Multi-Objective Parametric Query Optimization

Immanuel Trummer and Christoph Koch
{firstname}.{lastname}@epfl.ch
École Polytechnique Fédérale de Lausanne

ABSTRACT

Classical query optimization compares query plans according to one cost metric and associates each plan with a constant cost value. In this paper, we introduce the Multi-Objective Parametric Query Optimization (MPQ) problem where query plans are compared according to multiple cost metrics and the cost of a given plan according to a given metric is modeled as a function that depends on multiple parameters. The cost metrics may for instance include execution time or monetary fees; a parameter may represent the selectivity of a query predicate that is unspecified at optimization time.

MPQ generalizes parametric query optimization (which allows multiple parameters but only one cost metric) and multi-objective query optimization (which allows multiple cost metrics but no parameters). We formally analyze the novel MPQ problem and show why existing algorithms are inapplicable. We present a generic algorithm for MPQ and a specialized version for MPQ with piecewise-linear plan cost functions. We prove that both algorithms find all relevant query plans and experimentally evaluate the performance of our second algorithm in a Cloud computing scenario.

1. INTRODUCTION

Classical Query Optimization (CQ) models the cost of a query plan as a scalar cost value $c \in \mathbb{R}$. The optimization goal is to find the plan with minimal cost for a given query. Multi-Objective Query Optimization (MQ) [14, 22, 31] generalizes the classical model and associates each query plan with a cost vector $\mathbf{c} \in \mathbb{R}^n$ describing the cost of the plan according to multiple cost metrics. The optimization goal is to find a set of query plans that are all Pareto-optimal, meaning that no other plan has better cost according to all cost metrics at the same time. Parametric Query Optimization (PQ) [13, 17, 7] generalizes the classical model in a different way and associates each query plan with a cost function $c : \mathbb{R}^n \rightarrow \mathbb{R}$ describing the cost of the plan as function of multiple parameters whose values are not known at optimization time. The optimization goal is to find a plan set that contains an optimal plan for each possible combination of parameter values. In this paper, we introduce Multi-Objective Parametric Query Optimization (MPQ) and describe and analyze corresponding query optimization algorithms; MPQ generalizes and unifies the cost models of MQ and of PQ at the same time by representing the cost of a query plan as vector-valued function $\mathbf{c} : \mathbb{R}^n \rightarrow \mathbb{R}^m$. This allows to model multiple parameters as well as multiple cost metrics and is required in the following example scenarios.

Scenario 1. A Cloud provider lets users query a large scientific data set over a Web interface. Query processing takes place in the Cloud. User queries correspond to query templates such as `SELECT * FROM Table1 WHERE P1 AND P2` where `P1` and `P2` represent unspecified predicates; users submit queries by specifying those predicates in the Web interface. Query processing time in the Cloud can often be reduced when accepting higher monetary fees [22]. After having submitted a query, users are therefore provided with a visualization of possible tradeoffs between execution time and monetary fees (that are realized by alternative query plans) and can select their preferred tadeoff. To speed up this process, the Cloud provider calculates all relevant query plans for each query template in a preprocessing step. The selectivities of the predicates are unknown at preprocessing time and must be represented as parameters, execution time and monetary fees are the two cost metrics. A query plan is relevant if there is at least one point in the parameter space for which its time-fees tradeoff is Pareto-optimal, meaning that no alternative plan has both, lower fees and lower execution time. Figure 1 illustrates the preprocessing result in this scenario (for a query with two unspecified predicates).

Scenario 2. Embedded SQL queries are a classical use case for PQ [17, 7]: to avoid query optimization overhead at run time, all potentially relevant query plans are calculated in advance for a given query template. Parameters model the selectivity of unspecified predicates or the amount of buffer space that is available at run time. Execution time is the only cost metric in the classical setting. In the context of approximate query processing [3], execution time can however be traded against result precision. In such a scenario, the two metrics execution time and result precision both must be considered during optimization. The optimal query plan is selected at run time based not only on concrete parameter values but also on a policy that determines the optimal tradeoff between result precision and execution time, based for instance on the current system load or on minimum precision requirements for one specific invocation.

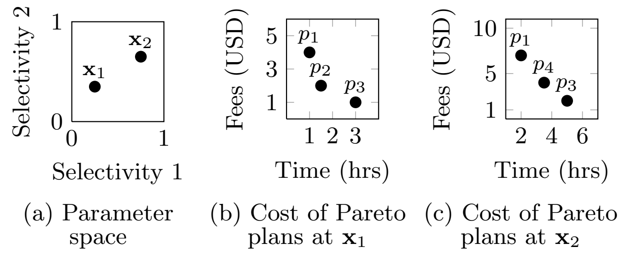


Figure 1: MPQ associates each point x in the parameter space with a set of Pareto-optimal query plans $\{p_i\}$ (the illustration uses cost metrics and parameters from Scenario 1)

The kind of query optimization that is described in the example scenarios requires to consider multiple parameters and multiple cost metrics; this is a novel variant of query optimization that we call MPQ. Figure 2 describes the context of MPQ: MPQ takes place before run time; the input to MPQ is a query associated with parameters. A parameter may represent any quantity that influences the cost of query plans and is unknown at optimization time. The goal of MPQ is to generate a complete set of relevant plans, meaning a set that contains a plan p^* for each possible plan p and each point in the parameter space $\mathbf{x}$ such that p^* has at most the same cost as p at $\mathbf{x}$ according to each cost metric. Formulated differently, the goal is to find a set of Pareto-optimal query plans for all points in the parameter space. As in PQ [17], all relevant query plans are generated in advance so that no query optimization is required at run time.

1.1 State-of-the-Art

MPQ is a generalization of MQ and of PQ; it is not possible to apply existing MQ or PQ algorithms to MPQ since PQ algorithms support only one cost metric and MQ algorithms do not support parameters. It may at first seem possible to model cost metrics as parameters; if all but one cost metric could be represented as parameters then PQ algorithms could be applied. Trying to model for instance monetary fees as a parameter in Scenario 1 (such that execution time becomes a function of predicate selectivities and monetary budget) leads however to the following problems: First, existing PQ algorithms [19, 13, 18, 5, 12, 9] usually assume that the value domain of each parameter is known in advance. This is realistic for predicate selectivity or the available amount of buffer space but not for monetary fees, as finding the minimal execution fees for a given query is a hard optimization problem all by itself. Second, cost metrics and parameters have different semantics: Assume for instance that alternative query plans for a given query have execution fees between 1 and 10 USD and that a plan p priced at 5 USD has lower execution time than all plans with higher fees. The result set of MPQ should only contain p but none of the more expensive plans since p is always preferable over them. A PQ algorithm (e.g., [13, 17]) would however generate plans with minimal execution time for each possible cost value between 6 and 10 USD, as the goal in classical PQ is to cover the whole parameter space by optimal plans (while the goal in MPQ is not to cover the whole cost space). The result set of PQ can be larger than the result set of MPQ by an arbitrary factor and result

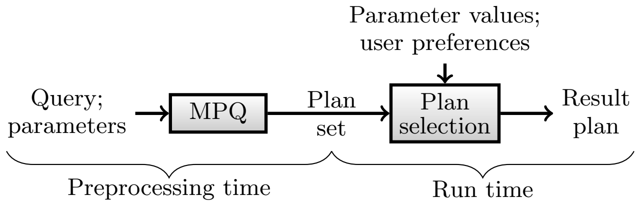


Figure 2: The context of MPQ

set size relates to optimization time. Additional problems arise since parameter domains are usually assumed to be connected intervals while cost values may be sparsely distributed in the total cost range. Altogether, transforming a MPQ problem into a PQ problem by modeling cost metrics as parameters seems inappropriate. A popular branch of PQ algorithms decomposes a PQ problem into multiple non-parametric CQ problems; it is however impossible to analogously decompose a MPQ problem into multiple non-parametric MQ problems for reasons outlined in Section 4. More related work is discussed in Section 3.

1.2 Contribution and Outline

We summarize our contributions before providing details:

- We **formally analyze the** MPQ problem with piecewise-linear (PWL) plan cost functions. We show in particular that the MPQ problem has no equivalent for certain fundamental properties of the PQ problem that have inspired the design of a broad class of PQ algorithms based on parameter space decomposition.

- We present **the first algorithms for MPQ**; those algorithms can deal with multiple cost metrics and parametric cost functions together. We present a generic MPQ algorithm that can deal with arbitrary plan cost functions and a specialization for PWL cost functions.

- We **formally analyze** our algorithms and show that both presented algorithms guarantee to generate all relevant query plans. We **experimentally evaluate** the algorithm for PWL cost functions in an example scenario similar to the one introduced in Example 1.

Section 2 introduces the formal model, Section 3 discusses related work. We analyze the MPQ problem in Section 4 and show that it differs from PQ in several important aspects. Section 5 presents and analyzes the Relevance Region Pruning Algorithm (RRPA). This is a generic algorithm for MPQ that can handle arbitrary plan cost functions. As many algorithms for CQ, MQ, and PQ, it is based on dynamic programming and generates and prunes query plans for joining table sets of increasing cardinality. The pruning function differs from prior approaches: Every query plan is associated with a region in the parameter space for which it is relevant (the Relevance Region, abbreviated RR). During pruning, this region is repeatedly reduced by comparisons with alternative plans. Plans are pruned once their RR becomes empty. We prove that RRPA formally guarantees to generate all relevant query plans for arbitrary queries.

The implementation of elementary RRPA operations such as adding cost functions and intersecting RRs depends on the considered class of cost functions. Most work on PQ focuses either on linear or on PWL cost functions which both can be stored and manipulated efficiently. Linear functions are however often a bad approximation for real plan cost functions [25] while PWL functions can approximate arbitrary cost functions up to an arbitrary degree of detail [17]. We therefore focus on PWL cost functions and present PWL-RRPA, a specialization of RRPA to PWL cost functions, in Section 6. We prove that all RRs that occur during the execution of PWL-RRPA belong to a limited class of shapes and propose data structures for representing cost functions and RRs. We provide pseudo-code for implementing all elementary operations of PWL-RRPA efficiently on those data structures and analyze the resulting complexity. PWL-RRPA was experimentally evaluated in a Cloud computing scenario; the results are discussed in Section 7.

2. DEFINITIONS

A **query** is represented by a set Q of tables that need to be joined. A **query plan** specifies the join order and the operators executing scan and join operations. The symbol $\mathbb{O}$ denotes the set of available operators. Let p_1 and p_2 be two query plans that join disjoint sets of tables and $o \in \mathbb{O}$ a join operator. The function $Combine(p_1, p_2, o)$ designates the query plan that joins the results of p_1 and p_2 using operator o. Plans p_1 and p_2 are called **sub-plans** of the resulting plan. The function $\mathbb{P}(Q)$ denotes the set of all possible plans for query Q. The execution **cost** of a query plan can depend on **parameters** whose exact values are not known at query optimization time. Parameters represent for instance predicate selectivities or the amount of available buffer space at query execution time. Parameter values for a fixed set of parameters are represented as a vector $\mathbf{x}$ (bold font distinguishes vectors from scalar values in the following). The **parameter space** $\mathbb{X}$ is the set of possible parameter vectors. Query plans are compared according to a set $\mathbb{M}$ of **cost metrics** for which analytic cost models are available. Let p be a query plan and $\mathbf{x}$ a parameter vector. The cost function $\mathbf{c}(p, \mathbf{x})$ estimates the cost of plan p under the circumstances described by parameter vector $\mathbf{x}$. The cost function yields a vector $\mathbf{c}$ that contains one value for each cost metric. Let $m \in \mathbb{M}$ be a cost metric, then $\mathbf{c}^m$ denotes the cost value for that metric. The notation $\mathbf{c}(p)$ designates the cost function for a constant plan p such that $\mathbf{c}(p)(\mathbf{x}) := \mathbf{c}(p, \mathbf{x})$.

EXAMPLE 1. *This example is based on Scenario 1. Consider a query template containing three predicates that are specified at run time. The selectivities of those three predicates are three parameters, the value domain of each parameter is the continuous interval* $[0, 1]$. *The selectivities of all three predicates together can be described by a three-dimensional vector (e.g.,* $\mathbf{x} = (0.1, 0.5, 0.2)$ *if the first predicate has selectivity 10%, the second predicate has selectivity 50%, and the third predicate has selectivity 20%). The parameter space containing all possible parameter vectors is the three-dimensional space* $\mathbb{X} = [0, 1]^3 \subseteq \mathbb{R}^3$. *The cost of a fixed query plan depends on the selectivities of the predicates and is measured according to the two cost metrics execution time and monetary fees, therefore* $\mathbb{M} = \{time, fees\}$. *The value domain for each of the two cost metrics is the set* $\mathbb{R}_+ \subseteq \mathbb{R}$ *of non-negative real numbers. The cost function* $\mathbf{c}(p)$ *of a fixed plan* p *therefore maps three-dimensional parameter vectors to two-dimensional cost vectors:* $\mathbf{c}(p) : \mathbb{X} \to \mathbb{R}_+^2$.

Plan quality metrics for which a higher value is better (e.g., result precision in Scenario 2) can always be transformed into cost metrics for which a lower value is better (e.g., replace result precision $\theta \in [0, 1]$ by precision loss $1-\theta$). Let p_1 and p_2 be two query plans that produce the same result. Plan p_1 **dominates** plan p_2 in all points of the parameter space in which p_1 has at most the same cost as p_2 according to each cost metric. The function $Dom(p_1, p_2) \subseteq \mathbb{X}$ yields the parameter space region where p_1 dominates p_2:

$$Dom(p_1, p_2) = \{\mathbf{x} \in \mathbb{X} | \forall m \in \mathbb{M} : \mathbf{c}^m(p_1, \mathbf{x}) \leq \mathbf{c}^m(p_2, \mathbf{x})\}$$

The plans p_1 and p_2 mutually dominate each other in parameter space regions where they have equivalent cost. Plan p_1 **strictly dominates** plan p_2 in all points of the parameter space in which p_1 dominates p_2 without having equivalent cost. The function $StD(p_1, p_2) \subseteq Dom(p_1, p_2)$ yields the parameter space region where p_1 strictly dominates p_2:

$$StD(p_1, p_2) = Dom(p_1, p_2) \setminus \{\mathbf{x} \in \mathbb{X} | \mathbf{c}(p_1, \mathbf{x}) = \mathbf{c}(p_2, \mathbf{x})\}$$

A plan's region of optimality is in PQ the parameter space region where no alternative plan has lower cost [17]. The multi-objective analogue to the region of optimality is the **Pareto region**; the Pareto region $pReg(p) \subseteq \mathbb{X}$ of plan p is the parameter space region where no alternative plan from $\mathbb{P}(Q)$ producing the same result as p strictly dominates p:

$$pReg(p) = \mathbb{X} \setminus (\bigcup_{p^* \in \mathbb{P}(Q)} StD(p^*, p))$$

A parametric optimal set of plans is in PQ a plan set that contains at least one cost-optimal plan for each point in the parameter space [17]. The multi-objective analogue is a **Pareto plan set (PPS)**; $P \subseteq \mathbb{P}(Q)$ is a PPS iff it contains for each possible plan $p^* \in \mathbb{P}(Q)$ and each parameter vector $\mathbf{x} \in \mathbb{X}$ at least one plan plan that dominates p^* for $\mathbf{x}$:

$$\forall p^* \in \mathbb{P}(Q)\ \forall \mathbf{x} \in \mathbb{X}\ \exists p \in P : \mathbf{x} \in Dom(p, p^*)$$

EXAMPLE 2. *Let* p_1, p_2, *and* p_3 *be three plans for the same query. Assume there is only one parameter* $\sigma \in [0, 1]$ *(*$\mathbb{X} = [0, 1]$*) that represents the selectivity of an unknown predicate. The two cost metrics time and monetary fees are considered:* $\mathbb{M} = \{time, fees\}$. *The plans have the following cost functions:* $\mathbf{c}^{time}(p_1) = 2\sigma$, $\mathbf{c}^{fees}(p_1) = 3$, $\mathbf{c}^{time}(p_2) = 0.5 + \sigma$, $\mathbf{c}^{fees}(p_2) = 2$, *and plan* p_3 *has the same cost as* p_2. *The following relationships hold among others: Plans* p_2 *and* p_3 *mutually dominate each other in the entire parameter space:* $Dom(p_2, p_3) = Dom(p_3, p_2) = [0, 1]$. *Plan* p_2 *strictly dominates* p_1 *for* $\sigma > 0.5$. *The Pareto region of* p_1 *is the selectivity interval* $[0, 0.5]$. *The Pareto regions of* p_2 *and* p_3 *are the entire parameter space. The sets* $\{p_1, p_2\}$ *and* $\{p_1, p_3\}$ *both form a PPS.*

A **Pareto plan** designates in the following a plan in a PPS. A **relevance mapping (RM)** for a PPS P maps each Pareto plan to a **relevance region (RR)** in the parameter space such that we can restrict our attention to the plans whose RR includes $\mathbf{x}$ whenever we need to find the best plans for a parameter space point $\mathbf{x} \in \mathbb{X}$:

$$\forall p^* \in \mathbb{P}(Q)\ \forall \mathbf{x} \in \mathbb{X}\ \exists p \in P : \mathbf{x} \in relM(p) \cap Dom(p, p^*)$$

The RR of a plan can be different from its Pareto region. The algorithm presented in Section 5 uses RMs and discards plans with empty RRs. The **Multi-objective parametric query optimization (MPQ) problem** is the focus of this

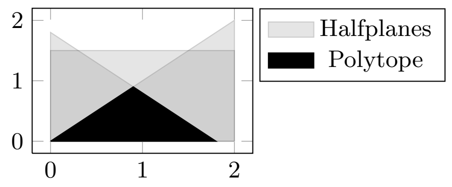


Figure 3: Two-dimensional convex polytope as intersection of three halfplanes

paper. An MPQ problem is defined by a query Q, a parameter space $\mathbb{X}$, and a set of cost metrics $\mathbb{M}$. Any PPS for Q is a solution to the MPQ problem.

We introduce a restricted variant of MPQ, the next definitions are prerequisites. An m-dimensional **convex polytope** is a set of points in $\mathbb{R}^m$ that *i)* is convex, meaning that any two points in the convex polytope are connected by a line segment that completely lies within the convex polytope again, and *ii)* corresponds to the intersection of a finite set of halfspaces, a halfspace being the set of solutions to a linear inequality of the form $\mathbf{w}^T \cdot \mathbf{x} \leq b$ with $\mathbf{w}, \mathbf{x} \in \mathbb{R}^m$ and $b \in \mathbb{R}$. Figure 3 illustrates how a convex polytope is constructed by intersecting three halfspaces in $\mathbb{R}^2$. The cost function $\mathbf{c}(p, \mathbf{x})$ of a plan p is **linear** in the entire parameter space, if for each cost metric $m \in \mathbb{M}$, there is a weight vector $\mathbf{w}_m$ and a constant $b_m \in \mathbb{R}$ such that $\mathbf{c}(p, \mathbf{x}) = \mathbf{w}_m^T \cdot \mathbf{x} + b_m$ for each $\mathbf{x} \in \mathbb{X}$. The cost function is **piecewise-linear (PWL)** if the parameter space can be partitioned into convex polytopes such that $\mathbf{c}(p, \mathbf{x})$ is linear in each polytope. Note that PWL cost functions may have discontinuities between regions in which they are linear. PWL functions are of high practical relevance since they can approximate arbitrary functions [18]. Most work on PQ (e.g., [13, 17]) restricts the PQ problem by assuming either linear or PWL cost functions. In analogy to that, we introduce a restricted variant of the MPQ problem: **PWL-MPQ** assumes that all vector-valued cost functions are PWL and that the parameter space itself forms a convex polytope (which is a standard assumption in PQ [17]). The PWL-MPQ problem is analyzed in Section 4 and a corresponding optimization algorithm is presented in Section 6. This algorithm exploits that the parameter space in PWL-MPQ can be partitioned into **linear regions** for a plan set P: a linear region is a convex polytope in the parameter space for which all plans in P have linear cost functions.

3. RELATED WORK

We introduced four different variants of query optimization in Section 1.1 (CQ, PQ, MQ, and MPQ) and justified why existing algorithms cannot be applied for MPQ. We discuss related work in PQ and MQ in more detail now.

PQ algorithms associate query plans with cost functions instead of cost values. The cost functions depend on parameters that represent for instance predicate selectivities. The goal in PQ is usually to generate a plan set that contains one optimal plan for each possible parameter value combination [13, 17, 18, 7]. Many approaches to PQ are based on parameter space decomposition [13, 17, 18, 12, 7]. They repeatedly invoke a standard optimizer to generate optimal plans for fixed parameter values (if the parameter values are fixed then the cost of a query plan can be modeled as a constant value again) in order to decompose the parameter space into regions in which a single plan is optimal. We will see in Section 4 why similar approaches fail for MPQ. Another branch of PQ algorithms [16, 11, 17, 18, 5, 9] is based on dynamic programming, similar to the CQ algorithm by Selinger [26]. They are specific to PQ since they consider only one cost metric during pruning (some approaches consider robustness in addition to execution time [4, 1] but robustness is directly derived from execution time and not an independent cost metric) and use data structures and corresponding manipulation functions that are intrinsically specific to assumptions that hold in PQ but not in MPQ (e.g., many PQ algorithms model the parameter space region in which a plan is optimal as convex polytope which works for PQ with PWL cost functions but not for MPQ with PWL cost functions as shown in Section 4). Using PQ algorithms for MPQ would require that the optimal plan according to one cost metric is always guaranteed to be optimal according to all other cost metrics. This case is unrealistic; even more so since many relevant cost metrics are anti-correlated (e.g., result precision and processing time in approximate query processing [3]). Ioannidis et al. [19] use randomized algorithms for PQ; they do not support multiple cost metrics. Randomized algorithms can never offer formal worst-case guarantees on generating complete plan sets, unlike the algorithms presented in this paper. Classical PQ deals with unknown parameter values by generating all plans that could be relevant. Other approaches define probability distributions over parameter values with the goal to generate one robust plan [4, 1] or one plan that minimizes expected cost [10]. In contrast to that, classical PQ aims at scenarios where new information becomes available at run time that should be considered during plan selection.

MQ algorithms compare query plans according to several cost metrics. The goal is to find a plan that represents the best compromise between conflicting metrics according to user preferences. The single-objective query optimization algorithm by Selinger has been generalized to MQ [14, 31]: plans producing the same result are compared according to multiple cost metrics during pruning and plans that are not Pareto-optimal are discarded. The latter approach can deal with a broad range of cost metrics but does not support parameters. Other MQ algorithms are tailored to specific combinations of cost metrics and user preference functions that allow efficient pruning [21, 32, 2, 3]. They allow for instance only cost metrics for which the cost of a query plan is calculated as weighted sum over the cost of its sub-plans [32]; this is however not possible in many relevant scenarios (e.g., the execution time of a plan equals the maximum over the execution times of its sub-plans if they are executed in parallel). None of those approaches supports parameters. The algorithms that we present in this paper place only minimal restrictions on the cost metrics (see Section 5.2) and allow parameters which is required to solve MPQ problems. Yet another branch of MQ algorithms separate multi-objective optimization from join ordering; they produce for instance a time-optimal join tree first and configure operators within that tree considering multiple cost metrics later [15, 24]. Such approaches are not applicable to MPQ since it is unrealistic to find one join tree that is optimal for all parameter values (parameters such as predicate selectivities clearly have strong influence on the optimal join order). Algorithms for multi-objective data flow optimization [27, 28, 22] are not applicable to query optimization with join reordering.

4. PROBLEM ANALYSIS

We analyze the newly introduced MPQ problem. The PQ problem (i.e., the MPQ problem with only one cost metric) was already analyzed in prior work [13]. The MPQ problem is a generalization of the PQ problem and the following analysis therefore focuses on pointing out differences between the PQ problem and the MPQ problem. We will see in Section 4.1 that having multiple cost metrics instead of only one changes many fundamental problem properties. This has important implications on the design of MPQ algorithms that we discuss in Section 4.2.

4.1 Analysis

Most work on PQ assumes that cost functions are PWL [13, 17, 12]. We make the same assumption in the following. Our comparison between PQ and MPQ focuses on three problem properties that have been shown to hold for PQ. Those three problem properties were already called the *guiding principles of PQ* [12] since many PQ algorithms exploit them in one way or another [13, 17, 12], assuming that they hold either over the whole parameter space [13, 17] or at least locally [12]. We will see that the guiding principles do not hold anymore for MPQ which makes many successful approaches to PQ inapplicable to MPQ. Table 1 summarizes the differences between PQ and MPQ. The left column contains statements about PQ that were proven by Ganguly [13]; the right column contains the adapted statements for MPQ that are proven next. All statements refer to linear regions (convex polytopes in the parameter space in which all compared cost functions are linear for each cost metric).

THEOREM 1. *The parameter space can be partitioned into linear regions for an arbitrary set of cost functions.*

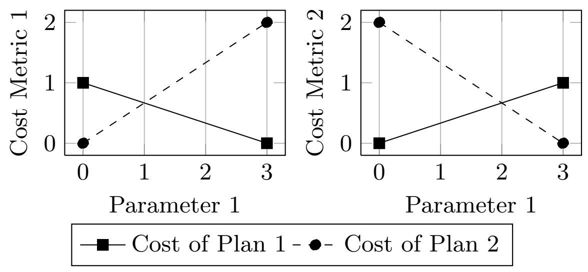
PROOF. Given only one cost metric, the parameter space can always be partitioned into linear regions according to results from PQ [17]. Denote by C_i the partitioning according to the i-th cost metric for $1 \leq i \leq M$ (represented as a set of polytopes). Then $\{c = c_1 \cap \ldots \cap c_M | c_i \in C_i\}$ is a partitioning of the parameter space into linear regions according to all cost metrics. The partitions are intersections of convex polytopes and therefore convex polytopes themselves. $\square$

We refer to the three statements about PQ by **S1**, **S2**, and **S3** in the following, and to the three statements about MPQ by **M1**, **M2**, and **M3**.

THEOREM 2. *Let p_1 and p_2 two arbitrary plans and $X \subseteq \mathbb{X}$ a linear region for $\{p_1, p_2\}$. Then the region D within X in which p_1 dominates p_2 forms a convex polytope.*

PROOF. Denote by $D_m \subseteq X$ the region in which p_1 is better or equivalent to p_2 according to cost metric $m \in \mathbb{M}$. Each region D_m forms a convex polytope (see results on PQ with linear cost functions [13]). Plan p_1 dominates p_2 in the region in which it is better or equivalent to p_2 according to all cost metrics. Region D corresponds therefore to the intersection of the D_m: $D = \cap_{m \in \mathbb{M}} D_m$. A convex polytope is an intersection of halfplanes. Therefore, the intersection of convex polytopes is a convex polytope again. $\square$

The following series of counter-examples proves the statements from Table 1. The multi-objective equivalent of an



Parameter Range	Pareto Plans
$[0, 1)$	Plan 1, Plan 2
$[1, 2)$	Plan 1
$[2, 3]$	Plan 1, Plan 2

Figure 4: If a plan is Pareto-optimal for two parameter values, it is not necessarily Pareto-optimal for the values in between.

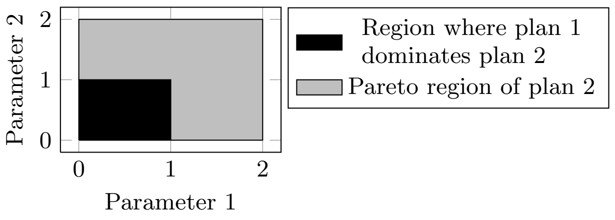


Figure 5: Pareto regions are not necessarily convex.

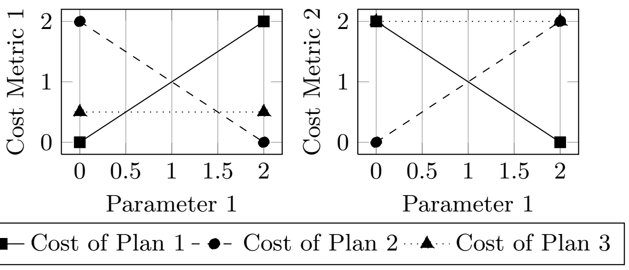
optimal plan is a Pareto-optimal plan. Statement **S1** about PQ does not generalize to the multi-objective case. Figure 4 shows a corresponding counter-example. The example shows the two-dimensional cost function of two plans within a one-dimensional parameter space. Plan 1 is Pareto-optimal in the whole parameter space (parameter value range $[0, 3]$). Plan 2 is however only Pareto-optimal for the parameter value ranges $[0, 1)$ and $[2, 3]$ but not for parameter values between 1 and 2. The example is minimal for MPQ since having less than two cost metrics leads to PQ and having less than one parameter leads to MQ. The negative result therefore applies to MPQ in general.

This example shows at the same time that Pareto regions are not necessarily connected (first part of **M2**). Figure 5 illustrates the second part of statement **M2**: the connected parts of the Pareto region are not necessarily convex. The example depicted in Figure 5 uses two plans and a two-dimensional parameter space. The example requires a two-dimensional parameter space since connected regions in a one-dimensional parameter space always form convex polytopes. Let $\mathbf{c}_1(x_1, x_2) = (x_1, x_2)$ be the two-dimensional cost function of plan 1 (the two-dimensional identity function) and $\mathbf{c}_2(x_1, x_2) = (1, 1)$ the cost function of plan 2. The region in which plan 1 dominates plan 2 forms a convex polytope as depicted in Figure 5. The remaining region is the Pareto region of plan 2. Figure 5 shows clearly that the Pareto region is not convex.

The example in Figure 4 also proves **M3a**. The example in Figure 6 proves **M3b**. Figure 6 shows cost functions of three plans for two cost metrics and one parameter. Plan 3 is Pareto-optimal for the parameter range $(0.5, 1.5)$ but nei-

Case of Single Cost Metric	**Case of Multiple Cost Metrics**
(S1) If the same plan is optimal for two points in a linear parameter space region, then that plan is also optimal on the line connecting those two points.	**(M1)** If the same plan is Pareto-optimal for two points in a linear parameter space region, then this plan is **not** necessarily Pareto-optimal on the line connecting those two points.
(S2) Each plan has one connected region within a linear parameter space region for which it is optimal. This region is either empty or forms a convex polytope.	**(M2)** The Pareto region of a plan within a linear region is **not** necessarily connected and the connected parts of it do **not** form convex polytopes in general.
(S3) If the same plan is optimal for all vertices of a convex polytope in a linear parameter space region, then that plan is optimal for all points within the polytope.	**(M3)** If all vertices of a convex polytope in a linear parameter space region have the same set of Pareto plans, then **(M3a)** those plans are **not** necessarily Pareto-optimal for all points of the polytope, and **(M3b)** plans can be Pareto-optimal within the polytope that are not Pareto-optimal on the vertices.

Table 1: Comparing the case of one cost metric and the case of multiple cost metrics in parametric query optimization; all statements refer to linear regions in the parameter space.



Parameter Range	**Pareto Plans**
$[0, 0.5]$	Plan 1, Plan 2
$(0.5, 1.5)$	Plan 1, Plan 2, Plan 3
$[1.5, 2]$	Plan 1, Plan 2

Figure 6: If a plan is not Pareto-optimal for two parameter values, it can still be Pareto-optimal for the values in between.

ther for the range $[0, 0.5]$ nor for the range $[1.5, 2]$. The cost functions in our examples are not monotone but the examples can be adapted (just turn the figures counterclockwise by 45 degrees). A common assumption in PQ is that plan cost functions are monotone in the parameters [4]. We see that this assumption does not change our negative results.

4.2 Implications on Algorithm Design

The three properties of the PQ problem that are listed in the left column of Table 1 have allowed to design PQ algorithms that split one PQ problem into several CQ problems. This approach has the advantage that an existing query optimizer for CQ can be turned into an optimizer for PQ with relatively low implementation overhead: the code of the existing CQ optimizer remains mostly unchanged (this is why such approaches to PQ are called *non-intrusive* [17]) and only a relatively small piece of code has to be added that splits the PQ problem into several CQ problems. We will see now, why such approaches fail for MPQ.

The Recursive Decomposition Algorithm proposed by Hulgeri and Sudarshan [17] is a non-intrusive PQ algorithm and works as follows: Given a convex polytope in the parameter space, the algorithm calculates an optimal plan for each vertex of that polytope (using a CQ query optimizer). If the same plan is optimal for each vertex, then that plan is optimal for every point within the polytope (according to statement **S3** from Table 1) and no further decomposition is necessary. If different plans are optimal for different vertices, then the polytope is decomposed into fragments and the algorithm is recursively applied to each fragment.

The described algorithm is representative for other non-intrusive approaches to PQ [13, 17, 18] since all of them successively decompose the parameter space into fragments in which only one plan is optimal. Statement **S3** is crucial for all those algorithms since it leads to a sufficient condition for checking whether further decomposition is unnecessary. Statement **M3** shows that no analogue condition can be found for MPQ: even if the same set of plans is Pareto-optimal for all vertices of a convex polytope in the parameter space, it may still be necessary to decompose that polytope further in order to find all Pareto plans (according to Statement **M3b**). This means that it is not possible to generalize non-intrusive algorithms for PQ to MPQ (which would allow to split one MPQ problem into several MQ problems to which existing MQ algorithms could be applied [14]). Motivated by this insight, we propose quite a different approach to MPQ in the following section.

5. GENERIC ALGORITHM

In this section, we present the Relevance Region Pruning Algorithm (RRPA) for MPQ. The algorithm associates each query plan with a RR in the parameter space that is used during pruning to detect irrelevant plans. The algorithm is generic and not specific to PWL cost functions. Section 5.1 describes the algorithm and Section 5.2 proves that RRPA finds complete PPSs for arbitrary MPQ problem instances. We do not explicitly describe how to deal with nested queries during optimization; techniques for decomposing complex SQL statements into simple SPJ query blocks have been proposed in prior work [26].

5.1 Outline of Algorithm

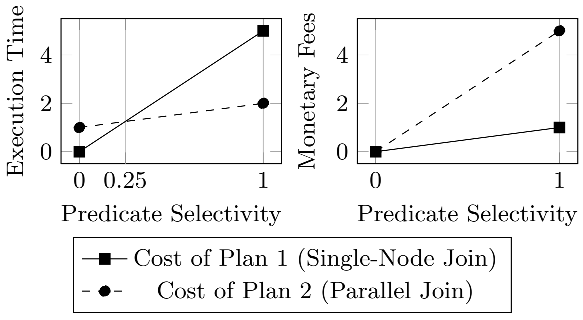
The analysis from the previous section has shown that trying to adapt non-intrusive PQ algorithms to MPQ is not a promising direction. We adopt a dynamic programming

(DP) based approach instead, calculating optimal plans for joining table sets out of optimal plans for joining subsets. Such an approach seems promising because DP has been widely used for designing algorithms in CQ [26], MQ [14], and PQ [17]. Algorithm 1 shows pseudo-code of RRPA. The main function takes a query Q as input and returns a PPS for Q. The algorithm uses two families of global variables: For each sub-query $q \subseteq Q$, variable $\mathcal{P}^q$ will eventually contain a PPS for q and variable $\mathcal{R}^q$ a corresponding RM (let $p \in \mathcal{P}^q$ a plan for q, then $\mathcal{R}^q(p)$ designates the RR of p). We assume that the plan sets are initially empty. RRPA first calculates PPSs and RMs for each base table $q \in Q$; it considers all possible scan plans for each base table and prunes out plans that are dominated in the entire parameter space. Details of the pruning function are discussed later. After the base tables, RRPA treats table sets in ascending order of cardinality. An auxiliary function generates the PPS for joining a table set $q \subseteq Q$ by considering all possible splits of q into two non-empty subsets (each split represents one specific pair of operands for the last join), all possible operators for the last join, and all pairs of plans for generating the inputs to the last join (those plans are selected out of the PPSs that were calculated before). A tentative plan is generated for every combination of operands, operator, and sub-plans. This plan is compared pairwise against all other plans that generate the same result and are already contained in $\mathcal{P}^q$. Those comparisons happen in the pruning function. The goal is to identify and discard suboptimal plans that are not required to form a PPS.

Pruning is based on the concept of RRs that was introduced in Section 2. Every plan is associated with a RR in the parameter space for which no alternative plan is known that has equivalent or dominant cost. The RR of a newly generated plan is initialized by the full parameter space. It is reduced during a series of comparisons between the newly generated plan and the old plans joining the same tables. At every comparison, the RR of the new plan is reduced by the points in the parameter space for which an old plan dominates the new plan. If the RR of the new plan becomes empty, it is discarded. Otherwise, the new plan is inserted. Before inserting the new plan, the RRs of the old plans are reduced by regions in which they are dominated by the new plan. Old plans with empty RRs are discarded. The following example illustrates the pruning method.

Example 3. *We revisit Scenario 1. Figure 7 shows the cost functions of two query plans that join the same two tables. The amount of data that needs to be joined depends linearly on the selectivity of one predicate; all cost functions therefore depend on this parameter. Plan 1 uses a single-node join while plan 2 uses a parallel join involving multiple nodes. Plan 1 executes faster than plan 2 for small amounts of input data since no data needs to be shuffled around in the network (assuming that all required input data resides initially on one node). Plan 2 executes faster for larger amounts of input data due to parallelization. The monetary fees of plan 2 are however always higher than the fees for plan 1, since the fees are proportional to the total work (summing up over different nodes) and the total amount of work increases by parallelization.*

Assume that plan 1 was generated before plan 2. The RR of plan 2 directly after its creation is the entire parameter space $[0, 1]$*. Plan 2 is pruned with all previously generated plans for joining the same tables, this is only plan 1 in our example. Plan 1 is preferable over plan 2 according to execution time and monetary fees at the same time as long as the selectivity is smaller than* 0.25*. The RR of plan 2 is therefore reduced by the interval* $[0, 0.25]$ *such that plan 2 remains relevant for the interval* $[0.25, 1]$*. Note that this example uses only linear cost functions that depend on only one parameter while* RRPA *can work with arbitrary cost functions that depend on an arbitrary number of parameters.*



Optimization Phase	Relevance Region of plan 2
After creating plan 2	$[0, 1]$
After pruning plan 2 with plan 1	$[0.25, 1]$

Figure 7: Illustration of pruning function

Algorithm 1 does not specify how elementary operations such as adding cost functions or intersecting relevance regions are implemented. The best way of implementing those operations depends on the considered class of cost functions (which also implicitly determines the class of RR shapes that one needs to consider). It is therefore impossible to specify an implementation for the generic case. For the same reason it is not possible to analyze the time complexity of RRPA. We will however present a specialized version of RRPA for PWL cost functions in Section 6 and analyze its complexity.

5.2 Proof of Completeness

We prove that RRPA generates complete PPSs for arbitrary input queries. We make the common assumption that the *Principle of Optimality* (POO) [14] holds for each cost metric: replacing a sub-plan p_S within a query plan p by an alternative sub-plan p'_S that has better or equivalent cost than p_S for a specific parameter vector $\mathbf{x}$ and according to a specific cost metric m, can only lead to a plan whose cost according to m is better than or equivalent to the one of p for $\mathbf{x}$. The POO restricts the cost function of a plan with regards to the cost functions of its sub-plans but it does not restrict the shapes of cost functions in general.

The proof that RRPA generates PPSs is an induction over the number of tables to join. The following lemma will be used for the inductive step.

Lemma 1. *If* RRPA *generates PPSs and corresponding RMs for all queries that join up to* N *tables then it also generates PPSs and corresponding RMs for queries that join up to* $N + 1$ *tables.*

Proof. Let Q be a query joining $N + 1$ tables ($|Q| = N + 1$), vector $\mathbf{x} \subseteq \mathbb{X}$ an arbitrary parameter vector, and

```
// Find a Pareto plan set for query $Q$
function GenericMPQ($Q$)
    // Initialize plan sets for base tables
    for $\langle q, p\rangle : q \in Q, p$ is plan for $q$ do
        Prune($\mathcal{P}, q, p$)
    end for
    // Consider table sets of increasing cardinality
    for $k \in 2..|Q|$ do
        // Iterate over table sets with given cardinality
        for $q \subseteq Q : |q| = k$ do
            $\mathcal{P}^q \leftarrow$GenerateParetoPlanSet($q$)
        end for
    end for
    return $\mathcal{P}^Q$
end function
// Generate Pareto plan set for joining $q$
function GenerateParetoPlanSet($q$)
    $\mathcal{P} \leftarrow \emptyset$
    // For all possible splits of table set $q$
    for $q_1, q_2 \subset q : q_1 \dot{\cup} q_2 = q$ do
        // For all sub-plans and operators
        for $p_1 \in \mathcal{P}^{q_1}, p_2 \in \mathcal{P}^{q_2}, o \in \mathbb{O}$ do
            // Construct new plan out of sub-plans
            $p_N \leftarrow$Combine($p_1, p_2, o$)
            // Accumulate cost of sub-plans
            $\mathbf{c}(p_N) =$AccumulateCost($o, p_1, p_2$)
            // Prune with new plan
            Prune($\mathcal{P}, q, p_N$)
        end for
    end for
    return $\mathcal{P}$
end function
// Prune plan set $\mathcal{P}$ for query $q$ with new plan $p_N$
procedure Prune($\mathcal{P}, q, p_N$)
    // Check whether the new plan is relevant
    $\mathcal{R}^q(p_N) \leftarrow \mathbb{X}$
    for $p \in \mathcal{P}^q$ do
        // Update relevance region of new plan
        $\mathcal{R}^q(p_N) \leftarrow \mathcal{R}^q(p_N) \setminus$Dom($p, p_N$)
        // Check if relevance region became empty
        if $\mathcal{R}^q(p_N) = \emptyset$ then
            return // Do not insert new plan
        end if
    end for
    // If we arrive here, the new plan will be inserted
    // Discard irrelevant old plans
    for $p \in \mathcal{P}^q$ do
        // Update relevance region of old plan
        $\mathcal{R}^q(p) \leftarrow \mathcal{R}^q(p) \setminus$Dom($p_N, p$)
        // Check if relevance region became empty
        if $\mathcal{R}^q(p) = \emptyset$ then
            $\mathcal{P}^q \leftarrow \mathcal{P}^q \setminus \{p\}$ // Discard old plan
        end if
    end for
    // Insert new plan into Pareto plan set
    $\mathcal{P}^q \leftarrow \mathcal{P}^q \cup \{p_N\}$
end procedure
```

Algorithm 1: The relevance region pruning algorithm for generic multi-objective parametric query optimization

p an arbitrary plan for Q. Plan p has two sub-plans, p_1 and p_2, that join at most N tables each. Therefore, RRPA generates a plan p_1^* that produces the same result as p_1 and dominates p_1 for $\mathbf{x}$. Additionally, $\mathbf{x}$ is included in the RR of p_1^*. RRPA also generates a plan p_2^* with the analogous properties relative to p_2. The plans p_1^* and p_2^* can be combined into a plan p^* that produces the same result as p and dominates p for $\mathbf{x}$ (due to the POO).

RRPA will generate p^* and initialize its RR with the full parameter space. Plan p^* is only pruned once its RR becomes empty during the pairwise comparisons with other plans. This can only happen, if RRPA keeps another plan that dominates p^* for $\mathbf{x}$ and $\mathbf{x}$ will be included in that plan's RR. RRPA generates a PPS for query Q and the corresponding RM since p and $\mathbf{x}$ were chosen arbitrarily. □

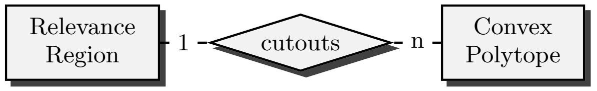


Figure 8: Representation of relevance regions if all cost functions are piecewise-linear

The following theorem is the main result of this subsection.

Theorem 3. RRPA *generates PPSs for arbitrary MPQ problem instances.*

Proof Sketch. The proof is an induction over the number of tables to join. Under the assumption that RRPA generates PPSs and corresponding RMs for single tables (the induction start), it also generates PPSs and corresponding RMs for arbitrary table sets according to Lemma 1 (the induction step). RRPA considers all possible plans for each base table and only discards plans that are dominated in the entire parameter space. This proves the induction start. □

6. ALGORITHM FOR PIECEWISE-LINEAR COST FUNCTIONS

RRPA presented in the last section is generic since it can deal with arbitrary cost functions. The pseudo-code of RRPA (Algorithm 1) left certain questions open such as how to represent RRs and how to efficiently intersect and reduce them; the answers to those questions depend on the considered class of cost functions. In this section, we present a specialized version of RRPA for PWL cost functions: PWL-RRPA. We propose data structures to represent cost functions and RRs in Section 6.1, show how elementary operations can be efficiently implemented on them in Section 6.2, and analyze the complexity of PWL-RRPA in Section 6.3. PWL-RRPA guarantees to generate PPSs for arbitrary PWL-MPQ problem instances as it is a specialization of RRPA.

6.1 Data Structures

Expressions of the form $\mathcal{R}^q(p)$ designate in Algorithm 1 the RR of a plan p joining a table set q. Figure 8 describes the internal representation of RRs as entity-relationship diagram. A RR is represented by a set of convex polytopes, called the cutouts, such that a parameter space vector is contained in a RR if it is not contained in any of the cutouts. The following theorem justifies this representation.

Theorem 4. *Any relevance region that occurs during the execution of* PWL-RRPA *can be represented as complement of a set of convex polytopes.*

Proof. The RR of a new plan is the entire parameter space and can therefore be represented as the complement of an empty set. After initialization, the RR can get reduced several times by regions in which a plan is dominated by another. When comparing two plans with PWL cost functions, the parameter space can be partitioned into linear regions according to Theorem 1. The region in which one plan dominates another within a linear region forms a

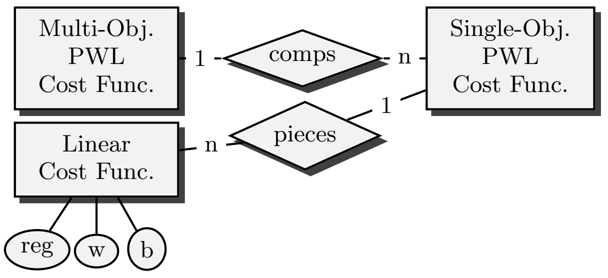


Figure 9: Representation of multi-objective piecewise-linear cost functions

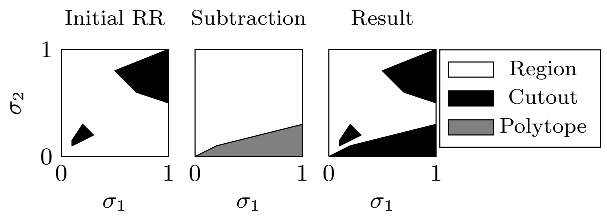


Figure 10: Polytopes are subtracted from a relevance region by adding them as cutouts.

convex polytope according to Theorem 2. Therefore, the RR can still be represented as complement of convex polytopes after reduction. □

The cost function of a plan p is represented by the expression $\mathbf{c}(p)$ in Algorithm 1. Figure 9 shows the internal representation of cost functions as entity-relationship diagram. A multi-objective PWL cost function is composed out of one single-objective PWL cost function per cost metric. The PWL cost function is linear within parameter space regions that form convex polytopes. Each PWL function is therefore represented as a set of linear functions; each linear function is characterized by the parameter space region to which it applies (attribute *reg* in Figure 9) and a weight vector (attribute $\mathbf{w}$ in Figure 9) with one weight per parameter together with the scalar base cost (b in Figure 9) that define the linear function. The parameter space regions of the linear pieces must not overlap; then the PWL function can be evaluated for a specific parameter vector $\mathbf{x}$ by identifying the unique piece whose region contains $\mathbf{x}$ and evaluating the formula $b + \mathbf{w}^T \cdot \mathbf{x}$ to obtain the cost value. A multi-objective PWL function is evaluated by evaluating all its components according to the aforementioned method.

PWL cost functions can approximate the real cost functions of single scan and join operations up to an arbitrary precision [17]. The accumulated cost of an entire query plan (using standard accumulation function such as minimum, maximum, and weighted sum) can therefore be represented as PWL function again; this fact has been used by prior PQ algorithms [17]. Generalizing this reasoning to the multi-objective case is trivial. Therefore, the representation proposed in Figure 9 covers each cost function that occurs during the execution of PWL-RRPA (assuming that the cost of single operations is approximated by PWL functions).

6.2 Implementation of Elementary Operations

PWL-RRPA performs two operations on RRs: it reduces the RR of a plan by the region in which it is dominated by another (e.g., Algorithm 1, Line 39) and checks whether a RR is empty (Algorithm 1, Line 41). Algorithm 2 shows pseudo-code for both operations. The field specifier *.cutouts* refers to Figure 8 and denotes the set of cutouts for a variable representing a RR. Convex polytopes are subtracted from a RR by adding them as cutouts, as illustrated in Figure 10.

```
 1: // Input: relevance region $rr$, convex polytopes $polys$
 2: // Effect: region $rr$ is reduced by $polys$
 3: procedure SubtractPolys($rr, polys$)
 4:     // Add polytopes to cutouts
 5:     $rr.cutouts \leftarrow rr.cutout \cup polys$
 6: end procedure
 7: // Input: relevance region $rr$
 8: // Output: true iff $rr$ is empty
 9: function IsEmpty($rr$)
10:     // Check whether union of cutouts is convex
11:     if $(\cup_{C \in rr.cutouts} C)$ is convex then
12:         // Calculate convex polytope covered by cutouts
13:         $CutPoly \leftarrow$ polytope formed by $(\cup_{C \in rr.cutouts} C)$
14:         // Check if cutouts cover whole parameter space
15:         if $\mathbb{X} \subseteq CutPoly$ then
16:             // Relevance region is empty
17:             return true
18:         end if
19:     end if
20:     // Cutouts do not cover whole parameter space
21:     return false
22: end function
```

Algorithm 2: Elementary operations on relevance regions

Function IsEmpty is based on the following theorem.

Theorem 5. *A relevance region is empty iff the union of its cutouts forms a convex polytope that covers the entire parameter space.*

Proof. Let $C_i \subseteq \mathbb{X}$ be the set of cutouts. The RR is empty iff $\forall \mathbf{x} \in \mathbb{X} \, \exists i : \mathbf{x} \in C_i$. This is the case iff $\mathbb{X} \subseteq \cup_i C_i$ which is equivalent to $\mathbb{X} = \cup_i C_i$ since all cutouts are contained within the parameter space $\mathbb{X}$. As $\mathbb{X}$ forms a convex polytope according to the definition of the PWL-MPQ problem (see Section 2), the union of the cutouts of an empty RR is a convex polytope. □

The union of the cutouts may not be convex and may not form a polytope. Checking whether a region of arbitrary shape (the union of the cutouts) contains the parameter space is inefficient. It is therefore crucial to note that the containment check is only necessary in the special case that the union of cutouts forms a convex polytope. The algorithm by Bemporad et al. [6] checks whether a union of convex polytopes is a convex polytope again and constructs the corresponding polytope in that case. Checking containment between two convex polytopes is a standard problem [20].

PWL-RRPA performs two operations on cost functions: It calculates the cost function of a new plan by accumulating the cost of its sub-plans (Algorithm 1, Line 26) and—given two cost functions—it calculates the region in which one dominates the other (e.g, Algorithm 1, Line 39). Algorithm 3 shows pseudo-code for both operations. The *comps* relationship (see Figure 9) associates a multi-objective cost function with one single-objective function for each cost metric. We treat the *comps* relationship as an array and refer to the single-objective cost function for metric m by the notation $.[m]$. The function AccumulateCost accumulates the cost of a new plan out of the cost of its sub-plans. It

```
// Input: a join operator $o$ and two plans $p_1$ and $p_2$
// Output: accumulated cost of executing $p_1$ and $p_2$
//     and joining their results using $o$
function AccumulateCost($o, p_1, p_2$)
    // Create new cost function
    $acCost \leftarrow$ new multi-obj. PWL cost func.
    // Iterate over all cost metrics
    for $m \in \mathbb{M}$ do
        // Initialize pieces of new cost function
        $newPcs \leftarrow \emptyset$
        // Iterate over cost function pieces of sub-plans
        for $fp_1 \in \mathbf{c}(p_1).comps[m].pieces$ do
            for $fp_2 \in \mathbf{c}(p_2).comps[m].pieces$ do
                // Intersect regions of the two pieces
                $r \leftarrow fp_1.reg \cap fp_2.reg$
                // Check if intersection is empty
                if $r \neq \emptyset$ then
                    // Add weight vectors
                    $\mathbf{w} \leftarrow fp_1.\mathbf{w} + fp_2.\mathbf{w} + o.\mathbf{w}$
                    // Add base costs
                    $b \leftarrow fp_1.b + fp_2.b + o.b$
                    // Construct new piece
                    $newPc \leftarrow$ new linear cost func. with
                      base cost $b$, weight $\mathbf{w}$, and region $r$
                    // Add new piece
                    $newPcs \leftarrow newPcs \cup \{newPc\}$
                end if
            end for
        end for
        $acCost.comps[m].pieces \leftarrow newPcs$
    end for
    return $acCost$
end function
// Input: two plans $p_1$ and $p_2$
// Output: a set of convex polytopes in the
//     parameter space where $p_1$ dominates $p_2$
function Dom($p_1, p_2$)
    // Calculate $p_1$'s dominant region for each metric
    for $m \in \mathbb{M}$ do
        // Initialize set of polytopes
        $polys_m \leftarrow \emptyset$
        // For all pairs of cost function pieces
        for $fp_1 \in \mathbf{c}(p_1).comps[m].pieces$ do
            for $fp_2 \in \mathbf{c}(p_2).comps[m].pieces$ do
                // Calculate intersection of regions
                $r \leftarrow fp_1.reg \cap fp_2.reg$
                // Calculate part where $p_1$ dominates $p_2$
                $rDom \leftarrow$ solutions to linear equations
                  $(fp_1.\mathbf{w} - fp_2.\mathbf{w})^T \mathbf{x} \leq fp_2.b - fp_1.b,\ \mathbf{x} \in r$
                // Add polytope if not empty
                if $rDom \neq \emptyset$ then
                    $polys_m \leftarrow polys_m \cup \{rDom\}$
                end if
            end for
        end for
    end for
    // Combine results from different metrics
    return $\{\cap_{m \in \mathbb{M}} p_m | p_m \in polys_m\}$
end function
```

Algorithm 3: Elementary operations on cost functions

iterates over all cost metrics and calculates the cost function for each metric separately. For each metric, it partitions the parameter space into regions in which both sub-plans have linear cost functions. Each nonempty linear region becomes a piece in the cost function of the new plan. The weight vector of the new piece corresponds to the component-wise sum of the weight vectors of the two sub-plans and the join cost vector (denoted by $o.\mathbf{w}$ in the pseudo-code) in the corresponding parameter space region[1]; Figure 11 illustrates this step for a two-dimensional parameter space with parameters σ_1 and σ_2, the two-dimensional weight vectors are shown at the interior of their linear regions. The base cost of the new piece is the sum over the join base cost ($o.b$) and the base costs of the sub-plans. Cost is therefore accumulated by adding the cost of the sub-plans. The function trivially generalizes to scenarios where cost is accumulated as weighted sum, minimum, or maximum of two cost functions.

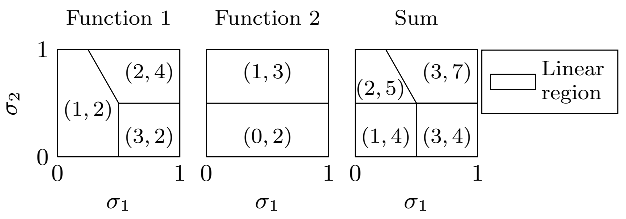


Figure 11: To add single-objective cost functions, their weight vectors are added in each linear region.

Function Dom returns a set of convex polytopes representing the region in which plan p_1 dominates plan p_2. A plan dominates another in regions in which it has better or equivalent cost according to each cost metric. Function Dom initially calculates for each cost metric m the set $DomPolys_m$ of convex polytopes in the parameter space in which p_1 is better than or equivalent to p_2 according to m. In a second step, the function intersects the polytope sets associated with specific cost metrics to obtain the region in which p_1 is better or equivalent according to all metrics.

We describe several refinements of the pseudo-code presented so far; those refinements led to significant performance improvements in our experiments. First, we simplify the internal representation of convex polytopes wherever possible by deleting redundant linear constraints. A linear constraint is redundant if it is implied by the other constraints of the same polytope. Second, we simplify the internal representation of RRs by deleting redundant cutouts. A cutout is redundant if it is covered by the other cutouts of the same RR. Third, we avoid unnecessary emptiness checks by associating each freshly created RR with a set of relevance points that are distributed across the entire parameter space. Each time that a new cutout is added to the RR, the relevance points that are contained within the new cutout are deleted from the point list. As long as the point list is not empty, the RR itself cannot be empty either and executing function IsEmpty can be avoided.

6.3 Complexity Analysis

The complexity of PQ, MQ, and MPQ algorithms depends heavily on the number of plans that are stored per table set. Prior work analyzing the complexity of PQ and MQ algorithms often considers the number of plans as random variable and derives upper bounds on its expected value [14,

[1]To simplify the pseudo-code, we made the strong assumption that the cost function of the final join is always linear in parameter space regions in which the cost functions of the two sub-plans are linear. This is not true in general but the code can easily be generalized by first accumulating the cost of the sub-plans, and then accumulating the resulting cost and the join cost in a second step.

13]. We adopt the same approach for analyzing the complexity of PWL-RRPA. We focus on the case of linear cost functions; the analysis can easily be generalized to PWL cost functions for a given number of pieces. Let n_X be the number of parameters. The linear cost function of a plan p can be described by a set of real-valued weights $w^p_{m,i} \in \mathbb{R}$ for $m \in \mathbb{M}$ and $i \in \{0, \ldots, n_X\}$. The cost of p according to metric m is given by the expression $w^p_{m,0} + \sum_{1 \leq i} w^p_{m,i} x_i$ where x_i designates the value of the i-th parameter. We say that a plan p_1 dominates a plan p_2 *parameter value independently* (p.v.i.) if $w^{p1}_{m,i} \leq w^{p2}_{m,i}$ for each metric m and for each $i \in \{0, \ldots, n_X\}$. If p_1 dominates p_2 p.v.i. then p_1 dominates p_2 (according to the definition in Section 2) for all possible (positive) parameter values. Given a concrete parameter space, a plan p_1 dominating another plan p_2 p.v.i. is a sufficient (but not a necessary) condition for p_1 dominating p_2 in the entire parameter space. We now derive an upper bound on the expected number of Pareto plans assuming that plan cost weights are chosen randomly; we assume that weights of different plans and different weights for the same plan are chosen independently. All those assumptions are common in the complexity analysis of PQ and MQ algorithms [14, 13]. By $n_M = |\mathbb{M}|$, we designate the number of cost metrics.

THEOREM 6. *The expected number of Pareto plans per table set is upper-bounded by* $2^{((n_X+1)\cdot n_M)}$.

PROOF SKETCH. The cost function of a plan is described by $(n_X + 1) \cdot n_M$ cost weights. Hence, a cost function can be thought of as a point in $(n_X + 1) \cdot n_M$-dimensional space. Ganguly et al. [14] derive an upper bound of 2^l on the size of the cover set when choosing an unspecified number of points in l-dimensional space (see Theorem 3 in their publication). Setting $l = (n_X + 1) \cdot n_M$, we can use that result to obtain an upper bound on the number of plans that are not dominated p.v.i. by any other plan. This is an upper bound on the number of plans that PWL-RRPA is expected to retain for any given table set after pruning (the bound is pessimistic since a plan that is not dominated p.v.i. may still be dominated in the entire concrete parameter space). □

The upper bound derived in Theorem 6 is consistent with prior results in the areas of PQ and MQ: the upper bound of 2^{n_M} on the expected number of plans derived for the case of n_M cost metrics and no parameters (MQ) [14] corresponds to a specialization of our result. Our bound grows exponentially in the number of parameters which is in line with prior results on PQ [18] (tighter bounds require additional assumptions [13]). We denote our bound on the number of plans per table set by n_P in the following, the number of scan and join operators by $n_O = |\mathbb{O}|$, and the number of tables by $n_Q = |Q|$. The function $\mathbf{lp}(a, b)$ represents the time for solving a linear program with matrix dimensions $a \times b$. An upper bound on the number of plans that PWL-RRPA generates per table set is given by $n_G = 2^{n_Q} n_P^2 n_O$.

LEMMA 2. *Function* ISEMPTY *has time complexity* $O(n_M^{n_G} \mathbf{lp}(n_M n_G, n_X))$.

PROOF. A cutout is a region in which one plan dominates another; a cutout is therefore defined by n_M linear constraints. Comparing one plan to another one during pruning adds at most one cutout to its RR. The total number of cutouts per RR is therefore bounded by n_G. The time complexity of ISEMPTY is dominated by the time for checking whether the union of polytopes is convex; Bemporad et al. [6] provide complexity results for their algorithm, we use them with n_G as bound on the number of polytopes and n_M as bound on the number of constraints per polytope. □

We denote the time complexity of ISEMPTY by T_{emp}.

THEOREM 7. PWL-RRPA *has time complexity* $O(3^{n_Q} n_P^3 n_O T_{emp})$.

PROOF. The time for emptiness checks dominates. Each newly generated plan is compared against $O(n_P)$ alternative plans which requires $O(n_P)$ emptiness checks. PWL-RRPA iterates over all subsets of Q. For a subset $q \subseteq Q$ containing $i = |q|$ tables, PWL-RRPA generates $O(2^i n_P^2 n_O)$ plans. Using $\sum_{i=1}^{n_Q} \binom{n_Q}{i} 2^i = 3^{n_Q}$ yields the total complexity. □

7. EXPERIMENTAL EVALUATION

We experimentally evaluate PWL-RRPA in a scenario similar to Scenario 1. We first describe the experimental setup, then present the results, and finally discuss them.

Experimental Setup. We consider a Cloud scenario with *two cost metrics*, execution time and monetary fees. A parallel hash join and a single-node hash join are available. The parallel hash join requires to shuffle the input data in the network. Parallelization therefore increases the total amount of work (which is proportional to monetary cost) while it can decrease execution time in comparison to a single-node join if the input relations are sufficiently large. This shows that a tradeoff exists between execution time and monetary fees and a query plan that minimizes one does not necessarily minimize the other. Base tables are associated with equality predicates whose selectivites are represented by parameters; one parameter is required for each table with a predicate. Indices are available for each column with a predicate. This makes an index seek preferable for low selectivity while a complete table scan is better for non-selective predicates; as predicate selectivity is a parameter, plans must often be kept for both cases which makes the benchmark even more challenging. We evaluate the performance of PWL-RRPA on randomly generated queries, using the generation method proposed by Steinbrunn [29] (and used recently in other publications [8]) to choose table cardinalities and join predicates; we assume that unique values occupy up to 10% of a table column. We separately evaluate the performance for star queries and for chain queries as the structure of the join graph is known to have significant impact on optimizer performance [29]. PWL-RRPA considers the full search space of bushy query plans but postpones Cartesian product joins as much as possible; this heuristic is commonly applied in state-of-the-art optimizers such as the Postgres optimizer[2]. Standard formulas are used to estimate join time; monetary cost are calculated according to the pricing system of Amazon EC2[3] and the properties of the simulated cluster nodes such as main memory size correspond to the ones of the general purpose medium instance in EC2. PWL-RRPA was implemented in Java 1.7, using Gurobi 5.6[4] as linear program solver. All experiments were executed on a commodity iMac equipped with an i5-3470S processor with 2.9 GhZ and 16 GB of RAM.

Experimental Results. The goal of the following experiments is to show how optimization time depends on

[2]http://www.postgresql.org/
[3]http://aws.amazon.com/de/ec2/
[4]http://www.gurobi.com/

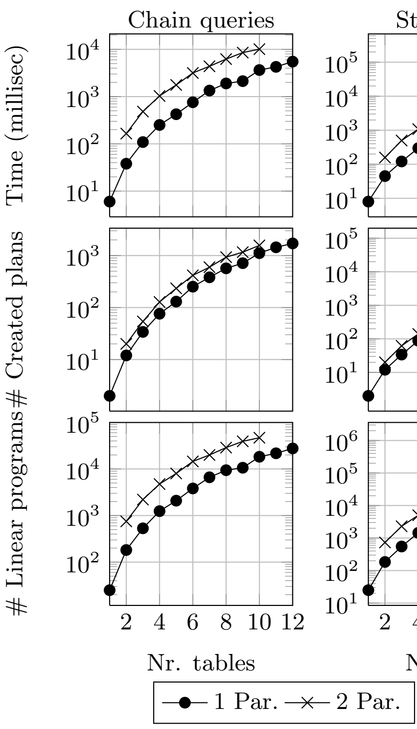


Figure 12: Optimization time, number of generated plans, and number of solved linear programs

query characteristics such as the number of tables, the number of parameters, and the join graph structure. We experimented with up to 12 tables for one parameter and up to 10 tables for two parameters. Figure 12 shows optimization time, the number of generated plans (including partial plans and plans that were pruned during optimization), and the number of solved linear programs (LPs). Each data point corresponds to the median of 25 randomly generated test cases. All three metrics are clearly correlated and increase in the number of tables as well as in the number of parameters. The number of solved LPs is much higher than the number of generated plans since operations such as comparing plans during pruning or checking emptiness of a plan's RR all require to solve several LPs. As in traditional query optimization, optimizing chain queries is faster than optimizing star queries when avoiding Cartesian product joins [23].

Discussion. MPQ is a generalization of MQ and PQ and computationally expensive. MPQ happens however before run time and it pays off as it avoids run time query optimization altogether. For same number of tables and parameters, our optimization times are higher but still comparable to optimization times of single-objective PQ algorithms that are often in the order of several seconds as well [18, 7].

8. CONCLUSION

We introduced MPQ, a novel variant of query optimization that allows to consider multiple cost metrics and parameters. We presented a first algorithm for this problem and evaluated it in a Cloud computing scenario. Our algorithm is exhaustive and guarantees to generate all relevant query plans.

9. REFERENCES

[1] M. Abhirama, S. Bhaumik, and A. Dey. On the stability of plan costs and the costs of plan stability. *PVLDB*, 2010.

[2] Z. Abul-Basher, Y. Feng, and P. Godfrey. Alternative Query Optimization for Workload Management. In *Database and Expert Systems Applications*, 2012.

[3] S. Agarwal, A. Iyer, and A. Panda. Blink and It's Done: Interactive Queries on Very Large Data. *PVLDB*, 2012.

[4] B. Babcock and S. Chaudhuri. Towards a Robust Query Optimizer: a Principled and Practical Approach. *SIGMOD Conf.*, 2005.

[5] S. Babu, P. Bizarro, and D. DeWitt. Proactive Re-Optimization. *SIGMOD Conf.*, 2005.

[6] A. Bemporad, K. Fukuda, and F. Torrisi. Convexity Recognition of the Union of Polyhedra. *Computational Geometry*, 2001.

[7] P. Bizarro, N. Bruno, and D. DeWitt. Progressive Parametric Query Optimization. *Trans. on KDE*, Apr. 2009.

[8] N. Bruno. Polynomial heuristics for query optimization. In *ICDE*, pages 589–600, 2010.

[9] N. Bruno and R. V. Nehme. Configuration-Parametric Query Optimization for Physical Design Tuning. *SIGMOD Conf.*, 2008.

[10] F. Chu, J. Halpern, and J. Gehrke. Least Expected Cost Query Optimization: What can we Expect? *SIGMOD Conf.*, 2002.

[11] R. Cole and G. Graefe. *Optimization of dynamic query evaluation plans.* 1994.

[12] A. Dey, S. Bhaumik, and J. Haritsa. Efficiently Approximating Query Optimizer Plan Diagrams. *PVLDB*, 2008.

[13] S. Ganguly. Design and Analysis of Parametric Query Optimization Algorithms. *PVLDB*, 1998.

[14] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query Optimization for Parallel Execution. In *SIGMOD Conf.*, 1992.

[15] M. Garofalakis and Y. Ioannidis. Multi-Dimensional Resource Scheduling for Parallel Queries. In *SIGMOD Conf.*, 1996.

[16] G. Graefe and K. Ward. Dynamic Query Evaluation Plans. *SIGMOD Conf.*, 1989.

[17] A. Hulgeri and S. Sudarshan. Parametric Query Optimization for Linear and Piecewise Linear Cost Functions. *PVLDB*, 2002.

[18] A. Hulgeri and S. Sudarshan. AniPQO: Almost Non-Intrusive Parametric Query Optimization for Nonlinear Cost Functions. *PVLDB*, 2003.

[19] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric Query Optimization. *PVLDB*, May 1997.

[20] V. Kaibel and M. Pfetsch. Some algorithmic problems in polytope theory. *Algebra, Geometry and Software Systems*, 1, 2003.

[21] S. Kambhampati, U. Nambiar, Z. Nie, and S. Vaddi. Havasu: A Multi-Objective, Adaptive Query Processing Framework for Web Data Integration. *ASU CSE*, 2002.

[22] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. E. Ioannidis. Schedule Optimization for Data Processing Flows on the Cloud. In *SIGMOD Conf.*, 2011.

[23] K. Ono and G. Lohman. Measuring the complexity of join enumeration in query optimization. *PVLDB*, pages 314–325, 1990.

[24] C. Papadimitriou and M. Yannakakis. Multiobjective Query Optimization. *PODS*, 2001.

[25] N. Reddy and JR. Haritsa. Analyzing plan diagrams of database query optimizers. *PVLDB*, 2005.

[26] 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 *SIGMOD Conf.*, pages 23–34, 1979.

[27] A. Simitsis, P. Vassiliadis, and T. Sellis. State-Space Optimization of ETL Workflows. *Trans. on KDE*, Oct. 2005.

[28] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing Analytic Data Flows for Multiple Execution Engines. *SIGMOD Conf.*, 2012.

[29] M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB J.*, 6(3):191–208, 1997.

[30] TPC. TPC-H Benchmark, 2013.

[31] I. Trummer and C. Koch. Approximation Schemes for Many-Objective Query Optimization. *SIGMOD Conf.*, 2014.

[32] Z. Xu, Y. C. Tu, and X. Wang. PET: Reducing Database Energy Cost via Query Optimization. *PVLDB*, 2012.