Handling Estimation Inaccuracy in Query Optimization

Chiraz Soussi Moumen, Franck Morvan, and Abdelkader Hameurlain

IRIT Laboratory, Paul Sabatier University,
118 Route de Narbonne, F-31062 TOULOUSE CEDEX 9, FRANCE
{moumen, morvan, hameurlain}@irit.fr

Abstract. Cost-Based Optimizers choose query execution plans using a cost model. The latter relies on the accuracy of estimated statistics. Unfortunately, compile-time estimates often differ significantly from run-time values, leading to a suboptimal plan choices. In this paper, we propose a compile-time strategy, wherein the optimization process is fully aware of the estimation inaccuracy. This is ensured by the use of intervals of estimates rather than single-point estimates of error-prone parameters. These intervals serve to identify plans that provide stable performance in several run-time conditions, so called robust. Our strategy relies on a probabilistic approach to decide which plan to choose to start the execution. Our experiments show that our proposal allows a considerable improvement of the ability of a query optimizer to produce a robust execution plan in case of large estimation errors. The produced plan is also competitive with those obtained using existing optimization methods when errors are small.

Keywords: Query Optimization · Robust Plans · Estimation Errors

1 Introduction

In database query processing, query optimization constitutes a critical stage due to its impact on query processing performance. During this stage, a query optimizer[1] enumerates several execution plans for a query, estimates the cost of each plan using a cost model and chooses the plan with the lowest estimated cost to execute the query [24]. This plan is called *best plan* [17, 19, 24]. Best plan selection requires accurate estimates of the costs of alternative plans. Unfortunately, these estimates are often significantly erroneous with respect to values encountered at run-time. Such errors, which may be in orders of magnitude [16], arise due to a variety of reasons [8]. One of the main reasons is the imprecision of information about data (e.g. sizes of temporary relations) as well as the use of outdated statistics. Indeed, values of parameters which are important for costing possible query plans may vary unpredictably between

[1] We focus our study on the Cost-Based Query Optimizers. In the rest of this paper, we will use the terms *query optimizer* and *cost-based query optimizer* interchangeably

compile-time and run-time (e.g. predicate selectivity, available memory amount). Outdated statistics cause the optimizer to inaccurately estimate the costs of alternative plans. Others sources of estimation errors are changes in data and/or system conditions (e.g. indexes are created or destroyed, memory contention) and the use of invalid assumptions like *Attribute Value Independence*, *Defaults Values*, etc. The use of these assumptions rarely satisfies the real nature of data (e.g. skewed and correlated data), leading to large estimation errors.

An adverse effect of estimation errors is to lead the optimizer to a sub-optimal plan choice, resulting in inflated response time. A considerable body of literature was dedicated to find solutions to this problem. These solutions include mainly: i) techniques for better quality of the statistical metadata (e.g. [7,12,14,22,26,27]), ii) run-time techniques (e.g. [5,17–20]) to monitor a query execution and trigger re-optimization of the plan when a sub-optimality is detected, and iii) compile-time (i.e. optimization-time) strategies (e.g. [1–3,9,13]) that permit the optimizer to generate an execution plan, taking into consideration the imprecision of used estimates.

Accurate cost estimates remain, though, very difficult as this requires a detailed and a prior knowledge about data (e.g. sizes of temporary relations, predicate selectivity) and run-time characteristics such as resource availability (e.g. memory amount, system load). Run-time techniques are able to adapt an execution plan to changes in run-time conditions with a risk of an important cost-increase resulting from many possible adaptations of the plan. Compile-time strategies proposed to date suppose that it is often possible to find a single execution plan whose performance remains stable regardless of changes in the run-time conditions compared to the compile-time expectations of the run-time conditions. This assumption is not always valid. In some situations, especially in case of large uncertainty about run-time characteristics, find such a plan becomes hard to achieve. In this paper, we focus on this issue. We propose a compile-time strategy for identifying plans, each of which provides stable performance over a range of possible run-time values of error-prone parameters. Throughout this paper, these plans are said *robust*.

Our proposed optimization strategy uses an interval of estimates for each uncertain parameter, rather than specifying estimates for single points. Such an interval fully quantifies estimation inaccuracy and is then used to generate robust plans. Note that a plan providing stable performance over the whole interval is a single robust plan. Otherwise, the interval is divided into sub-intervals so as to find execution plans, each of which is associated with the sub-interval within which the plan is robust. Our method relies on a probabilistic approach to decide which plan to choose to start the execution. The major contribution of our proposal is to maximize the ability of a query optimizer to produce a robust execution plan in the presence of large estimation errors.

The rest of this paper is organized as follows: Section 2 presents the overall problem background and motivations. Section 3 details our contribution. The experimental framework and evaluation results are highlighted in Section 4. Related work is overviewed in Section 5. We conclude and outline our future work in Section 6.

2 Context and Problem Position

In this section, we first present the problem background. Then, we point out the motivations for our work.

2.1 Problem Background

Motivated by the difficulty of providing precise estimates needed for costing query plans as well as the complexity of cost models, researchers focused their efforts on introducing new optimization algorithms. Their aim is to avoid significant performance regression caused by the use of erroneous estimates. Researches on this purpose fall into two main approaches [25]. A first approach, called *Single Point-Based Optimization* (e.g. [5, 17, 18, 20]) consists in monitoring a plan execution so as to detect estimation errors and a resulting sub-optimalty. This latter is corrected by interrupting the current execution and re-optimizing the remainder of the plan using up-to-date statistics. At each invocation, the optimizer considers the used estimates as though they were completely precise and accurate. It uses specific estimate for each parameter. The generated plan is thus the *best plan* for specific run-time conditions. The ability of methods relying on this approach to accurately collect statistics is limited [3]. Consequently, when re-optimizing, the optimizer may use new erroneous estimates. This may result in several plan re-optimization and so performance regression, i.e. large cost-increase.
While *Single Point-Based Optimization* uses exclusively single points for estimates ignoring possible estimation errors, a conceptually different approach called *Multi Point-Based Optimization* was introduced. Besides to providing a solution to estimation errors, this approach avoids performance regression due to several re-optimizations of a plan. Furthermore, contrary to the first approach which reacts after an estimation error is detected, *Multi Point-Based Optimization* aims to predict plan sub-optimalities and anticipate the reaction to this. The methods proposed as part of this approach consider the possibility of estimation errors at the optimization phase. They produce plans that are likely to perform reasonably well in many run-time conditions. These methods (e.g. [1–3, 9, 11]) aim at *preventing* rather than *correcting*. The key concept of these methods is the use of intervals of estimates, which models the estimation inaccuracy. In the literature, there are different techniques for computing such intervals. For instance, [6] uses strict upper and lower bounds, [3, 17] model estimation uncertainty by means of discrete buckets that depend on the way the estimate was derived, etc. Once computed, the interval of estimates is used by the optimizer to generate an execution plan that is likely to provide stable performance within this interval.

2.2 Motivations

Existing methods proposed as part of the *Multi Point-Based Optimization* approach suppose that it is always feasible to find a single robust plan within an interval of estimates. We believe that this assumption is not always valid. When the interval estimates is large, it becomes difficult to find a plan generating stable performance over the whole interval. Execution plans may be robust at only some points of the interval. In the remainder of this paper, we adopt the following definition for a robust plan : *let V_e be an estimate of an error-prone parameter, let I be an interval of estimates around V_e exhibiting the uncertainty about the estimate of this parameter. Let P_{best} be the best plan for a specific value V_i in I. Finally, let λ be a (user-defined) cost-increase threshold (expressed in percentage). A plan P_{alt} is robust with respect to estimation errors if :*

$$\forall\ V_i \in I, \frac{cost(P_{alt})}{cost(P_{best})} \leq 1 + \frac{\lambda}{100} \tag{1}$$

For instance, if users tolerate a minor cost increase (λ) of at most 20%, the cost of P_{alt} is at most 1,2 times the cost of the best plan.

3 Generation of a Robust Query Execution Plan

In this section, we detail our proposal called **Identification of Robust Plans (IRP)**, which is part of the *Multi Point-Based Optimization* approach. The key concept in our research is the uncertainty of estimates (especially the sizes of temporary relations) used by a query optimizer to choose an appropriate execution plan along with the difficulty to find a single robust execution plan when the estimation uncertainty is large, i.e. the interval of estimates is large. We present below an example that underlines our motivations before detailing our contribution.

3.1 A Motivating Example

We introduce first the case wherein the size of only one input-relation for a join operation is considered as error-prone. then, we extend it to the general case.
Example. Consider the query Q = *select * from customer as C, order as O where C.customerId = O.customerId and C.country='France' and C.city='Paris'*. Best plan selection for this query requires accurate estimate of selectivity of the predicates *C.country='France'* and *C.city='Paris'*. However, even with the existence of histograms on attributes *C.country* and *C.city*, the optimizer can not maintain multi-dimensional histograms for all possible combinations of attributes [23]. It may so assume independence to compute the joint selectivity of the two predicates. This assumption may lead to large estimation errors while costing plans due to an eventual correlation between the predicates. To avoid sub-optimal plan choice, resulting from these errors, we consider $\sigma(C)$, which is the result of *C.country='France' and C.city='Paris'*, as error-prone. We define

an interval of estimates around $\sigma(C)$. This interval is then used to select an appropriate execution plan. In this process, one of following cases may occur:

- There is a single plan that is robust at all points of I.
- There are several plans, each of which is robust at only some points of I.

Existing multi point-based optimization methods assume that the first case is often feasible, that is to say a plan providing stable performance within an interval of estimates can be founded. This assumption is not always valid. Consider the scenario of **Example**, suppose that there is an index on an attribute of the relation O. Through repeated invocations of a query optimizer, plans labelled "Plan1", "Plan2" and "Plan3" (Cf. Figure 1) are enumerated as possible execution plans for Q.

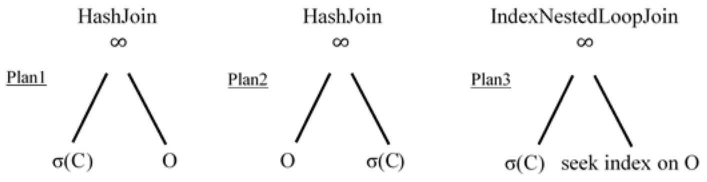


Figure 1. Possible execution plans for the query Q = σ(C) ⋈ O

Based on a cost model, a query optimizer estimates the costs of these plans with respect of variation of $\sigma(C)$ so as to determine the execution plan that offers robust performance whithin the interval of estimates. This plan must respect robustness definition presented in Section 1. Assume that the tolerated cost-increase (λ) is about 20%. This value is chosen based on the works of [1] and [3].

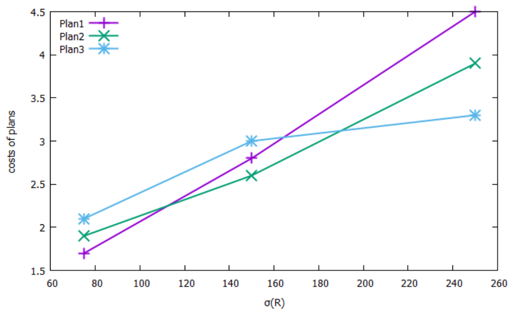


Figure 2. Costs of plans with respect to variations of $\sigma(C)$

Figure 2 plots the costs of the plans with respect to variations of $\sigma(C)$. The x-axis represents the values in the interval estimates I = [75, 250]. The latter models the size of $\sigma(C)$ in MB. In this Figure, we observe that for λ = 20%, none of plans P1, P2, and P3 is robust within I. This example highlights the necessity to divide the interval estimates into sub-intervals so as to find a robust plan within each sub-interval. An important question concerns the choice among these plans, the plan by which to start the execution. A solution could be to determine the sub-interval covering the most possible run-time values. The plan assigned to this sub-interval is then selected to execute the query. We detail this concept in the next subsection, which is devoted to describe our proposal.

3.2 Robust Optimization Method

The above mentioned discussion motivates the necessity of incorporating estimation uncertainty in the query optimization process. In order to design and develop such an optimization method, it is suitable to offer, first, a method for one-join operation. This method constitutes the building block for a robust execution of a multi-join query. In this regard, we present a method that consists of two main modules: (1) Identification of Robust Plans, and (2) Selection of an Execution Plan. We will examine them more in detail in the subsections below.

Identification of Robust Plans: IRP is a compile-time strategy for identifying robust plans. It relies on intervals of estimates to model the estimation uncertainty. Such intervals are used to select the plan that minimizes a potential sub-optimality resulting from possible estimation errors. The computation of these intervals is out of the scope of this paper. Methods to calculate the upper and lower bound of such an interval can be found in [3, 6, 17].
Let Q = Join (T, R1) be a join query between T and R1, where T is a temporary relation and R1 is a base relation. Let us assume that the interval of estimates around T$_{estimated}$ is defined, i.e. I = [L, U] where L and U are the lower and the upper bound of the interval I. Let S be a set of possible query execution plans. This module consists in determining -for each plan P in S- the interval within which the plan is robust (Cf. Algorithm 1 in Appendix). Calculate this interval may be converted into a numerical solving problem. We compute the lower bound by solving the inequality :

$$Cost(P, T_{estimated}, R1_{value}) \leq (1 + \frac{\lambda}{100}) \times CostBestPlan(S, T_{estimated}, R1_{value}) \quad (2)$$

, where $T_{estimated}$ is considered as the variable of the inequality. We choose then the minimum root. $T_{estimated}$ refers to the point estimate of the temporary relation size, and $R1_{value}$ is the size of the base relation R1.
$CostBestPlan(S, T_{estimated}, R1_{value})$ returns the cost of the best plan in S for $T_{estimated}$ and $R1_{value}$ while $Cost(P, T_{estimated}, R1_{value})$ returns the execution cost of P. To avoid a large computational complexity, browsing through the interval is made using the principle of the secant method which is very effective and can be extended to n-parameters space [21]. Algorithm 3 shows our modified Secant method for computing upper bound. It processes similarly compared to Algorithm 2. Note that the Secant method terminates as soon as we get a cost inversion or if it diverges. Otherwise, a new position in the interval is computed. The upper bound is determined by solving the inequality :

$$(1 + \frac{\lambda}{100}) \times CostBestPlan(S, T_{estimated}, R1_{value}) < Cost(P, T_{estimated}, R1_{value}) \quad (3)$$

, also looking for the minimum root.

Selection of an Execution Plan: After identifying robust plans, only one plan should be selected to start the query execution. This plan is chosen based on a probabilistic approach. We propose to compute, for each plan, its probability $Prob(P)$ to handle run-time variation of estimates. The greater is the probability the higher is the likelihood of the plan to minimize sub-optimality. We process estimates as random variables and compute $Prob(P)$ as:[2]

$$Prob(P) = \frac{length\ of\ the\ robustness\ range\ associated\ with\ P}{length\ of\ the\ interval\ of\ estimates\ I} \tag{4}$$

The plan with the higher probability is chosen to start execution. When two plans have equal probabilities, the plan that maximizes the worst-case performance is selected (Cf. Algorithm 4).

Extension to Multiple Parameters. So far, we have assumed that the size of only one join input-relation is error-prone. In this subsection, we address the situation wherein the above approach is generalized to the multi parameter case, i.e. the sizes of both inputs-relations are prone to estimation errors. In this case, the size of each relation is modelled by an interval of estimates. the intersection of these intervals forms a bounding box. We compute -for each plan- a robustness box rather than a robustness range. In addition, the plan associated with the robustness box covering the most space of uncertainty is chosen to start the execution. Notwithstanding these changes, the basic mechanism of the IRP algorithm is similar to the first case (Cf. Algorithm 5 and Algorithm 6).

4 EXPERIMENTS

In this section we describe the experimental evaluation of our proposed optimization method, termed IRP. We compare IRP with a method that relies on the single point-based optimization approach (termed TRAD) and with RIO. RIO is a method using the multi point-based optimization approach. We study the behaviours of these methods with respect to error in estimates. We consider that an estimation uncertainty may be high, medium or low. We conducted our experiments using a simulation model that we describe below.

4.1 Simulation Model

To perform our experiments, we used a query builder and a simulator. The simulator includes the simulated optimization method and a meta-base. The simulated method consists of two main modules: 1)Query Optimizer, and 2)Query Executor. The meta-base includes information describing the run-time environment (e.g. CPU performance, available memory amount) as well as information

[2] : $\mathrm{P}(X \in [x, x+d]) = \int_x^{x+d} (1/(b-a))\mathrm{d}y = d/(b-a)$, where $[x, x+d] \subset [a, b]$

about data (e.g. sizes of base relations). The dataset used in the experiments is shown in Table 1.

Table 1. Summary of dataset used in the experiments

Parameter	**Value**	**Parameter**	**Value**
Buffer size	400 MB	CPU performance	100 000 MIPS
Disk bandwidth	100 MB/s	Disk latency	2 ms
Size of a page on disk	4 KB	Size of a record	1 - 3 KB
Size of a relation	100 - 1000 MB	Size of an attribute	10 - 500 Bytes

In the next subsections, we present the results of our experiments. First, we study the case wherein only one input-relation for a join is prone to an estimation error. Second, we extend our experiments to the case wherein both inputs-relations are prone to estimation errors. During our experiments, we consider that the cost-increase threshold for robustness condition is 20%.

4.2 One relation prone to an estimation error

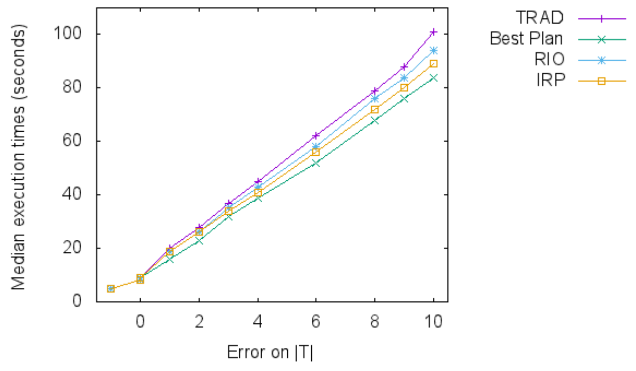
For this experiment, we use a set of 100 queries which we call SQ. Queries in SQ include only one join operation and verify all, the followings conditions: (i) One of the inputs relations is a base relation, denoted R1, and (ii) One of the inputs relations is a temporary relation, denoted T and is resulting from a selection operation on a base relation named R2. The following query Q' is a sample of queries in SQ:

*Q' = select * from R1, R2 Where R1.a = R2.b and R2.c < V1 and R2.d > V2*

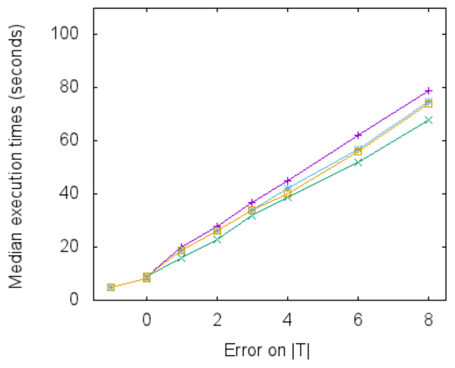
Q' can be represented as *Join(R1, T)*, where T is the result of *"select * from R2 Where R2.c < V1 and R2.d > V2"*. Estimates of the temporary relations sizes while costing execution plans for queries in SQ, may be erroneous. This is due to eventual correlations between the selection predicates. For each query, the uncertainty about the temporary relation size, i.e., |T|, is modelled by an interval of estimates. To calculate such an interval, we use the method proposed in [3]. We consider three uncertainty levels: high, medium, and low. Note that a high uncertainty in estimates involves a large interval of estimates. This interval is then used by RIO and IRP to generate a robust query execution plan. As for TRAD, it relies on a single-point estimate of |T| to select an execution plan expected to be the best plan.

Impact of estimation errors on execution times: The first experiment consists in assessing the impact of estimation errors on the execution time of each method. Figure 3 shows the variation of the execution time of plans produced by TRAD, RIO and IRP for queries in SQ. We vary the error between the estimate of the temporary relation size and its actual value. This is repeated for each query in SQ. Figure 3 also shows the execution time of the best plan.
The execution time for a method is computed as the median of the execution times of all queries in SQ by this method. We decided to use the median

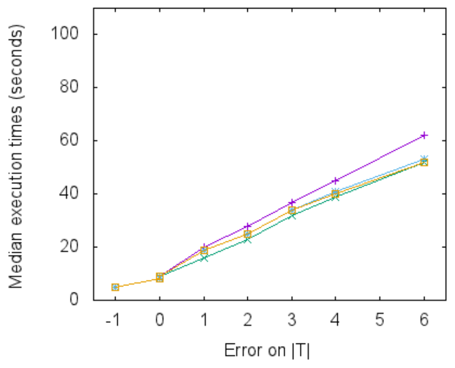
rather than the mean value because some extreme values of execution times can strongly pull the mean value up or down. This results on a wrong interpretation of most execution times. Error in estimates plotted on the x-axis is calculated as $(|T_{actual}|/|T_{estimated}|)$ - 1 [3]. A positive error indicates an underestimate while a negative error indicates an overestimate of the actual run-time value compared to the compile-time estimated value.
Figure 3 shows that the performance of TRAD remains acceptable when the error on the temporary relation size, i.e. |T|, is very low. However, when the error becomes large, (i.e. greater than 2), we observe a significant increase in the execution times of plans produced by TRAD. Indeed, TRAD selects an execution plan whose performance is heavily dependent on the accuracy of compile-time estimates. This plan is optimal for only the single-point estimate of |T|. Plans chosen by TRAD are very sensitive to estimation errors.



(c) High uncertainty



(a) Medium uncertainty



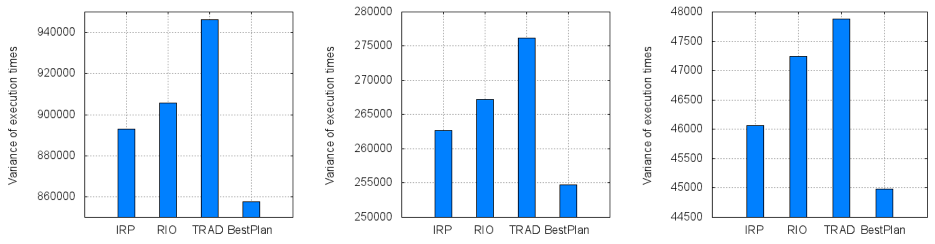
(b) Low uncertainty

Figure 3. Variation of execution times with respect to errors on |T|

Figure 3 shows that RIO generates better performance compared with TRAD. However, the performance of RIO may deteriorate. Indeed, when RIO does not find a single robust plan within the interval of estimates of |T|, it behaves like

TRAD. It produces a plan expected to be optimal for the compile-time estimate of |T|. Unlike RIO, IRP remains based on multi-point estimates. It generated an execution plan that provides stable performances within a sub-interval of the interval of estimates. Using a sub-interval of estimates rather than a single-point estimate for |T| allows to IRP to produce plans whose execution times are usually more stable compared with RIO and TRAD. We also notice in Figure 3 that when the uncertainty is low, the performance of TRAD and IRP becomes close. This is because when the interval of estimates is narrow, it becomes more feasible for RIO to find a plan that is robust within an interval.

Impact of estimation errors on the consistency of methods: The consistency of a method refers to its ability to cope with estimation errors and/or changes in run-time conditions compared to compile-time expectation of run-time conditions. A method is said highly consistent if its performance does not degrade significantly in the presence of estimation errors [28]. To measure the consistency of TRAD, RIO and IRP, we compute the variance of performance of each method and compare it to the variance in case of best performance. Note that the consistency is inversely proportional to the variance [28]. To achieve this experiment, we use the same set of queries previously used, i.e. SQ. The results of this evaluation are shown in Figure 4.



(a) High uncertainty (b) Medium uncertainty (c) Low uncertainty

Figure 4. Variance of execution times with respect to errors on |T1|

As we can see in Figure 4, the variances of RIO and IRP are close when the estimation uncertainty is low. However, when the uncertainty is large (high-/medium), the gap between the variances of methods increase. A high variance means a significant dispersion of execution times. Figure 4 demonstrates that plans generated by IRP are more stable than those generated by RIO. This figure also proves that estimation errors have a pronounced impact on the consistency of TRAD. This figure confirms the evaluation results previously obtained.

– **Impact of estimation errors on the effectiveness of methods:**

The effectiveness of a method is related to its ability to achieve its objectives. The objective of TRAD is to find an execution plan that generates optimal performance based on single estimation values. As for RIO and RPI, they share the goal of generating execution plans whose performance is robust for many possible values of error-prone parameters. To evaluate the effectiveness of TRAD, we calculate the percentage of plans actually optimal compared to the number of estimated optimal plans during compile-time. As for RIO and RPI, we calculate for each method the times where the number of expected robust plans matchs that of actually robust plans. We present the results in Figure 5 by means of a percentage of truly robust plans against total estimated robust plans.

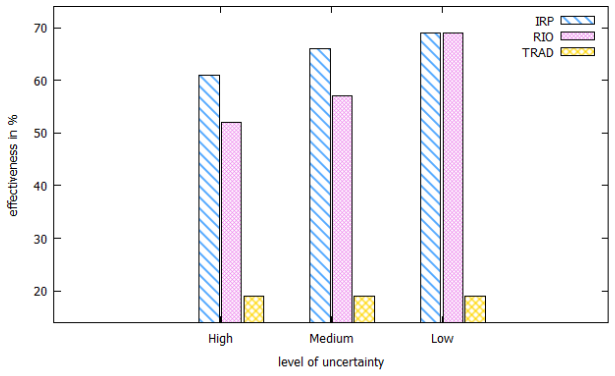
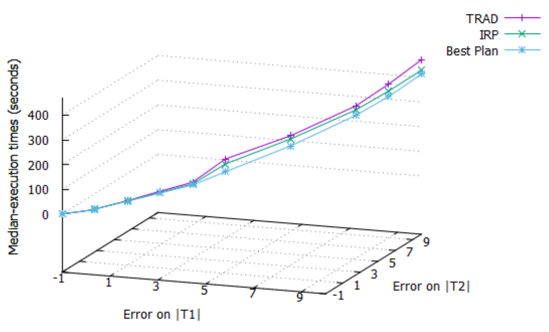


Figure 5. Effectiveness of methods

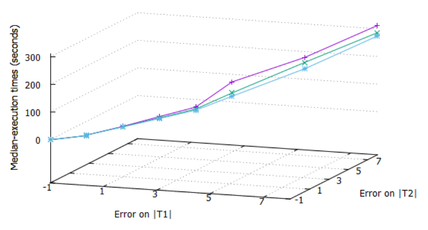
Figure 5 above shows the effectiveness of methods TRAD, RIO and RPI. To conduct this experiment, we used the set of queries, S. At runtime, we varied the error on estimates about the temporary relation size of each query. The results show that regardless of the level of uncertainty, the RPI method has a high effectiveness ($> 60\%$). As for RIO, its effectiveness is less important than RPI when uncertainty is medium or high. However, the effectiveness of RIO and RPI are close when uncertainty is low. TRAD has a very low effectiveness. This is because it often generates plan estimated optimal for only expected execution conditions, unlike RIO and RPI that consider several possible execution conditions when choosing a plan execution.

4.3 Both relations prone to estimation errors

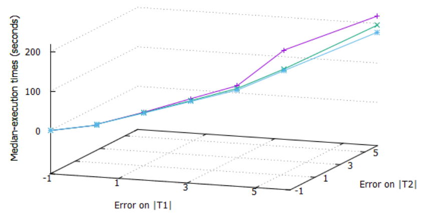
This subsection provides the results of our experiments under the assumption that the sizes of both join inputs-relations denoted T1 and T2, are prone to estimation errors. As RIO assumes that at least one of the inputs-relations is a base relation [3], we compare our method only with TRAD. The experiment is done using the same dataset (Cf. Table 1) as previously. Figure 6 below shows the impact of estimation errors on |T1| and |T2| on the executions times of IRP and TRAD.



(a) High uncertainty



(b) Medium uncertainty



(c) Low uncertainty

Figure 6. Variation of execution times with respect to error on |T1| and |T2|

Unsurprisingly, Figure 6 confirms the results obtained in the first case. IRP generates execution plans that provide stable performance in the presence of estimation errors.

5 RELATED WORK

A variety of optimization strategies have been proposed in the literature to identify robust plans. [29] overviews these works. In this section, we compare our proposal with the most closely related prior research, i.e., [3, 15, 19]. Similarly to this paper, these methods address the problem of query optimization due to the uncertainty in estimates. These methods make use of intervals to manage estimation uncertainty. These intervals are then used by the optimizer to pick an appropriate execution plan. However, our work defers from these works on several aspects.

In the initial optimization phase of Rio [3], if the plan chosen by the optimizer at the lower and the upper bounds of the interval is the same as that chosen for the single-point estimate, then the plan is assumed *robust* within the whole interval. In our method, we adopt a different way to check the robustness of a plan. We verify the robustness of a plan at different points in the interval.

These points are determined using our modified secant method, which provides reduced computation complexity and precise results. In addition, Rio assumes that at least one of the join inputs-relations is a base relation. We extend this by considering the case wherein both relations are prone to errors.
Among previous related research, the work by Markl et al. [19]. In [19], and later in [4], validity ranges are computed for each sub-plan of a query execution plan. The best plan is first chosen based on single-point estimates. The upper and lower bounds wherein the plan remains optimal are then computed. These methods aim for optimal performance while IRP aims for stability. Contrary to these methods, IRP uses intervals of estimates to generate robust execution plans . In the presence of large estimation errors, these plans may be more stable that plans expected to be optimal.
The use of intervals may seem similar to the principle of parametric optimization [10,15] where different plans are generated for different intervals of the optimization parameters. The main limitation of this approach is the large number of plans to generate, store and compare at the optimization phase. We reduce this complexity by computing robustness ranges. We accept a minor cost-increase that reduces the number of plans generated. In addition, parametric optimization is particularly attractive in the case of queries that are compiled once and executed several times, possibly with minor modifications of parameters. Unlike our work, its objective being to avoid compiling multiple instances of the same query, but not to ensure robustness against estimation errors.

6 Conclusion and Future Work

This paper proposes an optimization strategy to handle uncertainty in compile-time estimates. Uncertainty is modelled by means of intervals of estimates. These intervals are then used to identify robust plans. We characterize a plan as robust if its cost remains stable and acceptable in case of estimation errors. Our method relies on a probabilistic approach to select the plan by which to start the execution. The experiments show that the proposed strategy may improves the ability of a query optimizer to generated a robust execution plan, especially in the presence of large estimation errors.
The proposed method performs in a uniprocessor environment. As future work, we plan to extend it to a multiprocessor environment. It would be interesting to study the importance of the parallelism degree choices over the identification of a robust query execution plan.

A Appendix

A.1 Algorithm 1. Robustness-Range Computing

```
program Robustness-Range Computing (Computation of the
         robustness-range of each execution plan in S)
{Assuming as inputs: S, $\lambda$, R1$_{value}$, L, U};
var   T$_{estimated}$, CostOfP$_{best}$, LowerBound, UpperBound: Real;
Begin
  for each P in S do
  { T$_{estimated}$ := L;
    CostOfP$_{best}$ := CostBestPlan(S,T$_{estimated}$,R1$_{value}$);
    While (Cost(P,T$_{estimated}$,R1$_{value}$)>(1+$\lambda$/100)$\times$CostOfP$_{best}$
             and T$_{estimated}$ < U) do
      T$_{estimated}$ := ComputeLowerBound(P,T$_{estimated}$,R1$_{value}$,S,$\lambda$);
    if (T$_{estimated}$ < U) then
     {
     LowerBound := T$_{estimated}$;
     CostOfP$_{best}$ := CostBestPlan(S,T$_{estimated}$,R1$_{value}$);
     While (Cost(P,T$_{estimated}$,R1$_{value}$)<=(1+$\lambda$/100)$\times$CostOfP$_{best}$
             and T$_{estimated}$ < U)
       T$_{estimated}$ := ComputeUpperBound(P,T$_{estimated}$,R1$_{value}$,S,$\lambda$);
     UpperBound := T$_{estimated}$;
     P.setRobustnessRange(LowerBound,UpperBound);
     }
    else
     break; % plan rejected
   }
End
```

A.2 Algorithm 2. Lower-Bound Computing

```
program ComputeLowerBound (incremented value of T$_{estimated}$)
{Assuming as inputs: P, T$_{estimated}$, R1$_{value}$, S, $\lambda$};
var   CostOfP$_{best}$, OldDiff, NewDiff: Real;
Begin
  CostOfP$_{best}$ := CostBestPlan(S,T$_{estimated}$,R1$_{value}$);
  OldDiff := Cost(P,T$_{estimated}$,R1$_{value}$)-(1+$\lambda$/100)$\times$CostOfP$_{best}$;
  $T_{estimated} := T_{estimated} \times 1,05$;
  CostOfP$_{best}$ :=CostBestPlan(S,T$_{estimated}$,R1$_{value}$);
  NewDiff := Cost(P,$T_{estimated}$,R1$_{value}$)-(1+$\lambda$/100)$\times$CostOfP$_{best}$;
  if (NewDiff<=OldDiff and NewDiff>0) then
     T$_{estimated}$ := T$_{estimated}$$\times$[1+NewDiff$\div$(21$\times$(OldDiff-NewDiff))]
  else
     break; %method is diverging
  return T$_{estimated}$;
End
```

A.3 Algorithm 3. Upper-Bound Computing

```
program ComputeLowerBound (incremented value of $T_{estimated}$)
{Assuming as inputs: P, $T_{estimated}$, $R1_{value}$, S, $\lambda$};
var   CostOfP$_{best}$, OldDiff, NewDiff: Real;
Begin
  CostOfP$_{best}$ := CostBestPlan(S,$T_{estimated}$,$R1_{value}$);
  OldDiff := $(1+\lambda/100)\times$CostOfP$_{best}$-Cost(P,$T_{estimated}$,$R1_{value}$);
  $T_{estimated} := T_{estimated} \times 1,05$;
  CostOfP$_{best}$ :=CostBestPlan(S,$T_{estimated}$,$R1_{value}$);
  NewDiff := $(1+\lambda/100)\times$CostOfP$_{best}$-Cost(P,$T_{estimated}$,$R1_{value}$);
  if (NewDiff<=OldDiff and NewDiff>0) then
     $T_{estimated}$ := $T_{estimated}\times$[1+NewDiff$\div$(21$\times$(OldDiff-NewDiff))]
  else
     break; %method is diverging
  return $T_{estimated}$;
End
```

A.4 Algorithm 4. SelectionOfExecutionPlan

```
program SelectionofExecutionPlan()
{Assuming as inputs: S, I};
var   prob, probMax:Real; i,j,verif:Integer; founded:Boolean;
      ListOfRobustPlans:List<Plan>; QEP:Plan;
Begin
probMax := 0;
For each P in S do
{
   prob := P.getStabilityInterval().getlength() ÷ I.getlength();
   P.setProbability(prob);
   //compute probMax;
   if (prob > probMax)
     probMax := prob
}
For each P in S do
{
   if (P.getProbability() == probMax)
     ListOfRobustPlans.add(P)
}
i := 0;
founded := false;
While (i<ListOfRobustPlans.size() and founded==false)
{
   verif := 0;
   item := ListOfRobustPlans.get(i);
   For j:=0 to ListOfRobustPlans.size()
   {
        if (item.getInterval().getUpperBound() >=
          ListOfRobustPlans.get(j).getInterval().getUpperBound())
            verif := verif+1
   }
   if (verif==ListOfRobustPlans.size())
   {
      founded := true;
      QEP := P;
   }
   else
      i := i+1
}
return QEP;
End
```

A.5 Algorithm 5. Robustness-Box Computing

```
program Robustness-Box Computing (computation of robustness-box
for each plan P)
{Assuming as inputs: S, $\lambda$, L$_r$, U$_r$, L$_s$, U$_s$};
var   StopT1, StopT2: Integer;
      T1$_{estimated}$, T2$_{estimated}$: Real;
Begin
For each P in S do
{
   StopT1 := 0; StopT2 := 0;
   T1$_{estimated}$ := Lr; T2$_{estimated}$ := Ls;

   While (Cost(P,T1$_{estimated}$,T2$_{estimated}$)> $(1+\lambda)\times$
         CostBestPlan(S,T1$_{estimated}$,T2$_{estimated}$) and StopT1+StopT2<2) do
   {
    if (T1$_{estimated}$ < Ur)
          T$_{estimated}$ := ComputeLowerBound(P,T1$_{estimated}$,T2$_{estimated}$,S,$\lambda$);
    else
          stopT1 := 1;

    if (T2$_{estimated}$ < Us)
          T2$_{estimated}$  ComputeLowerBound(P,T1$_{estimated}$,T2$_{estimated}$,S,$\lambda$);
    else
          stopT2 := 1;
   }

   if (StopR+StopS < 2)
   {
      LowerBound.setLeft(T1$_{estimated}$);
```

```
        UpperBound.setLeft(T2$_{estimated}$);
        While (Cost(P,T1$_{estimated}$,T2$_{estimated}$) $<= (1+\lambda)\times$
          CostBestPlan(S,T1$_{estimated}$,T2$_{estimated}$) and StopR+StopS < 2)
        {
         if (T1$_{estimated}$ < Ur)
              T1$_{estimated}$ := ComputeUpperBound(P,T1$_{estimated}$,T2$_{estimated}$,S,$\lambda$);
         else
              stopR := 1;

         if (T2$_{estimated}$ < Us)
              T2$_{estimated}$ := ComputeUpperBound(P,T1$_{estimated}$,T2$_{estimated}$,S,$\lambda$);
        else
              stopS := 1;
        }
        LowerBound.setRight(T1$_{estimated}$);
        UpperBound.setRight(T2$_{estimated}$);
        P.setBoundingBox(LowerBound,UpperBound);
   }
   else
     break //plan rejected, it is not robust within the bounding box;
}
End
```

A.6 Algorithm 6. SelectionOfExecutionPlanInBox

```
program SelectionOfExecutionPlanInBox()
{Assuming as inputs: S};
var   surface, surfaceMax:Real; i, j, verif:Integer; founded:Boolean;
      ListOfRobustPlans:List<Plan>; QEP: Plan;
Begin
surfaceMax := 0;
For each P in S do
{
   surface := P.getBoundingBox().getIntervalR().getlength() $\times$
        P.getBoundingBox().getIntervalS().getlength();
   P.setSurface(surface);
   //compute surfaceMax;
   if (surface > surfaceMax)
     surfaceMax := surface;
}
For each P in S do
{
   if (P.getSurface()==surfaceMax)
     ListOfRobustPlans.add(P)
}
i := 0;
```

```
founded := false;
While (i < ListOfRobustPlans.size() and founded==false)
{
   verif := 0;
   P := ListOfRobustPlans.get(i);
   For j := 0 to ListOfRobustPlans.size()
   {
    if (P.getBoundingBox().getIntervalR().getRight() >=
    RobustPlans.get(j).getBoundingBox().getIntervalR().getRight()
    and
    P.getBoundingBox().getIntervalS().getRight() >=
    RobustPlans.get(j).getBoundingBox().getIntervalS().getRight())
      verif := verif+1
   }
   if (verif == ListOfRobustPlans.size())
   {
    founded := true;
    QEP := P;
   }
   else
    i := i+1
}
return QEP;
End
```

References

1. Abhirama, M., Bhaumik, S., Dey, A., Shrimal, H., Haritsa, J.R.: On the stability of plan costs and the costs of plan stability. In: Proc. VLDB Endow. 3, pp. 1137–1148 (2010)
2. Babcock, B., Chaudhuri, S.: Towards a robust query optimizer: A principled and practical approach. In: Proc. of ACM SIGMOD International Conference on Management of Data, pp.119–130 (2005)
3. Babu, S., Bizarro, P., DeWitt, D.: Proactive re-optimization. In: Proc. of ACM SIGMOD International Conference on Management of Data, pp.107–118 (2005)
4. Bizarro, P., Bruno, N., DeWitt, D.J.: Progressive parametric query optimization. IEEE Trans. Knowl. Data Eng. 21, pp. 582–594 (2009)
5. Bruno, N., Jain, S., Zhou, J.: Continuous cloud-scale query optimization and processing. PVLDB 6, pp. 961–972 (2013)
6. Chaudhuri, S., Narasayya, V., Ramamurthy, R.: Estimating progress of Long Running SQL Queries. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, pp.803–814 (2004)
7. Chen, C.M., Roussopoulos, N.: Adaptive selectivity estimation using query feedback. In: Proc. of ACM SIGMOD International Conference on Management of Data, pp.161–172 (1994)
8. Christodoulakis, S.: Implications of certain assumptions in database performance evaluation. In: ACM Trans. Database Syst. 9, pp. 163–186 (1984)
9. Chu, F.C., Halpern, J.Y., Seshadri, P.: Least expected cost query optimization: An exercise in utility. In: Proc. of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia pp.138–147 (1999)
10. Cole, R.L., Graefe, G.: Optimization of dynamic query evaluation plans. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, pp.150–160 (1994)
11. D., H., Darera, P.N., Haritsa, J.R.: Identifying robust plans through plan diagram reduction. In: Proc. VLDB Endow. 1, pp. 1124–1140 (2008)
12. Deshpande, A., Garofalakis, M.N., Rastogi, R.: Independence is good: Dependency-based histogram synopses for high-dimensional data. In: ACM SIGMOD Conference, pp.199–210 (2001)
13. Dutt, A., Neelam, S., Haritsa, J.R.: Quest: An exploratory approach to robust query processing. In: Proc. VLDB Endow. 7, 1585–1588 (2014)
14. Getoor, L., Taskar, B., Koller, D.: Selectivity estimation using probabilistic models. I:n Proc. of ACM SIGMOD International Conference on Management of Data, p.461–472 (2001)
15. Hulgeri, A., Sudarshan, S.: Parametric query optimization for linear and piecewise linear cost functions. In: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB Endowment, pp.167–178 (2002)
16. Ioannidis, Y.E., Christodoulakis, S.: On the propagation of errors in the size of join results. In: Proc. of SIGMOD International Conference on Management of Data, pp.268–277 (1991)
17. Kabra, N., DeWitt, D.J.: Efficient mid-query re-optimization of sub-optimal query execution plans. In: Proc. of ACM SIGMOD International Conference on Management of Data, pp.106–117 (1998)
18. Karanasos, K., Balmin, A., Kutsch, M., Ozcan, F., Ercegovac, V., Xia, C., Jackson, J.: Dynamically optimizing queries over large scale data platforms. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, pp.943–954 (2014)

19. Markl, V., Raman, V., Simmen, D., Lohman, G., Pirahesh, H., Cilimdzic, M.: Robust query processing through progressive optimization. In: Proc. of ACM SIGMOD International Conference on Management of Data, pp.659–670 (2004)
20. Neumann, T., Galindo-Legaria, C. A.: Taking the edge off cardinality estimation errors using incremental execution. (DBIS), Germany., pp.73–92 (2013)
21. Papakonstantinou, J. M. Tapia, R. A.: Origin and evolution of the secant method in one dimension. The American Mathematical Monthly, 500–518 (2013)
22. Poosala, V., Haas, P.J., Ioannidis, Y.E., Shekita, E.J.: Improved histograms for selectivity estimation of range predicates. In: Proc. of ACM SIGMOD International Conference on Management of Data, pp.294–305 (1996)
23. Poosala, V., Ioannidis, Y.E.: Selectivity estimation without the attribute value independence assumption. In: Proc. of 23rd International Conference on Very Large Data Bases, pp.486–495 (1997)
24. Selinger, P.G., Astrahan, M. M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, pp.23–34 (1979)
25. Soussi, M.C., Morvan, F., Hameurlain, A.: Estimation Error-Aware Query Optimization: An Overview. In: The International Journal of Computer Systems Science and Engineering, *In press.*
26. Tzoumas, K., Deshpande, A., Jensen, C.S.: Lightweight graphical models for selectivity estimation without independence assumptions. In: PVLDB (2011)
27. Tzoumas, K., Deshpande, A., Jensen, C.S.: Efficiently adapting graphical models for selectivity estimation. In: The VLDB Journal 22, pp.3–27 (2013)
28. Wiener, J.L., Kuno, H., Graefe, G.: Benchmarking Query Execution Robustness. In: TPC Technology Conference on Performance Evaluation and Benchmarking. pp. 153-166 (2009)
29. Yin, S., Hameurlain, A., Morvan, F.: Robust query optimization methods with respect to estimation errors: A survey. In: SIGMOD Rec. 44, 25–36 (2015)