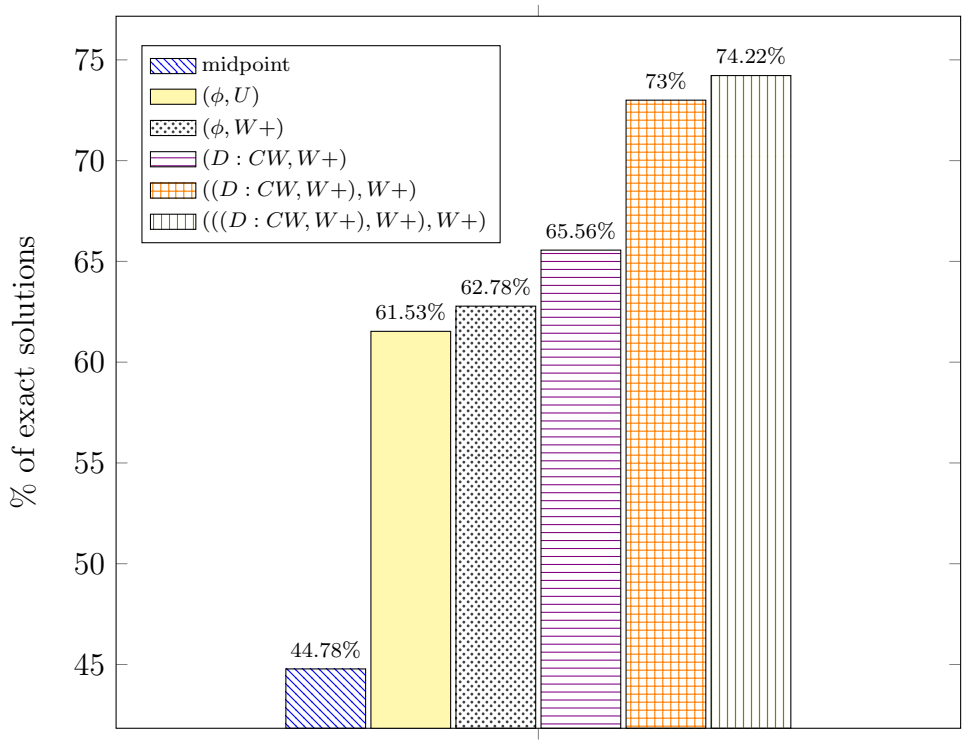


Figure 5.1: Overall percentage of exact solutions for the synthetic data set.

5.2.2 Synthetic experimental results

We now discuss the experimental evaluation for the max-min heuristic when using the synthetic data set. As wase mentioned, we generated synthetic data sets with different settings. However, here we only present the results of the synthetic data set with a *mixed* set of operators, and from now on the term "synthetic data set" will be used to refer to the *mixed* set of operators unless mentioned otherwise. This is because the *mixed* setting seems to be the most representative of real situations in which we have no prior information on relationships between operators. Moreover, the baseline version of the max-min heurstic $H(\emptyset, U)$ finds the exact optimal minmax regret solution for all cases in the strictly dominant and the overlapped dominant data sets; this shows the power of the max-min heuristic in these cases. For the rest of this chapter, the max-min heuristic will be referred to simply as (p, q), for initial plan p and remaining operators q in some specific order.

Before we discuss the results of the max-min heuristic, let us present the results of the pessimistic and optimistic heuristics that we considered. These heuristics find

the optimal plan under a single scenario which is the scenario where all operators are assigned their maximum selectivities or minimum selectivities for the pessimistic and optimistic heuristics respectively. These heuristics produced very bad results. The overall worst case ratio for the pessimistic approach is over 17, while that for the optimistic approach is over 129. We also tested the midpoint heuristic that simply orders the intervals in non-decreasing order of their midpoints (not going through all max-min scenarios). It performs much better than both the pessimistic and optimistic heuristics: its overall worst ratio is approximately 1.86. Therefore, in the following discussion, the midpoint heuristic is used in comparison with the max-min heuristic. More detailed results for the experimental evaluation of the midpoint, pessimistic and optimistic heuristics using synthetic data can be found in Appendix B.1.3.

We started our evaluation with the baseline version of the max-min heuristic $(\emptyset, U)$. This version starts with an empty initial plan $(\emptyset)$ and inserts operators without any specific order (U), where (U) stands for an unordered set. The max-min heuristic $(\emptyset, U)$ was often better than running midpoint heuristic.

We also wanted to test various orders for passing the operators to the heuristic. We considered the following orders (recall the following notation from Section 4.2.2): non-decreasing width $(W+)$, non-increasing width $(W-)$, non-decreasing midpoint $(M+)$, and non-increasing midpoint $(M-)$. The results for midpoint ($M-$ and $M+$) and non-increasing width ordering $(W-)$ show far worse performance than $(W+)$. For example, $(M+)$ and $(M-)$ generated plans whose regret ratio was above three. This is because when using the midpoint ordering, two operators with the same midpoint but different widths cannot be distinguished. Moreover, comparing the $(W+)$ ordering with the $(W-)$ ordering showed that the $(W+)$ ordering performed much better than the $(W-)$ ordering. A possible explanation for this is that, when using $(W+)$, the operator with the smallest width will be processed first and the operator with the largest width will be processed last. The fact that this last operator is the most uncertain and that the heuristic considers it in each possible position of the current plan may explain the improved results. Figures 5.1, 5.2 and 5.3 show improvements in the result of $(\emptyset, W+)$ compared to $(\emptyset,U)$ for all of the three measuring

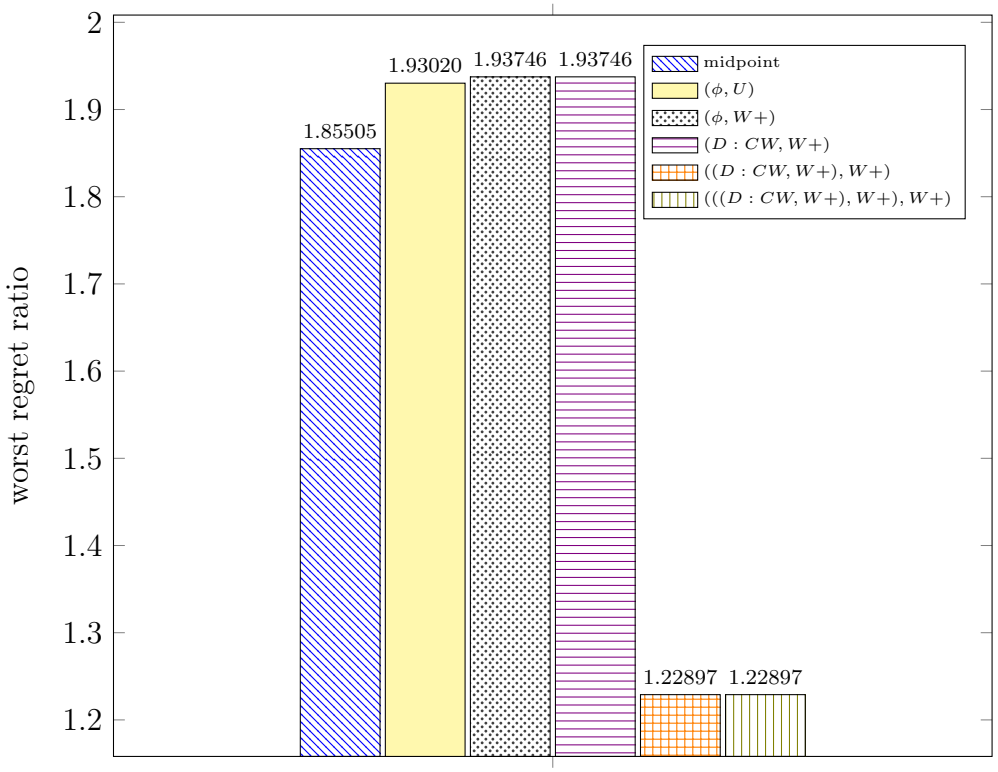


Figure 5.2: Overall worst regret ratio for the synthetic data set.

criteria: the percentage of exact solutions, the overall worst ratio and the overall average ratio.

While W+ ordering performs better than M+, M-, and W-, it is still not significantly better than the random ordering. In the next phase of our experimental evaluation we seeded our heuristic with an initial plan. We tested the heuristic using different criteria for choosing the initial plan. Recall the notation for these criteria from Section 4.2.1 which can be summarised as follows: (D:C) (Dominant:Cardinality), (D:W) (Dominant:Width) and (D:CW) (Dominant: first Cardinality, then Width). We also considered the option of using the result of another heuristic (such as midpoint, pessimistic or optimistic) as the initial plan for the max-min heuristic. The criterion (D:CW) stands for choosing the largest subset of dominant operators, and in case of a tie, choosing the one with the greatest total width of the operators. The results for (D:CW) produced the best results. The superiority of (D:CW) compared to the alternatives can be explained by the fact that (D:CW) feeds the heuristic with a partial plan that preserves the relative order of the maximum num-

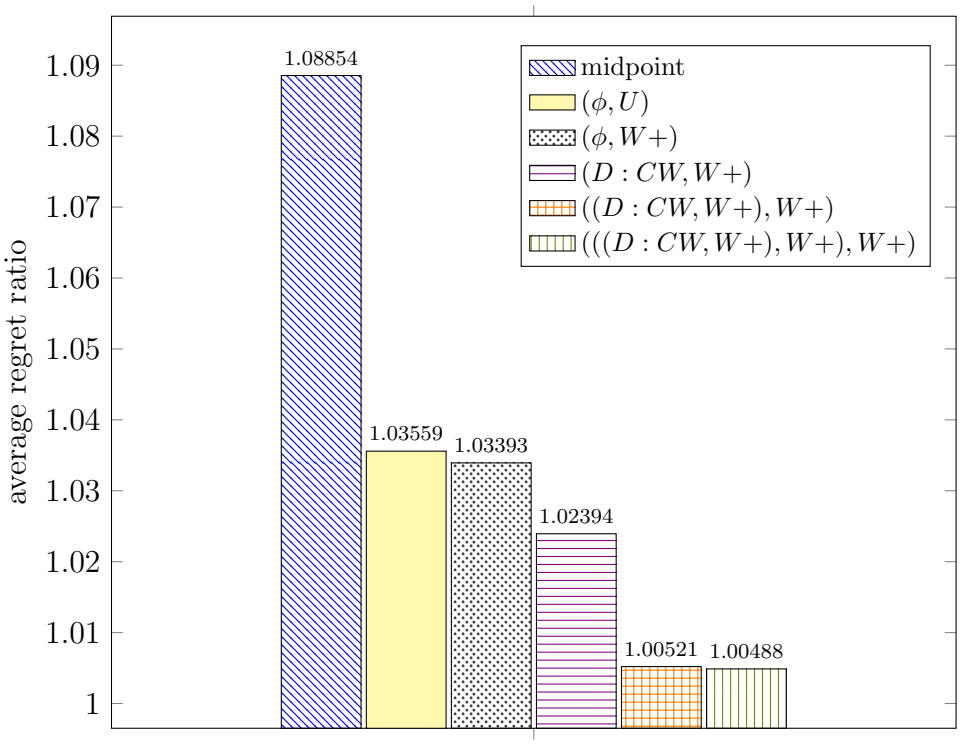


Figure 5.3: Overall average regret ratio for the synthetic data set.

ber of dominant operators as they should appear in the optimal plan. In the case of finding two groups of dominant operators with the same number of operators, (D:CW) chooses the one with the greatest total width. The intuition behind choosing the greatest total width is that operators with larger width are more uncertain than those with smaller width. Hence, preserving their relative order will have a greater positive impact on the final solution compared to operators with a smaller width. We obtained better results using (D:$CW, W+$) compared to ($\emptyset, W+$) and ($\emptyset$,U) when considering the percentage of exact solutions (Figure 5.1) and the average regret ratio (Figure 5.3). The results for the worst case regret ratio (Figure 5.2) were rather inconclusive when compared with the result of ($\emptyset, W+$), so we tried to improve on this by running multiple phases of our heuristic.

The bar charts in Figures 5.1, 5.2 and 5.3 also show the results for running our heuristic multiple times. This means that we take the output of running one phase of our heuristic and use it as the initial plan for the next phase. The figures show the results for starting off by running (D:$CW, W+$) first and then executing two more phases. As can be seen, this variant clearly outperforms the baseline ($\emptyset, U$)

algorithm and the other variants in all respects. The overall worst regret ratio for $((D{:}CW, W+), W+)$ (i.e. running the heuristic and passing its result as initial plan for one more iteration) is less than 1.23 and the overall average ratio is approximately 1.005, compared to approximately 1.93 and 1.036, respectively, for $(\emptyset, U)$. The experiments showed that running one additional phase, $((D{:}CW, W+), W+)$, improves the quality of the generated plan significantly, but running another phase after that, $(((D{:}CW, W+), W+), W+)$, makes almost no difference. Moreover, the max-min heuristic using $((D{:}CW, W+), W+)$ significantly outperforms the midpoint heuristic in all respects. The midpoint heuristic found the exact solution in only 44.78% of cases, while $((D{:}CW, W+), W+)$ found 73% of the cases. The overall worst ratio for the midpoint heuristic is approximately 1.86, while its overall average ratio is approximately 1.089. Comparing these results with the results of $((D{:}CW, W+), W+)$, clearly show that running $((D{:}CW, W+), W+)$ always produces higher quality results. Appendix B.1.2 presents the full results for the discussed variations of the max-min heuristic.

To investigate the effectiveness of the max-min heuristic, the results of the midpoint, pessimistic and optimistic heuristics were fed to the max-min heuristic as initial plans, after which the quality of the resultant solutions were studied. The experimental results showed the effectiveness of the max-min heuristic in taking a poor quality plan and improving it with respect to all measuring criteria. The improvement in the results were substantial, as can be seen in Table B.1.3 in Appendix B. For example, after passing the midpoint, pessimistic and optimistic heuristics to the max-min heuristic, the worst ratio dropped to less than 1.5 (recall that they were initially approximately 1.9, 17.2 and 129.2, respectively).

5.3 The Star Schema Benchmark data set

In this section we introduce the Star Schema Benchmark (SSB), describe how the data set was prepared, and present the main results of evaluating the max-min heuristic on the SSB data set.

5.3.1 Preparing the SSB data set

The Star Schema Benchmark (SSB) is a variation of the well-known TPC-H benchmark [100]. The benchmark simulates a supply chain business model of a warehouse. It models the relationships between suppliers, customers, items and orders. We generated skewed SSB benchmark data with a scaling factor of 1, meaning that the central facts table, *lineOrder*, contains 6,000,197 tuples. We then joined the dimensional tables, namely *part*, *customer*, *supplier* and *date*, to the *lineOrder* table.

After generating the SSB data set, the next step was to create histograms for each attribute in the central facts table, *lineOrder*. This was done by dividing the domain of an attribute into equi-width buckets, then counting the number of tuples that fall into each bucket. We did not keep any further information on the distribution of tuples within each bucket of a histogram. For example, Figure 5.4 shows the histogram for the attribute `ordtotalprice`. This histogram consists of 20 buckets, each covering roughly 18,000 different values. For example, bucket #1 covers the range from 1 to 17,673 as shown in Table 5.1, which has the full range of buckets for `ordtotalprice` with their total numbers of tuples.

After building a histogram for each attribute in the generated table, eleven attributes were chosen to be used in generating random selection operators namely: `orderKey`, `linenumber`, `suppkey`, `quantity`, `ordtotalprice`, `revenue`, `supplycost`, `brand`, `size`, `container`, and `custkey`. Once again, we were limited to eleven operators, since solving any generated case optimally using the brute-force approach requires checking $11! * 2^{11}$ ($\approx$ 81.7 billion) different costs.

The basic information from the histograms allows us to determine intervals for the selectivities of each selection operator. For a "less than" ($<$) and "greater than" ($>$) operator, we know that all histogram buckets exclusively covering smaller/larger values have to be included fully. However, for the bucket in which the predicate value falls, we do not know precisely how many elements will be selected. In extreme cases, none or all of the elements satisfy the predicate, thus giving us the lower and upper bounds for the selectivity. Example 5.3.1 below illustrates with concrete values the calculation of selectivity bounds for given predicates.

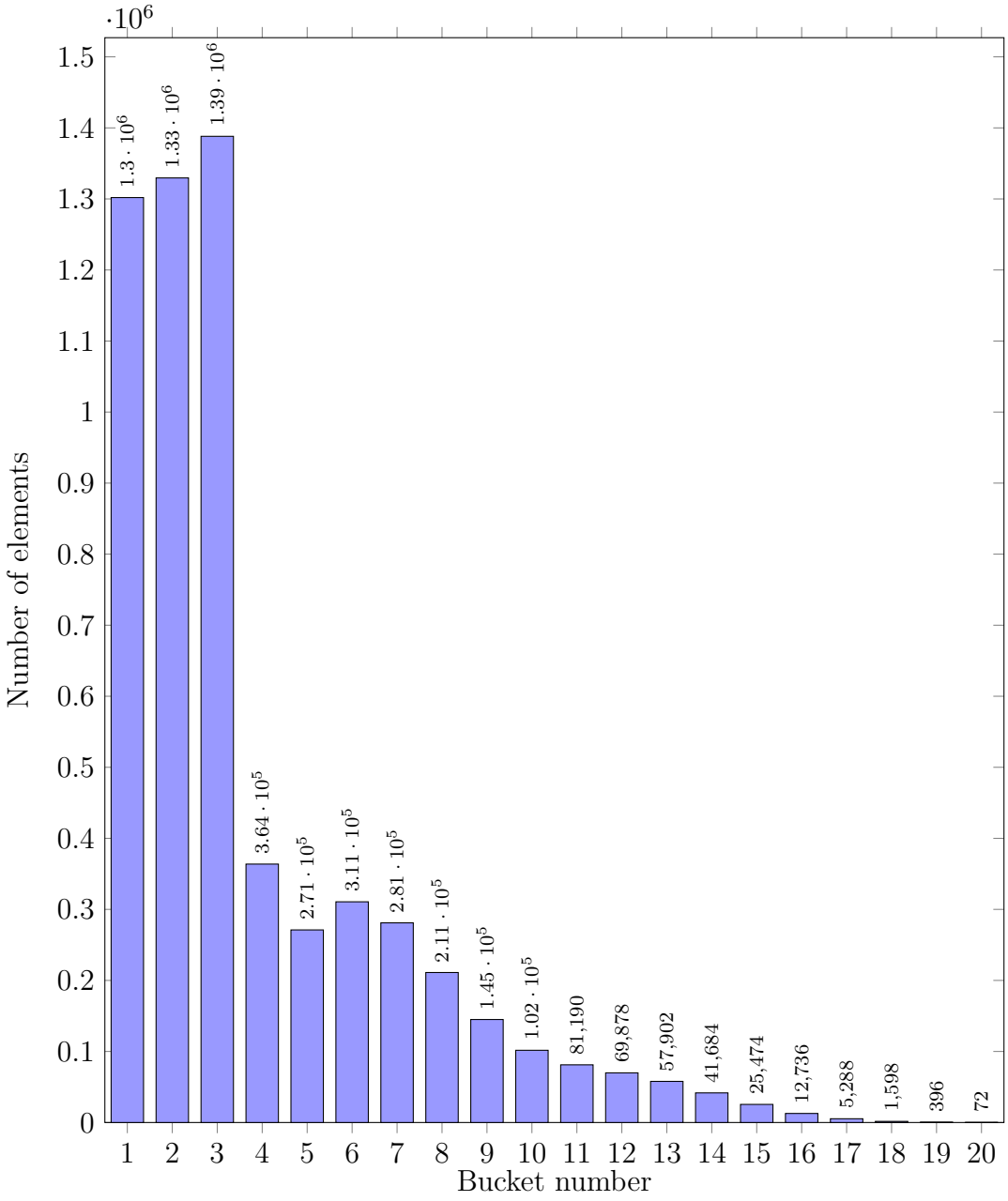


Figure 5.4: Histogram for attribute `ordtotalprice`.

bucket #	start value	end value	total # of tuples
1	1	17673	1301927
2	17674	35346	1329577
3	35347	53019	1388148
4	53020	70692	363752
5	70693	88365	271088
6	88366	106038	310664
7	106039	123711	281044
8	123712	141384	211126
9	141385	159057	144979
10	159058	176730	101674
11	176731	194403	81190
12	194404	212076	69878
13	212077	229749	57902
14	229750	247422	41684
15	247423	265095	25474
16	265096	282768	12736
17	282769	300441	5288
18	300442	318114	1598
19	318115	335787	396
20	335788	353460	72

Table 5.1: Range of values for attribute `ordtotalprice` histogram.

Example 5.3.1 Consider the histogram for the `ordtotalprice` attribute as shown in Table 5.1. Given the predicate `ordtotalprice` < 40000, what are the selectivity interval bounds?

The value 40000 falls into bucket 3. So we know that all tuples in buckets 1 and 2 must be included, with a lower bound given by including nothing from bucket 3, and an upper bound given by including all tuples from bucket 3. Hence the lower and upper bounds for the selectivity are computed as follows:

$$\begin{aligned} \text{Lower bound} &= \frac{1301927 + 1329577}{6000197} = \frac{2631504}{6000197} \\ &= 0.4386 \end{aligned}$$

$$\begin{aligned} \text{Upper bound} &= \frac{1301927 + 1329577 + 1388148}{6000197} \\ &= \frac{4019652}{6000197} \\ &= 0.6699 \end{aligned}$$

Therefore, the selectivity interval for the predicate `ordtotalprice` < 40000 is [0.4386, 0.6699].

Now consider the predicate `ordtotalprice` > 260000, which covers the six buckets from bucket 15 up to bucket 20. For the lower bound, the total number of tuples in buckets 16 to 20 will be used, while for the upper bound the total number of tuples in all six buckets will be considered as follows:

$$\begin{aligned} \text{Lower bound} &= \frac{12736 + 5288 + 1598 + 396 + 72}{6000197} = \frac{20090}{6000197} \\ &= 0.0033 \end{aligned}$$

$$\begin{aligned} \text{Upper bound} &= \frac{25474 + 12736 + 5288 + 1598 + 396 + 72}{6000197} \\ &= \frac{45564}{6000197} = 0.0076 \end{aligned}$$

As a result, the selectivity interval for the predicate `ordtotalprice` < 260000 is $[0.0033, 0.0076]$. ◇

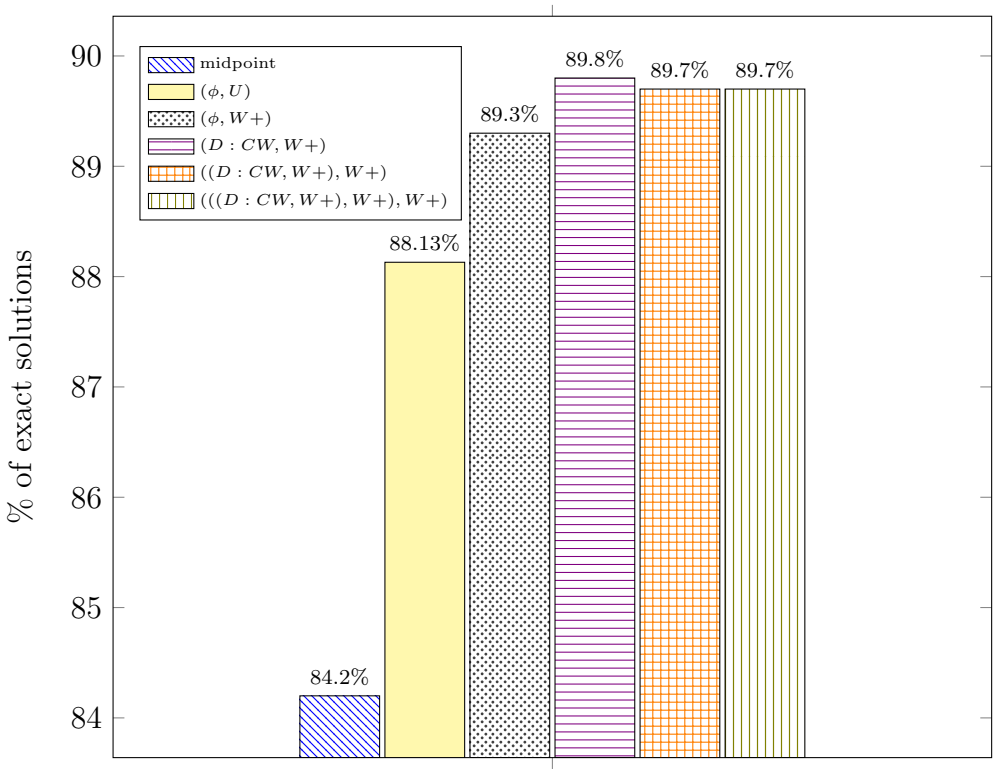


Figure 5.5: Overall percentage of exact solutions (SSB).

We used our tool to generate random queries with k predicates, where $k \in [2, 11]$. Each query consists of predicates using different attributes. Queries basically consist of a conjunctive predicate whose clauses are made up of the selected attributes compared to a random value taken from the attribute's domain, using a less-than or greater-than operator. The following query is an example generated in our experiments:

$$\texttt{orderKey} < 2964443 \text{ AND } \texttt{linenumber} > 5 \text{ AND } \texttt{quantity} < 29$$

We used our tool to generate 100 random cases for each query size k, giving a total of 1000 test cases for all k sizes, where $k \in [2, 11]$.

5.3.2 The SSB experimental results

This section presents the experimental results of our heuristics using the SSB data set. We optimised the generated SSB queries using minmax regret optimisation, as well as the midpoint, pessimistic and optimistic approaches.

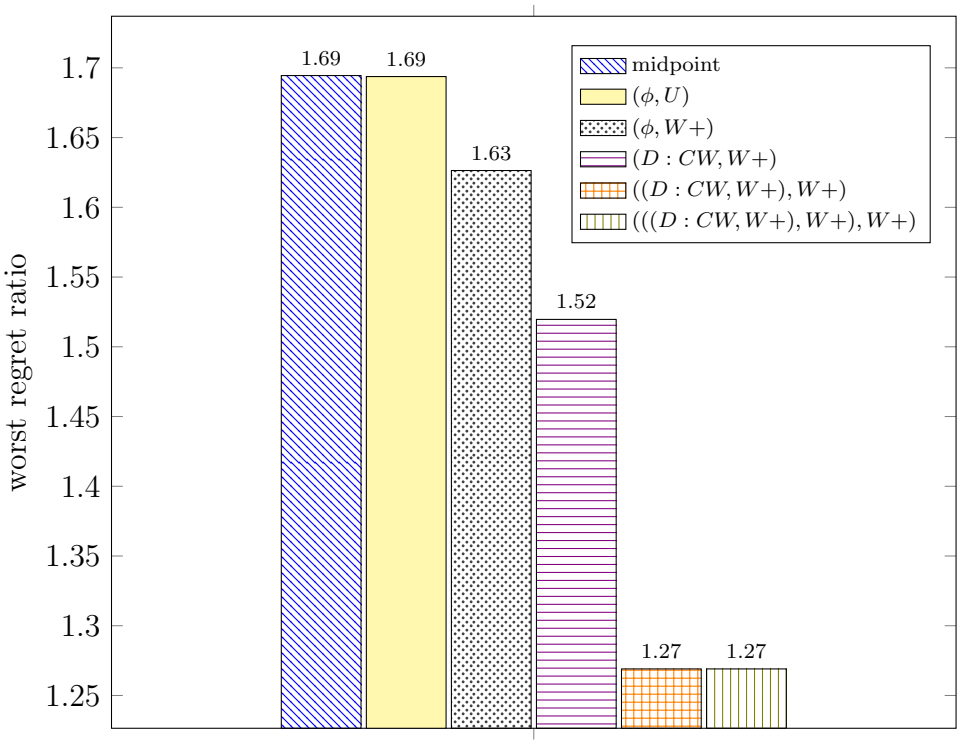


Figure 5.6: Overall worst regret ratio (SSB).

The pessimistic and optimistic approaches showed much worse performance compared with their performance on the synthetic data set. The overall worst regret ratios on SSB are over 3204 and 89182 for the pessimistic and optimistic approaches respectively. In addition, the overall average regret ratio for the pessimistic and optimistic approaches in SSB are approximately 12.08 and 555.33 respectively. Once again both pessimistic and optimistic approaches showed much worse performance than the midpoint approach. Therefore, we used the midpoint approach to compared with the max-min heuristic. Appendix B.2.2 presents more results about the experimental evaluation using the SSB data set for the midpoint, pessimistic, and optimistic approaches.

After finding the optimal minmax regret solution for all generated cases, we started evaluating the max-min heuristic using the $(\emptyset, U)$ version as a baseline. Recall that the $(\emptyset, U)$ version of the max-min heuristic starts without any initial plan $(\emptyset)$ and is passed the operators in no specific order (U). It is interesting to note that the max-min heuristic performs better on the SSB data set compared to the synthetic data set, even with the baseline $(\emptyset, U)$. For example, $(\emptyset, U)$ finds

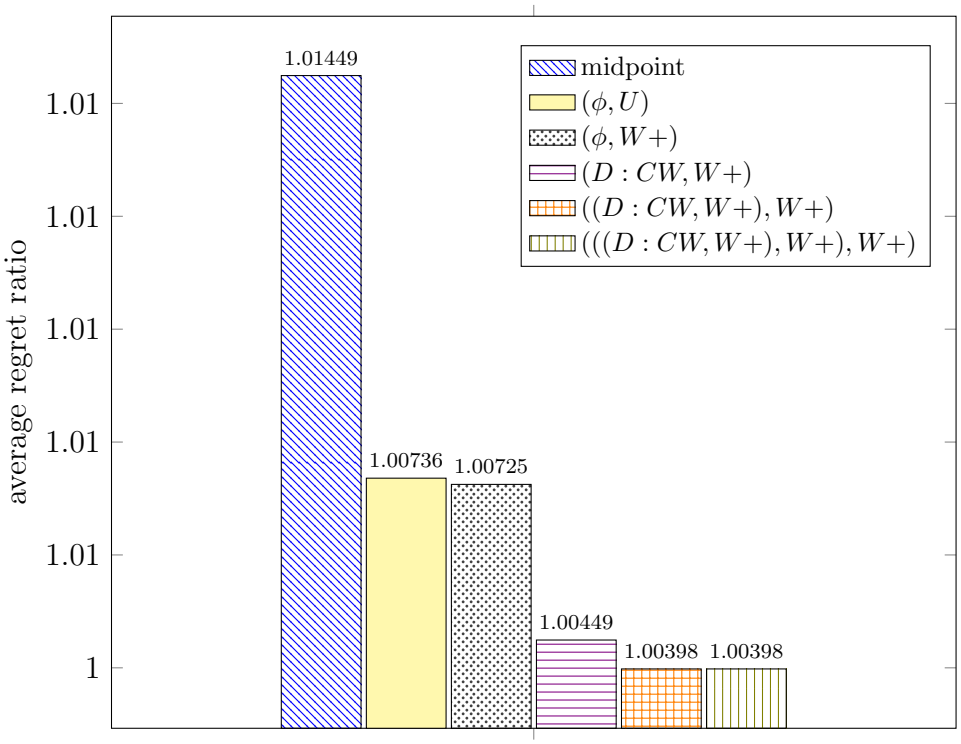


Figure 5.7: Overall average regret ratio (SSB).

the exact optimal solution in 88.13% of the cases in the SSB data set compared to approximately 61.53% of the cases in the synthetic data set. $(\emptyset, U)$ outperforms the midpoint approach when considering the criterion of the overall percentage of exact solutions (Figure 5.5) and the criterion of the overall average ratio (Figure 5.7). However, for the overall worst regret ratio criterion (Figure 5.6), the result was inconclusive since both $(\emptyset, U)$ and the midpoint approach had 1.69 as an overall worst regret ratio. Therefore, we tested passing the operators in non-decreasing order according to the width of their selectivity interval $(W+)$. As a result, the max-min heuristic $(\emptyset, W+)$ showed improvements in all criteria.

Figures 5.5, 5.6 and 5.7 clearly illustrate the improvement in the performance of the max-min heuristic with respect to all measuring criteria when it was seeded with initial plan (D:CW). As before, using the initial plan in the max-min heuristic produced the best results. Once again, we tested our heuristic with multiple iterations. Specifically, we passed the result of (D:$CW, W+$) as an initial plan for a second run of the heuristic. This improved the results, specially for the worst regret ratio. For example, the overall worst regret ratio dropped from 1.52 for (D:$CW, W+$) to 1.27

for $((D{:}CW, W+), W+)$ as shown in Figure 5.6. Similar to the results using the synthetic data set, running a third phase of the heuristic, $(((D{:}CW, W+), W+), W+)$, produces almost the same results as $((D{:}CW, W+), W+)$. The full results can be found in Appendix B.2.1.

We also wanted to see how the max-min heuristic performs when it is fed with a bad initial plan. Therefore, we challenged the max-min heuristic by feeding it with the results of the midpoint, pessimistic and optimistic results as initial plans. The improvements were impressive with respect to all measuring criteria. For example, the worst regret ratio dropped from 1.7, 3205 and 89183 for the midpoint, pessimistic and optimistic approaches, respectively, to 1.3 and approximately 1.4 for both the pessimistic and optimistic approaches. The complete results for this experiment can be found in Appendix B.2.2.

5.4 The Enron data set

The third data set used to evaluate minmax regret optimisation of selection operators was the Enron email data set. We first discuss the process of preparing the data set for the experimental evaluation and how the queries were generated. Then the main results of the experimental evaluation are presented.

5.4.1 Preparing the Enron data set

The Enron data set was originally acquired and made public as a result of the investigation by the Federal Energy Regulatory Commission after the collapse of the company. The version of the data set that we used contains over 255000 emails [35]. Estimating the selectivity of string predicates which perform substring matching using SQL `like` is known to be difficult [29]. We chose the Enron data set to test our heuristic in such an environment, where string predicate selectivities are known to fall within some interval. The Enron email data set records the email communications sent and received by Enron employees. It has five tables which are as follows:

- The `person` table stores employees' personal details.

- The `email_address` table stores the email addresses and whether each belongs to an Enron employee or not.
- The `message` table includes email details such as the subject and when the email was sent.
- The `emailreceiver` table associates each message with its recipient list of addresses.
- The `body` table which associates each email message with its body.

In order to prepare the data set for our experiments, we first nominated suitable string attributes from different tables, since not all string attributes would be useful for our experiments. For example, we did not use name attributes (i.e. first and last name) from the `person` table because the `person` table has only the names of 156 Enron employees. This would severely limit the size of the data that would be used in the experiments. On the other hand, we used the `subject` attribute of the emails from the `message` table because it is a string attribute with widely varying values and each email in the data set has a subject. We chose attributes with a variety of string lengths, from attributes with relatively short strings, such as the subject of the emails, to long strings such as the body of emails. Specifically, we chose the following attributes: `sender` email address, `recipient` email addresses, `subject` and `body` of the emails. After choosing these string attributes from different tables, we joined the tables to form one big table called `emailDetails`. This allowed us to perform a select query on a single table.

Before we generated the queries, we needed to model the selectivity ranges of the predicates. In order to do this, we nominated sets of keywords for each attribute. These keywords would be used in the queries. While choosing the keywords, we tried to have a range of keywords, including popular and less popular keywords. This in turn produced different ranges of selectivity intervals which enabled us to test our heuristic thoroughly, as we will see in Section 5.4.2.

In our experiments, we assumed that queries use selection predicates such as `subject like '%work%'`. We also assumed that the database maintains indexes on words and on the n-grams (e.g. 2-grams) of words which allows the database also

#	2-gram	Selectivity
1	sc	0.46867
2	ch	0.75389
3	he	0.90265
4	ed	0.83887
5	**du**	**0.39394**
6	ul	0.60242
7	le	0.84724

Table 5.2: The 2-grams for the keyword 'schedule' for the `body` attribute in Example 5.4.1.

to provide selectivities for these predicates. In a predicate such as `subject like '%work%'` the selectivity for the word '`work`' will underestimate the true selectivity. This is because this predicate will not match the word '**work**' only, but will also match words such as: 'net**work**', '**work**ing', '**work**load' and '**work**day' (and many others). Therefore, we form an interval selectivity for a word by considering the exact match as a lower selectivity estimate. For the upper estimate, we use the minimum selectivity of all the n-grams of the word (after fixing the value of n). This is because any string containing the whole word must contain all of its n-grams as well. The following example illustrates this idea.

Example 5.4.1 Assume we want to find the selectivity interval for the keyword 'schedule' for the `body` attribute. Assume also that the database system stores the selectivity of all 2-grams for keywords, such as shown in Table 5.2. The 2-grams will be used to find the upper selectivity range of the keyword.

For the lower selectivity estimate, we consider the number of exact matches for 'schedule' divided by the total number of rows, which gives 0.04548. For the upper selectivity estimate, we need to consider the 2-grams for the word 'schedule' as shown in Table 5.2. The smallest value among the 2-grams, which happens to be the value of 'du' in this case, will be divided by the total number of rows to form the upper selectivity for 'schedule'. As a result, the selectivity range for 'schedule'

is $[0.04548, 0.39394]$. $\diamond$

We calculated the upper selectivities of the keywords using n-grams for $n \in \{2, 3, 4, 5\}$. We decided to use 2-grams to calculate the upper selectivities since larger gram sizes will not help in generating interval selectivities for small words. For example, using 5-grams to find the upper selectivity for a word with three letters for instance would end up having the same selectivity as the exact word (i.e. its minimum selectivity). Moreover, we noticed that the selectivity interval width (i.e. the difference between the upper and lower selectivity values) when using a bigger gram size is much smaller than the width when using a smaller grams size. Having selectivity intervals with different widths is important in testing the effectiveness of our approach.

We calculated the selectivity intervals for all chosen keywords as explained in Example 5.4.1. Due to the nature of the email addresses of the `sender` and `recipients` in this data set, the selectivity intervals for keywords from these attributes are very small. However, the selectivity intervals of the keywords for the `subject` and `body` attributes are more representative and have a variety of interval widths and selectivity values. Therefore, the queries were generated using keywords from the `subject` and `body` attributes. This gave rise to a range of intervals, including those with small values such as $[0.0004, 0.01]$ for the 'progress' keyword in the `subject` attribute, those with larger values such as $[0.6, 0.7]$ for the 'you' keyword in the `body` attribute, and those with large width such as $[0.07, 0.6]$ for the 'price' keyword in the `body` attribute.

We developed a tool to generate queries with different numbers of predicates. For each number of predicates k, where $k \in [2, 11]$, we generated 20 queries randomly. We prepared a list of 40 keywords from the `subject` attribute and a list of 45 keywords from the `body` attribute. The full lists of keywords for `subject` and `body` attributes can be found in Table B.5 and Table B.6, respectively, in Appendix B.3.1. The tool randomly picked one keyword from the `subject` list to form one predicate and picked $k - 1$ distinct keywords from the `body` list to form a query with k predicates. Each generated query was checked by the tool to make sure that it returned a non-empty answer. The following is a sample of a generated query:

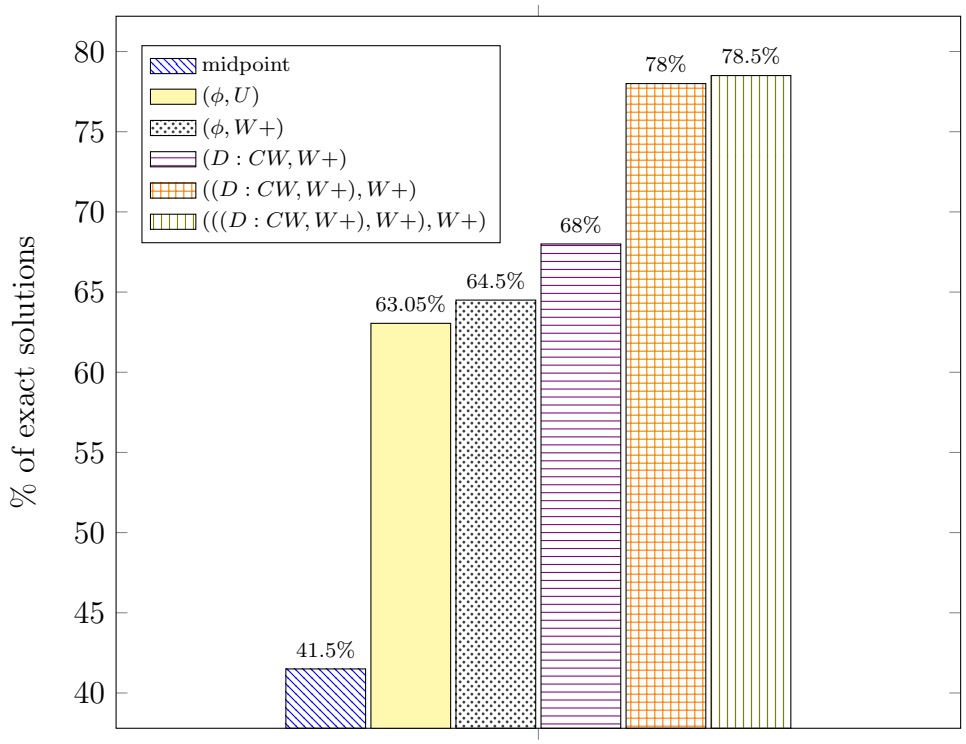


Figure 5.8: Overall percentage of exact solutions (Enron).

```
select *
from   emailDetails
where  body like '%action%' AND
       body like '%schedule%' AND
       body like '%meet%' AND
       subject like '%market%' ;
```

5.4.2 The Enron experimental results

This section presents the experimental results of our heuristics using the Enron data set. As before, we first found the optimal minmax regret solution for all generated queries using the brute-force approach.

We started by testing the pessimistic and optimistic heuristics. They continued to perform poorly when compared with the midpoint heuristic as shown in Table B.8 in Appendix B.3.3. Therefore, the midpoint is used as a basis of comparison with the max-min heuristic. It is interesting to note that if the selectivities were simply

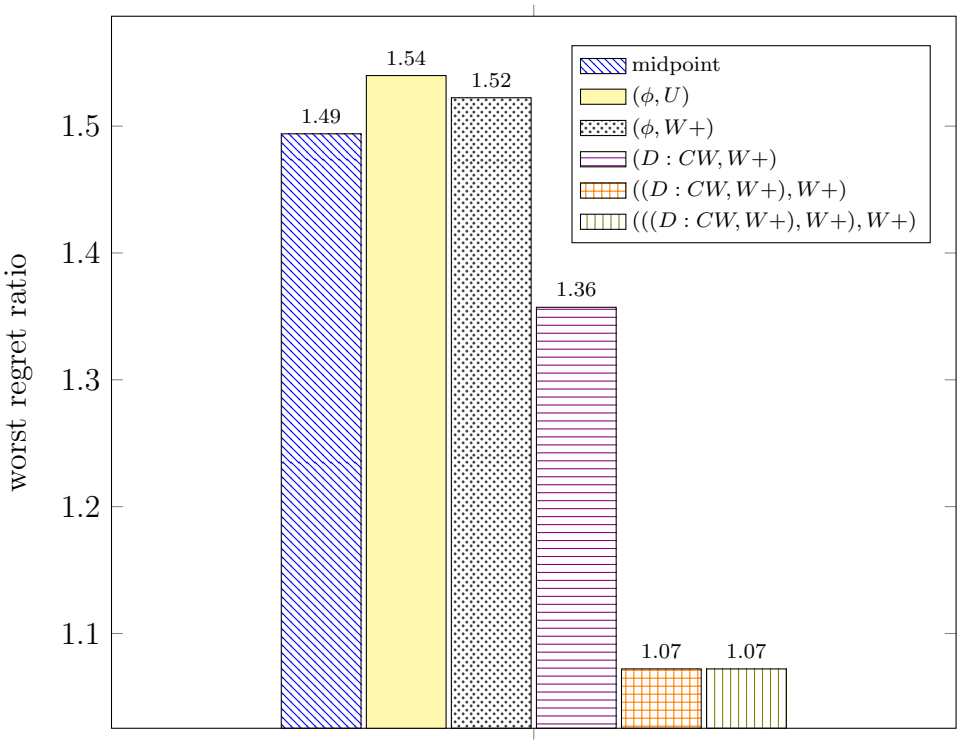


Figure 5.9: Overall worst regret ratio (Enron).

calculated based on whole keywords without their n-grams, then that would be equivalent to considering only the minimum selectivity values of the intervals and this is exactly what the optimistic approach does. This poor choice of selectivity by the optimistic approach results in very poor performance, for example a worst regret ratio of more than 31.

In general the results on the Enron data set showed similar trends to the other data sets, but were more impressive as shown in Figures 5.8, 5.9 and 5.10. Going from the baseline of the max-min heuristic $(\emptyset, U)$ to $(\emptyset, W+)$ and then to $(D{:}CW, W+)$ showed a good improvement in the results. The max-min heuristic with multiple iterations also boosted the quality of the results. The $((D{:}CW, W+), W+)$ and $(((D{:}CW, W+), W+), W+)$ variants of the max-min heuristic found the minmax optimal solution in 78% and 78.5% of cases respectively, both having a worst regret ratio of only 1.07, and an average regret ratio of less than 1.001. By contrast, the midpoint heuristic had a worst regret ratio of over 1.49, an average of 1.06, and, for example, did not find a single minmax optimal solution in all cases of 10 operators. Overall the midpoint heuristic found the optimal minmax solution in only

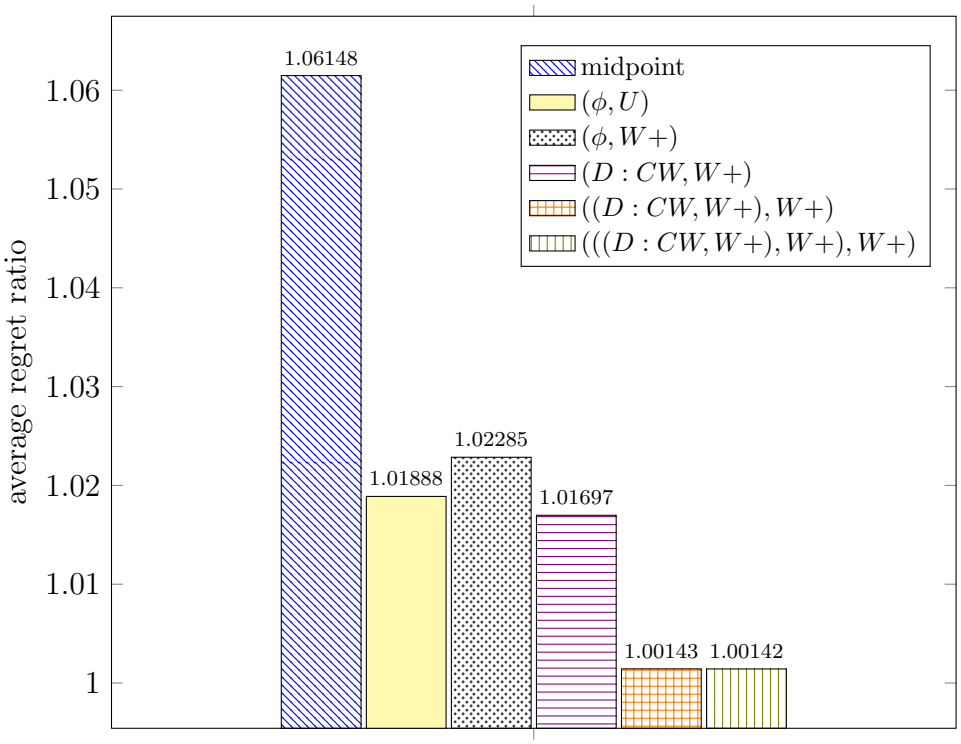


Figure 5.10: Overall average regret ratio (Enron).

41.5% of cases. The full results for the max-min heuristic can be found in Table B.7 in Appendix B.3.2.

The max-min heuristic also showed good performance in refining and improving bad plans by being fed, as initial plan, the results of the midpoint, pessimistic and optimistic heuristics. It is interesting that, with the max-min heuristic refinement, the results of these heuristics became closer to the result of $(((D{:}CW, W+), W+), W+)$. For example, the percentage of exact solutions was 41.5% for the midpoint heuristic, and 31.5% for both the pessimistic and optimistic heuristics. However, after the max-min heuristic refinement the percentage of exact solutions improved to more than 76%. Table B.8 in Appendix B.3.3 shows the full results for the midpoint, pessimistic and optimistic heuristics.

5.5 Discussion of experimental results

In order to study the max-min heuristic and the impact of its parameters, namely the choice of initial plan and the order in which the operators are passed to the

heuristic, we started with the baseline $(\emptyset, U)$ which has no initial plan and passes the operators in no specific order. As we discussed in Section 2.3 the aim for most query optimisers is finding the optimal plan (not avoiding worst plans as in our approach). Recall also from Section 2.6 that some existing systems use single value estimate such as the midpoint to deal with imprecision in the statistical information. This approach is compared to our heuristic in our experimental evaluation. The results of our experiments showed that even the basic version of the max-min heuristic $(\emptyset, U)$ outperforms an approach that finds the optimal plan using the midpoint as a single point estimate in the measures of the percentage of exact solution found and the average regret ratio. For example on the Enron data set, $(\emptyset, U)$ shows a more than 34% improvement in finding the optimal minmax regret plan when compared to the midpoint heuristic.

Now let us consider the ordering parameter for the max-min heuristic (recall the different ordering criteria from Section 4.2.2). The aim of sorting is to pass the operators to the heuristic based on their selectivity in order to build a good solution. Our experimental evaluation showed that sorting the operators according to the width of their selectivities is much better that sorting the operator according to their midpoint selectivities. One reason for this is that the midpoint criterion does not reflect the level of imprecision for the selectivities of the operators, since two operators may have same midpoint selectivity but vary in their selectivity interval width which cannot be distinguished using the midpoint. The ordering criterion $(W+)$, which sorts the operators in non-decreasing order according to the width of their selectivity intervals, showed the best performance. The intuition behind this is that with $(W+)$, the operator with the smallest width will be processed first while the operator with the largest width (hence most imprecise and influential in the cost calculation) will be processed last. This allows the heuristic to consider the operators with larger selectivity width in more possible positions in the current plan, helping to improve the result of $(\emptyset, W+)$ when compared with $(\emptyset, U)$.

The results are further improved when using $(D{:}CW, W+)$, which starts with the initial plan $(D{:}CW)$ and then passes the remaining operators in non-decreasing order according to the width of their selectivity intervals. Starting with $(D{:}CW)$ as an ini-

tial plan allows the heuristic to find the correct order for the largest set of dominant operators in terms of operator cardinality. In case of having more than one dominant set with the same maximum cardinality, the heuristic chooses the one whose total width of the selectivity intervals is greatest. As a result this dominant set includes more operators with imprecise selectivities compared to other sets. Starting with the correct order for such operators, which have the same relative order in the optimal plan, gives the heuristic a good start and helps to improve the overall solution. The remaining operators have the chance to be tested in more positions starting from the least precise to the most imprecise operators. The $((D{:}CW, W+), W+)$ version of the heuristic improves the quality of the result significantly. This takes the good result of $(D{:}CW, W+)$ and refines it by adding another iteration which allows the operators to be tested in further possible positions.

As seen in the experiments, $(((D{:}CW, W+), W+), W+)$ introduces a slight improvement in the overall percentage of exact solutions for both synthetic and Enron data sets when compared with $((D{:}CW, W+), W+)$. However, it does not show any significant improvements in terms of the overall worst and average regret ratio in all data sets. One explanation for this is that when we use $(((D{:}CW, W+), W+), W+)$, we first find the correct order for the biggest set of dominant operators $(D{:}CW)$. Finding the correct order for such operators provides a good start for building the solution knowing that this subset of operators preserve their relative order as in the optimal minmax regret plan. This in turn, allows the remaining operators to be tested in more possible positions. Eventually after the first and second iteration (i.e. $(D{:}CW, W+)$ and $((D{:}CW, W+), W+)$ respectively) all operators will find their best position under the tested max-min scenarios. Therefore, no much room for further improvement is left for $(((D{:}CW, W+), W+), W+)$ and that might explains the slight improvement after $((D{:}CW, W+), W+)$. Moreover, during our experimental evaluation we noticed that adding more iterations after $(((D{:}CW, W+), W+), W+)$ does not improve the results at all. Even though the $(((D{:}CW, W+), W+), W+)$ is tested with more iterations using different data sets and with different query sizes (up to 11 selection operators in our experiment), however it is hard to have a general claim that no more improvement will be gained by adding more iterations after

Measure	Synthetic		SSB		Enron	
	mid.	max-min	mid.	max-min	mid.	max-min
% exact	45%	73%	84%	90%	42%	78%
Worst	1.86	1.23	1.69	1.27	1.49	1.07
Average	1.09	1.01	1.01	1.004	1.06	1.001

Table 5.3: Comparing the max-min heuristic with the midpoint heuristic on all data sets.

$(((D{:}CW, W+), W+), W+)$.

It is true that $(D{:}CW, W+)$ and $((D{:}CW, W+), W+)$ require additional computations (see Appendix B.1.2 for sample run times for versions of the max-min heuristic), but they showed a significant improvement in the quality of results. So, they trade time for quality. If the aim is to have a quick solution with reasonable results, then the max-min heuristic versions $(\emptyset, W+)$ or $(D{:}CW, W+)$ are recommended. On the other hand, if the aim is to have high-quality plans by spending extra time, then the max-min heuristic $((D{:}CW, W+), W+)$ is recommended. The max-min heuristic runs at query optimisation stage, so the extra time required by $((D{:}CW, W+), W+)$ in the optimisation process would not affect the run time but improves the quality of the result.

The main target in MRO is to avoid bad plans under worst case scenarios. The max-min heuristic showed impressive results in avoiding such bad plans. In fact it not only avoids bad plans; it even found the exact optimal minmax regret solution for a large percentage of cases. For example, it found the exact minmax regret solution in 89.7% of the cases in the SSB data set. More interesting, even the basic version of the max-min heuristic $(\emptyset, W+)$ gave very good performance when compared with the midpoint, pessimistic and optimistic heuristics.

Overall the max-min heuristic performed very well on all data sets. The synthetic data set was used to stress test the heuristic with cases that might not appear commonly in real life, such as operators with purely nested selectivity intervals (recall the definition of nested operators from Section 3.1). This might explain the slightly reduction in performance on the synthetic data set compared to the SSB

and Enron data sets for some criteria. Table 5.3 summarises the results of the max-min heuristic $((D{:}CW, W+), W+)$ compared to the midpoint heuristic on all data sets. It outperforms the midpoint heuristic on all measures. For example, the $((D{:}CW, W+), W+)$ shows up to 46.8% improvement (in Enron data set) comparing to the midpoint heuristic in terms of the percentage of finding the exact optimal minmax regret plan.

5.6 Conclusion

This chapter has presented an experimental evaluation of various approaches to optimise the selection ordering problem using MRO. The brute-force approach for computing the optimal minmax regret solution was used to evaluate the performance of the various heuristics. We have compared the performance of our novel max-min heuristic to other baseline heuristics. The midpoint, pessimistic and optimistic heuristics all consider a single point from each selectivity interval of the selection operators to find the optimal solution. The midpoint heuristic uses the midpoint selectivity of the selection operators instead of considering the whole selectivity interval. The pessimistic heuristic considers only the maximum value of the selectivity interval. The optimistic heuristic, on the other hand, uses the minimum selectivity value of the selectivity interval for the selection operators.

Three different data sets were used in the experimental evaluation. One data set was generated synthetically, while for the second data set we used the Star Schema Benchmark (SSB), a variation of the well-known TPC-H benchmark [100]. The third data set was the Enron email data set [35]. This chapter has discussed the pre-processing of the data sets, how the queries were generated for each data set, and the measuring criteria that were used in the experimental evaluation. The experimental evaluation using the three data sets showed that the max-min heuristic clearly outperforms the other heuristics.

Chapter 6

Applying the Max-min Heuristic to the Total Flow Time Problem

In Section 2.9.1 we discussed the job scheduling problem on a single machine where the objective is to find the order which minimises the total flow time (TFT). Then in Section 2.9.2, we presented the MRO version of the problem with interval processing times, and discussed some approximation heuristics such as the 2-approximation algorithm. We mentioned that there are some similarities between MRO for the TFT problem and MRO for the selection ordering problem. However, the cost function for the TFT problem is linear (Equation 2.9.1 in Section 2.9.1), while that for the selection ordering problem is non-linear (Equation 2.4.2 in Section 2.4.1). Moreover, the midpoint heuristic leads to 2-approximation for the TFT problem, while it does not guarantee a solution whose regret is no more than twice the optimal regret for the selection ordering problem, as discussed in Section 3.4.4. Nevertheless, we decided to apply the max-min heuristic to the TFT problem since it performed very well on the selection ordering problem, as shown in Chapter 5. This chapter describes the implementation and experimental evaluation of the max-min heuristic on the TFT problem.

6.1 Max-min heuristic

Similar to the selection ordering problem, the brute-force approach to MRO for job scheduling is not practical, as discussed in Section 2.9.1. This is because it requires an exponential number of calculations to examine $n!$ scheduling plans and 2^n extreme scenarios for a set of n jobs. The max-min heuristic considers only $n+1$ max-min scenarios for each plan instead of all extreme scenarios. A *max-min scenario* in the TFT problem is an extreme scenario in which the first k jobs in a plan are assigned their maximum processing time and the following $n-k$ jobs are assigned their minimum processing time, where $0 \leq k \leq n$.

One motivation to apply max-min heuristic in the job scheduling problem is its encouraging performance in the selection ordering problem as seen in Chapter 5. Moreover, max-min scenarios play a special role in MRO for the job scheduling problem. Lebedev and Averbakh define an important class of job scheduling problems where the worst case scenarios are max-min scenarios [75]. This class considers a set of jobs with nested processing times with the same midpoints and no common interval boundaries (this class was discussed in Section 2.9.2).

In this section, we adapted the basic algorithm of the max-min heuristic as described in Section 4.1 for the selection ordering problem. Algorithm 6 shows the implemented max-min heuristic, $H(u, q)$, for the job scheduling problem. Apart from processing jobs rather than selection operators, the only other difference is the calculation of the regret in line 8.

As before, the algorithm is parameterised by two inputs: u, a (possibly empty) starting plan and q, an order in which to process jobs. Algorithm 6 starts with the outer *for* loop in line 1 and takes one job at a time from q, checking if it already exists in u, as seen in line 2. If so, the job is removed from u and tested in each position in u; otherwise, it directly checks the job in each position in u. Each time a job is checked in a position, a new plan is formed. This plan is considered under all max-min scenarios. The solution for the current stage will be the plan with the smallest maximum regret which in turn will be the starting plan for the next iteration. Ultimately, the algorithm will return u as a solution.

As for selection ordering, the max-min heuristic can start with some partial plan

Algorithm 6: $H(u, q)$

```
1  foreach job j from the sequence q do
2      if j is in u then
3          remove j from u
4      Assume u currently comprises i jobs;
5      foreach position k, 1 ≤ k ≤ i + 1, in u do
6          Temporarily insert j in position k in u;
7          foreach max-min scenario for u do
8              Calculate the regret of plan u;
9              Store the maximum regret for position k;
10     Choose as the final position for j in u the one that minimises the
       maximum regret;
11 Return u;
```

u. Section 2.9.2 discussed the importance of the domination relationship between jobs, since their relative order in any optimal plan is known. Therefore, for a given set of jobs J with subset J' of dominant jobs, we could start by assigning J' to u after sorting the dominant jobs in non-decreasing order. If we have more than one dominant set, then we could use one of the following: the subset S' with the maximum cardinality denoted by *D:C* (i.e. **D**ominant:**C**ardinality), the subset S' whose jobs have the largest total width of their processing time intervals denoted by *D:W* (i.e. **D**ominant:**W**idth), or the subset S' with the maximum cardinality whose total width of the processing time intervals is greatest, which is denoted by *D:CW* (i.e. first **D**ominant:**C**ardinality, then **W**idth).

The heuristic also accepts different orders for the jobs in q. They can be ordered based on the interval width of their processing times and that can be in non-decreasing or non-increasing order, denoted as $W+$ and $W-$ respectively. Alternatively, the jobs can have non-decreasing or non-increasing order according to the midpoints of their processing time intervals, denoted by $M+$ and $M-$ respectively. As before, the output of $H(u, q)$ can be assigned to u as an initial plan for

another iteration of the algorithm. In the following section we present the experimental evaluation for various versions of the max-min heuristic.

6.2 Experimental evaluation

The max-min heuristic for TFT has been implemented and tested experimentally. Moreover, the brute-force algorithm as well as the midpoint (2-approximation) algorithm have been implemented in order to evaluate the performance of the max-min heuristic. The testing environment (i.e. hardware and software configurations) was the same as that described in Chapter 5.

Below we first identify the measuring criteria used in the experimental evaluation, then describe how the data set was generated, and finally present the experimental results showing the performance of various versions of the max-min heuristic on the TFT problem.

6.2.1 Measuring criteria

The measuring criteria are the same as those for the selection ordering problem, as described in Section 5.1. We repeat them here for convenience. The *percentage of exact solutions found* is the number of cases where the heuristic generates a plan equivalent to the optimal minmax regret solution. The *regret ratio* $\lambda(J)$ is defined as follows:

$$\lambda(J) = \frac{R(H(u,q), X(J))}{R(P(J), X(J))} \tag{6.2.1}$$

The regret ratio was treated here in the same way as in Section 5.1. We calculated the *average regret ratio* for all cases with the same number number of jobs and also found the *worst regret ratio* (i.e. the maximum $\lambda(J)$) over them.

In order to evaluate the stability of the experiments, the variance and the confidence intervals are calculated for the percentage of exact solutions found and the overall average regret ratio. The margin of error for the percentage of exact solutions and the average regret ratio measures was less than ± 2 percentage points and ± 0.01

respectively, while their variance was less than 0.02. Full results for the statistical measures can be found in Appendix C.2.

6.2.2 Generating test data

A synthetic data set was used to evaluate the performance of the max-min heuristic for the TFT problem. It was generated in a similar way to the one described in Section 5.2.1.

Each test case consists of a set of k jobs, with k ranging from 2 to 10, and we say the test case belongs to group k. We generated a hundred different cases for each group k. In order to generate a single test case with k jobs, we generate $2k$ uniformly distributed random numbers from the range $[0, 100]$. Then we take a pair of values from the $2k$ random numbers in order to specify the processing time bounds for a single job, where the smaller value is associated with the lower bound and the larger value is associated with the upper bound. We call this setting a *mixed* setting, which means that we do not specify any special relationships between the generated jobs (e.g. nested or dominant). This setting is the most general and so fully tests the heuristic.

6.2.3 Experimental results

This section presents the results of the experimental evaluation for various versions of the max-min heuristic for the job ordering problem, comparing them with those of the midpoint, pessimistic and optimistic heuristics. For simplicity, throughout this section we will refer to the max-min heuristic with the chosen initial plan u and order q as simply (u, q) instead of $H(u, q)$.

Let us start with the pessimistic and optimistic heuristics. These heuristics consider a single scenario to find the solution. The pessimistic heuristic considers the scenario in which all jobs are assigned their maximum processing time, while the optimistic heuristic considers only the scenario in which all jobs are assigned their minimum processing time. Both heuristics find the exact minmax regret optimal solution for approximately 23% of the cases. However, the pessimistic heuristic has

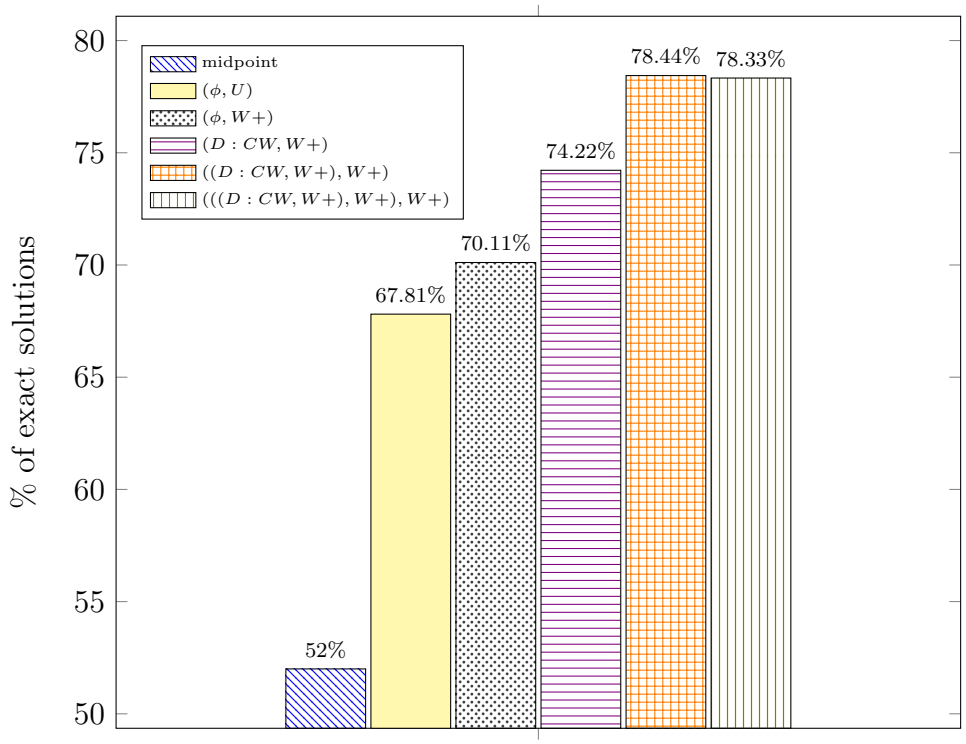


Figure 6.1: Overall percentage of exact solutions for the synthetic data set.

an overall worst regret ratio of 8.5, while that for the optimistic heuristic is 11.94. They are far worse than the midpoint heuristic which sorts the jobs in non-decreasing order according to the midpoints of their processing times (recall more details from Section 2.9), where the regret ratio is guaranteed to be no more than 2 [71]. In fact, the midpoint heuristic outperforms both the pessimistic and optimistic heuristics on all measures, so we will use it in comparison with the max-min heuristic. The overall worst regret ratio for the midpoint is 1.68, while its overall average regret ratio is 1.045689. It found the exact minmax regret optimal solution in 52% of the cases. The full results for the midpoint, pessimistic and optimistic heuristics can be found in Appendix C.1.

As for selection ordering, we evaluated the max-min heuristic experimentally with various settings. These included starting with an empty initial plan and considering random operator ordering $(\emptyset, U)$, ordering by midpoint ($M-$ and $M+$) and ordering by interval width ($W-$ and $W+$). Overall, the performance of non-decreasing order ($M+/W+$) was better than non-increasing order. The experimental evaluation showed that the $W+$ ordering (non-decreasing width) performed best,

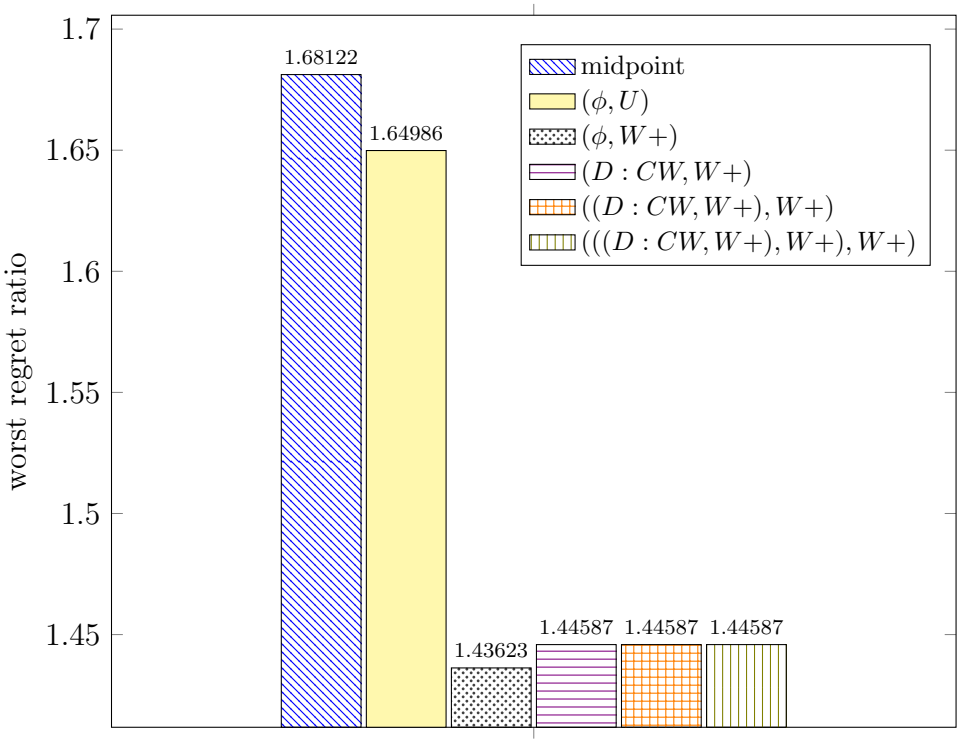


Figure 6.2: Overall worst regret ratio for the synthetic data set.

so it will be used in this section. We also tested the heuristic by starting with initial plan D:CW. Recall from Section 6.1 that D:CW stands for the largest subset of dominant jobs, and, in case of a tie, the one with the greatest total width of processing time will be chosen. Furthermore, we applied multiple phases where the result of the previous phase is passed into the next phase as an initial plan.

The $(\emptyset, U)$ version is used as a baseline. It outperforms the midpoint heuristic on all measures as shown in Figures 6.1, 6.2 and 6.3. The $(\emptyset, U)$ version finds the exact solution in 67.81% of the cases. The $(\emptyset, W+)$ version showed the effectiveness of ordering. The percentage of exact solutions found jumped to 70.11%, while the worst regret ratio dropped to 1.44 from 1.65 for $(\emptyset, U)$ and 1.68 for the midpoint heuristic as shown in Figure 6.2.

Starting with initial plan D:CW improved the percentage of exact solutions found to more than 74.22%, but did not improve the overall worst regret ratio or the overall average regret ratio. However, the average regret ratio can be improved by performing another iteration (i.e. $((D$:$CW, W+), W+)$) which improved the overall average regret ratio to 1.015 as shown in Figure 6.3. Figure 6.1, shows that the per-

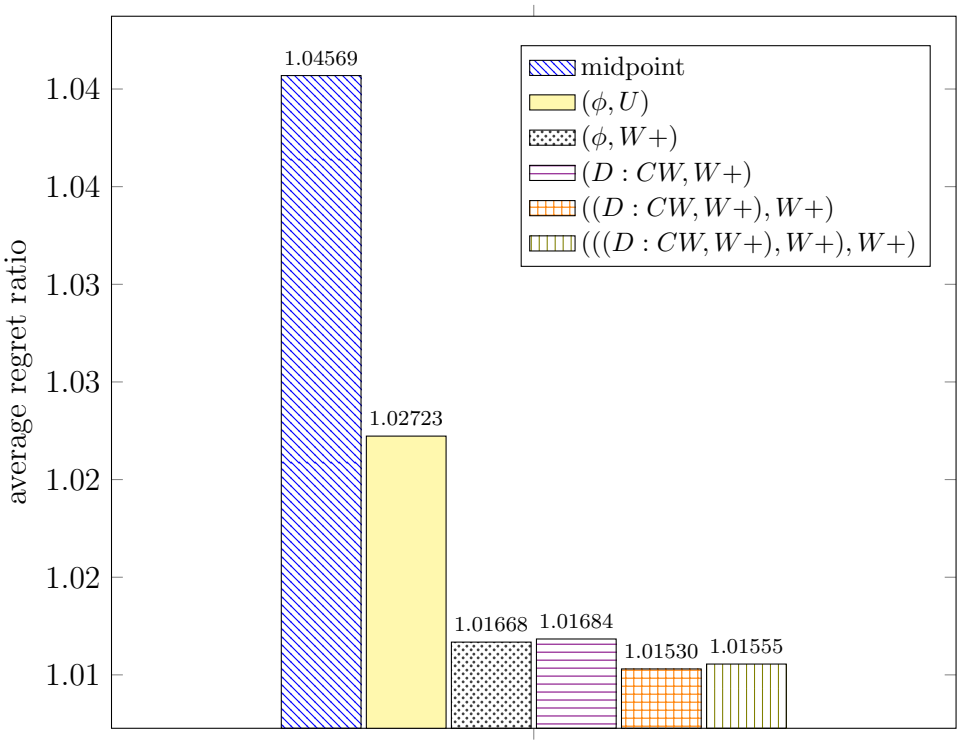


Figure 6.3: Overall average regret ratio for the synthetic data set.

centage of exact solutions found was also improved using the $((D{:}CW, W+), W+)$ version, reaching 78.44%. The $(((D{:}CW, W+), W+), W+)$ version does not improve the results when compared to the $((D{:}CW, W+), W+)$ version. During our experimental evaluation, we performed up to six iterations of the max-min heuristic $(D{:}CW, W+)$, but the results were the same as for $(((D{:}CW, W+), W+), W+)$ on all measures. Table C.1 in Appendix C.1 shows the full results for the max-min heuristic applied to the job scheduling problem.

6.3 Discussion of experimental results

In our experiments we started with $(\emptyset, U)$ version of the max-min heuristic. By starting with $(\emptyset, U)$ we could evaluate the basic version of the max-min heuristic without any parameters (i.e. initial plan and ordering criteria) and then study the effectiveness of the various parameters. Version $(\emptyset, U)$ outperformed the midpoint heuristic on all measures. Therefore, if the aim is to find the optimal order or to avoid bad orders as often as possible, then the basic version of max-min heuristic is

a better choice than midpoint.

The experimental evaluation shows the effectiveness of choosing an appropriate ordering. Version $(\emptyset, W+)$ improved the results by passing the jobs in non-decreasing order based on the width (i.e. imprecision) of their processing times. This makes sense because by using non-decreasing ordering, the max-min heuristic delays passing the most uncertain jobs to the algorithm, which in turn means they are tested in more positions and this improves the result. Using initial plan $D{:}CW$ improved the percentage of exact solutions found. However, adding an extra iteration using $((D{:}CW, W+), W+)$ showed better performance in terms of worst regret ratio and overall average regret ratio. This extra iteration takes the good result of $(D{:}CW, W+)$ and enhances it by allowing all jobs to be tested again, this time in all positions of the plan.

The experiments show that in a few cases the worst regret ratio for $(\emptyset, W+)$ is better than that for $(D{:}CW, W+)$, which in turns affects the average regret ratio. A possible explanation is that sometimes the initial plan has only a few jobs, yet large processing time intervals, while the remaining jobs have small processing time intervals. In these cases, the jobs with big intervals are fixed in their position by the initial plan, while the jobs with smaller intervals have the chance to be tested in every position of the plan. This leads to the same flaw as for the non-increasing order of jobs. Misplacing the jobs with the most uncertain processing times has a negative impact on the quality of the solution. Therefore, it is better to pass them to the heuristic later, allowing them to be tested in more positions with a better chance of finding the correct position.

6.4 Conclusion

An advantage of the midpoint heuristic for job scheduling is that it guarantees a bound for the generated solution which does not exceed two times the optimal solution. It is also the case that the max-min heuristic is more complex than the midpoint heuristic. However, the experimental evaluation showed that the max-min heuristic gives higher quality results and avoids plans with high regret when

compared to the midpoint heuristic. According to the experiments, the max-min heuristic found the exact optimal solution in more than 78% of the cases, while the midpoint heuristic found just 52%. As a result, the midpoint heuristic can be used for a quick solution with reasonable quality. However, better results can be obtained using $(\emptyset, W+)$ at the expense of extra time. On the other hand, if time is not a constraint, then $((D{:}CW, W+), W+)$ is the best option for the highest quality results.

This chapter applies the max-min heuristic to the TFT problem. Then, it presented the experimental evaluation for the max-min heuristic with various settings and compare them to the midpoint, pessimistic and optimistic heuristics. In the next chapter we discuss the problem of ordering join operators when their selectivities are only known to fall within intervals.

Chapter 7

The Join Ordering Problem

In previous chapters we studied the selection ordering problem and our proposed heuristic to solve that problem. We also studied the effectiveness of the developed heuristic for the selection ordering problem experimentally.

This chapter discusses the join ordering problem, where similarly to the selection ordering problem, we assume that the selectivity of each join predicate is known to be in some interval. Using the same conventions and definitions introduced in Section 2.5, this chapter first defines the minmax regret optimisation problem for the join ordering problem in Section 7.1, followed by some properties of the problem in Section 7.2. A heuristic for the join ordering problem, based on the max-min heuristic for selection ordering and Algorithm 1 in Section 2.5.3, is presented in Section 7.3.

7.1 Join minmax regret optimisation

Before defining the minmax regret optimisation problem, we define the join ordering problem where the selectivities of the join predicates are defined partially and fall within some particular interval of values. For the join ordering problem, we will consider only connected chain queries (recall Definition 2.5.4 in Section 2.5.1), and their associated left-deep join trees. Let $Q = (R, P)$ be a connected chain query, where $R = \{R_0, R_1, \ldots, R_n\}$ is a set of relations, $P = \{p_1, p_2, \ldots, p_n\}$ is a set of predicates, and p_i represents the join predicate between R_{i-1} and R_i, $1 \leq i \leq$

n. Recall that we can represent the selectivities of join predicates in P as a set $S = \{s_1, s_2, \ldots, s_n\}$, where s_i represents the selectivity between relations R_{i-1} and R_i. Suppose now that each selectivity is defined by a closed interval, such that $s_i = [\underline{s}_i, \overline{s}_i]$ where $\underline{s}_i, \overline{s}_i \in [0, 1]$ for $1 \leq i \leq n$.

Recall that an assignment of concrete values to all n selectivities is called a *scenario* and is defined by a vector $x = (s_1, s_2, ..., s_n)$, with $s_i \in [\underline{s}_i, \overline{s}_i]$. Recall also that $X(Q) = \{x \mid x \in [\underline{s}_1, \overline{s}_1] \times [\underline{s}_2, \overline{s}_2] \times \cdots \times [\underline{s}_n, \overline{s}_n]\}$ denotes the set of all possible scenarios, and that π^n denotes the set of all possible permutations over $1, 2, \ldots, n$. For $\pi_j \in \pi^n$, $\pi_j(i)$ denotes the i-th element of π_j. Let π^v, where $\pi^v \subseteq \pi^n$, be the set of all permutations associated with valid left-deep join trees.

Definition 7.1.1 A query execution plan ρ_j for a connected chain query $Q = (R, P)$ is a permutation $p_{\pi_j(1)}, p_{\pi_j(2)}, \ldots, p_{\pi_j(n)}$ of the n join predicates in P. The set of all possible query execution plans associated with left-deep join trees is given by:

$$J(Q) = \{\rho \mid \rho = p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)} \text{ such that } \pi \in \pi^v\}.$$

Recalling the cost formula in Equation (2.5.3), the cost for evaluating plan ρ_j under a given scenario x is:

$$Cost(\rho_j, x) = \sum_{i=1}^{n} \left(\prod_{j=0}^{i} (s_j * r_j) \right) \tag{7.1.1}$$

Let $\rho_{opt(x)}$ be the optimal plan for scenario x, which is the query execution plan in $J(Q)$ which has the minimal cost for scenario x, and let $\pi_{opt(x)}$ be the permutation of the join predicates for this plan. Since we are facing multiple scenarios, we use minmax regret optimisation, as in Section 3.2, to determine the quality of a plan ρ_j.

Definition 7.1.2 Given a plan ρ and a scenario x, the absolute *regret* $\gamma(\rho, x)$ of ρ for x is:

$$\gamma(\rho, x) = Cost(\rho, x) - Cost(\rho_{opt(x)}, x) \tag{7.1.2}$$

The maximal regret of a plan is the regret for its worst-case scenario and is simply defined as $\max_{x \in X}(\gamma(\rho, x))$.

Definition 7.1.3 Given the set $J(Q)$ of all possible join execution plans for query Q and the set $X(Q)$ of all possible scenarios, minimising the maximal regret is done as follows (where $R(J(Q), X(Q))$ is the optimal regret):

$$R(J(Q), X(Q)) = \min_{\rho \in J}(\max_{x \in X}(\gamma(\rho, x)))$$

Then the *minmax regret optimisation* problem for Q, which we denote $MRO(Q)$, is to find a plan whose maximum regret matches $R(J(Q), X(Q))$. For simplicity and when there is no confusion, we also use $MRO(Q)$ to denote a plan which minimises $R(J(Q), X(Q))$.

7.2 Precedence adjacency property

In this section we will study the precedence adjacency property, which was defined in Lemma 2.5.11, in the case of interval selectivities. This property is useful because it reduces the number of plans which need to be considered by any heuristic. We use the precedence adjacency property in our heuristic, as we discuss in Section 7.3.

Consider a precedence graph G where each join predicate is associated with an interval selectivity. Dependent on the particular precedence graph, relation R_i is associated with either selectivity s_i (if its parent in G is R_{i-1}) or selectivity s_{i+1} (if its parent in G is R_{i+1}). Based on the considered scenario, the selectivity for R_i can be either its minimum or maximum value. For simplicity, we use $Rank(\underline{R}_i)$ and $Rank(\overline{R}_i)$ to refer to the rank for relation R_i when the associated selectivity takes its minimum and maximum value, respectively. In the following, we define some relationships between relations in precedence graph G_i based on their rank (recall Definition 2.5.9).

Definition 7.2.1 Given two relations $R_M, R_N \in R$ in precedence graph G_i, we say that R_M *dominates* R_N if $Rank(\underline{R}_M) \leq Rank(\underline{R}_N)$ and $Rank(\overline{R}_M) \leq Rank(\overline{R}_N)$.

The following definitions define two special kinds of domination.

Definition 7.2.2 Given two relations $R_M, R_N \in R$ in precedence graph G_i, we say that R_M *strictly dominates* R_N if $Rank(\overline{R}_M) < Rank(\underline{R}_N)$.

Definition 7.2.3 Given two relations $R_M, R_N \in R$ in precedence graph G_i, we say that R_M and R_N are *dominant overlapped* operators if $Rank(\underline{R}_M) \leq Rank(\underline{R}_N)$ and $Rank(\overline{R}_M) \leq Rank(\overline{R}_N)$ as well as $Rank(\overline{R}_M) \geq Rank(\underline{R}_N)$.

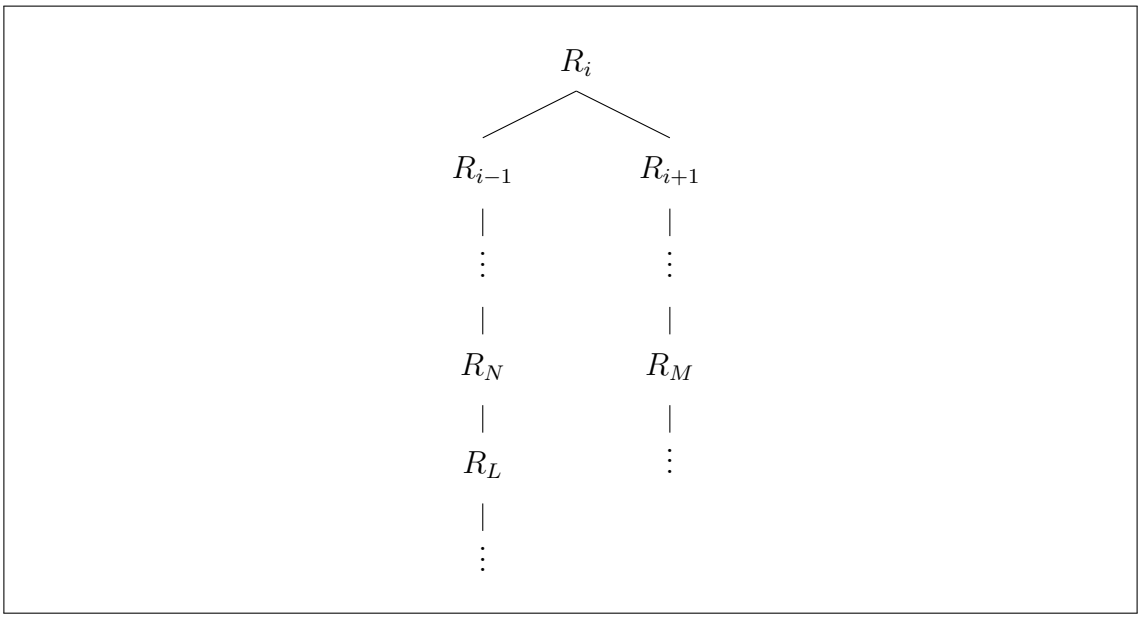


Figure 7.1: Precedence graph G_i.

Next we first study the precedence adjacency property in the case of strictly dominant relations in G_i, before considering the general case of domination in Section 7.2.2.

7.2.1 Strict domination

Given a precedence graph G_i with relations R_L, R_M and R_N as shown in Figure 7.1, suppose that the relationships between the ranks are as follows: $Rank(\overline{R}_L) < Rank(\underline{R}_M)$ and $Rank(\overline{R}_M) < Rank(\underline{R}_N)$. Therefore, under any scenario R_L strictly dominates R_M and R_M strictly dominates R_N, as illustrated in Figure 7.2. Assuming that the minmax regret optimal plan starts with R_i, we will show that no relation can appear between R_N and R_L in the minmax regret optimal plan.

Lemma 7.2.4 The precedence adjacency property holds for the case of strict domination.

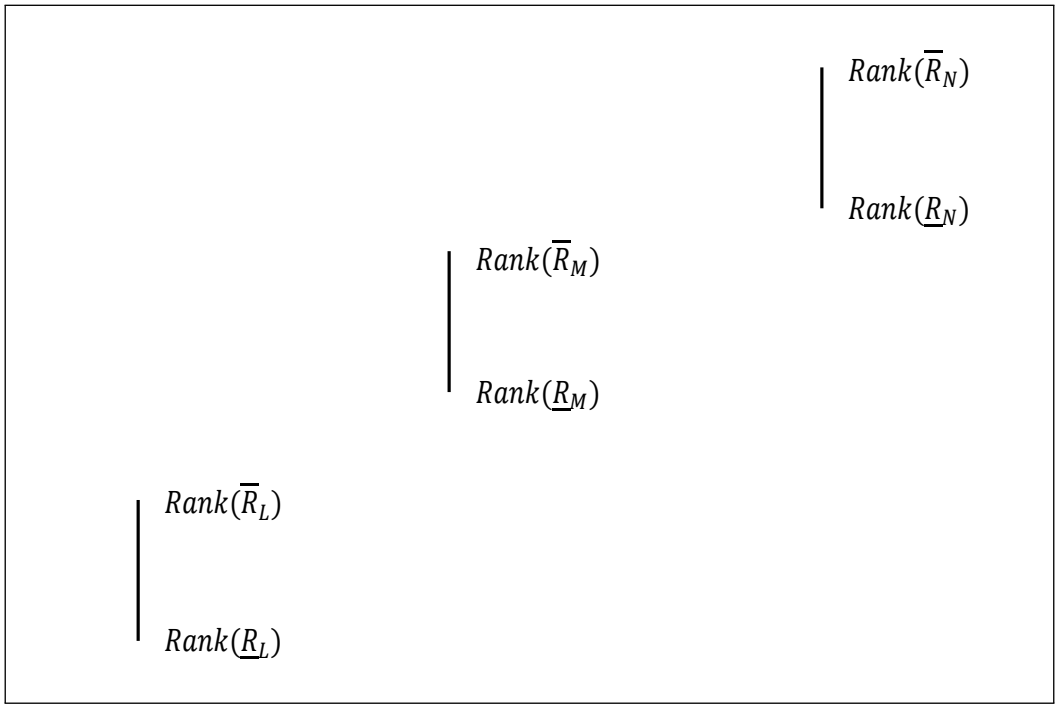


Figure 7.2: Strict domination between relations R_L, R_M and R_N.

Proof: Let G_i be a precedence graph with relations R_L, R_M and R_N described as above. Due to the precedence constraints in G_i, R_N must precede R_L in any optimal plan that minimises the maximum regret. For any sequences of relations A and B, assume plan $\rho = AR_NR_MR_LB$, in which R_M is in between R_N and R_L, is a plan that minimises the maximum regret (proof by contradiction). Let us swap R_M with its neighbours to generate the following plans: $\rho' = AR_NR_LR_MB$ and $\rho'' = AR_MR_NR_LB$. Using Lemma 2.5.10, the following are true under any scenario x:

$$C_{out}(\rho', x) < C_{out}(\rho, x) \quad \text{because} \quad Rank(R_L) < Rank(R_M) \tag{7.2.1}$$

$$C_{out}(\rho'', x) < C_{out}(\rho, x) \quad \text{because} \quad Rank(R_M) < Rank(R_N) \tag{7.2.2}$$

In order to compare the regret of ρ' (or ρ'') with ρ under any scenario x we only need to compare their costs. This is because the optimal plan for x is the same for both regrets. Since the inequality in (7.2.1) is true for any scenario, it is also true for the worst-case scenario y' of plan ρ'. Using Equation (7.2.1), we get the following:

$$C_{out}(\rho', y') - C_{out}(\rho, y') < 0 \tag{7.2.3}$$

Equation (7.2.3) shows that the maximum regret for plan ρ' is smaller than the maximum regret for plan ρ which contradicts the assumption that ρ is the optimal plan. The same argument can be made analogously using Equation (7.2.2) to show that the maximum regret for plan ρ'' is smaller than the maximum regret for plan ρ which is also a contradiction. Therefore, R_N must be followed immediately by R_L in the plan that minimises the maximum regret. □

7.2.2 Domination

After proving that the precedence adjacency property holds in the case of strict domination, we investigate the property in the general case of domination. The case of strict domination is somewhat similar to the case of constant selectivities. This similarity lies in the fact that the relative optimal order of the strictly dominant relations, based on rank, is preserved throughout all scenarios, as if the predicates had constant selectivities. On the other hand, the relative optimal order of dominant overlapped relations, based on rank, can change depending upon the scenario.

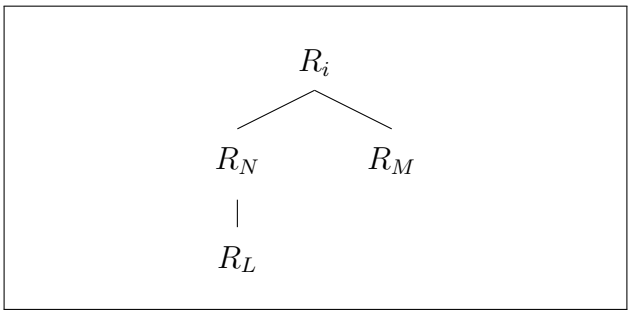


Figure 7.3: Counter-example precedence graph used in Lemma 7.2.5.

Lemma 7.2.5 The precedence adjacency property does not hold for the general case of domination.

Proof: We will construct an example to show that the precedence adjacency property does not hold for overlapped domination. Suppose we are given a chain query Q with the following order of relations: R_L, R_N, R_i, R_M. Let G_i be the precedence graph with R_i as a root, and relations R_L, R_M and R_N as shown in Figure 7.3, where all relations have the same cardinality of 1. Let R_L strictly dominate R_N,

	s_i $\underline{s}_L$ $\underline{s}_N$ $\underline{s}_M$	s_i $\underline{s}_L$ $\underline{s}_N$ $\overline{s}_M$	s_i $\underline{s}_L$ $\overline{s}_N$ $\underline{s}_M$	s_i $\underline{s}_L$ $\overline{s}_N$ $\overline{s}_M$	s_i $\overline{s}_L$ $\underline{s}_N$ $\underline{s}_M$	s_i $\overline{s}_L$ $\underline{s}_N$ $\overline{s}_M$	s_i $\overline{s}_L$ $\overline{s}_N$ $\underline{s}_M$	s_i $\overline{s}_L$ $\overline{s}_N$ $\overline{s}_M$	Max Regret
$R_iR_MR_NR_L$	0.00	**0.60**	0.00	0.44	0.00	0.45	0.00	0.17	0.60
$R_iR_NR_MR_L$	0.10	0.30	0.50	**0.54**	0.10	0.15	0.50	0.27	**0.54**
$R_iR_NR_LR_M$	0.00	0.00	0.32	0.0	0.14	0.00	**0.58**	0.00	0.58

Table 7.1: The regret for each plan under each scenario for the example in Lemma 7.2.5.

so $\overline{s}_L < \underline{s}_N$. Consider the case in which R_L dominates R_M, and R_M dominates R_N, so that $\underline{s}_L \leq \underline{s}_M$ and $\overline{s}_M \leq \overline{s}_N$. Assume that the optimal minmax regret plan for Q starts with R_i. We note that if we make the difference $(\overline{s}_M - \underline{s}_L)$ large and make $\overline{s}_L$ as close as possible to $\underline{s}_N$, then we can generate an optimal minmax regret plan where R_M is in between R_N and R_L. Consider the following values for G_i as a counter-example.

Since R_i is the root of G_i, its selectivity is $s_i = [1, 1]$. Let $s_L = [0.2, 0.49]$, $s_M = [0.4, 0.8]$ and $s_N = [0.5, 0.9]$. Table 7.1 presents the regret of each plan consistent with G_i under each scenario. The minmax regret solution for this example is $R_iR_NR_MR_L$ which violates the precedence adjacency property. As a result, the precedence adjacency property does not hold for the general case of domination. □

7.3 Max-min heuristic for join ordering

This section presents our max-min heuristic for solving the join ordering problem that was described in Section 7.1. It is important to mention that the heuristic presented here is different from the one described in Chapter 4. The only common feature they share is that both use max-min scenarios when finding a solution. Some concepts from the algorithm discussed in Section 2.5.3, which deals with precise selectivity values, in addition to the findings from the previous section, are used in this heuristic.

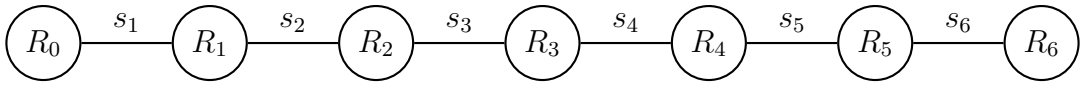


Figure 7.4: The graph of the connected chain query Q in Example 7.3.1.

The max-min heuristic for the join ordering problem accepts a connected chain query graph. It finds a plan that performs well by considering max-min scenarios. Algorithm 7 presents the max-min heuristic formally. In general, the heuristic considers one precedence graph at a time. For each precedence graph, it finds the best plan under max-min scenarios, and from those it chooses the plan with the smallest maximum regret to be the overall solution.

When the heuristic considers a precedence graph G_i, it calls $getTreeMaxminPlan$ in Algorithm 7. This function starts by forming sequence p which consists of the root followed by the relations of the longer branch, preserving their relative order, as shown in line 2. The relations of the shorter branch form sequence q as in line 3, with their relative order also preserved. The union of p and q will eventually form the complete plan.

In the max-min heuristic, we took advantage of our finding in Lemma 7.2.4 regarding the precedence adjacency property in the case of strict domination. This property states that if there are two relations M and N where N is a child of M and $Rank(\underline{M}) \geq Rank(\overline{N})$, then no relation can appear between them in the optimal plan. The max-min heuristic applies this property. It scans both sequences p and q and groups together any strictly dominant relations that satisfy the precedence adjacency property. The grouping mechanism enables us to place the grouped sequence of relations together and make sure that no relation (or sequence of relations) can be placed in between them during the processing of the heuristic. This in turns reduces the number of possible places in which any relation/sequence needs to be checked later on as well. The equivalent grouped sequence for p, is assigned to p' in line 4, while that for q is assigned to q' in line 5. The following example demonstrates the generation of p' and q' using some concrete values.

Example 7.3.1 Let Q be a connected chain query whose graph is shown in Figure 7.4. Let $R = \{R_0, R_1, R_2, R_3, R_4, R_5, R_6\}$ be the set of relations involved in Q,

Algorithm 7: *Max-min Heuristic*

```
1 getQueryPlan(Q)
    Input: connected chain query Q
    Output: Max-min heuristic plan heuristicPlan for Q
    // Let G_0, ..., G_{n-1} be the precedence graphs for Q
2   heuristicPlan = getTreeMaxminPlan(G_0);
3   for 1 ≤ i ≤ n − 1 do
4       tempPlan = getTreeMaxminPlan(G_i);
5       if Regret(heuristicPlan) > Regret(tempPlan) then
6           heuristicPlan = tempPlan;
7   return heuristicPlan;

1 group(C)
    Input: A sequence C of relations
    Output: A sequence C' comprising groups of relations from C
2   while there is a relation M with a child N such that
    Rank(M) ≥ Rank(N) do
3       Combine M and N in one group;
4   return C';

1 ungroup(C')
    Input: A sequence C' comprising groups of relations
    Output: A sequence C of relations resulting from ungrouping the group
            sequence in C'
2   Let C be the sequence of relations resulting from replacing each group in
    C' by its equivalent sequence;
3   return C;
```

```
1  getTreeMaxminPlan(G_i)
       Input: precedence graph G_i
       Output: Max-min heuristic plan treePlan for G_i
2      Let p be the sequence starting with the root relation followed by the
       relations of the longer branch in G_i;
3      Let q be the sequence of relations from the shorter branch in G_i;
4      p' = group(p);
5      q' = group(q);
6      treePlan = p';
       // initialise j with the first position in treePlan to attempt
          inserting a group from q'
7      j = 1;
8      foreach group O in q' do
9          for j ≤ i ≤ |treePlan| + 1 do
10             Temporarily insert O in position i in treePlan;
11             foreach max-min scenario for treePlan do
12                 Calculate the regret of plan treePlan;
13                 Store the maximum regret for position i;
14         Choose as the final position k for O in treePlan that which minimises
           the maximum regret;
15         j = k + 1;
16     return ungroup(treePlan);
```

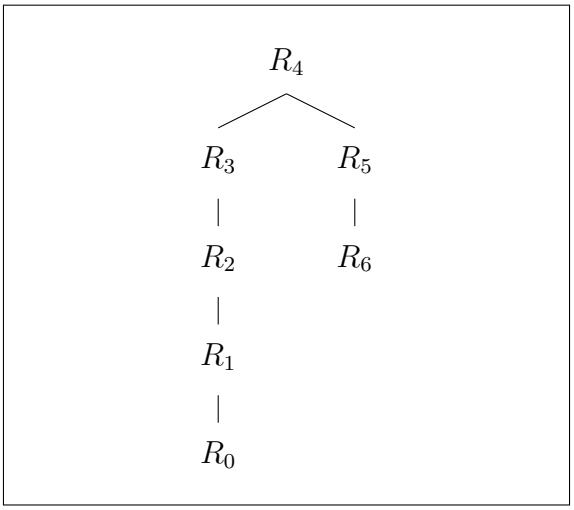


Figure 7.5: The precedence graph G_4 used in Example 7.3.1.

R_i	$Rank(\underline{R}_i)$	$Rank(\overline{R}_i)$
R_0	0.9983870968	0.9985074627
R_1	0.9999994118	0.9999998701
R_2	0.9999996552	0.9999998795
R_3	0.6666666667	0.9891304348
R_4	root	
R_5	0.9750000000	0.9850746269
R_6	0.9995454545	0.9996666667

Table 7.2: The minimum and maximum ranks for relations in G_4 in Example 7.3.1.

with the cardinalities of the relations as follows: $r_0 = 10^3$, $r_1 = 10^7$, $r_2 = 10^7$, $r_3 = 10^2$, $r_4 = 10^3$, $r_5 = 10^2$ and $r_6 = 10^4$. Let $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ be the set of selectivities for the join predicates in Q, with selectivity intervals as follows: $s_1 = [0.62, 0.67]$, $s_2 = [0.17, 0.77]$, $s_3 = [0.29, 0.83]$, $s_4 = [0.03, 0.92]$, $s_5 = [0.08, 0.67]$ and $s_6 = [0.22, 0.30]$.

Assume that Algorithm 7 is considering the precedence graph G_4 as shown in Figure 7.5. The first step taken by $getTreeMaxminPlan$ is to generate sequences p and q. The root, along with the left branch, is assigned to p, since the left branch is the longer branch, while the right branch is assigned to q. Therefore, $p = R_4, R_3, R_2, R_1, R_0$ and $q = R_5, R_6$.

The next step in *getTreeMaxminPlan* is to generate sequences p' and q' using *group*. Table 7.2 shows the ranks for all relations using Equation (2.5.1)[1]. The precedence adjacency property holds for one pair only, namely R_0 and R_1, since $Rank(\underline{R}_1) > Rank(\overline{R}_0)$. Therefore, they will be grouped in p', so no relation/sequence from q' is placed in between them. As a result, $p' = R_4, R_3, R_2, (R_1, R_0)$ where the grouped relations are enclosed in parentheses, and $q' = R_5, R_6$. ◇

The heuristic then takes the first group from q' and places it in each possible position in p' to form a temporary plan, which is what the outer loop in *getTreeMaxminPlan* does at line 8. For each temporary plan formed, the regret is calculated under all max-min scenarios (the inner *for* loop starting at line 9), and the plan with the smallest maximum regret is stored (line 14). The chosen plan from each iteration of the outer loop is used as the basis of the following iteration. In each subsequent iteration, the next group from q' is placed at each position after its parent in the plan that was generated in the previous step. The newly generated plan is also tested under all max-min scenarios and the plan with the smallest maximum regret will be chosen. This process continues until all groups of q' are placed in *treePlan*. Finally, *ungroup* is called on the resulting plan to form the heuristic solution. In the ungroup operation, any group is expressed in its original form as a sequence of relations. Algorithm 7 describes the full process of our max-min heuristic for the join ordering problem.

It is interesting to note that applying the max-min heuristic is straightforward for G_0 and G_{n-1}. This is because there is only one plan to consider for both cases. On the other hand, for any other precedence graph G_i we have i relations on the left branch and $n-i-1$ relations on the right branch. Therefore, if $i \geq n-i-1$ then the relations on the left branch will be assigned to parameter p in the heuristic and the relations on the right branch will be assigned to q, or vice versa if $i < n-i-1$. Then the heuristic proceeds as described above.

[1]The high precision is needed to distinguish between relations' ranks in Table 7.2 due to the large cardinality values (recall Equation (2.5.1) for the rank formula). Without high precision, some relations would incorrectly have same rank, which would adversely affect the quality of the result.

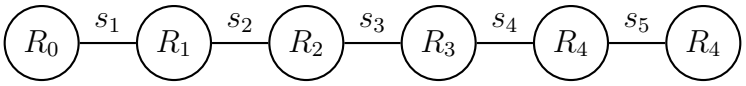


Figure 7.6: The graph of the connected chain query in Example 7.3.2.

It is clear that our heuristic runs in polynomial-time. We are considering $n+1$ precedence graphs, each one of them being passed to $getTreeMaxminPlan$. The initial plan p formed in $getTreeMaxminPlan$ consists of at least $\lceil n/2 \rceil$ and at most n relations. The remaining relations/groups are considered in $i+1$ possible positions (that is $O(n)$) under $i+2$ max-min scenarios (i.e. $O(n)$), where i is the length of sequence q'. The regret of the considered plan can be computed in $O(n^2 \log n)$ time (this complexity is mainly for finding the optimal plan for the considered scenario) [73]. Therefore, the overall complexity is $O(n^6 \log n)$. However, in practice we found that the heuristic runs quickly, especially when it finds relations satisfying the precedence adjacency property and groups them, which in turn reduces the number of possible plans. The following example shows that the execution time of the heuristic at the logical query optimisation stage is a small fraction of the execution time of the query itself, specially when dealing with queries joining a number of large relations.

Relation	Relation name	Cardinality
R_0	`kind_type`	7
R_1	`aka_title`	361472
R_2	`title`	2528312
R_3	`cast_info`	36244344
R_4	`name`	4167491
R_5	`aka_name`	901343

Table 7.3: Cardinalities of relations in Figure 7.6.

Example 7.3.2 Join queries are commonly used when querying the Internet Movie Data Base (IMDB) where users require a combination of information about movies, genres, actors, directors, producing companies etc. A snapshot of the IMDB from

2013 has 21 relations and is 3.6 GB in size [76]. The cardinality of the relations vary, with some containing as many as 36 millions records.

We tested the running time for the chain query in Figure 7.6 which joins the six relations whose cardinalities are shown in Table 7.3. This query took 30 seconds to execute, using same testing environment (i.e. hardware and software configurations) of Chapter 5. However, only 0.2 seconds is needed by our heuristic to find the optimal minmax regret order in which to join the relations. Spending less than 1% of the query running time for optimisation is not a large overhead in real life, given the possible outcomes. ◇

7.4 Conclusion

This chapter started by formulating the join ordering problem under imprecise environment and defining the minmax regret optimisation version of the problem assuming that the selectivity values are known to fall within intervals. After presenting some properties of the problem, we introduced our novel heuristic which leveraged the properties we had found. In the following chapter, we evaluate the max-min heuristic experimentally.

Chapter 8

Experimental Evaluation of the Join Ordering Heuristic

The previous chapter introduced the join ordering problem and presented our max-min heuristic, which we applied to the join ordering problem for a given connected chain query. This chapter discusses our experimental evaluation of the heuristic.

Similar to the experimental evaluation in Chapter 5 for the selection ordering problem, we tested the max-min join heuristic in a controlled environment, separate from the query optimisers in available database servers. This allows us to study the performance of the heuristic itself before extending it to involve more operators and integrating it into existing query optimisers.

For the experimental evaluation, we used a commodity PC with a 3.19 GHz Intel Core i5 processor and 8 GB RAM. The operating system was Windows 7 Enterprise (64-bit). The Java programming language was used to implement both the minmax regret brute-force algorithm and the max-min heuristic. The Java code was compiled using the Eclipse IDE (Juno release) that is JDK compliance and uses the JavaSE-1.7 execution environment.

8.1 Measuring criteria

In order to evaluate the join max-min heuristic, we used the same measuring criteria that we used in Sections 5.1 and 6.2.1 when conducting the experimental evaluation

for the selection ordering problem and TFT problem respectively. Recall that these criteria are the percentage of exact solutions found by the heuristic, the average regret ratio and the worst regret ratio (which is simply the overall maximum regret ratio value over all test cases).

The join max-min heuristic was also compared with three baseline heuristics which consider only a single selectivity value for each join predicate instead of considering the entire selectivity interval. These heuristics are the midpoint, pessimistic and optimistic heuristics which choose the midpoint, maximum and minimum selectivity value, respectively, for each join predicate.

The statistical measures of variance and margin of error were used to evaluate the stability of the experimental results. These measures were calculated for the percentage of exact solutions found as well as for the overall average regret ratio. In both cases the variance and margin of error were significantly low (see Table D.2 in Appendix D.2).

8.2 Generating test data

In this experimental evaluation we use a synthetic data set, in which we randomly generate connected chain queries. The selectivities of the join predicates are generated using an approach similar to that for generating *mixed* sets of selectivities for the selection operators as described in Section 5.2.1. On the other hand, the cardinality for each relation in a query is chosen randomly from the set $\{10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$, which provides for a wide range of cardinalities.

For each generated test case, we checked that each predicate joining a pair of relations returns at least one tuple; if not, we randomly choose new cardinalities for the relations and generate a new selectivity interval for the join predicate. The following example illustrates the problem.

Example 8.2.1 Consider a test case with only one join predicate p_1, such that the cardinalities of R_0 and R_1 are both 10, while p_1 has the selectivity interval $s_1 = [0.002, 0.56]$. Using Equation 2.5.1 in Section 2.5.1 to calculate the number of tuples in the result of the join when s_1 takes its minimum and maximum values, we

have:

$$|R_0 \bowtie_{\underline{s}_1} R_1| = \quad 0.002 \times 10 \times 10 \quad = 0.2$$
$$|R_0 \bowtie_{\bar{s}_1} R_1| = \quad 0.56 \times 10 \times 10 \quad = 56$$

Because less than one tuple is returned when s_1 takes its minimum value, this case will be discarded and a new case will be generated. ◇

We generated ten groups of test cases. Each group comprises queries with k relations, where k varies from 3 to 12. Therefore, a test case from group k has $k-1$ predicates joining k relations (to form a connected chain query). We generated 100 different test cases for each group k, to yield 1000 test cases overall.

8.3 Synthetic experimental results

As mentioned earlier we used the same hardware and software environment as the one we used in conducting the experiments for the selection ordering and TFT problems. The brute-force algorithm for MRO of the join ordering problem was implemented in order to evaluate the quality of plans produced by the max-min join heuristic. Similar to Chapters 5 and 6, the heuristic for the join ordering problem was compared to the midpoint, pessimistic and optimistic heuristics.

Let us first consider the pessimistic and optimistic heuristics. As in the experiments for selection ordering in Chapter 5, both heuristics perform worse than the midpoint and max-min heuristics. In term of finding the exact solution, they perform better than in previous experiments. The pessimistic heuristic found the optimal solution in 90.7% of the cases, better than the optimistic heuristic which found 81.5%, as shown in Figure 8.1. However, they performed poorly with respect to the overall worst regret ratio measure. The overall worst regret ratio for the pessimistic heuristic was 6.08 and the value jumped to more than 242 in the case of the optimistic heuristic, as shown in Figure 8.2. Figure 8.3 shows the overall average regret ratio for both the pessimistic and optimistic heuristics, with values of 1.07 and 2.37, respectively.

The midpoint heuristic, which considers only scenarios where all predicates are

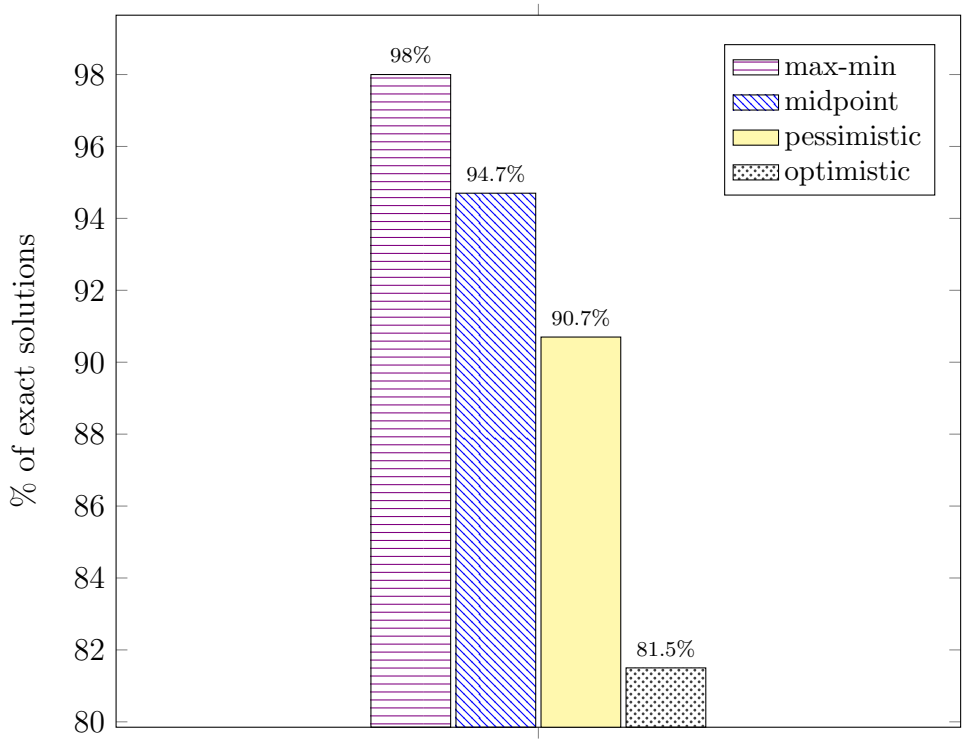


Figure 8.1: Overall percentage of exact solutions for the synthetic data set.

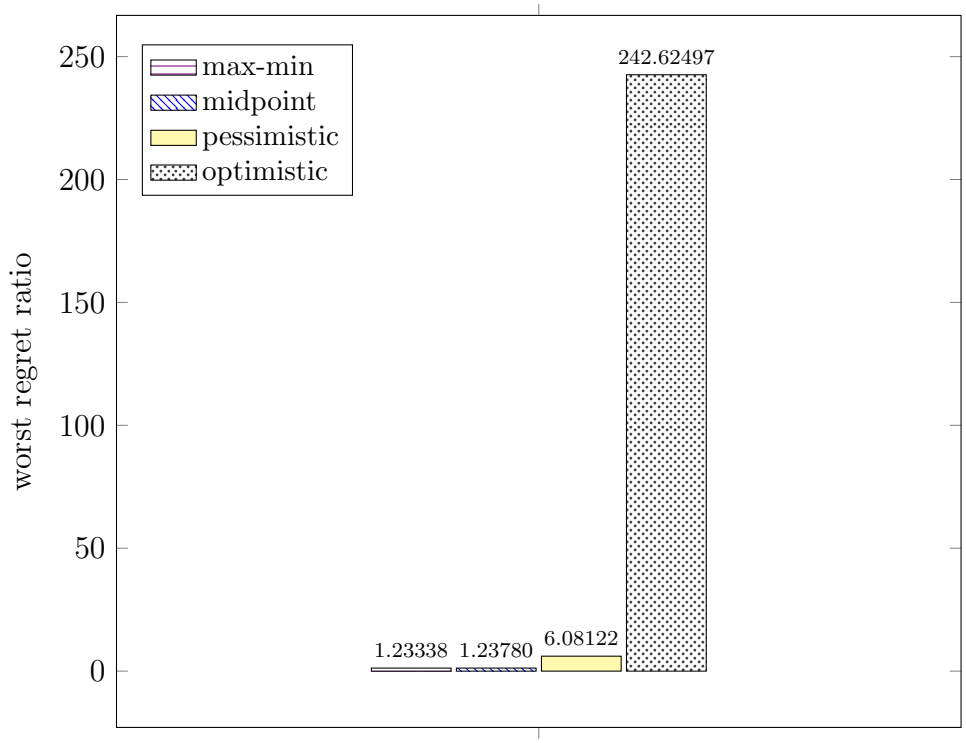


Figure 8.2: Overall worst regret ratio for the synthetic data set.

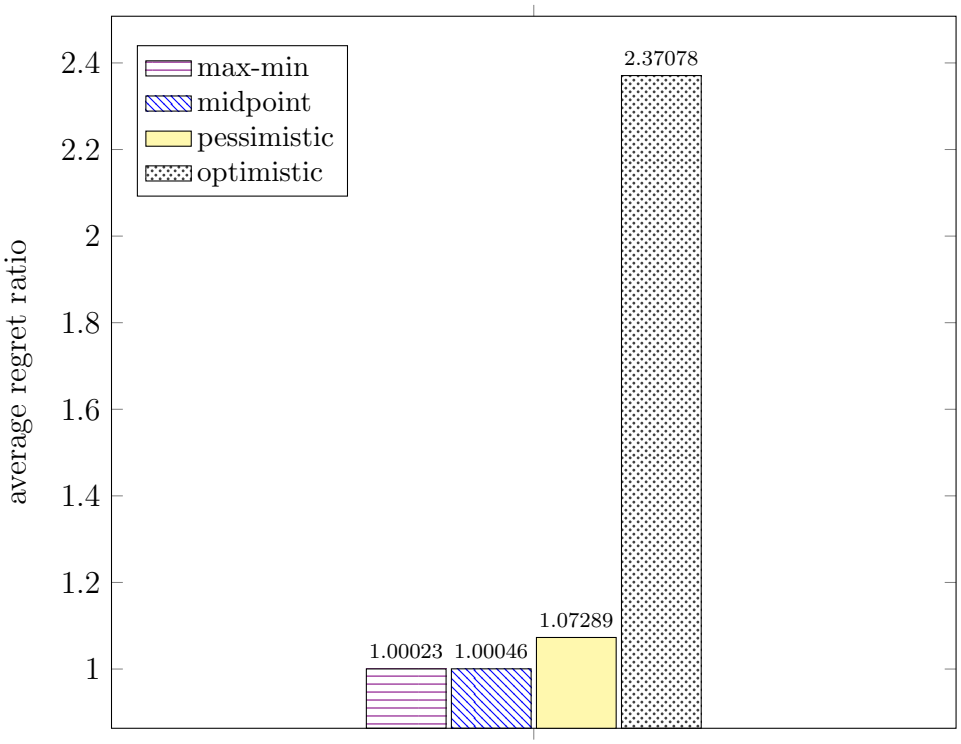


Figure 8.3: Overall average regret ratio for the synthetic data set.

assigned their midpoint selectivities, performed better than both the pessimistic and optimistic heuristics on all criteria. The midpoint heuristic found the optimal solution in 94.7% of the tested cases, with overall average regret ratio of 1.00046 and worst case regret ratio of 1.2378.

The experimental evaluation demonstrates the superior performance of the max-min heuristic compared to the other heuristics on all measuring criteria. Figure 8.1 shows that the max-min heuristic finds 98% of optimal solutions. In fact, our heuristic found the optimal minmax regret solution in all test cases using 3 to 6 relations as well as those with 8 relations (refer to Table D.1 in Appendix D.1). In comparison, the midpoint found all optimal solutions only for test cases using 3 to 5 relations. Even though the max-min heuristic performs better than the midpoint heuristic in terms of the worst regret ratio and average regret ratios as shown in Figures 8.2 and 8.3 respectively, the improvement was not large. The overall worst regret ratio of the max-min heuristic is 1.23338 (compared to 1.2378), while its average regret ratio is 1.00023 (compared to 1.00046). Our heuristic did not find the optimal solution in only 20 test cases (out of 1000), and, apart from the single case with the

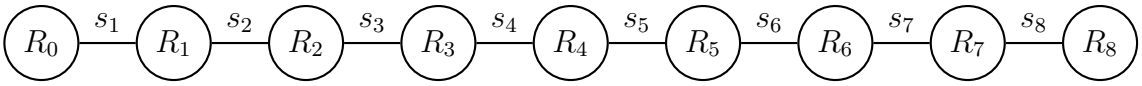


Figure 8.4: The graph of the connected chain query Q in Example 8.3.1.

overall worst regret ratio, the regret ratio for each of the remaining 19 cases is less than 1.0000000000013. More detailed results for the experimental evaluation of our heuristic and the compared heuristics can be found in Table D.1 in Appendix D.1. The following example is from our experimental evaluation, showing the regret of the various heuristics.

Example 8.3.1 Consider the connected chain query $Q = (R, P)$ whose graph is shown in Figure 8.4. The cardinalities of the relations in R are as follows: $r_0 = 10^6$, $r_1 = 10^6$, $r_2 = 10$, $r_3 = 10^2$, $r_4 = 10^5$, $r_5 = 10^3$, $s_6 = 10^4$, $r_7 = 10$ and $r_8 = 10$. The join predicates in P have the following selectivity intervals:

$s_1 = [0.06801, 0.72516]$

$s_2 = [0.18788, 0.89047]$

$s_3 = [0.22310, 0.92875]$

$s_4 = [0.09419, 0.78224]$

$s_5 = [0.00797, 0.68855]$

$s_6 = [0.74758, 0.75892]$

$s_7 = [0.41458, 0.73522]$

$s_8 = [0.44373, 0.88772]$

The optimal minmax regret solution for this case is $R_6R_7R_5R_4R_3R_2R_8R_1R_0$. The following are the plans found by the results of heuristics, with their regret ratios:

- Max-min heuristic: $R_6R_7R_5R_4R_3R_2R_8R_1R_0$ with regret ratio = 1.
- Midpoint heuristic: $R_2R_3R_4R_5R_6R_7R_8R_1R_0$ with regret ratio = 1.04316.
- Pessimistic heuristic: $R_7R_8R_6R_5R_4R_3R_2R_1R_0$ with regret ratio = 1.11189.
- Optimistic heuristic: $R_2R_3R_4R_5R_6R_7R_8R_1R_0$ with regret ratio = 1.04316.

So in this case our heuristic found the optimal solution, while the midpoint and optimistic heuristics found the same plan as each other. ◇

8.4 Discussion of experimental results

It is interesting to note that all the heuristics perform better on the join ordering problem than the selection ordering problem. One possible explanation is that, due to the setting we considered for the join ordering problem, we were able to determine the relative order for at least $\lceil n/2 \rceil$ relations for any precedence graph (where n is the number of relations in the query graph). This in turn reduces the number of possible plans and enhances the quality of the heuristic results.

If we consider the percentage of exact solutions criterion, we notice that all heuristics had good performance. However, in terms of average regret ratio and specially worst regret ratio, the pessimistic and optimistic heuristics performed badly with a worst regret ratio of 6 and 243 respectively. This shows the unreliability of the pessimistic and optimistic heuristics. Even though they found the exact solution in a good number of cases, they performed very poorly when they missed the exact solution. On the other hand, the max-min heuristic found the exact solution in more cases and even when it missed the exact optimal plan, it provided a solution very close to the optimal plan (regret ≈ 1). Such cases highlight the importance of applying MRO in query optimisation, thereby finding a reliable solution that performs well regardless of the encountered scenario, even a worst-case scenario.

The experimental evaluation showed that both the max-min and midpoint heuristics managed to avoid bad plans well, with worst regret ratio less than 1.24. In fact, in most cases they found the exact minmax regret optimal solution (Figure 8.1) and when they missed the optimal solution, on average they found a plan which is very close to the optimal one (Figure 8.3). In practice, the midpoint heuristic seems to be sufficient. However, if finding the exact solution is more important than optimisation time, then the max-min heuristic is recommended. Considering more settings with different query graphs and join trees may reveal more variation in the performance between the max-min and midpoint heuristic as well as the other heuristics.

8.5 Conclusion

This chapter discussed the experimental evaluation of the max-min join heuristic that was presented in Section 7.3. Using a synthetic data set, our heuristic was compared to the midpoint, pessimistic and optimistic heuristics. Our heuristic outperformed all of them on all measuring criteria used, although in some cases the differences were small. Testing our heuristic with other data sets would be more representative and may show a larger discrimination between the performance of the various heuristics. Also, extending the class of join queries considered beyond connected chain queries may reveal different results.

The next chapter concludes the thesis by summarising the thesis and its contributions, followed by identifying some limitations of the thesis and a discussion of future directions.

Chapter 9

Conclusions and Future Work

This chapter concludes the thesis. It first summarises the thesis and its main contributions. Then it identifies some limitations of the thesis. Finally, the chapter discusses some directions for future work.

9.1 Thesis summary and contributions

This thesis considers the unreliability of statistical information that is used by an optimiser during the logical query optimisation process. We used minmax regret optimisation (MRO) as a measure of optimality in this unreliable environment. The main focus of the thesis was on the selection ordering problem and the join ordering problem. In this section we summarise the main contributions of the thesis.

In this thesis we defined and studied the problem of ordering selection operators when the selectivities of the selection predicates are assumed to fall within intervals. A number of useful properties of the problem were identified (e.g. that only extreme scenarios need to be considered). We also found some special cases in which the optimal solution can be found in polynomial time (e.g. dominant sets of operators).

We developed a novel, polynomial-time heuristic for the selection ordering problem. The heuristic was tested using three different data sets: a synthetic data set, the Star Schema Benchmark (SSB) and the Enron email data set. It was also compared with three baseline heuristics, namely the midpoint, pessimistic and optimistic heuristics. Our heuristic performed very well and outperformed all the

baseline heuristics. Considering all the data sets we used, our heuristic overall finds the optimal solution ranging from 73% of all test cases in the synthetic data set to 90% of all cases in the SSB data set. More importantly, it avoids bad plans, producing a worst regret ratio of 1.27 over all data sets. Its average regret ratio varied between 1.01 on the synthetic data set to 1.001 on the Enron data set.

In addition, we applied the selection ordering heuristic to the TFT job scheduling problem, where the processing times of jobs are known to fall in intervals. The problem is NP-hard and a 2-approximation algorithm is already known for the problem in the literature. This approximation uses the midpoint of the processing time intervals instead of considering the entire interval. We evaluated our heuristic experimentally on a synthetic data set and compared it with the 2-approximation algorithm. The results showed that our heuristic performed better than a 2-approximation algorithm. Our heuristic, for example, found the optimal solution in more than 78% of the tested cases, compared to just 52% for the 2-approximation algorithm.

Finally, we studied the join operator ordering problem where the selectivities for join predicates are known to fall in intervals. We investigated the precedence adjacency property which allowed us to identify the relative order of neighbouring relations in precedence graphs. We also developed a heuristic for the join ordering problem, which leveraged the precedence adjacency property to reduce the number plans considered. The heuristic was tested experimentally on a synthetic data set and compared with the midpoint, pessimistic and optimistic heuristics. Our heuristic performed well and found the exact optimal plan in 98% of the tested cases. Its average regret ratio was 1.00023 which shows that the heuristic often finds solutions very close to the optimal solution.

9.2 Thesis limitations

In this section we discuss some limitations of the thesis.

- In Chapter 3, we studied the selection ordering problem and we assumed that the selectivities of all selection predicates are independent of each other. However, in practice there are correlations between the selectivities of some pred-

icates.

- The join ordering problem is known to be NP-hard problem in general. Therefore, when we considered interval selectivities for join predicates in Chapter 7, we restricted our study to connected chain queries (where an optimal join order can be found in polynomial time in the case without intervals). Moreover, we only considered plans that correspond to left-deep join trees.

- Due to hardware limitations and the complexity of the brute-force approach for MRO of both the selection and join ordering problems, in our experimental evaluation we were limited to 11 selection operators for the selection ordering heuristic (Chapter 5) and to 12 relations for the join ordering heuristic (Chapter 8).

- Because of the difficulty of optimisation under imprecise statistical data, this thesis studied the selection ordering problem (Chapter 3) and join ordering problem (Chapter 7) independently of each other, and did not consider any other operators. However, in real life, queries make use of all the relational algebra operators.

- We assumed that the selectivities of selection and join predicates are known to fall within intervals while other parameters, such as the costs of operators or cardinalities of relations, are known precisely. However, statistical information about these parameters can also be unreliable in real life.

9.3 Directions for future work

In this section we discuss some directions for future work.

- As mentioned in Section 9.2, we assumed that the selectivities of the selection and join predicates are independent. In the future, we could relax this assumption and assume that the system stores some joint selectivities. New heuristics would have to be designed to make use of such information.

- In Chapter 8, we evaluated our join ordering heuristic using a synthetic data set. In the near future, we would like to extend our experimental evaluation to use benchmarks such as TPC-H or the one proposed in [76].

- As mentioned in Section 9.2, we considered the join ordering problem only for connected chain queries and produced plans corresponding to left-deep join trees (Chapter 7). As an extension of our work, different join trees, such as bushy trees, could be considered. Also different settings for the join ordering problem could be considered as well, such as allowing Cartesian products or extending the class of join queries.

- Another future direction for our work is to try to find an approximation algorithms with proven bounds for the selection ordering and join ordering problems. Alternatively, we could try to design a new polynomial-time heuristic with lower computational complexity and the same quality of results. For example, it would be interesting to try to design a new heuristic for the selection ordering problem with complexity closer to the midpoint heuristic and result quality as good as, or very close to, our selection ordering heuristic described in Chapter 4.

- As mentioned in Section 9.2, we assumed in Chapters 3 and 7 that the cost of performing the select and join operators are known precisely. Future studies should extend our approach to consider unreliability in the cost estimates of operators and model the costs as intervals, similar to the selectivities for both select and join predicates. Moreover, this could be extended to include unreliability in other parameters such as relation cardinalities.

- In this thesis, we used the minmax regret optimisation approach to measure the optimality of a plan. This measure uses the *absolute* regret value. In the future we could use minmax *relative* regret. This measure divides the cost of a plan by the cost of the scenario's optimal plan.

- This thesis has undertaken initial work towards building a general framework for query optimisation under imprecise statistics. As mentioned in Section 9.2,

we considered queries with only selection operators and queries with only join operators. As a future direction, this work should be extended to include other operators and queries that involve a mix of operators.

Appendix A

Further Investigation of the Selection Ordering Problem

This section presents more details on some further investigations of the selection ordering problem that were briefly discussed in Section 3.6. Some of these investigations did not lead to a good heuristic for the selection ordering problem. However, they improved our understanding and helped us in designing our novel heuristic as presented in Chapter 4.

A.1 Towards an approximation algorithm

Selection operators can have special relationships between them based on their selectivity intervals. Recall from Section 3.1 the definition of dominant operators and nested operators. Identifying the relationship between selection operators can help in finding the optimal order for them or at least finding the relative order for some of the operators in the optimal solution, as discussed in Section 3.6.1. Therefore, it is important to recognise the relationship between operators in the first place, before finding an optimal solution. This section introduces a special way of modelling a set of selection operators in order to identify the relationships between the operators. Finding the relationships between operators is important as discussed in Section 3.5 and is useful for finding an initial plan for the max-min heuristic, as shown in Section 4.2.1.

Definition 3.6.1 states that operators σ_j and σ_k are entangled if they have selectivities such that $\overline{s}_k \leq \overline{s}_j$ and $\underline{s}_k \geq \underline{s}_j$. Such selectivities imply that σ_j and σ_k are either nested or equal operators. On the other hand, if σ_j and σ_k are not entangled, then they have a dominant relationship. In the case of equal operators, the entanglements can be avoided by making sure that equal operators preserve their relative order in both lists, L and U (recall the definitions of the lower list L and the upper list U in Section 3.6.1).

Proposition A.1.1 Let S be a set of selection operators with each operator in subset $S' \subseteq S$ having equal selectivity. There will be no entanglements between any operators in S' if they have the same relative order in both the lower and upper lists.

By maintaining the order suggested in Proposition A.1.1, we can say that if any operators σ_j and σ_k are entangled, then they are nested operators, otherwise one dominates the other.

Proposition A.1.2 Let S be a set of selection operators containing operators σ_j and σ_k. The relationship between σ_j and σ_k can be identified as follows:

- If σ_j and σ_k are entangled, then they have a nested relationship.
- If σ_j and σ_k are not entangled, then they have a dominant relationship.

It is interesting to note that, for any set of selection operator S, if $L(i) = U(i)$ for $1 \leq i \leq n$, then S is a set of dominant of operators. This modelling approach has been implemented in our software. Now consider Example A.1.1 which describes the approach of modelling and detecting relationships between a set of operators with some concrete values.

Example A.1.1 Let $S = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ be a set of selection operators with the selectivity intervals $s_1 = [0.14, 0.58]$, $s_2 = [0.76, 0.81]$, $s_3 = [0.32, 0.90]$ and $s_4 = [0.20, 0.26]$, as shown in Figure A.1.

To model the operators and recognise their relationships, lists L and U should be constructed by sorting the operators in non-decreasing order according to their

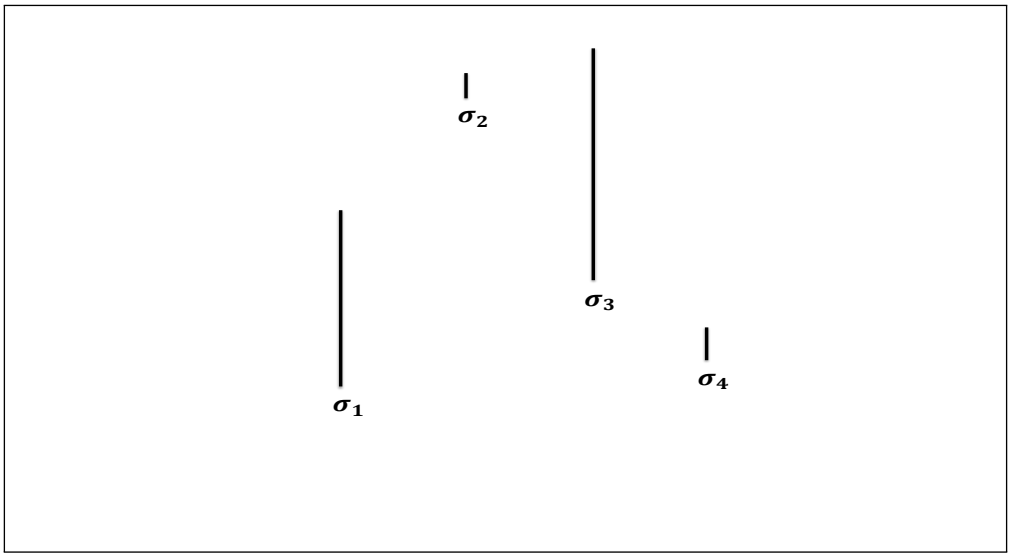


Figure A.1: Selectivity intervals for selection operators in Example A.1.1.

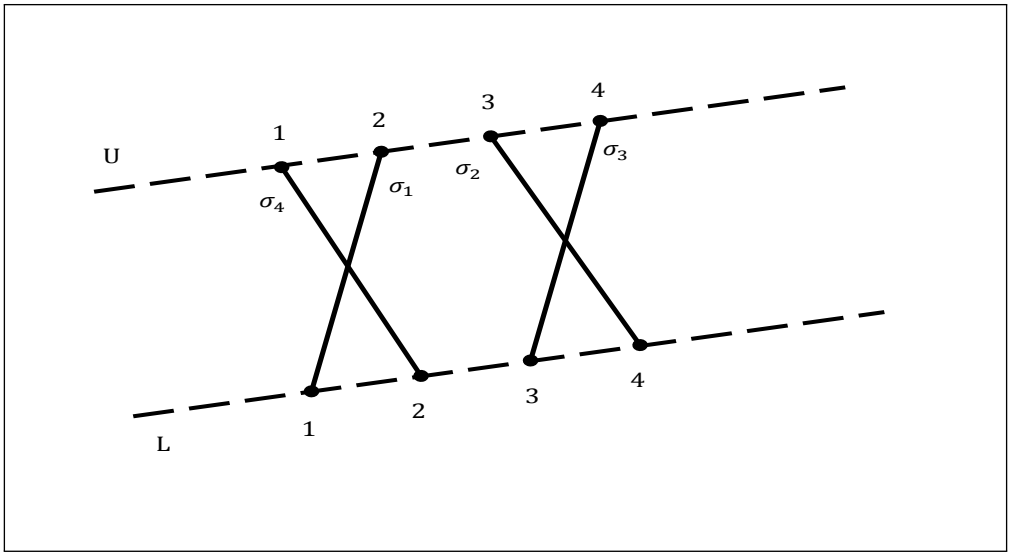


Figure A.2: Entangled operators in Example A.1.1.

minimum and maximum selectivity respectively. As a result, $L = \{\underline{s}_1, \underline{s}_4, \underline{s}_3, \underline{s}_2\}$ and $U = [\overline{s}_4, \overline{s}_1, \overline{s}_2, \overline{s}_3]$. Figure A.2 shows L and U as well as how the operators are related by connecting their positions in L and U. The figure shows that σ_1 and σ_4 are entangled, which indicates that they have a nested relationship, and similarly σ_2 and σ_3 have a nested relationship. On the other hand, any two operators which are not entangled with each other have a dominant relationship. For example, σ_1 is not entangled with σ_2 nor with σ_3, which means that σ_1 has a domination relationship with both σ_2 and σ_3. ◇

Now let us discuss the disentangling algorithm, which was briefly described in Section 3.6.1, in more detail. The following lemmas identify some important properties for the algorithm.

Lemma A.1.1 If entanglement exists, then there is at least one neighbouring pair involved in the Upper and Lower list.

Proof: Recall from Sections 3.6.1 that the operators in lists L and U are sorted in non-decreasing order according to their lower and upper selectivities respectively. An entanglement exists if $L(i) \neq U(i)$, for some $1 \leq i \leq n$. Let v be the first index where $L(v) \neq U(v)$ and let $L(v) = \sigma_j$ and $U(v) = \sigma_k$ (see Figure A.3). Since $L(i) = U(i)$ for $1 \leq i < v - 1$, then the upper bound of σ_j and the lower bound of σ_k must be at an index greater than v, which means $U(w) = \sigma_j$ and $L(y) = \sigma_k$, where $w, y > v$. As a result, σ_j and σ_k are entangled according to the definition.

Consider the selection operator that is located just before σ_j in U, and let this operator be σ_h such that $U(w-1) = \sigma_h$. It could be that $w - 1 = v$; consequently $\sigma_h = \sigma_k$ (see Figure A.3). The lower bound of σ_h must be at index greater than v in L. This is because each operator having index less than v has the property of $L(i) = U(i)$, $1 \leq i < v - 1$. Therefore, σ_h and σ_j are entangled (due to the definition) as well as neighbouring in U.

Analogously, consider the selection operator that is located just before σ_k in L, and let this operator be σ_g, such that $L(y-1) = \sigma_g$. It could be that $y-1 = v$; thus $\sigma_g = \sigma_j$. The upper bound of σ_g must be at an index greater than v in U. This is because each operator having an index less than v has the property of $L(i) = U(i)$,

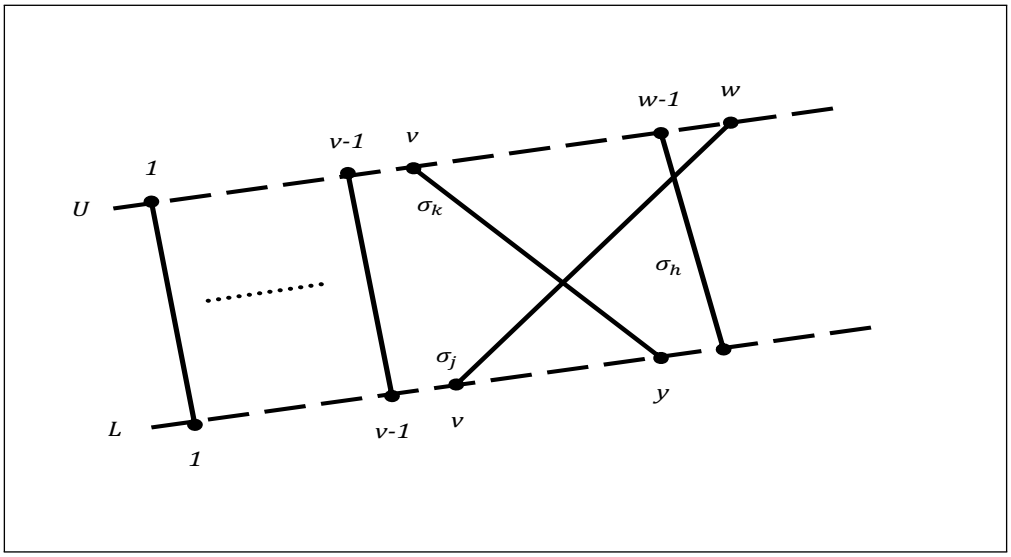


Figure A.3: Neighbouring entanglement in proof of Lemma A.1.1.

$1 \leq i < v-1$. Consequently, σ_g and σ_k are entangled (due to the definition) as well as neighbouring in L.

The above shows that if there is an entanglement, then there is at least one neighbouring pair involved in both the Upper and Lower list. □

As mentioned earlier, entanglement represents a nested relationship between two operators. The main idea in this approach is to modify the minimum or maximum selectivities of the nested operators in order to make them dominant. Once they are dominant, we know their relative order in an optimal solution. This change in the selectivities is represented by swapping the position of operators in either L or U, which is called disentangling (recall Definition 3.6.2). The aim is that any disentanglement does not create a new entanglement. It is true that, after swapping the position of any two entangled operators, lists L and U may not be sorted any more. However, this does not create a problem because we will ensure that any disentanglement does not introduce a new entanglement. The following lemma takes care of that.

Lemma A.1.2 Disentangling a neighbouring pair only removes the entanglement between this neighbouring pair and does not create any new entanglement.

Proof: Let σ_j and σ_k be two operators which are entangled and neighbours with respect to their upper bounds which means that they have consecutive indexes in

list U. Assume $U(w) = \sigma_j, L(x) = \sigma_j$ and $U(w+1) = \sigma_k, L(y) = \sigma_k, y < x$ as shown in Figure A.4.

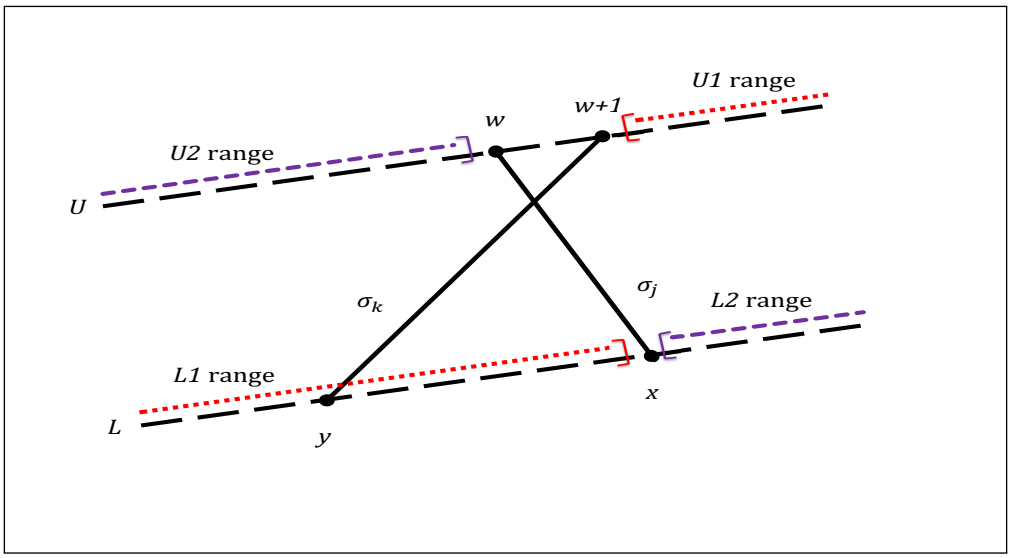


Figure A.4: Entangling range for operator σ_j in the proof of Lemma A.1.2.

First let us consider the effect of the disentanglement on operator σ_j. The change in the index ranges of the possible entangled operators with σ_j other than σ_k should be tested. If the ranges after disentanglement do not increase, then such a disentanglement does not create any new entanglement. However, if index ranges increase then this means that the disentanglement causes new entanglement. Let us consider four ranges as shown in Figure A.4 and described as follows:

- Range $U1 \subset U$ such that $U1(m), w + 1 < m \leq n$.
- Range $U2 \subset U$ such that $U2(m), 1 \leq m < w$.
- Range $L1 \subset L$ such that $L1(o), 1 \leq o < x$.
- Range $L2 \subset L$ such that $L2(o), x < o \leq n$.

Operators σ_l and σ_j are entangled if σ_l is any operator which has an upper index in $U1$ and a lower index in $L1$ or has its upper index in $U2$ and its lower index in $L2$ (see Figure A.4). After disentangling σ_j and σ_k by switching their indices in list U, ranges $U1$, $U2$, $L1$ and $L2$ do not change. As a result, disentangling σ_j and σ_k does not create any new entanglement with operator σ_j.

Analogously, consider the effect of the disentanglement on operator σ_k. Now, let us define the following new ranges as follows:

- Range $U1 \subset U$ such that $U1(m), w+1 < m \leq n$.
- Range $U2 \subset U$ such that $U2(m), 1 \leq m < w$.
- Range $L1 \subset L$ such that $L1(o), 1 \leq o < y$.
- Range $L2 \subset L$ such that $L2(o), y < o \leq n$.

Operators σ_k and σ_f are entangled if σ_f is any operator which has an upper index in $U1$ and a lower index in $L1$ or it has its upper index in $U2$ and its lower index in $L2$. After disentangling σ_j and σ_k by switching their indices in U, ranges $U1$, $U2$, $L1$ and $L2$ have not changed. This means that disentangling σ_j and σ_k does not create any new entanglement with σ_k.

Analogously, the same procedure can be applied for any two operators which are entangled and neighbours in L. As a result, disentangling a neighbouring pair only removes the entanglement between these neighbours and does not cause any new entanglement. □

A.2 More details on average midpoint heuristic

This section provides more details on the average midpoint heuristic which was discussed in Section 3.6.2. The formal algorithm of the heuristic as well as it experimental evaluation are provided in this section.

During our study of the selection ordering problem we noticed that misplacing the operator with the widest selectivity interval has a large negative effect on the solution compared to misplacing operators with smaller interval. Therefore, if a heuristic can find the correct position of this operator or close to the correct position in the minmax regret solution, then this will improve the quality of its solutions. This strategy has been used in the *average midpoint heuristic*. From our study of cases with multiple nested operators and based on experiments done using our software, we have found that:

# of operators	midpoint			average midpoint		
	% exact	Worst	Average	% exact	Worst	Average
2	100.00%	1	1	1	1	1
3	56%	1.70071	1.09758	63%	1.43579	1.03889
4	24%	1.71191	1.13823	44%	1.54615	1.06474
5	6%	1.70666	1.21026	31%	1.43163	1.08642
6	3%	1.82569	1.28381	18%	1.40406	1.10005
7	0%	2.05576	1.35777	13%	1.92507	1.19155
8	0%	2.21590	1.41479	6%	2.04063	1.19945
9	0%	2.40672	1.4420	10%	2.10694	1.21256
10	0%	2.26314	1.53108	6%	2.12993	1.26580
overall	mean	max	mean	mean	max	mean
summary	21%	2.40672	1.27506	35.63%	2.12993	1.11171

Table A.1: Experimental results for the average midpoint heuristic using a synthetic data set of nested operators.

- if the average midpoint selectivity of all operators is equal to or smaller than the midpoint selectivity of the widest interval, then the operator with the widest selectivity interval will be at the middle or further towards the end of the minmax regret solution.

- if the average midpoint selectivity of all operators is greater than the midpoint selectivity of the widest interval, then the operator with the widest selectivity interval will move away from the middle and towards the beginning of the minmax regret solution.

Therefore, the operators with the largest selectivity intervals tend to be somewhat towards the middle of the minmax regret solution for nested operators. This is somehow similar to the class of uniform orders for the job scheduling problem, as discussed in Section 2.9.

Table A.1 shows the result of an experimental evaluation of the average midpoint heuristic when compared to the midpoint heuristic using a synthetic data set of

nested operators. We tested the heuristic with k operators, $k \in [2, 10]$, and for each k we generated 100 different cases. We used three measuring criteria, namely the percentage of cases where the heuristic generates a plan equivalent to the optimal minmax regret solution, the worst regret ratio and the average regret ratio (more details on these measures can be found in Section 5.1). Overall, the average midpoint heuristic performs better than the midpoint heuristic on all measuring criteria when handling a set of nested operators.

Algorithm 8: Average midpoint heuristic

```
1  averageMidpoint(W)
       Input: A list W of n operators sorted in non-increasing order according to their
              selectivity interval widths.
       Output: A solution order sol for the average midpoint heuristic.
       // Assume that the index of all lists starts at 1.
2      Let averageMidpoint be the average midpoint for the selectivities of all n operators;
3      Let lists L and R be the lists holding the sorted plans to the left and the right of the
       operator with the widest selectivity interval respectively;
4      for 2 ≤ i ≤ W.length() do
5          if midpoint(W[i]) < averageMidpoint then
6              addToLeftList(W[i], L);
7          else
8              addToRightList(W[i], R);
9      return sol = L + W[1] + R;

1  addToLeftList(o, L)
       Input: The operator o and the list L into which o should be placed.
       Output: The list L with o placed in the correct position.
       // Assume that the index of all lists starts at 1.
2      boolean isPlaced = false;
3      int i = L.length();
4      while (i > 0 and !isPlaced) do
5          if (!nested(o, L[i])) then
               // i.e. not nested.
6              if L[i]dominates(o) then
7                  shiftRight(L, i + 1, L.length()); // This function takes a sublist
                       of list L from index i + 1 to index L.length() and shifts
                       each operator one position to the right.
8                  L[i + 1] = o;
9                  isPlaced = true;
           // else i.e. nested, so check o with next operator in L.
10         i = i − 1;
11     if (!isPlaced) then
           // i.e. o is nested with operator L[1] or L[1] dominates o, so o
              should be at the beginning of L.
12         L[1] = o;
```

```
1  addToRightList(o, R)
       Input: The operator o and the list R into which o should be placed.
       Output: The list R with o placed in the correct position.
       // Assume that the index of all lists starts at 1.
2      boolean isPlaced = false;
3      int i = 1, listLength = R.length();
4      while (i ≤ listLength And !isPlaced) do
5          if (!nested(o, R[i])) then
               // i.e. not nested.
6              if o.dominates(R[i]) then
7                  shiftRight(R, i, R.length()); // This function takes a sublist of
                       list R from index i to index R.length() and shifts each
                       operator one position to the right.
8                  R[i] = o;
9                  isPlaced = true;
           // else i.e. nested, so check o with next operator in R.
10         i = i + 1;
11     if (!isPlaced) then
           // i.e. o is nested with operator R[R.length()] or R[R.length()]
                dominates o, so o should be at the end of R.
12         R[R.length()+1] = o;
```

Appendix B

Experimental Results for the Selection Ordering Heuristic

B.1 Results for the synthetic data set

B.1.1 Run Time

Figure B.1 shows the run time of the W+ ordering variant (single and multiple phases) together with the baseline algorithm ($\emptyset$,U) when generating plans for up to 200 operators. The run times are for the basic algorithm described in Section 4.1. Unsurprisingly, the variants ($\emptyset$,U) and (D:$CW, W+$) have the fastest run times, as they only execute a single operator insertion phase. Furthermore, it can be clearly seen that the additional run time of $(((D : CW, W+), W+), W+)$ does not pay off, since it produces plans that are only marginally better than those of $((D : CW, W+), W+)$.

B.1.2 Max-min heuristic results (synthetic data)

Table B.1 presents the full results of evaluating the max-min heuristic with various parameters using the synthetic data set as discussed in Section 5.2. The table shows the result of each version of the max-min heuristic for each group k of selection operators, with k ranging from 2 to 10. The last column of the table shows the overall result for the three measuring criteria.

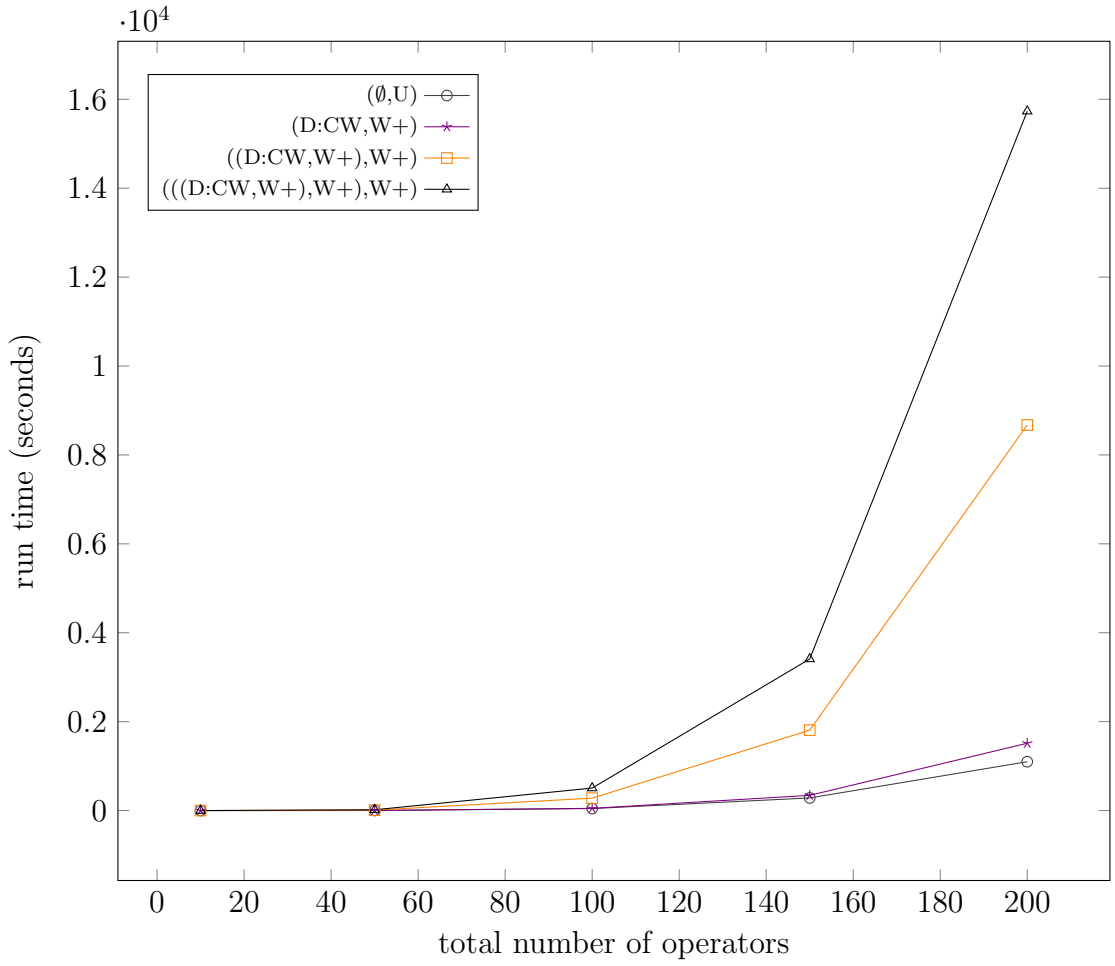


Figure B.1: Run time of the max-min heuristic for selection ordering.

Heuristic	Measure	2	3	4	5	6	7	8	9	10	Max\Average	
(ϕ, U)	% exact	100%	97.70%	83.70%	77%	57.70%	51.50%	37.10%	28.10%	21%	Ave.	61.53%
	Worst	1	1.607 47	1.665 53	1.705 97	1.529 30	1.645 73	1.930 20	1.547 72	1.799 11	Max	1.930 20
	Average	1	1.006 37	1.026 46	1.031 89	1.031 97	1.043 26	1.046 11	1.054 75	1.079 46	Ave.	1.035 59
$(\phi, W+)$	% exact	100%	100%	86%	77%	60%	56%	41%	30%	15%	Ave.	62.78%
	Worst	1	1	1.526 73	1.583 28	1.251 54	1.645 73	1.425 74	1.517 40	1.937 46	Max	1.937 46
	Average	1	1	1.020 57	1.021 31	1.019 70	1.051 94	1.051 64	1.035 40	1.104 80	Ave.	1.033 93
(D:CW, W+)	% exact	100%	100%	87%	81%	66%	64%	40%	34%	18%	Ave.	65.56%
	Worst	1	1	1.237 42	1.562 21	1.662 56	1.395 82	1.425 74	1.287 33	1.937 46	Max	1.937 46
	Average	1	1	1.010 54	1.015 54	1.018 36	1.029 29	1.036 32	1.027 60	1.077 80	Ave.	1.023 94
((D:CW, W+),W+)	% exact	100%	100%	92%	85%	76%	73%	48%	44%	39%	Ave.	73%
	Worst	1	1	1.132 16	1.159 22	1.047 79	1.159 46	1.140 15	1.167 32	1.228 97	Max	1.228 97
	Average	1	1	1.003 11	1.004 51	1.002 62	1.005 27	1.010 08	1.009 89	1.011 43	Ave.	1.005 21
(((D:CW, W+),W+), W+)	% exact	100%	100%	92%	85%	76%	75%	51%	45%	44%	Ave.	74.22%
	Worst	1	1	1.132 16	1.159 22	1.047 79	1.159 46	1.140 15	1.167 32	1.228 97	Max	1.228 97
	Average	1	1	1.003 11	1.004 51	1.002 62	1.004 29	1.009 63	1.009 42	1.010 33	Ave.	1.004 88

Table B.1: Full results for the max-min heuristic (synthetic data).

B.1.3 Results for other heuristics (synthetic data)

As discussed in Section 5.2, the synthetic data set was also used to test the midpoint, pessimistic and optimistic heuristics. The max-min heuristic was used to improve the results of these heuristics when their results are passed to the max-min heuristic as initial plans. Similar to Table B.1, Table B.2 shows the full results of each group $k \in [2, 10]$ of selection operators using each of the three measuring criteria, namely the percentage of exact solutions found, the average regret ratio and the worst regret ratio.

Heuristic	Measure	2	3	4	5	6	7	8	9	10	Max\Average	
Midpoint heuristic	% exact	100%	91%	74%	55%	32%	22%	19%	7%	3%	Ave.	44.78%
	Worst	1	1.607 47	1.711 24	1.855 05	1.529 30	1.789 94	1.520 74	1.727 91	1.530 96	Max	1.855 05
	Average	1	1.016 87	1.066 75	1.089 63	1.108 97	1.143 44	1.101 08	1.124 19	1.145 93	Ave.	1.088 54
((Midpoint, W+),W+)	% exact	100.00%	100.00%	92.00%	84.00%	75.00%	71.00%	47.00%	39.00%	34.00%	Aver.	71.33%
	Worst	1	1	1.109 22	1.159 22	1.147 69	1.159 46	1.140 15	1.150 42	1.163 62	Max	1.163 62
	Average	1	1	1.002 46	1.005 30	1.004 96	1.007 19	1.013 98	1.015 84	1.011 96	Aver.	1.006 85
(((Midpoint, W+),W+), W+)	% exact	100.00%	100.00%	92.00%	84.00%	77.00%	71.00%	50.00%	45.00%	40.00%	Aver.	73.22%
	Worst	1	1	1.109 22	1.159 22	1.147 69	1.159 46	1.140 15	1.116 80	1.163 62	Max	1.163 62
	Average	1	1	1.002 46	1.005 30	1.003 43	1.006 11	1.012 26	1.010 15	1.009 53	Aver.	1.005 47
Pessimistic heuristic	% exact	86%	56%	39%	12%	5%	5%	0.00%	1.00%	0.00%	Ave.	22.67%
	Worst	4.071 83	3.833 92	17.184 19	3.305 68	3.687 90	2.139 31	2.141 15	2.370 38	2.339 03	Max	17.184 19
	Average	1.089 46	1.354 43	1.474 36	1.381 95	1.429 00	1.323 38	1.301 32	1.291 27	1.299 75	Ave.	1.327 21
((Pessim., W+),W+)	% exact	100%	100%	90%	76%	68%	59%	42%	29%	29%	Ave.	65.89%
	Worst	1	1	1.132 16	1.190 35	1.126 69	1.321 81	1.221 73	1.392 33	1.225 37	Max	1.392 33
	Average	1	1	1.004 37	1.011 38	1.007 04	1.021 67	1.018 37	1.031 18	1.026 08	Ave.	1.013 34
(((Pessim., W+),W+), W+)	% exact	100%	100%	90%	81%	76%	70%	53%	44%	44%	Ave.	73.11%
	Worst	1	1	1.132 16	1.159 22	1.045 88	1.321 81	1.221 73	1.392 33	1.116 00	Max	1.392 33
	Average	1	1	1.004 37	1.007 32	1.002 51	1.010 41	1.013 19	1.013 41	1.006 76	Ave.	1.006 44

Heuristic	Measure	2	3	4	5	6	7	8	9	10	Max\Average	
Optimistic heuristic	% exact	79%	65%	34%	26%	9%	6%	9%	2%	1%	Ave.	25.67%
	Worst	9.179 92	8.396 41	5.633 88	9.461 57	9.290 71	24.829 89	11.078 60	129.1523	9.730 07	Max	129.1523
	Average	1.235 58	1.324 49	1.594 44	1.915 85	2.307 79	2.768 38	2.456 75	4.270 93	2.712 31	Ave.	2.287 39
((Optim., W+),W+)	% exact	100%	99%	92%	80%	68%	61%	43%	33%	29%	Ave.	67.22%
	Worst	1	1.266 76	1.237 42	1.190 35	1.227 92	1.321 81	1.140 15	1.454 82	1.327 17	Max	1.454 82
	Average	1	1.002 67	1.004 16	1.009 18	1.013 57	1.021 44	1.018 56	1.024 13	1.020 48	Ave.	1.012 69
(((Optim., W+),W+), W+)	% exact	100%	100%	93%	85%	78%	71%	53%	43%	39%	Ave.	73.56%
	Worst	1	1	1.109 22	1.110 36	1.087 58	1.321 81	1.140 15	1.454 82	1.327 17	Max	1.454 82
	Average	1	1	1.001 79	1.004 07	1.002 64	1.012 88	1.011 56	1.015 36	1.014 27	Ave.	1.006 95

Table B.2: Full results for the midpoint, pessimistic and optimistic heuristics with their refinements using the max-min heuristic (synthetic data).

B.2 Results for the Star Schema Benchmark data set

B.2.1 Max-min heuristic results (SSB)

As discussed in Section 5.3.2, various versions of the max-min heuristic were evaluated experimentally using the SSB data set. Table B.3 presents the full experimental results for the baseline version (ϕ, U), as well as the $(\phi, W+)$, $(D{:}CW, W+)$, $(((D{:}CW, W+), W+)$ and $(((D{:}CW, W+), W+), W+)$ versions. The table shows the results for k selection operators, where $k \in [2, 11]$. The measuring criteria are the percentage of exact solutions, the worst regret ratio and the average regret ratio criteria. In this data set, the third iteration shows no improvement at all; both $(((D{:}CW, W+), W+)$ and $(((D{:}CW, W+), W+), W+)$ versions of the max-min heuristic have the same results for each criterion.

Heuristic	Measure	2	3	4	5	6	7	8	9	10	11	Max\Average	
(ϕ, U)	% exact	100%	99.90%	98.60%	96.50%	96.60%	89.70%	83%	77.20%	74.10%	65.70%	Ave.	88.13%
	Worst	1	1.149 82	1.343 33	1.156 86	1.297 13	1.521 30	1.693 74	1.367 98	1.485 20	1.626 38	Max	1.693 74
	Average	1	1.000 15	1.001 44	1.001 63	1.002 31	1.010 76	1.013 39	1.014 58	1.012 52	1.016 83	Ave.	1.007 36
$(\phi, W+)$	% exact	100%	100%	99%	96%	96%	90%	87%	79%	76%	70%	Ave.	89.30%
	Worst	1	1	1.074 84	1.053 59	1.046 33	1.219 47	1.295 00	1.367 98	1.234 94	1.626 38	Max	1.626 38
	Average	1	1	1.000 75	1.000 64	1.000 76	1.008 47	1.007 96	1.017 07	1.008 91	1.027 94	Ave.	1.007 25
(D:CW, W+)	% exact	100%	100%	100%	97%	98%	92%	87%	82%	75%	67%	Ave.	89.80%
	Worst	1	1	1	1.009 64	1.003 39	1.219 47	1.269 04	1.205 66	1.115 58	1.519 63	Max	1.519 63
	Average	1	1	1	1.000 11	1.000 03	1.007 35	1.009 70	1.008 01	1.005 65	1.014 08	Ave.	1.004 49
((D:CW, W+),W+)	% exact	100%	100%	100%	97%	98%	90%	86%	78%	76%	72%	Ave.	89.70%
	Worst	1	1	1	1.009 64	1.003 39	1.219 47	1.269 04	1.242 15	1.088 93	1.110 25	Max	1.269 04
	Average	1	1	1	1.000 11	1.000 03	1.007 28	1.007 50	1.014 54	1.004 79	1.005 56	Ave.	1.003 98
(((D:CW, W+),W+), W+)	% exact	100%	100%	100%	97%	98%	90%	86%	78%	76%	72%	Ave.	89.70%
	Worst	1	1	1	1.009 64	1.003 39	1.219 47	1.269 04	1.242 15	1.088 93	1.110 25	Max	1.269 04
	Average	1	1	1	1.000 11	1.000 03	1.007 28	1.007 50	1.014 54	1.004 79	1.005 56	Ave.	1.003 98

Table B.3: Full results for the max-min heuristic (SSB data).

B.2.2 Results for other heuristics (SSB)

The SSB data set was also used to test the performance of the midpoint, pessimistic and optimistic approaches. Table B.4 shows the full experimental evaluation for these approaches. The max-min heuristic was fed with the results of these approaches as initial plans to measure the effectiveness of the max-min heuristic in improving existing results. The results in Table B.4 show how max-min heuristic improved the results of the midpoint, pessimistic and optimistic approaches. For example, the max-min heuristic improved the overall worst regret ratio from 1.69 to 1.27 for the midpoint heuristic, from 3205 to 1.42 for the pessimistic heuristic, and from 89183 to 1.37 for the optimistic heuristic.

Heuristic	Measure	2	3	4	5	6	7	8	9	10	11	Max\Average	
Midpoint heuristic	% exact	100%	99%	96%	94%	91%	87%	80%	75%	66%	54%	Ave.	84.20%
	Worst	1	1.149 82	1.343 33	1.164 76	1.297 13	1.413 51	1.565 24	1.353 10	1.367 00	1.694 45	Max	1.694 45
	Average	1	1.001 50	1.006 24	1.004 17	1.007 65	1.013 74	1.022 82	1.019 16	1.029 73	1.039 92	Ave.	1.014 49
((Midpoint, W+),W+)	% exact	100%	100%	100%	97%	98%	90%	86%	78%	75%	71%	Ave.	89.50%
	Worst	1	1	1	1.009 64	1.003 39	1.219 47	1.269 04	1.242 15	1.088 93	1.110 25	Max	1.269 04
	Average	1	1	1	1.000 11	1.000 03	1.007 28	1.007 50	1.013 82	1.005 49	1.005 60	Ave.	1.003 98
(((Midpoint, W+),W+), W+)	% exact	100%	100%	100%	97%	98%	90%	86%	78%	75%	72%	Ave.	89.60%
	Worst	1	1	1	1.009 64	1.003 39	1.219 47	1.269 04	1.242 15	1.088 93	1.110 25	Max	1.269 04
	Average	1	1	1	1.000 11	1.000 03	1.007 28	1.007 50	1.014 54	1.005 49	1.005 56	Ave.	1.004 05
Pessimistic heuristic	% exact	96%	86%	81%	67%	64%	51%	41%	38%	25%	28%	Ave.	57.70%
	Worst	4.250 70	177.344	3204.974	2605.269	1391.897	3.370 32	8.644 87	22.388 80	4.330 01	5.319 01	Max	3204.974
	Average	1.055 74	2.874 81	36.581 84	53.916 95	19.929 84	1.232 37	1.390 47	1.466 76	1.170 09	1.180 67	Ave.	12.079 96
((Pessim., W+),W+)	% exact	100%	100%	100%	97%	98%	87%	84%	76%	76%	72%	Ave.	89%
	Worst	1	1	1	1.009 64	1.003 39	1.419 29	1.269 04	1.242 15	1.088 93	1.138 11	Max	1.419 29
	Average	1	1	1	1.000 11	1.000 03	1.013 27	1.007 52	1.015 41	1.004 90	1.006 16	Ave.	1.004 74
(((Pessim., W+),W+), W+)	% exact	100%	100%	100%	97%	98%	89%	86%	78%	76%	73%	Ave.	89.70%
	Worst	1	1	1	1.009 64	1.003 39	1.419 29	1.269 04	1.242 15	1.088 93	1.138 11	Max	1.419 29
	Average	1	1	1	1.000 11	1.000 03	1.011 47	1.007 50	1.014 54	1.004 79	1.005 71	Ave.	1.004 42

Heuristic	Measure	2	3	4	5	6	7	8	9	10	11	Max\Average	
Optimistic heuristic	% exact	94%	92%	77%	70%	60%	52%	44%	35%	25%	16%	Ave.	56.50%
	Worst	2.388 71	89 182.7	20 833.33	3204.974	3205.012	74 670	20 703.87	58 232.86	22 762.16	46 874.91	Max	89 182.65
	Average	1.046 11	1023.64	252.7106	89.562 38	52.899 96	1387.124	410.5541	903.3359	464.4910	967.9778	Ave.	555.3341
((Optim., W+),W+)	% exact	100%	100%	100%	96%	98%	90%	86%	78%	74%	72%	Ave.	89.40%
	Worst	1	1	1	1.374 39	1.003 39	1.219 47	1.269 04	1.242 15	1.088 93	1.110 25	Max	1.374 39
	Average	1	1	1	1.003 85	1.000 03	1.007 28	1.007 50	1.014 54	1.005 57	1.005 45	Ave.	1.004 42
(((Optim., W+),W+), W+)	% exact	100%	100%	100%	96%	98%	90%	86%	78%	74%	73%	Ave.	89.50%
	Worst	1	1	1	1.374 39	1.003 39	1.219 47	1.269 04	1.242 15	1.088 93	1.110 25	Max	1.374 39
	Average	1	1	1	1.003 85	1.000 03	1.007 28	1.007 50	1.014 54	1.005 57	1.005 40	Ave.	1.004 42

Table B.4: Full results for the midpoint, pessimistic and optimistic heuristics with their refinements using the max-min heuristic (SSB data).

B.3 Enron sample data and experimental results

B.3.1 Keyword selectivity ranges

Table B.5 lists the keywords that were chosen from the `subject` attribute. The minimum and maximum selectivities for these keywords in this table were calculated as discussed in Section 5.4.

#	Keyword	Selectivity(min)	Selectivity(max)
1	word	7.94E-04	0.019868405
2	work	0.002061557	0.019868405
3	progress	4.42E-04	0.014376134
4	price	0.005194968	0.10603915
5	schedule	0.00937676	0.039509921
6	request	0.01401233	0.023213056
7	action	0.002613131	0.085145835
8	staff	0.00309038	0.019234681
9	meeting	0.029065219	0.126588221
10	enron	0.031995212	0.049739469
11	reminder	0.00265225	0.060145052
12	interview	0.002124147	0.025442824
13	day	0.004619922	0.066795237
14	notification	0.002190649	0.029421199
15	notice	0.004052701	0.046300933
16	not	0.002597484	0.046300933
17	for	0.080122989	0.129185704
18	start	0.01451305	0.099533705
19	email	0.002765694	0.055290417
20	market	0.007319115	0.031154159
21	update	0.016140389	0.024163641
22	interest	8.10E-04	0.18942073
23	view	6.88E-04	0.043930337

#	Keyword	Selectivity(min)	Selectivity(max)
24	data	0.005777837	0.114770451
25	use	7.94E-04	0.05872113
26	you	0.011203605	0.034999531
27	best	0.001040558	0.049598642
28	all	0.002386243	0.076793985
29	some	7.71E-04	0.044622739
30	are	0.003919697	0.160015178
31	new	0.021922138	0.055192621
32	the	0.036583839	0.110733398
33	real	0.002010703	0.110741222
34	time	0.004295237	0.033677317
35	date	0.001596044	0.130222351
36	week	0.00413485	0.018006353
37	hour	0.001283094	0.06824263
38	out	0.00561745	0.049672967
39	market	0.007319115	0.031154159
40	but	5.40E-04	0.020009232

Table B.5: Keyword list for the `subject` attribute (Enron).

The keywords chosen for the `body` attribute, along with their minimum and maximum selectivities, are presented in Table B.6.

#	Keyword	Selectivity(min)	Selectivity(max)
1	progress	0.010405583	2.42E-01
2	price	0.065907242	6.28E-01
3	schedule	0.045479439	0.393937379
4	request	0.047959567	0.216166208
5	action	0.025783157	0.68534847
6	staff	0.024507886	0.288281592
7	meet	0.045854979	0.723821744

#	Keyword	Selectivity(min)	Selectivity(max)
8	remind	0.003798429	0.589507573
9	interview	0.009173343	0.251846404
10	day	0.077001314	0.676421575
11	notification	0.005789573	0.582012424
12	notice	0.025888778	6.19E-01
13	regard	0.007197847	4.24E-01
14	attache	6.26E-05	0.610713839
15	from	0.351348814	6.14E-01
16	sent	0.079598798	0.820425455
17	subject	0.043930337	0.478735057
18	thank	0.051844057	0.56120517
19	group	0.069005445	0.438978688
20	please	0.350953715	0.655309194
21	there	0.156118952	0.892364806
22	the	0.78957251	0.902653032
23	not	0.341264005	0.618799681
24	for	0.624413219	0.778736621
25	start	0.046711679	0.620278369
26	email	0.101790856	0.614860424
27	market	0.073938318	0.42336249
28	update	0.033336984	0.13001111
29	interest	3.61E-02	0.820425455
30	view	0.031987388	0.422310196
31	data	0.043034518	0.722550385
32	use	0.111574451	0.659835232
33	you	0.616006603	0.724302904
34	like	0.169286318	0.272059054
35	best	0.056651749	0.738354353
36	all	0.250410747	0.796965951

#	Keyword	Selectivity(min)	Selectivity(max)
37	some	0.150133786	0.651639857
38	are	0.43255148	0.84691275
39	real	0.030555642	0.788222914
40	time	0.148729424	0.524629154
41	date	0.047826563	0.722550385
42	hour	0.017075327	0.713384866
43	out	0.180971083	0.605632315
44	market	0.073938318	0.42336249
45	but	0.211882706	0.427759435

Table B.6: Keyword list for the `body` attribute (Enron).

B.3.2 Max-min heuristic results (Enron)

Section 5.4.2 discussed the experimental evaluation of the max-min heuristic using the Enron data set. Various versions of the max-min heuristic were tested to show the effectiveness of ordering, starting with initial plans and having multiple iterations. Table B.7 shows the full results of our experiments using k selection operators, where k ranges from 2 to 11.

Heuristic	Measure	2	3	4	5	6	7	8	9	10	11	Max\Average	
(ϕ, U)	% exact	100%	100%	94.50%	68.50%	70%	63.50%	44.50%	46.50%	31.50%	11.50%	Ave.	63.05%
	Worst	1	1	1.493 92	1.319 09	1.323 11	1.421 17	1.383 24	1.332 29	1.463 61	1.539 85	Max	1.539 85
	Average	1	1	1.007 12	1.011 86	1.013 92	1.027 60	1.023 95	1.020 80	1.035 01	1.048 58	Ave.	1.018 88
$(\phi, W+)$	% exact	100%	100%	90%	80%	70%	60%	45%	50%	40%	10%	Ave.	64.50%
	Worst	1	1	1.071 83	1.215 32	1.193 54	1.421 17	1.522 37	1.191 32	1.288 93	1.246 93	Max	1.522 37
	Average	1	1	1.005 71	1.012 65	1.012 24	1.053 00	1.042 93	1.014 52	1.040 17	1.047 25	Ave.	1.022 85
(D:CW, W+)	% exact	100%	100%	100%	75%	65%	75%	55%	55%	40%	15%	Ave.	68%
	Worst	1	1	1	1.215 32	1.193 54	1.166 19	1.155 90	1.191 32	1.252 29	1.357 16	Max	1.357 16
	Average	1	1	1	1.013 60	1.012 66	1.017 12	1.012 61	1.013 44	1.036 52	1.063 80	Ave.	1.016 97
((D:CW, W+),W+)	% exact	100%	100%	100%	90%	80%	95%	60%	60%	70%	25%	Ave.	78%
	Worst	1	1	1	1.029 92	1.004 36	1.000 10	1.050 81	1.010 06	1.038 55	1.072 06	Max	1.072 06
	Average	1	1	1	1.001 62	1.000 72	1.000 00	1.003 00	1.000 88	1.002 42	1.005 63	Ave.	1.001 43
(((D:CW, W+),W+), W+)	% exact	100%	100%	100%	90%	80%	95%	65%	60%	70%	25%	Ave.	78.50%
	Worst	1	1	1	1.029 92	1.004 36	1.000 10	1.050 81	1.010 06	1.038 55	1.072 06	Max	1.072 06
	Average	1	1	1	1.001 62	1.000 72	1.000 00	1.002 93	1.000 88	1.002 42	1.005 63	Ave.	1.001 42

Table B.7: Full results for the max-min heuristic (Enron data).

B.3.3 Results for other heuristics (Enron)

The performance of the midpoint, pessimistic and optimistic heuristics were tested using the Enron data set as discussed in Section 5.4.2. The outputs of these heuristics were used as initial plans for the max-min heuristic. This was done to study the effectiveness of the max-min heuristic in improving the quality of bad results. The experiments show the power of the max-min heuristic, as presented in Table B.8. For example, the max-min heuristic improved the overall percentage of optimal plans found from 40.5% to 77.5% for the midpoint heuristic, from 31.5% to 79.5% for the pessimistic heuristic, and from 31.5% to 77.5% for the optimistic heuristic.

Heuristic	Measure	2	3	4	5	6	7	8	9	10	11	Max\Average	
Midpoint heuristic	% exact	100%	90%	80%	45%	45%	20%	10%	20%	0.00%	5%	Ave.	41.50%
	Worst	1	1.439 06	1.493 92	1.366 91	1.172 12	1.266 06	1.383 24	1.315 76	1.318 44	1.322 53	Max	1.493 92
	Average	1	1.037 69	1.054 96	1.064 37	1.043 54	1.058 71	1.097 50	1.087 24	1.099 22	1.071 61	Ave.	1.061 48
((Midpoint, W+), W+)	% exact	100%	100%	100%	90%	75%	100%	65%	60%	45%	30%	Ave.	76.50%
	Worst	1	1	1	1.029 92	1.047 76	1	1.052 92	1.014 90	1.083 64	1.181 59	Max	1.181 59
	Average	1	1	1	1.001 62	1.003 04	1	1.005 69	1.001 59	1.010 23	1.015 29	Ave.	1.003 75
(((Midpoint, W+), W+), W+)	% exact	100%	100%	100%	90%	80%	95%	60%	60%	65%	25%	Ave.	77.50%
	Worst	1	1	1	1.029 92	1.004 36	1.000 10	1.052 92	1.010 06	1.083 64	1.072 06	Max	1.083 64
	Average	1	1	1	1.001 62	1.000 72	1.000 00	1.003 80	1.000 84	1.005 84	1.004 74	Ave.	1.001 76
Pessimistic heuristic	% exact	100%	85%	75%	20%	15%	15%	5%	0.00%	0.00%	0.00%	Ave.	31.50%
	Worst	1	1.823 54	1.493 92	1.737 43	1.475 91	1.558 74	1.471 05	1.332 29	1.463 61	1.537 44	Max	1.823 54
	Average	1	1.078 86	1.065 36	1.152 85	1.164 97	1.177 06	1.148 02	1.154 87	1.184 47	1.152 38	Ave.	1.127 88
((Pessim., W+), W+)	% exact	100%	100%	100%	90%	75%	100%	70%	65%	40%	20%	Ave.	76%
	Worst	1	1	1	1.029 92	1.047 76	1	1.058 12	1.018 91	1.270 27	1.072 06	Max	1.270 27
	Average	1	1	1	1.001 62	1.003 04	1	1.005 29	1.001 61	1.025 69	1.011 00	Ave.	1.004 83
(((Pessim., W+), W+), W+)	% exact	100%	100%	100%	90%	80%	95%	70%	60%	75%	25%	Ave.	79.50%
	Worst	1	1	1	1.029 92	1.004 36	1.000 10	1.003 86	1.010 06	1.007 18	1.072 06	Max	1.072 06
	Average	1	1	1	1.001 62	1.000 72	1.000 00	1.000 49	1.000 84	1.000 50	1.004 17	Ave.	1.000 83

Heuristic	Measure	2	3	4	5	6	7	8	9	10	11	Max\Average	
Optimistic heuristic	% exact	90%	85%	25%	40%	15%	20%	5%	15%	15%	5%	Ave.	31.50%
	Worst	5.648 56	1.583 99	16.527 66	29.766 26	15.653 25	5.607 60	24.513 53	19.638 76	31.342 79	17.781 10	Max	31.342 79
	Average	1.358 45	1.049 64	2.286 23	4.754 08	2.883 56	1.688 87	5.626 53	3.416 14	8.828 84	4.061 37	Ave.	3.595 37
((Optim., W+),W+)	% exact	100%	100%	100%	85%	80%	95%	60%	55%	65%	25%	Ave.	76.50%
	Worst	1	1	1	1.029 92	1.004 36	1.000 10	1.050 81	1.034 56	1.038 55	1.072 06	Max	1.072 06
	Average	1	1	1	1.002 01	1.000 72	1.000 00	1.005 91	1.003 76	1.003 66	1.004 17	Ave.	1.002 02
(((Optim., W+),W+), W+)	% exact	100%	100%	100%	85%	80%	95%	65%	60%	65%	25%	Ave.	77.50%
	Worst	1	1	1	1.029 92	1.004 36	1.000 10	1.050 81	1.010 06	1.038 55	1.072 06	Max	1.072 06
	Average	1	1	1	1.002 01	1.000 72	1.000 00	1.003 97	1.000 88	1.002 93	1.004 17	Ave.	1.001 47

Table B.8: Full results for the midpoint, pessimistic and optimistic heuristics with their refinements using the max-min heuristic (Enron data).

B.4 Statistical measures

Two different measures were used to study the stability of our experimental evaluations. We used the variance to measure the stability of the overall percentage of exact solutions found by the heuristics and the overall average regret ratio. The smaller the value of the variance is the better, since this indicates that the results are condensed around the values of the two evaluation measures (i.e. the percentage of exact solutions and the average regret ratio). Moreover, we wanted to evaluate the margin of error in the two evaluation measures, so we used the confidence interval measure with 95% confidence level. If the margin of error is small then the result is highly stable.

Heuristic	Measure	Synthetic		SSB		Enron	
		variance	marg. err.	variance	marg. err.	variance	marg. err.
(ϕ, U)	% exact	0.077 68	± 1.82%	0.013 79	± 0.73%	0.080 69	± 1.76%
	Average	0.000 52	± 0.001 49	4.16×10^{-5}	± 0.000 40	0.000 22	± 0.000 91
$(\phi, W+)$	% exact	0.082 20	± 1.88%	0.010 74	± 0.64%	0.076 23%	± 1.71%
	Average	0.000 95	± 0.002 01	7.62×10^{-5}	± 0.000 54	0.000 38	± 0.001 21
(D:CW,W+)	% exact	0.077 60	± 1.82%	0.012 44	0.69%	0.071 10	± 1.65%
	Average	0.000 50	± 0.001 47	2.41×10^{-5}	± 0.000 30	0.000 35	± 0.001 16
((D:CW,W+), W+)	% exact	0.051 04	± 1.48%	0.010 92	± 0.65%	0.054 10	± 1.44%
	Average	1.67×10^{-5}	± 0.000 27	2.16×10^{-5}	± 0.000 29	3.01×10^{-6}	± 0.000 11
(((D:CW,W+), W+),W+)	% exact	0.045 24	± 1.39%	0.010 92	± 0.65%	0.052 53	± 1.42%
	Average	1.44×10^{-5}	± 0.000 25	2.16×10^{-5}	± 0.000 29	2.99×10^{-6}	± 0.000 11

Table B.9: Variance and margin of error for % of exact solutions and average regret ratio of the max-min heuristic experiments.

Table B.9 presents the variance and margin of error for the percentage of exact solutions and the average regret ratio evaluation measures when testing various versions of the max-min heuristics. All the variances were less than 0.1. The margin of error was less than ±2% and ±0.01 for the percentage of exact solutions and the average regret ratio respectively. Therefore, we have high stability results

Heuristic	Measure	Synthetic		SSB		Enron	
		variance	marg. err.	variance	marg. err.	variance	marg. err.
Midpoint heuristic	% exact	0.118 48	± 2.25%	0.021 04	± 0.90%	0.122 53	± 2.17%
	Average	0.002 40	± 0.003 21	0.000 16	± 0.000 78	0.000 82	± 0.001 78
((Mid. W+), W+)	% exact	0.058 62	± 1.58%	0.011 57	± 0.68%	0.059 53	± 1.51%
	Average	3.08×10^{-5}	± 0.000 36	2.03×10^{-5}	± 0.000 28	2.45×10^{-5}	± 0.000 31
(((Mid. W+), W+),W+)	% exact	0.048 46	± 1.44%	0.011 20	± 0.66%	0.055 13	± 1.46%
	Average	1.76×10^{-5}	± 0.000 27	2.18×10^{-5}	± 0.000 29	4.41×10^{-6}	± 0.000 13
Pessimistic heuristic	% exact	0.084 71	± 1.90%	0.056 00	± 1.47%	0.138 03	± 2.31%
	Average	0.010 56	± 0.006 72	320.1	± 1.110 23	0.003 20	± 0.003 51
((Pessim. W+), W+)	% exact	0.070 61	± 1.74%	0.011 64	± 0.67%	0.070 90	± 1.65%
	Average	0.000 12	± 0.000 71	3.07×10^{-5}	± 0.000 34	5.91×10^{-5}	±0.000 48
(((Pessim. W+) ,W+),W+)	% exact	0.043 45	± 0.14%	0.010 58	± 0.64%	0.050 73	± 1.40%
	Average	2.37×10^{-5}	± 0.000 32	2.60×10^{-5}	± 0.000 32	1.47×10^{-6}	$\pm\ 7.54\times10^{-5}$
Optimistic heuristic	% exact	0.073 02	± 1.78%	0.065 53	± 1.59%	0.087 53	± 1.84%
	Average	0.780 79	± 0.057 81	209997.7	± 28.4369	5.044 65	± 0.139 38
((Optim. W+), W+)	% exact	0.068 00	± 1.71%	0.011 36	± 0.66%	0.056 03	± 1.47%
	Average	7.22×10^{-5}	± 0.000 56	2.01×10^{-5}	± 0.000 28	4.35×10^{-6}	± 0.000 13
(((Optim. W+) ,W+),W+)	% exact	0.049 82	± 1.46%	0.011 03	± 0.65%	0.052 63	± 1.42%
	Average	3.68×10^{-5}	± 0.000 40	2.01×10^{-5}	± 0.000 28	2.5×10^{-6}	$\pm\ 9.90\times10^{-5}$

Table B.10: Variance and margin of error for % of exact solutions and average regret ratio for the midpoint, pessimistic and optimistic heuristics experiments with their refinements using the max-min heuristic.

The statistical measures also show stability in the experimental evaluation of the midpoint heuristic and its refinements using the max-min heuristic. Table B.10 shows the variance and margin of error for the percentage of exact solutions and the average regret ratio in these experiments. The pessimistic and optimistic heuristics had a large variant in the SSB data set. This is because they performed extremely bad and their average regret ratio was fluctuating while the variance is sensitive for outliers. The margin of error in this case for the optimistic heuristic is large as well since it is the worst heuristic in our experiments but the margin of error for the pessimistic heuristic is small with a value ∓ 1.11. Refining the results of

pessimistic and optimistic heuristics using the max-min heuristic improved the result significantly. From Table B.10 we can see that in these experiments the margin of error is less than $\pm 2\%$ and ± 0.001 for both the percentage of exact solutions and the average regret ratio measures respectively while the variance for both of them is less than 0.1.

Appendix C

Experimental Results for Total Flow Time

C.1 Max-min heuristic additional results (TFT)

As discussed in Section 6.2, different variations of the max-min heuristic for the total flow time (TFT) problem were evaluated experimentally using a synthetic data set. Table C.1 presents the full experimental results for the versions (ϕ, U), $(\phi, W+)$, $(D{:}CW, W+)$, $(((D{:}CW, W+), W+)$ and $(((D{:}CW, W+), W+), W+)$. The table shows the result under each group k of jobs, where $k \in [2, 10]$. It also shows the overall results using the percentage of exact solutions, worst regret ratio and average regret ratio criteria. Table C.2 on the other hand, shows the full result for the other heuristics, namely the 2-approximation (midpoint), pessimistic and optimistic heuristics, using the same data set.

Heuristic	Measure	2	3	4	5	6	7	8	9	10	Max\Average	
(ϕ, U)	% exact	100%	96%	85.60%	81.10%	71.80%	57.90%	45.80%	41.60%	30.50%	Ave.	67.81%
	Worst	1	1.402 61	1.567 30	1.465 52	1.618 24	1.385 51	1.649 86	1.457 65	1.473 84	Max	1.649 86
	Average	1	1.003 99	1.020 13	1.018 06	1.028 45	1.037 04	1.043 12	1.039 99	1.054 26	Ave.	1.027 23
$(\phi, W+)$	% exact	100%	97%	88%	86%	78%	62%	45%	36%	39%	Ave.	70.11%
	Worst	1	1.188 21	1.436 23	1.225 03	1.229 67	1.251 71	1.196 50	1.315 58	1.167 70	Max	1.436 23
	Average	1	1.003 63	1.018 10	1.010 01	1.015 48	1.022 68	1.024 65	1.030 29	1.025 26	Ave.	1.016 68
(D:CW, W+)	% exact	100%	99%	92%	87%	81%	66%	51%	48%	44%	Ave.	74.22%
	Worst	1	1.107 90	1.412 51	1.302 18	1.445 87	1.214 44	1.262 27	1.176 00	1.294 89	Max	1.445 87
	Average	1	1.001 08	1.011 92	1.012 59	1.018 83	1.028 82	1.029 41	1.022 73	1.026 16	Ave.	1.016 84
((D:CW, W+),W+)	% exact	100%	100%	96%	84%	83%	70%	64%	61%	48%	Ave.	78.44%
	Worst	1	1	1.166 92	1.225 03	1.445 87	1.257 46	1.185 50	1.201 79	1.274 45	Max	1.445 87
	Average	1	1	1.004 03	1.012 93	1.014 39	1.025 25	1.021 59	1.024 03	1.035 50	Ave.	1.015 30
(((D:CW, W+),W+), W+)	% exact	100%	100%	96%	84%	83%	70%	64%	61%	47%	Ave.	78.33%
	Worst	1	1	1.166 92	1.225 03	1.445 87	1.257 46	1.185 50	1.201 79	1.274 45	Max	1.445 87
	Average	1	1	1.004 03	1.012 93	1.014 39	1.026 91	1.021 59	1.024 26	1.035 87	Ave.	1.015 55

Table C.1: Full results for the max-min heuristic (TFT).

Heuristic	Measure	2	3	4	5	6	7	8	9	10	Max\Average	
Midpoint heuristic	% exact	100%	89%	72%	69%	56%	31%	17%	19%	15%	Ave.	52%
	Worst	1	1.490 86	1.609 14	1.681 22	1.381 62	1.293 66	1.366 52	1.278 54	1.445 60	Max	1.681 22
	Average	1	1.017 33	1.059 32	1.041 87	1.042 57	1.052 67	1.062 66	1.062 75	1.072 03	Ave.	1.045 69
Pessimistic heuristic	% exact	81%	65%	33%	18%	8%	6%	1%	0.00%	0.00%	Ave.	23.56%
	Worst	7.333 27	8.497 31	7.539 01	4.245 55	3.035 50	2.703 93	2.962 79	2.827 15	2.634 02	Max	8.497 31
	Average	1.273 88	1.388 44	1.579 33	1.479 51	1.546 50	1.450 61	1.528 41	1.625 19	1.614 86	Ave.	1.498 53
Optimistic heuristic	% exact	84%	65%	27%	19%	8%	5%	2%	0.00%	1%	Ave.	23.44%
	Worst	11.943 89	3.781 74	7.401 58	5.238 60	2.833 16	3.202 61	2.826 46	3.528 03	2.454 98	Max	11.943 89
	Average	1.233 65	1.219 60	1.593 55	1.518 09	1.546 00	1.460 65	1.592 75	1.638 95	1.609 78	Ave.	1.490 33

Table C.2: Full results for the midpoint, pessimistic and optimistic heuristics (TFT).

C.2 Statistical measures

Heuristic	Measure	Variance	Margin of error
(ϕ, U)	% exact	0.055 57	± 1.54%
	Average	0.000 30	± 0.001 12
$(\phi, W+)$	% exact	0.056 43	± 1.55%
	Average	9.50×10^{-5}	± 0.000 64
(D:CW,W+)	% exact	0.044 57	± 1.38%
	Average	0.000 11	± 0.000 69
((D:CW,W+), W+)	% exact	0.031 11	± 1.15%
	Average	0.000 14	± 0.000 76
(((D:CW,W+),W+),W+)	% exact	0.031 80	± 1.17%
	Average	0.000 14	± 0.000 78

Table C.3: Variance and margin of error for % of exact solutions and average regret ratio of the max-min heuristic experiments (TFT).

For the results in Tables C.1 and C.2 we calculated the variance and the confidence intervals with 95% confident level to measure the margin of error and the stability of the result. This statistical study were performed on the percentage of exact solution and the average regret ratio measuring criteria. Table C.3 presents the statistics for the max-min heuristic with different variations. The margin of error for the percentage of exact solutions and the average regret ratio measures was less than ±2% and ±0.01 respectively while their variance was less than 0.02. This indicates a high stability in the experimental results of the max-min heuristic. A similar stability for the experiments of the midpoint, pessimistic and optimistic heuristics is shown in Table C.4.

Heuristic	Measure	Variance	Margin of error
Midpoint heuristic	% exact	0.094 47	± 2.01%
	Average	0.000 49	± 0.001 45
Pessimistic heuristic	% exact	0.081 18	± 1.86%
	Average	0.011 62	± 0.007 05
Optimistic heuristic	% exact	0.083 54	± 1.89%
	Average	0.022 38	± 0.009 79

Table C.4: Variance and margin of error for % of exact solutions and average regret ratio of the midpoint, pessimistic and optimistic heuristics experiments (TFT).

Appendix D

Experimental Results for the Join Ordering Heuristic

D.1 Max-min heuristic additional results

This section presents the full results of evaluating the max-min heuristic using the synthetic data set as discussed in Section 8.3. The heuristic is also compared with the midpoint, pessimistic and optimistic heuristics.

Our heuristic performed well on this synthetic data set. It outperforms the midpoint, pessimistic and optimistic heuristics on all measuring criteria. Our heuristic found the optimal solution in 98% of the tested cases. The regret ratio for each of the cases where our heuristic did not find the optimal solution is less than 1.0000000000013, apart form the single case of the overall worst regret ratio of 1.23. This value of the overall worst regret ratio shows how well our heuristic does in terms of avoiding bad plans. On this criterion, the midpoint heuristic is marginally worse than our heuristic with a ratio of 1.24, and both were far better than the pessimistic and optimistic heuristics with values of 6 and 243, respectively.

Heuristic	Measure	3	4	5	6	7	8	9	10	11	12	Max\Average	
Max-min heuristic	% exact	100%	100%	100%	100%	97%	100%	100%	99%	94%	90%	Ave.	98%
	Worst	1	1	1	1	1.233 38	1	1	1[a]	1[b]	1.000 01	Max	1.233 38
	Average	1	1	1	1	1.002 33	1	1	1[c]	1[d]	1.000 00	Ave.	1.000 23
Midpoint heuristic	% exact	100%	100%	100%	79%	95%	97%	99%	96%	93%	88%	Ave.	94.70%
	Worst	1	1	1	1.002 47	1.000 04	1.100 86	1.043 16	1.079 29	1.237 80	1.000 01	Max	1.237 80
	Average	1	1	1	1.000 02	1.000 00	1.001 01	1.000 43	1.000 79	1.002 38	1.000 00	Ave.	1.000 46
Pessimistic heuristic	% exact	100%	99%	91%	92%	93%	90%	86%	89%	89%	78%	Ave.	90.70%
	Worst	1	1.000 05	3.549 03	3.272 57	2.193 34	6.081 22	3.717 06	5.734 37	2.684 75	4.744 32	Max	6.081 22
	Average	1	1.000 00	1.091 69	1.067 47	1.039 88	1.150 78	1.107 63	1.082 83	1.053 43	1.135 21	Ave.	1.072 89
Optimistic heuristic	% exact	98%	92%	86%	85%	77%	86%	82%	75%	64%	70%	Ave.	81.50%
	Worst	22.1453	35.6211	65.458 22	17.846 90	242.6250	6.125 71	73.761 46	18.343 45	40.521 73	23.592 90	Max	242.6250
	Average	1.235 85	1.718 00	2.397 60	1.834 96	7.071 39	1.140 42	2.380 63	1.678 22	2.607 47	1.643 30	Ave.	2.370 78

Table D.1: Full results for the max-min, midpoint, pessimistic and optimistic heuristics (synthetic data).

[a]The precise value is: 1.000000000000095
[b]The precise value is: 1.0000000000035
[c]The precise value is: 1.00000000000000095
[d]The precise value is: 1.000000000000035

D.2 Statistical measures

Heuristic	Measure	Variance	Margin of error
Max-min heuristic	% exact	0.001 06	± 0.20%
	Average	4.90×10^{-7}	$\pm 4.34 \times 10^{-5}$
Midpoint heuristic	% exact	0.004 04	± 0.39%
	Average	5.34×10^{-7}	$\pm 4.53 \times 10^{-5}$
Pessimistic heuristic	% exact	0.003 52	± 0.37%
	Average	0.002 37	± 0.003 02
Optimistic heuristic	% exact	0.009 37	± 0.60%
	Average	2.667 65	± 0.101 35

Table D.2: Variance and margin of error for % of exact solutions and average regret ratio for experiments of the max-min midpoint, pessimistic and optimistic heuristics.

To measure the stability of the experimental evaluations, we used the variance and the confidence interval based on 95% confidence level. Table D.2 shows the stability of the percentage of the exact solution and the overall average regret ratio measuring criteria for the max-min, midpoint, pessimistic and optimistic heuristics. The small variance indicates that the results are condensed around the values of the percentage of the exact solution and the overall average regret ratio and hence a higher stability. The margin of error for the percentage of the exact solution is less than 1% on all heuristics. Generally, the margin of error for the average regret ratio is very small on all heuristics however it is slightly larger for the optimistic heuristic with value less than 0.11.

Appendix E

Developed Software

In order to study the selection ordering and join ordering problems, we developed various pieces of software. The software also helped us to understand various problems as well as to verify and test different properties and heuristics.

Recall from Section 5.2.1 that the software was implemented in Java. The Eclipse IDE (Juno release), which is JDK compliant and uses the JavaSE-1.7 execution environment, was used to compile the Java code. Section E.1 discusses the main functionalities implemented, while Section E.2 presents how we detected a subset of dominant operators for a given set of selection operators, which was used to find initial plans for the selection operator heuristic, as discussed in Section 4.2.1. Section E.3 discusses how we dealt with precision problems in Java when we implemented and tested our join ordering heuristic presented in Section 7.3.

E.1 General functionality

In this section, we briefly mention the main information provided by, and functions implemented by, our software.

- We developed a program to draw the selectivity intervals of selection operators for any plan in order to visualise relationships between operators. Figure E.1(a) shows operator intervals using their actual values for the y-axis, while Figure E.1(b) draws the operators according to the width of their selectivities with their midpoint aligned.

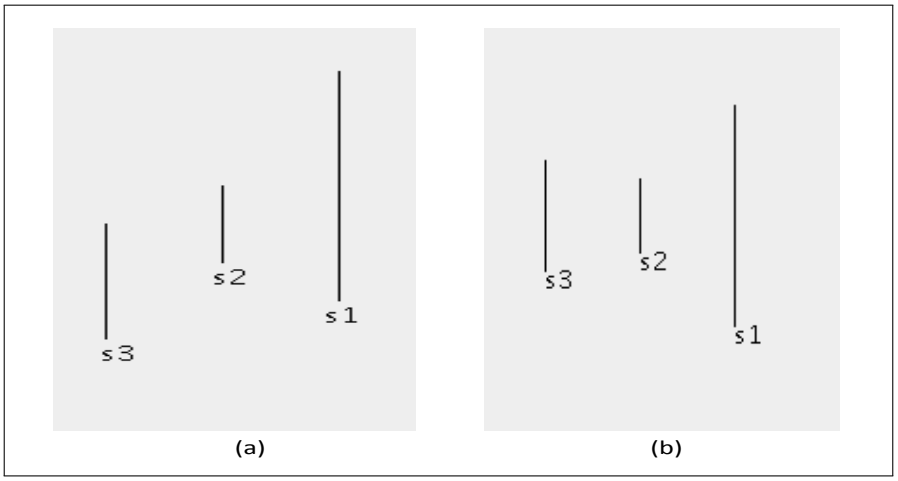


Figure E.1: Two ways in which to visualise three selection operators.

- We implemented the brute-force algorithms for MRO of the selection ordering, join ordering and TFT problems. Figure E.2 shows a full report after running the brute-force algorithm for MRO of a set of three selection operators. The report includes the cost and regret tables for each plan under each extreme scenario. In addition, the maximum regret of each plan is identified and the details of the optimal solution is reported.

- We developed programs to generate synthetic data sets for the selection ordering problem, TFT problem and join ordering problem as described in Sections 5.2.1, 6.2.2 and 8.2, respectively. Figure E.3 shows how properties can be specified before generating a random set of selection operators.

- We also implemented and tested some relevant algorithms during our study, such as the disentangling algorithm and average midpoint heuristic that we discussed in Sections 3.6.1 and 3.6.2 respectivelyy.

- Experimental evaluations discussed in Chapters 5, 6 and 8 were conducted using the software we implemented. Figure E.4 shows a sample report generated after testing our selection ordering heuristic $(((optimistic, W+), W+), W+)$ with 100 different test cases. In this experiment, our heuristic starts with the result of the optimistic heuristic as an initial plan and then performs three iterations.

```
---------- Selection operator list ----------
selection operator S1  [0.4 0.99]   cost = 1.0
selection operator S2  [0.6 0.95]   cost = 1.0
selection operator S3  [0.1 0.7]    cost = 1.0

---------- Cost Table ----------
        S1 min S2 min S3 min   S1 min S2 min S3 max   S1 min S2 max S3 min   S1 min S2 max S3 max   S1 max S2 min S3 min   S1 max S2 min S3 max   S1 max S2 max S3 min   S1 max S2 max S3 max
S1 S2 S3          16.4                  16.4                  17.8                  17.8                  25.84                  25.84                  29.305                 29.305
S1 S3 S2          14.4                  16.8                  14.4                  16.8                  20.89                  26.83                  20.89                  26.83
S2 S1 S3          18.4                  18.4                  23.3                  23.3                  21.94                  21.94                  28.905                 28.905
S2 S3 S1          16.6                  20.2                  20.45                 26.15                 16.6                   20.2                   20.45                  26.15
S3 S1 S2          11.4                  19.8                  11.4                  19.8                  11.99                  23.93                  11.99                  23.93
S3 S2 S1          11.6                  21.2                  11.95                 23.65                 11.6                   21.2                   11.95                  23.65

---------- Regret Table ----------
        S1 min S2 min S3 min   S1 min S2 min S3 max   S1 min S2 max S3 min   S1 min S2 max S3 max   S1 max S2 min S3 min   S1 max S2 min S3 max   S1 max S2 max S3 min   S1 max S2 max S3 max
S1 S2 S3          5.0                   0.0                   6.4                   1.0                   14.24                  5.64                   17.355                 5.655
S1 S3 S2          3.0                   0.4                   3.0                   0.0                   9.29                   6.63                   8.94                   3.18
S2 S1 S3          7.0                   2.0                   11.9                  6.5                   10.34                  1.74                   16.955                 5.255
S2 S3 S1          5.2                   3.8                   9.05                  9.35                  5.0                    0.0                    8.5                    2.5
S3 S1 S2          0.0                   3.4                   0.0                   3.0                   0.39                   3.73                   0.04                   0.28
S3 S2 S1          0.2                   4.8                   0.55                  6.85                  0.0                    1.0                    0.0                    0.0

---------- Maximum Regret List ----------
S1 S2 S3   17.355
S1 S3 S2   9.29
S2 S1 S3   16.955
S2 S3 S1   9.35
S3 S1 S2   3.73
S3 S2 S1   6.85

---------- Solution ----------
Minimal regret solution: S3 S1 S2 with worst case scenario S1 max S2 min S3 max and  regret = 3.73
```

Figure E.2: MRO brute force approach calculation.

```
* randomType =  "random", "notNested", "nested" or "overlapped"
* outputType = "print" or "save"
* fileName is optional input for save operation
* if the given COST is negative then the generated operators will have random
  COST otherwise it will have a fix COST as the provided positive number
* input format = no_OfOperator randomType COST outputType fileName*
Please enter your command:
4 random -1 print
S1      0.1322961449623108      0.15253937244415283     7.0
S2      0.16894972324371338     0.5501213669776917      44.0
S3      0.41359013319015503     0.5910953283309937      44.0
S4      0.24094700813293457     0.6358627676963806      59.0
```

Figure E.3: Random selection operator generator.

- We implemented software to process and prepare the SSB and Enron data sets. For example, the implemented software created the histograms for the SSB data set. For the Enron data set, our software was used to compute the minimum and maximum selectivity intervals for the chosen keywords.

E.2 Detecting domination in selection operators

In Section 4.2.1 we mentioned that we could use the approach in Section 3.6.1 to find the subsets of dominant operators for a given set of selection operators, but instead we used the Bellman-Ford algorithm. The Bellman-Ford algorithm is implemented in a free Java library called JGraphT [90]. This Java library specialises in graph theory and its algorithms. To find a subset of dominant operators, we create a graph to represent domination between operators.

In order to find the subset with the maximum cardinality (D:C), we create a graph where vertices represent selection operators and edges represent domination relationships between operators. The longest path in the graph yields the subset with the maximum cardinality (D:C).

The Bellman-Ford algorithm is a well-known algorithm for finding the shortest path in a weighted graph [56]. However, we want to find the longest path. Therefore, we set the weight of all edges representing a domination to -1. Then we use the Bellman-Ford algorithm to find the shortest path with negative weights, which in turn returns the subset of vertices connected by the longest path (i.e. the subset

```
------------- Selection Operator list 100 -------------
S1 [ 0.06969624757766724, 0.23373466730117798]   cost = 1.0
S4 [ 0.3181183934211731, 0.8949145078659058]     cost = 1.0
S2 [ 0.4567340612411499, 0.6063461303710938]     cost = 1.0
S3 [ 0.7969278693199158, 0.9513689279556274]     cost = 1.0

--------------------- Solution ----------------------

Minmax Regret solution:         S1 S2 S4 S3 with minmax regret value: 0.06736881420296115
        & worst case scenario:                                  S1max S4min S2max S3min
Initial optimistic solution:            S1 S4 S2 S3 with minmax regret value: 0.11287848373048948
        Acctual worst case scenario for max-min solution:       S1max S2min S3min S4max

optimistic ratio:               1.675530214772995

Max-min solution is different than minmax regret solution               ############

-----

Iteration # 1: initial plan =   S1 S4 S2 S3

Max-min Algo. solution:         S1 S2 S4 S3 with minmax regret value: 0.06736881420296115
        Algorithm's worst case scenario:                        S1max S4min S2max S3min
        Acctual worst case scenario for max-min solution:       S1max S4min S2max S3min
Max-min ratio:          1.0


-----

Iteration # 2: initial plan =   S1 S2 S4 S3

Max-min Algo. solution:         S1 S2 S4 S3 with minmax regret value: 0.06736881420296115
        Algorithm's worst case scenario:                        S1max S4min S2max S3min
        Acctual worst case scenario for max-min solution:       S1max S4min S2max S3min
Max-min ratio:          1.0


Max-min solution and minmax regret solution are the same
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

============================================= 100 cases of 4 Oper ================================================
|| The following are the summary after performing 2 iterations:                                               ||
|| Number of match cases between Max-min algorithm and the minmax regret approach =          93                ||
|| Maximum ratio for Max-min algorithm=                                1.1092230475317548                      ||
|| Average ratio for Max-min algorithm=                                1.0017900573555492                      ||
==================================================================================================================
```

Figure E.4: Max-min heuristic report for version $(((optimistic, W+), W+), W+)$ after testing 100 cases with 4 operators.

with the maximum cardinality).

If we are looking for the subset with the maximum cardinality and largest total width (D:CW), we first find the subset with the maximum cardinality. If there is only one subset with maximum cardinality, then it is the one that satisfies (D:CW). Otherwise, we find the total width of the selectivity intervals in each subset and return the one with the largest total width.

On the other hand, if we want to find the subset with the largest total width (D:W), we create a different graph. In this graph, we create two vertices for each selection operator, namely $\underline{s}_i$ and $\overline{s}_i$, representing the minimum and maximum selectivities of operator σ_i, where $1 \leq i \leq n$ and n is the total number of operators in the given set. For each pair of vertices $\underline{s}_i$ and $\overline{s}_i$, there is an edge with weight equal to $-(\overline{s}_i - \underline{s}_i)$, which represents the negative width of the selectivity interval for σ_i. If σ_i dominates σ_j, then we create an edge with zero weight between $\overline{s}_i$ and $\underline{s}_j$ to represent the domination. Then we use the Bellman-Ford algorithm to find the subset with the largest total width.

E.3 Dealing with precision in Java

The largest primitive decimal data type in Java is *double*, with a precision of 64-bits as specified in the IEEE Standard for Binary Floating-Point Arithmetic 754 [1]. The *double* type can handle large numbers (in scientific representation only, e.g. 2.35 E15) but with a lack of precision and rounding control for the decimal part of numbers. This caused problems for us when we compared the cost of plans in the join ordering problem since we deal with large numbers (e.g. joining up to 12 relations with cardinality up to 10^7), as illustrated in the following example.

Example E.3.1 Consider plan ρ and scenario x, and assume that we want to calculate the regret of plan ρ under scenario x. Assume the following values for $Cost(\rho, x)$ and $Cost(\rho_{opt(x)}, x)$, which is the cost of the optimal plan for x.

$$\begin{aligned} Cost(\rho, x) &= 161562424665980150959763226 49.79014 \\ Cost(\rho_{opt(x)}, x) &= 161562424665980150955397204 20.91022 \end{aligned}$$

The regret for plan ρ under scenario x is:

$$\gamma(\rho, x) = 436602228.879928587$$

However, the *double* data type in Java represents both of the values of $Cost(\rho, x)$ and $Cost(\rho_{opt(x)}, x)$ as follows:

$$Cost(\rho, x) = Cost(\rho_{opt(x)}, x) = 1.6156242466598015 \text{ E28}$$

Therefore, the regret $\gamma(\rho, x)$ is zero when the *double* data type is used in Java. Obviously, this is not correct. ◇

To overcome this problem, we used the *BigDecimal* class from the *java.math* library. This class allows the programmer to specify the scale in terms of decimal places (i.e. the number of digits after the decimal point) and the rounding method. Basic arithmetic in this class can be performed via specific methods. We used the *BigDecimal* class for the calculations used in the join ordering heuristic.

Bibliography

[1] The Java Tutorials. `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html`, December 2014.

[2] H. Aissi, C. Bazgan, and D. Vanderpooten. Approximating min-max (regret) versions of some polynomial problems. In *Proceedings of the 12th annual International Conference on Computing and Combinatorics*, pages 428–438, 2006.

[3] H. Aissi, C. Bazgan, and D. Vanderpooten. Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438, 2009.

[4] A. Allahverdi and J. Mittenthal. Scheduling on m parallel machines subject to random breakdowns to minimize expected mean flow time. *Naval Research Logistics (NRL)*, 41(5):677–682, 1994.

[5] A. Allahverdi and J. Mittenthal. Scheduling on a two-machine flowshop subject to random breakdowns with a makespan objective function. *European Journal of Operational Research*, 81(2):376–387, 1995.

[6] A. Allahverdi, C. Ng, T. E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[7] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 493–500, Philadelphia, PA, USA, 1997.

[8] K. H. Alyoubi, S. Helmer, and P. T. Wood. Query optimisation based on measures of regret. In *Proceedings of the 7th Saudi Students Conference in the UK (SSC)*, Edinburgh, United Kingdom, 2014.

[9] K. H. Alyoubi, S. Helmer, and P. T. Wood. Ordering selection operators under partial ignorance. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 310–317, 2015.

[10] K. H. Alyoubi, S. Helmer, and P. T. Wood. Ordering selection operators using the minmax regret rule. *CoRR*, abs/1507.08257, 2015.

[11] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[12] L. Ashdown, M. Colgan, T. Kyte, et al. Oracle database SQL tuning guide. `https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL192`, December 2014.

[13] I. Averbakh. On the complexity of a class of combinatorial optimization problems with uncertainty. *Mathematical Programming*, 90(2):263–272, 2001.

[14] I. Averbakh. Minmax regret linear resource allocation problems. *Operations Research Letters*, 32(2):174–180, 2004.

[15] I. Averbakh. Computing and minimizing the relative regret in combinatorial optimization with interval data. *Discrete Optimization*, 2(4):273 – 287, 2005.

[16] R. Avnur and J. Hellerstein. Eddies: Continuously adaptive query processing. *SIGMOD Rec.*, 29(2):261–272, May 2000.

[17] B. Babcock and S. Chaudhuri. Towards a robust query optimizer: a principled and practical approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 119–130, 2005.

[18] S. Babu, P. Bizarro, and D. DeWitt. Proactive Re-optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 107–118, New York, NY, USA, 2005. ACM.

[19] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 407–418, 2004.

[20] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49–71, 2003.

[21] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *ACM Sigmod Record*, 29(1):68–79, 2000.

[22] J. Boulos, Y. Viemont, and K. Ono. A neural networks approach for query cost evaluation. *Transactions in Information Processing Society of Japan*, 12:38, 2001.

[23] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A multidimensional workload-aware histogram. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 211–222, 2001.

[24] N. Bruno, Y. Kwon, and M.-C. Wu. Advanced join strategies for large-scale distributed computation. *Proceedings of the VLDB Endowment*, 7(13):1484–1495, 2014.

[25] A. Candia-Vjar, E. lvarez Miranda, and N. Maculan. Minmax regret combinatorial optimization problems: an algorithmic perspective. *RAIRO - Operations Research*, 45:101–129, 4 2011.

[26] D. R. Carr, J. H. Greenberg, E. W. Hart, G. Konjevod, E. Lauer, H. Lin, T. Morrison, and A. C. Phillips. Robust optimization of contaminant sensor placement for community water systems. *Mathematical Programming*, 107(1):337–356, 2006.

[27] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1):99–128, 1982.

[28] S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 34–43. ACM, 1998.

[29] S. Chaudhuri, V. Ganti, and L. Gravano. Selectivity estimation for string predicates: Overcoming the underestimation problem. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 227–238. IEEE, 2004.

[30] T.-Y. Chen and C.-W. Tsui. Optimism and pessimism in decision making based on intuitionistic fuzzy sets. In *11th Joint International Conference on Information Sciences*. Atlantis Press, 2008.

[31] F. Chu, J. Halpern, and J. Gehrke. Least expected cost query optimization: What can we expect? In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 293–302, 2002.

[32] F. Chu, J. Y. Halpern, and P. Seshadri. Least expected cost query optimization: an exercise in utility. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 138–147, 1999.

[33] S. Cluet and G. Moerkotte. On the complexity of generating optimal left-deep processing trees with cross products. In *Proceedings of the International Conference on Database Theory (ICDT)*, volume 893, pages 54–67. Springer, 1995.

[34] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[35] W. W. Cohen. Enron email dataset. `https://www.cs.cmu.edu/~./enron/`.

[36] E. Conde. On a constant factor approximation for minmax regret problems using a symmetry point scenario. *European Journal of Operational Research*, 219(2):452–457, 2012.

[37] A. Condon, A. Deshpande, L. Hellerstein, and N. Wu. Flow algorithms for two pipelined filter ordering problems. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 193–202, 2006.

[38] T. Connolly and C. Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, 4th edition, 2005.

[39] H. D, P. N. Darera, and J. R. Haritsa. On the production of anorexic plan diagrams. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 1081–1092, 2007.

[40] R. L. Daniels and P. Kouvelis. Robust scheduling to hedge against processing time uncertainty in single-stage production. *Management Science*, 41(2):363–376, Feb. 1995.

[41] M. S. Daskin, S. M. Hesse, and C. S. Revelle. α-reliable p-minimax regret: A new model for strategic facility location modeling. *Location Science*, 5(4):227–246, 1997.

[42] X. Deng, H.-N. Liu, J. Long, and B. Xiao. Competitive analysis of network load balancing. *Journal of Parallel and Distributed Computing*, 40(2):162–172, Feb. 1997.

[43] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 143–154, 2005.

[44] A. Deshpande, Z. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, Jan. 2007.

[45] D. Dubois, H. Prade, and R. Sabbadin. Decision-theoretic foundations of qualitative possibility theory. *European Journal of Operational Research*, 128(3):459–478, 2001.

[46] A. Dutt and J. R. Haritsa. Plan bouquets: A fragrant approach to robust query processing. *ACM Trans. Database Syst.*, 41(2):11:1–11:37, May 2016.

[47] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Pearson, 6th edition, 2011.

[48] O. Etzioni, S. Hanks, T. Jiang, R. M. Karp, O. Madani, and O. Waarts. Efficient information gathering on the internet. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 234–243, Oct 1996.

[49] L. Fegaras. A new heuristic for optimizing large queries. In *Database and Expert Systems Applications*, pages 726–735. Springer, 1998.

[50] P. Furtado and H. Madeira. Summary grids: building accurate multidimensional histograms. In *Proceedings of the 6th International Conference on Database Systems for Advanced Applications*, pages 187–194. IEEE Computer Society, Apr 1999.

[51] S. Ganguly. Design and analysis of parametric query optimization algorithms. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 228–238, 1998.

[52] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Prentice Hall, Upper Saddle River, 2000.

[53] M. Garey. Optimal task sequencing with precedence constraints. *Discrete Mathematics*, 4(1):37–56, 1973.

[54] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[55] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 476–487, 2002.

[56] A. V. Goldberg and T. Radzik. A heuristic improvement of the Bellman-Ford algorithm. *Applied Mathematics Letters*, 6(3):3–6, 1993.

[57] R. Goldman and J. Widom. WSQ/DSQ: A practical approach for combined querying of databases and the web. *SIGMOD Rec.*, 29(2):285–296, May 2000.

[58] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[59] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys (CSUR)*, 25(2):73–169, 1993.

[60] S. O. Hansson. *Decision Theory: A Brief Introduction*. Department of Philosophy and the History of Technology, Royal Institute of Technology (KTH), Stockholm, 2005.

[61] J. M. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, 22(2):267–276, June 1993.

[62] T. Ibaraki and T. Kameda. On the optimal nesting order for computing n-relational joins. *ACM Trans. Database Syst.*, 9(3):482–502, Sept. 1984.

[63] Y. Ioannidis. The history of histograms (Abridged). In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 19–30, 2003.

[64] Y. E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. *SIGMOD Rec.*, 20(2):268–277, 1991.

[65] Y. E. Ioannidis and Y. C. Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. *ACM SIGMOD Record*, 20(2):168–177, 1991.

[66] Y. E. Ioannidis and V. Poosala. Histogram-based solutions to diverse database estimation problems. *IEEE Data Engineering*, 18(3):10–18, September 1995.

[67] S. Johnson. Optimal sequential testing. RAND Research Memorandum RM1652, RAND Corporation, Santa Monica, California, 1956.

[68] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. *SIGMOD Rec.*, 27(2):106–117, June 1998.

[69] A. Kasperski. *Discrete Optimization with Interval Data - Minmax Regret and Fuzzy Approach*. Springer, 2008.

[70] A. Kasperski and P. Zielinski. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Information Processing Letters*, 97(5):177–180, 2006.

[71] A. Kasperski and P. Zielinski. A 2-approximation algorithm for interval data minmax regret sequencing problems with the total flow time criterion. *Operations Research Letters*, 36(3):343–344, 2008.

[72] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*, volume 14. Springer Science & Business Media, 2013.

[73] R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of nonrecursive queries. In *Proceedings of the 12th International Conference on Very Large Data Bases (VLDB)*, pages 128–137, 1986.

[74] P.-A. Larson, W. Lehner, J. Zhou, and P. Zabback. Cardinality estimation using sample views with quality assurance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 175–186, 2007.

[75] V. Lebedev and I. Averbakh. Complexity of mnimizing the total flow time with interval data and minmax regret criterion. *Discrete Appl. Math.*, 154(15):2167–2177, Oct. 2006.

[76] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015.

[77] J. Lenstra, A. R. Kan, and P. Brucker. Complexity of machine scheduling problems. In *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.

[78] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, January/February 1978.

[79] G. Lohman. Is query optimization a "solved" problem? `http://wp.sigmod.org/?p=1075`, 2014.

[80] C.-C. Lu, S.-W. Lin, and K.-C. Ying. Robust scheduling on a single machine to minimize total flow time. *Computers & Operations Research*, 39(7):1682–1691, 2012.

[81] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent selectivity estimation via maximum entropy. *The Very Large Data Bases (VLDB) Journal*, 16(1):55–76, January 2007.

[82] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic. Robust query processing through progressive optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 659–670, 2004.

[83] A. Mazeika, M. H. Bőhlen, N. Koudas, and D. Srivastava. Estimating the selectivity of approximate string queries. *ACM Trans. Database Syst.*, 32(2), June 2007.

[84] J. B. Mazzola and A. W. Neebe. Resource-constrained assignment scheduling. *Operations Research*, 34(4):560–572, July 1986.

[85] G. Moerkotte. Building query compilers. http://pi3.informatik.uni-mannheim.de/∼moer/querycompiler.pdf, September 2009.

[86] G. Moerkotte, D. DeHaan, N. May, A. Nica, and A. Boehm. Exploiting ordered dictionaries to efficiently construct histograms with Q-error guarantees in SAP HANA. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 361–372, 2014.

[87] G. Moerkotte and W. Scheufele. Constructing optimal bushy processing trees for join queries is NP-hard. Technical report, 1996.

[88] C. L. Monma and J. B. Sidney. Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research*, 4(3):pp. 215–224, 1979.

[89] A. Motro. Management of uncertainty in database systems. In *Modern Database Systems*, pages 457–476. ACM Press/Addison-Wesley Publishing Co., 1995.

[90] B. Naveh and et al. JGraphT Java class library. `http://jgrapht.org/`, January 2012.

[91] T. Neumann and C. A. Galindo-Legaria. Taking the edge off cardinality estimation errors using incremental execution. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 73–92, 2013.

[92] T. Neumann, S. Helmer, and G. Moerkotte. On the optimal ordering of maps and selections under factorization. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 490–501, 2005.

[93] F. Olken and D. Rotem. Simple random sampling from relational databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 160–169, 1986.

[94] K. Ono and G. M. Lohman. Measuring the complexity of join enumeration in query optimization. In *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, pages 314–325, 1990.

[95] M. Peterson. *An Introduction to Decision Theory*. Cambridge University Press, 2009.

[96] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. *ACM Sigmod Record*, 14(2):256–276, June 1984.

[97] S. Plotkin. Competitive routing of virtual circuits in ATM networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1128–1136, Aug. 1995.

[98] N. Polyzotis and M. Garofalakis. Statistical synopses for graph-structured XML databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 358–369, 2002.

[99] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. *SIGMOD Rec.*, 25(2):294–305, 1996.

[100] T. Rabl, M. Poess, H.-A. Jacobsen, P. O'Neil, and E. O'Neil. Variations of the star schema benchmark to test the effects of data skew on query performance. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 361–372, 2013.

[101] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, Berkeley, CA, USA, 3rd edition, 2003.

[102] R. T. Rockafeller. *Convex Analysis*. Princeton University Press, 1970.

[103] L. J. Savage. The theory of statistical decision. *Journal of the American Statistical Association*, 46(253):55–67, 1951.

[104] D. A. Schneider and D. J. DeWitt. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, pages 469–480, 1990.

[105] N. Schweikardt, T. Schwentick, and L. Segoufin. Algorithms and theory of computation handbook. Database theory: query languages chapter. Chapman & Hall/CRC, 2010.

[106] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 23–34, New York, NY, USA, 1979. ACM.

[107] M. Siepak and J. Jozefczyk. Solution algorithms for unrelated machines min-max regret scheduling problem with interval processing times and the total flow time criterion. *Annals of Operations Research*, pages 1–17, 2014.

[108] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 6th edition, 2011.

[109] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[110] Y. Sotskov and T.-C. Lai. Minimizing total weighted flow time under uncertainty using dominance and a stability box. *Computers and Operations Research*, 39(6):1271–1289, 2012.

[111] Y. N. Sotskov, N. G. Egorova, T.-C. Lai, and F. Werner. The stability box in interval data for minimizing the sum of weighted completion times. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 14–23, 2011.

[112] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 250–258, 2005.

[113] K. Tzoumas, A. Deshpande, and C. S. Jensen. Efficiently adapting graphical models for selectivity estimation. *The VLDB Journal*, 22(1):3–27, Feb. 2013.

[114] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[115] S. D. Viglas. Write-limited sorts and joins for persistent memory. *Proceedings of the VLDB Endowment*, 7(5):413–424, 2014.

[116] A. Volgenant and C. W. Duin. Improved polynomial algorithms for robust bottleneck problems with interval data. *Computers and Operations Research*, 37(5):909–915, May 2010.

[117] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Selectivity estimation on streaming spatio-textual data using local correlations. *Proceedings of the VLDB Endowment*, 8(2), 2014.

[118] W. Wu, X. Wu, H. Hacigümüş, and J. F. Naughton. Uncertainty aware query execution time prediction. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 7(14):1857–1868, October 2014.

[119] J. Yang and G. Yu. On the robust single machine scheduling problem. *Journal of Combinatorial Optimization*, 6(1):17–33, 2002.

[120] S. Yin, A. Hameurlain, and F. Morvan. Robust query optimization methods with respect to estimation errors: A survey. *ACM SIGMOD Record*, 44(3):25–36, Dec. 2015.

[121] V. Zadorozhny, L. Raschid, M. E. Vidal, T. Urhan, and L. Bright. Efficient evaluation of queries in a mediator for WebSources. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 85–96, 2002.

[122] N. Zhang, P. J. Haas, V. Josifovski, G. M. Lohman, and C. Zhang. Statistical learning techniques for costing XML queries. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 289–300, 2005.