Least Expected Cost Query Optimization: What Can We Expect?

Francis Chu[*] Joseph Halpern Johannes Gehrke

Department of Computer Science
Upson Hall, Cornell University
Ithaca, NY 14853-7501, USA

Paper ID: 108

Abstract

A standard assumption in the database query optimization literature is that it is adequate to optimize for the "typical" case—that is, the case in which various parameters (e.g., the amount of available memory, the selectivities of predicates, etc.) take on their "typical" values. In [CHS99], we argued that we could do better by choosing plans based on their *expected cost.* Here we investigate this issue more thoroughly. We show that in many circumstances of interest, a "typical" value of the parameter often does give acceptable answers, provided that it is chosen carefully and we are interested only in minimizing expected running time. However, by minimizing the expected running time, we are effectively assuming that if plan p_1 runs three times as long as plan p_2, then p_1 is exactly three times as bad as p_2. An assumption like this is not always appropriate (for example, for time-critical data). We show that the focusing on least expected cost can lead to significant improvement for a number of cost functions of interest.

1 Introduction

Database queries are typically specified declaratively, so it is up to the DBMS to choose a good plan to carry out the query [SAC$^+$79]. Given a query, a cost-based optimizer has to pick a plan of least cost. To do this, the optimizer has to estimate the cost of a plan. One problem is that the cost of a plan depends on various parameters (e.g., the amount of available memory, the selectivities of predicates, the sizes of the tables, etc.), some of whose value the optimizer might not (or cannot) know at query optimization time. Traditionally, an optimizer assumes (at optimization time) that the parameters will take on certain specific values (at run time), and then computes an optimal plan based on the assumption that the parameters actually take on these values. How these values are chosen varies from implementation to implementation. In [CHS99], we termed this approach *least specific cost* (*LSC*) *optimization.* Of course, if the performance of the plan chosen can vary significantly, depending on the parameter value, then the plan chosen by an LSC optimizer might be far from optimal.

We may be able to do better if we have a probability distribution on the possible parameter values (which is often a quite reasonable assumption in practice). In this case, one obvious choice of specific value for an LSC optimizer to take is the expected value of the parameter (or perhaps the modal value—that is, the one that occurs most frequently). However, if the goal is to find a plan of least expected running time, then choosing this specific value is not necessarily the right thing to do. In general, the best plan under the assumption that the parameter takes on its expected value is not necessarily a plan with least expected running time, as the following example shows.

Example 1.1: Suppose that we have two plans, p_1 and p_2, such that if there are at least 1000 pages of memory available, p_1 takes 280 seconds to run and p_2 takes 300 seconds to run, and if the amount of memory is between 633 pages and 1000 pages, p_1 takes 560 seconds to run and p_2 (still) takes 300 seconds to run. Suppose that, by observation, the probability of having 2000 pages is 0.8 and the probability of having 700 pages is 0.2. The expected value of the amount of memory available is 1740 pages. If the optimizer assumes that there are 1740 pages of memory, it will choose p_1. However, the expected running time of p_1 is $0.8(280) + 0.2(560) = 336$, while the expected running time of p_2 is 300. ▮

In light of such examples, we advocated in [CHS99] what we called *least expected cost* (*LEC*) *optimization*,

[*]This is the contact author. Phone: (607)255-4574 FAX: (607)255-4428 Email: `fcc@cs.cornell.edu`

where plans are chosen based on their expected cost instead of their cost at a specific parameter setting. The advantage of LEC optimization is that the plan it picks has the least expected cost among the plans under consideration. So if the query is compiled once and executed many times (as is often the case), then the average running time of the query is likely to be the least among all the candidates considered. We also showed how the LEC plan could be computed given a probability distribution on the parameters, by extending techniques currently used in LSC optimizers, with relatively little overhead in running time. If the probability distribution over the parameters is given by histograms, the actual overhead depends on the number of buckets used in the histograms; LSC optimization is essentially LEC optimization with one bucket.

In this paper we investigate the relation between LEC optimization and LSC optimization more thoroughly, both through empirical experiments and analytical explorations. Our contributions are as follows:

1. One surprise that we encountered is that LSC optimization can produce LEC plans (at least in principle) in a wide variety of situations. This prompted us to characterize when LSC optimization can yield LEC plans.
2. We exhibit, both through experiments and analytical results, scenarios in which the above conditions hold. This is of special interest for practitioners, since under these conditions, they can obtain the LEC plan without modifying the core of the existing optimizers.
3. The scenarios above all depend on the cost being linear in terms of running time (which is a standard assumption in the literature). We argue that this is not always appropriate through motivating examples, and we investigate some consequences of having non-linear cost functions.

The rest of this paper is organized as follows. In Section 2.1 we report the results of experiments showing that LSC optimization can produce LEC plans in centralized databases. In Section 2.2, we show that this continues to be true even in a distributed setting in which the transmission time can be modeled probabilistically. Then in Section 2.3 we characterize when it is feasible to produce LEC plans using LSC optimizers. The results in Section 2 depend critically on the assumption that the cost is just running time. In Section 3, we show that full LEC optimization is in general necessary if the cost function is not linear in the running time. We then discuss, in Section 4, some challenges that arises when the cost function is not linear in the running time. Finally, in Section 5 we offer some concluding remarks and possible future directions.

2 LEC Through LSC

We originally planned to construct an LEC prototype to benchmark the performance of LEC optimization using the TPC-H database on a commercial DBMS. One of the things we did was to fine-tune the cost formula we found in textbooks. (It is well known that textbook formulas do not accurately describe the cost of joins; see [GBC98, LG98].) As we were doing these experiments, however, we ran into some findings that altered the course of our research.

2.1 The Experiments

We ran all our experiments on a machine equipped with two Pentium Pro processors running at 200MHz, 128MB of RAM, Windows NT 4.0 SP5 1381, and an 8.9GB SCSI hard disk. The page size of the DBMS is 8KB and we may set the available buffer memory from 4MB to 127MB. (Due to overhead of the operating system, we varied the buffer pool size only from 4MB to 80MB. Note that the DBMS refuses to run certain memory-intensive plans if the size of the buffer pool is below 4MB.) The DBMS allows users to force a given plan to be used by giving "hints" to the optimizer. The database we used was the TPC-H database with the scale factor set to one. (We refer the reader to [TPPC99] for the details of TPC-H.)

2.1.1 Amount of Available Memory

Since the amount of available memory is difficult to predict [G. Lohman, private communication, 1998], we focused on this parameter in [CHS99]. When we ran the experiments to fine-tune the cost formulas we have for join methods, we discovered that there is often a dominant plan; that is, sort-merge (or indexed nested-loop) is always better, even though both indexed nested-loop and sort-merge exhibit discontinuous behavior, as one would expect, when the amount of memory becomes low (see Figure 1). Of course, we also have cases in which no join method dominates another; that is, the curves cross.

Clearly there is no need for LEC optimization when there is a dominant plan. Using any value of the parameter in LSC optimization will give the optimal plan. How much LEC optimization helps when there is a crossover depends on the probability distribution of memory sizes. For example, consider TPC-H Query 12, whose running

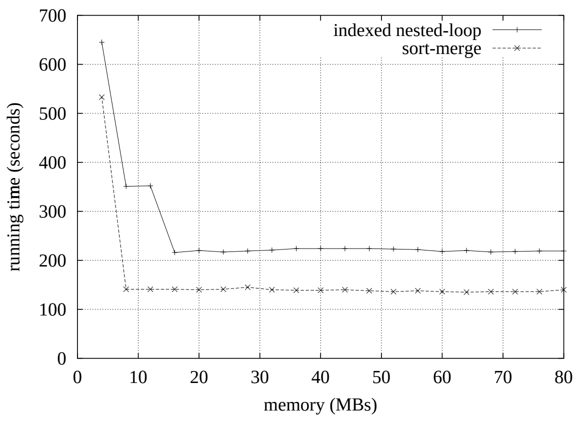

Figure 1: Running time for ORDERS $\bowtie$ LINEITEM

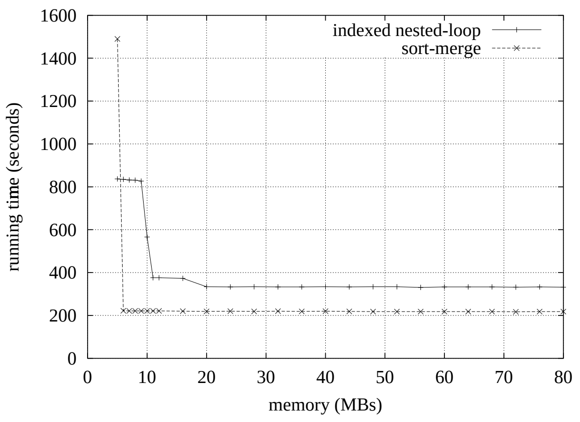

Figure 2: Running time for TPC-H Query 12

time for different values of memory is shown in Figure 2.[1] Note that sort-merge does better than indexed nested-loop in all memory settings except at 5MB (the DBMS refuses to execute either join method below 5MB). So if there is significant probability that the amount of memory available is low (in this case, around 5MB), then LEC optimization will help.

In our experiments, the crossovers for memory typically happen at the low end of the scale. This suggests that, if the parameter of interest is memory, as long as the probability of a memory shortage is low enough for this possibility to be safely ignored, then there will typically be a dominant plan. (This comes out in all our other experiments, not just the experiments shown in Figures 1 and 2.) Thus, in this case, LSC optimization suffices.

One might wonder if this phenomenon is just an artifact of the experimental setup we used. This is not so. The textbook cost formula for indexed nested-loop is essentially the number of pages of the outer relation plus the number of tuples in the outer relation that satisfies the selection predicate, since for each of these tuples we will need to read a page from the inner relation [Ull89]. (The textbook formula is not running time, but is supposed to be directly proportional to running time.) This quantity is clearly independent of memory. The rationale is that if we are using indexed nested-loop, we assume that the index fits in memory, so that we have to read only a page for each tuple of the outer relation (that satisfies the selection predicates). Once the amount of memory becomes so low that the index no longer fits in memory, the running time will clearly change. Figures 1 and 2 show that this is essentially the case. The textbook formula for sort-merge involves logarithms of the amount of memory, so they are not constant with respect to memory (unlike indexed nested-loop). However, the bases of these logarithms are the number of pages available, so if we have a large amount of memory, the formula is essentially constant for a large range of memory values. The bottom line is that, according to textbook formulas, there typically is a dominant plan when the amount of memory available is sufficiently large.

[1]Actually, this is the running time for a slight modification of Query 12; we added "and o_totalprice < 15000" to the predicate, since sort-merge dominates indexed nested-loop for the original query.

2.1.2 Selectivity

Another parameter we investigated is selectivity. Unlike memory, the running time is not a step function, and it increases as selectivity increases (whereas the running time *decreases* as memory increases). While it is often the case that an optimizer might not know the selectivity of a particular predicate due to a lack of statistics on the tables, this problem can be alleviated by keeping more detailed statistics. There is another source of uncertainty that cannot be dealt with by keeping statistics on the database itself. Often an optimizer is faced with a problem of optimizing a *query template*; that is, a query with (user) input variables. (This can happen with embedded SQL that contains host language variables; it can also happen with stored procedures that accept inputs.) In this case, the optimizer does not know the value of the inputs and must choose a plan in the absence of that information. The problem is that the selectivities of predicates in the query now depend on the input values, and this uncertainty cannot be alleviated by keeping detailed statistics about the database itself. However, if we have distributions on the input values (either collected by the DBMS or furnished by the users), then we can perform

```
select
    n_name,
    sum(l_extendedprice * (1 − l_discount)) as revenue
from
    customer, orders, lineitem, supplier, nation, region
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = [REGION]
    and o_orderdate >= [DATE]
    and o_orderdate < [DATE] + interval 1 year
group by
    n_name
order by
    revenue desc;
```

Figure 3: TPC-H Query 5

LEC optimization.[2]

As an example of a query template, consider Query 5 from the TPC-H benchmark (reproduced in Figure 3). The query contains two parameters: [REGION] and [DATE]. Since the TPC-H database is uniformly populated, the selectivity is essentially independent of the parameter values (especially for those values used in the actual benchmark itself). This is hardly surprising, since TPC-H is designed to evaluate traditional optimizers, which use LSC optimization. However, it is known that, in practice, databases are often *not* uniformly populated. Let us see what could happen if this is the case.

Example 2.1: Suppose we have a database populated as follows:

- 20% of the orders have o_orderdate between 1992-01-01 and 1996-12-31,
- 80% of the orders have o_orderdate between 1997-01-01 and 1998-12-31, and
- within each of the above groups the distribution on o_orderdate is uniform.

Suppose that the input distribution on [DATE] is $\Pr(\text{[DATE]} = y\text{-01-01}) = 0.2$ for $y \in \{1993, 1994, 1995, 1996, 1997\}$. If the optimizer uses the expected value of the input, it will take 1995-01-01 as the representative. This translates to having selectivity 0.04, since the interval is one year and 4% of the orders have o_orderdate in 1995. However, the expected selectivity is actually $0.4(0.2) + 0.04(0.8) = 0.112$. If we apply LSC optimization under the assumption that the selectivity is 0.04, we are unlikely to get an optimal plan. ∎

[2]There are other ways of dealing with this problem; for example, [INSS92] proposes an approach called *parametric query optimization*. In that approach, an LSC plan is produced for each parameter value. The problem is that the plan size grows, in the worst case, with the number of possible parameter values. Another problem is that they might output plans for parameter values with very low probability. It is possible that most of the plans in the output rarely get used.

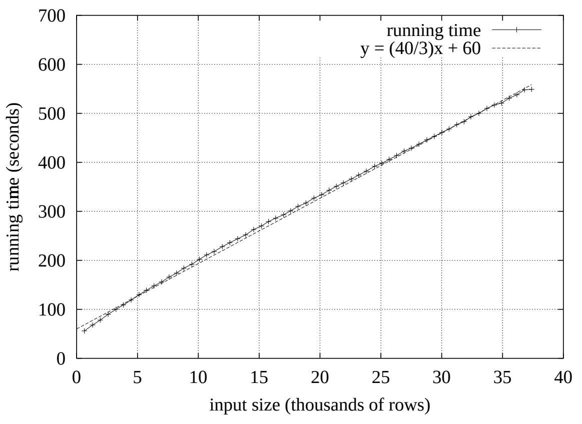

Figure 4: Running time for TPC-H Query 5

As Example 2.1 shows, the selectivity that corresponds to the expected value of the input need not be the expected selectivity in general. (Indeed, sometimes it does not even make sense to talk about the "expected value" of the input, since the input might not be numeric, as in the case of [REGION].) Why should we care about the expected selectivity at all?

Proposition 2.2: *If the cost of a plan is linear in the (intermediate) table sizes, then the LSC plan for the expected selectivity is an LEC plan.*

How reasonable is it to assume that the cost is linear in the input table sizes? If we have enough memory, the textbook formulas say sort-merge is essentially linear in the input table sizes. The textbook formula for indexed nested-loop is also essentially linear in the size of the outer input table. Since our focus is on selectivity in this subsection, we do assume that we have plenty of memory. (For sort-merge, "enough memory" essentially means the amount of memory is more than the square root of the larger relation. So if pages are 8KB, 80MB of memory will handle tables up to about 8GB. For indexed nested-loop "enough memory" essentially means that the index fits in memory.) In Figure 4 we show the running time of TPC-H Query 5 against selectivity; note that the running time is in fact almost linear in the input size. The x-axis shows the number of rows in the

ORDERS table that got selected. So we can convert the x-axis to selectivity if we divide by the number of rows in the ORDERS table, namely 1,500,000. (Since TPC-H is uniformly populated, the different values of [DATE] will select roughly the same number of rows, so we made the length of the interval a variable and varied that instead. The number of rows selected is roughly directly proportional to the length of the interval, because the TPC-H database is uniformly populated.)

We said that to use an LSC optimizer to get LEC plans we need to convert the input distribution to a distribution on selectivity. To do this, we need to evaluate the selectivity of each input. This can be done offline for a stored procedure. For example, the input distribution for Example 2.1 is $\Pr(\text{[DATE]} = y\text{-01-01}) = 0.2$ for $y \in \{1993, 1994, 1995, 1996, 1997\}$. The selectivity distribution is $\Pr(\sigma = 0.4) = 0.2$ and $\Pr(\sigma = 0.04) = 0.8$, where σ denotes the selectivity. If the number of input values is large and it is infeasible to convert the distribution of input values to the distribution of selectivities in one fell swoop, then we can do it incrementally. (In fact, if the DBMS actually gets the distribution on the input values by collecting statistics on user inputs, as opposed to being given a distribution by fiat, then the DBMS can do the conversion while collecting the statistics.)

2.2 Distributed Databases

As we saw in the previous section, careful selection of parameter values enables LSC optimization to produce LEC plans when we have uncertainty about selectivity. In this section, we show that the same thing happens when we have uncertainty about communication cost in a distributed or federated database.

A relatively recent trend has been a focus on distributed databases or federated databases, such as Garlic [HKWY97, ROH99]. In these databases, the tables are stored on (geographically) separate sites, connected by LANs or WANs, so one issue that comes up when executing queries is where operations get performed and which tables and intermediate results are shipped across the network. Even if we are ultimately interested only in the running time of a query, part of the running time is the time it takes to transmit tables, which depends on the characteristics of the network. For "local area clusters" or "server farms" there is a generally accepted model.[3] Let ℓ denote latency (essentially, this counts the time to set up the connections, which is roughly the time to transmit an empty packet), let s denote the size of the table, and let b denote the available bandwidth. Then the transmission time is

$$T(\ell, s, b) = \ell + s/b \tag{2.1}$$

[K. Birman and R. van Renesse, private communication, 2001].

The transmission time is certainly not linear in b, since b occurs in the denominator; so, in general

$$\mathbf{E}(T(\ell, s, b)) \neq T(\mathbf{E}(\ell), \mathbf{E}(s), \mathbf{E}(b)).$$

Unlike selectivity, we can no longer take the expected values of these parameters if we want LEC plans. However, it turns out that by choosing the appropriate parameters, we can still use LSC optimization to produce LEC plans, as the following proposition shows.

Proposition 2.3: $\mathbf{E}(T(\ell, s, b)) = \mathbf{E}(\ell) + \mathbf{E}(s)\mathbf{E}(1/b)$.

Once again, the moral of the story is that, if we choose the right parameters (in this case s, ℓ, and $1/b$—note that $\mathbf{E}(1/b) \neq 1/\mathbf{E}(b)$ in general, so we cannot use b) and the right specific values, we can use LSC optimization to produce LEC plans.

2.3 When Is One Bucket Enough?

The examples in this section have shown that it is often possible to produce LEC plans using LSC optimization (i.e., using just one bucket). It is natural to wonder exactly when it suffices to use just one bucket.

A necessary condition is that there exists a parameter setting whose LSC plan is an LEC plan: if such a setting does not exist, then we cannot possibly produce an LEC plan by using just one bucket. In theory, this is also a sufficient condition. In practice, however, it is not enough that *there exists* some parameter setting: we must be able to *find one* efficiently, say in linear time. Some conditions that allow us to find such a parameter value in linear time are the following:

C1: If there is a dominant plan (e.g., see Section 2.1.1), then any value will do.

C2: If the cost of a plan is essentially linear in the parameter(s) of interest (e.g., see Section 2.1.2), then we can use the expected value of the parameter(s), since expectation is a linear operator.

C3: If the cost of a plan is a (sum of) product(s) of independent parameters (e.g., see Section 2.2), then we can use the expected value again, since the expected value of the product is the product of the expected values for independent parameters.

[3] For the general case, such as the Internet, there is no probability distribution that describes the transmission time [K. Birman and R. van Renesse, private communication, 2001], so we focus on the cases when the uncertainty can be represented by a probability distribution.

Sometimes parameters that do not fit the above criteria can be transformed so that they do (see Section 2.2).

Note that while it is possible to produce LEC plans using LSC optimization in the cases above, existing optimizers (including the DBMS we used in the experiments) do not take advantage of C2 and C3, since they do not use the expected value of the appropriate parameters. Thus, existing LSC optimizers are in general *not* producing LEC plans. These results show that, in many cases, with relatively little overhead, they could.

So far each scenario we covered satisfies one of the conditions above. In the following sections, we investigate scenarios in which LSC optimization, no matter how cleverly done, cannot produce an LEC plan.

3 More General Cost Functions

In [CHS99] and so far in this paper, we have implicitly assumed that "cost" is essentially "running time". While this assumption certainly seems reasonable and requires little motivation, is it always appropriate? If "cost" means "money" and we are paying a fixed amount per time unit, then "running time" and "cost" *are* essentially interchangeable. By minimizing expected running time, we are in fact minimizing expected cost. However, this is not always true, as the following examples show. In these examples, given a plan p, let $\mathbf{r}(p)$ be the random variable describing the running time of p ($\mathbf{r}(p)$ is a random variable since it is a function from parameters such as the amount of memory available or the selectivity to the actual running time). We also assume some distribution $\Pr$ on these underlying parameters. Thus, we can talk about $\Pr(\mathbf{r}(p) = 1$ minute)—the probability of the set of underlying parameters for which the running time of p is 1 minute.

Example 3.1: An investor may need the results of a query to decide whether to sell a certain stock within 10 minutes. After that, he stands to lose millions of dollars (either by not selling a stock that is going down or by selling a stock that is going up). In this case, the user does not care if a plan runs for 10 seconds or 9 minutes—either will meet the deadline. The user also does not care if a plan runs for 11 minutes or an hour—either will miss the deadline. Now suppose that the optimizer has to choose between plan p_1 and plan p_2 for the query. Suppose further that

- $\Pr(\mathbf{r}(p_1) = 9$ minutes$) = 1$,
- $\Pr(\mathbf{r}(p_2) = 10$ seconds$) = .5$, and
- $\Pr(\mathbf{r}(p_2) = 11$ minutes$) = .5$.

It is easy to see that p_2 has lower expected running time. However, p_2 also misses the deadline 50% of the time; the user would surely prefer p_1 over p_2 while, unfortunately, an optimizer that minimizes expected running time would just as surely pick p_2 over p_1. In this case, clearly, minimizing the expected running time is *not* the right thing to do. ▮

Example 3.2: Suppose an optimizer has to choose between plan p_1 and plan p_2 for some query. Suppose further that

- $\Pr(\mathbf{r}(p_1) = .5$ minutes$) = .9$,
- $\Pr(\mathbf{r}(p_1) = 10$ minutes$) = .1$,
- $\Pr(\mathbf{r}(p_2) = 2$ minutes$) = .9$, and
- $\Pr(\mathbf{r}(p_2) = 3$ minutes$) = .1$.

It is easy to check that the expected running time of p_1 is 1.45 minutes, which is less than the expected running time of p_2 (2.1 minutes). Thus, an optimizer that minimizes expected running time will pick p_1. But is this necessarily the "right" choice? A user could get upset if a query that usually takes 30 seconds were to suddenly take 10 minutes. So while there is no deadline to meet in this case, it is not so clear that minimizing expected running time is the right thing to do. ▮

The LEC approach deals naturally with situations where our goal is not necessarily to minimize running time. All that is required is that the user specify a function characterizing the "cost" of each possible running time. For example, in Example 3.1, we can take the "cost" of getting an answer in less than 10 minutes to be 1, and the "cost" of getting it in more than 10 minutes to be 1,000,000. Once we have a cost function, we simply choose the plan of least expected cost. Of course, there is no reason to assume that the cost of a plan depends only on running time. For example, it may matter whether the plan is blocking or produces results at a constant rate. Sometimes the order of the results may matter for display purposes. All of these factors can be taken into account in the cost function. As long as the plan induces a probability on each of the relevant events and a cost for each of them (e.g., a probability of blocking and a cost for blocking), we can still sensibly define the notion of a plan of least expected cost. We focus for simplicity here on cases where the cost function depends only on running time. Although this is somewhat restrictive, it does seem to cover many cases of interest.

Formally, we assume that there is a *plan-cost function* that takes as input a plan p and returns a random variable $\mathbf{c}(p)$, the cost of plan p as a function of the parameter settings. Up to now, we have assumed that $\mathbf{c}(p) = \mathbf{r}(p)$. Now we insist only that $\mathbf{c}(p)$ be a function of $\mathbf{r}(p)$. However, we do not require that $\mathbf{c}(p)(s)$ depend only on $\mathbf{r}(p)(s)$, where s is a parameter setting. We allow $\mathbf{c}(p)(s)$ to depend on global properties of $\mathbf{r}(p)$, such as its variance. Our goal is still to find LEC plans, but with respect to a cost function that is more general than $\mathbf{r}(p)$. We write $\mathbf{E}_{\mathbf{c}}^{\Pr}(p)$ to denote the expected value of $\mathbf{c}(p)$ (given the underlying distribution $\Pr$ on the parameter space).

One obvious way to get a plan-cost function is to define a *time-cost function,* that is, a function that characterize the cost of running for t time units.

Example 3.3: Consider Example 3.1. As we said above, one time-cost function that captures the cost of time to the investor is

$$c(t) = \begin{cases} 1 & \text{if } t \leq 10 \text{ and} \\ 10^6 & \text{if } t > 10. \end{cases}$$

Given a time-cost function c, we can define a plan-cost function $\mathbf{c}$ by taking $\mathbf{c}(p)(s) = c(\mathbf{r}(p)(s))$.

With such a plan cost function, it is easy to check that $\mathbf{E}_{\mathbf{c}}^{\Pr}(p) = \sum_t c(t) \Pr(\mathbf{r}(p) = t)$. Thus, in this case,

$$\mathbf{E}_{\mathbf{c}}^{\Pr}(p_1) = 1 < 0.5(10^6) + 0.5 = \mathbf{E}_{\mathbf{c}}^{\Pr}(p_2).$$

So if the optimizer minimized expected cost instead of expected running time, it would pick p_1 instead of p_2. ▮

Of course, once the cost function is no longer linear in running time, the expected cost of a plan is not the cost of the expected running time of the plan, so we can no longer use just one bucket in general.

The real question is often “Where are the costs coming from?” In a situation like Example 3.1, they clearly need to be obtained somehow from the user. This is a nontrivial problem. In a case such as Example 3.1, it may be quite possible for the user to provide a cost function. Moreover, in this case, even a qualitative description of the cost function may be enough to provide useful guidance in choosing a plan (and, in particular, to steer the system away from the obvious plan which just minimizes expected running time). However, in Example 3.2, it is not immediately clear how to choose a plan-cost function that captures users’ annoyance regarding variation in running time. We could, of course, just take the plan cost to be the variance. Then the plan of least cost would certainly be the one of minimum variance. However, in that case, one “optimal” plan would be to just slow down the computation to that of the worst case.[4] While this will minimize variance, it will probably not make users that happy. While not perfect, one way of capturing both preferences of the users (i.e., minimizing running time and minimizing variance) is to use a plan-cost function based on a time-cost function that is exponential in the running time. In Example 3.2, this will have the intended effect. To see this, let $c(x) = 2^x$ and $\mathbf{c}$ be based on c. Then

$$\begin{aligned} \mathbf{E}_{\mathbf{c}}^{\Pr}(p_1) &= 0.9(2^{0.5}) + 0.1(2^{10}) \\ &> 0.9(2^2) + 0.1(2^3) \\ &= \mathbf{E}_{\mathbf{c}}^{\Pr}(p_2), \end{aligned}$$

so the optimizer would pick p_2 instead of p_1 if it minimized expected cost. (Alternatively, we could just have the plan-cost function take the variance of $\mathbf{r}(p)$ into account; we consider this approach in Section 4.2.)

[4]While we may not always know the worst case, as long as there is some upper bound on running time, we can use that.

4 Least Expected User Cost Query Optimization

As we saw in the previous section, the plan-cost function $\mathbf{c}(p)(s) = \mathbf{r}(p)(s)$ does not always adequately capture the preferences of the user. If the plan-cost function is linear in running time (i.e., $\mathbf{c}(p)(s) = f(\mathbf{r}(p)(s))$ for some linear function f), then a dynamic programming algorithm (DPA) à la System R would produce an LEC plan. Such an algorithm does not, however, produce LEC plans in general, as the following example shows.

Example 4.1: Suppose that we adopt the plan-cost function from Example 3.3. Consider a two-stage join. Suppose that we have two possible plans, p_1 and p_2, for the first stage, and one possible plan, p_3, for the second stage. Suppose further that

- $\Pr(\mathbf{r}(p_1) = 1 \text{ minute}) = 0.9$,
- $\Pr(\mathbf{r}(p_1) = 11 \text{ minutes}) = 0.1$, and
- $\Pr(\mathbf{r}(p_2) = 9 \text{ minutes}) = 1$.

Note that this translates to $\mathbf{E}_{\mathbf{c}}^{\Pr}(p_1) = 0.9(1) + 0.1(10^6)$ and $\mathbf{E}_{\mathbf{c}}^{\Pr}(p_2) = 1$, so it seems like we should pick p_2 at the first stage.

Now suppose that $\Pr(\mathbf{r}(p_3) = 2 \text{ minutes}) = 1$. Then

- $\Pr(\mathbf{r}(p_1; p_3) = 3 \text{ minutes}) = 0.9$,
- $\Pr(\mathbf{r}(p_1; p_3) = 14 \text{ minutes}) = 0.1$, and
- $\Pr(\mathbf{r}(p_2; p_3) = 11 \text{ minutes}) = 1$.

So $\mathbf{E}_{\mathbf{c}}^{\Pr}(p_1; p_3) = 0.9(1) + 0.1(10^6)$ while $\mathbf{E}_{\mathbf{c}}^{\Pr}(p_2; p_3) = 10^6$. The upshot is that, although p_2 is the LEC plan at the first stage, $p_1; p_3$ is the global LEC plan. ▮

Thus, in general, a (naïve) DPA à la System R will not work, since keeping only a local LEC plan at each level might not give a global LEC plan. One way around this problem is to restrict our search space by using the "black box" approach of [CHS99]. The key idea there is to generate a relatively small set of candidate plans, in the hope that this set includes the optimal plan (or, at least, a plan close to optimal). We then compute the expected cost of each of the plans generated, and choose the best one. The approach to generating candidate plans used in [CHS99] was to run a standard LSC query optimizer as a black box, for a set of possible values of the parameters of interest. (That is, for each setting of the parameters, we compute the optimal LSC plan for that setting.) This approach can clearly be applied with more general cost functions as well. Another approach is to understand when dynamic programming will produce LEC plans. We consider this issue in the remainder of this section.

4.1 When Does the DPA Produce LEC Plans?

Let p be a left-deep plan that computes the join $A_1 \bowtie A_2 \bowtie \cdots \bowtie A_n$. (As is standard in the literature, we restrict to left-deep plans.) Let $p.sp$ be the subplan of p that computes $A_1 \bowtie \cdots \bowtie A_{n-1}$, let $p.ap$ be the method used to access A_n, and let $p.jm$ be the top-level join method of p (i.e., the method used to join $A_1 \bowtie \cdots \bowtie A_{n-1}$ with A_n). Note that

$$\mathbf{r}(p) = \mathbf{r}(p.sp) + \mathbf{r}(p.ap) + \mathbf{r}(p.jm).$$

A plan-cost function $\mathbf{c}$ is *additive* iff for all p_1 and p_2,

$$\mathbf{c}(p_1;p_2) = \mathbf{c}(p_1) + \mathbf{c}(p_2),$$

where the addition on the right is pointwise addition (that is, $(\mathbf{c}(p_1)+\mathbf{c}(p_2))(s) = \mathbf{c}(p_1)(s)+\mathbf{c}(p_2)(s)$). Of course, if we identify the cost with the running time, then the plan-cost function is certainly additive. Denote the plan picked by the DPA the *DPA plan.*

Theorem 4.2: *If $\mathbf{c}$ is additive, then the DPA plan is an LEC plan.*

The plan-cost functionin Example 4.1 is not additive, which is why the DPA does not work. Since in [CHS99] the plan-cost function is $\mathbf{r}(p)$, we were able to use the DPA to obtain LEC plans. Given that plan-cost function are not additive in general, there are two questions.

Q1: How non-optimal is the DPA plan?

Q2: How do we get LEC plans in general?

To answer the first question, let

$$\Delta_{\mathbf{c}}(p_1,p_2) \overset{\text{def}}{=} \mathbf{c}(p_1;p_2) - (\mathbf{c}(p_1) + \mathbf{c}(p_2)).$$

Note that $\Delta_{\mathbf{c}}(p_1,p_2)$, like $\mathbf{c}(p_1)$ and $\mathbf{c}(p_2)$, is a random variable, whose value depends on the settings of the underlying parameters. As the next theorem shows, we can bound the non-optimality of the DPA plan by a bound on $\Delta_{\mathbf{c}}(p_1,p_2)$.

Theorem 4.3: *Suppose that $\zeta_* \le \mathbf{E}^{\Pr}_{\Delta_{\mathbf{c}}}(p_1,p_2) \le \zeta^*$ for all p_1, p_2, and s. Let p_d be the DPA plan for a $(k+2)$-way join and let p_e be an LEC plan for that join; then $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_d) - \mathbf{E}^{\Pr}_{\mathbf{c}}(p_e) \le 2k(\zeta^* - \zeta_*)$.*

Now consider Q2. The key problem is that, in general, we could have $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1;p) < \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2;p)$ even though $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1) \ge \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2)$; this means that if we prune p_1 because $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1) \ge \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2)$, then we will not consider $p_1;p$, which is better than $p_2;p$. The following proposition shows that, if $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1) - \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2)$ is big enough, then the reversal cannot happen.

Proposition 4.4: *For all p_1, p_2, and p,*

1. $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1;p) \ge \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2;p)$ *iff*
2. $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1) - \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2) \ge \mathbf{E}^{\Pr}_{\Delta_{\mathbf{c}}}(p_2,p) - \mathbf{E}^{\Pr}_{\Delta_{\mathbf{c}}}(p_1,p)$.

Here is an easy corollary of Proposition 4.4.

Corollary 4.5: *Suppose that $\zeta_* \le \mathbf{E}^{\Pr}_{\Delta_{\mathbf{c}}}(p_1,p_2) \le \zeta^*$ for all p_1 and p_2. Then $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1) - \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2) \ge \zeta^* - \zeta_*$ implies $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1;p) \ge \mathbf{E}^{\Pr}_{\mathbf{c}}(p_2;p)$ for all p_1, p_2, and p.*

Corollary 4.5 suggests a modification of the DPA that will yield a global LEC plan. Suppose that the set of tables in the join is $S = \{T_1,\ldots,T_n\}$. For each $U \subseteq S$, let c_U be the expected cost of the LEC plans for U. For each U, keep all plans p such that $\mathbf{E}^{\Pr}_{\mathbf{c}}(p) - c_U \le (|S| - |U|)(\zeta^* - \zeta_*)$. An easy induction shows that in fact we do have that for each set U, the set of plans that we keep for U includes an LEC plan. The following example shows how the modified algorithm works for a two-stage join.

Example 4.6: Suppose that

$$c(t) = \begin{cases} t & \text{if } t \le 10 \text{ and} \\ t+10 & \text{if } t > 10. \end{cases}$$

Let $\mathbf{c}$ be the plan-cost function based on c. It is easy to check that $-10 \le \Delta_{\mathbf{c}}(p_1,p_2) \le 10$ for all p_1 and p_2. Consider a two-stage join. Suppose that we have three possible plans, p_1, p_2, and p_3 for the first stage and

- $\Pr(\mathbf{r}(p_1) = 5 \text{ minutes}) = 1.0$,
- $\Pr(\mathbf{r}(p_2) = 2 \text{ minutes}) = 0.5$,
- $\Pr(\mathbf{r}(p_2) = 11 \text{ minutes}) = 0.5$,
- $\Pr(\mathbf{r}(p_3) = 20 \text{ minutes}) = 0.9$, and
- $\Pr(\mathbf{r}(p_3) = 7 \text{ minutes}) = 0.1$.

Then we have

- $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1) = 5(1.0) = 5$,
- $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_2) = 2(0.5) + 21(0.5) = 11.5$, and
- $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_3) = 30(0.9) + 7(0.1) = 27.7$.

Thus, p_1 is the LEC plan at stage 1 and that $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_3) - \mathbf{E}^{\Pr}_{\mathbf{c}}(p_1) = 22.7 \geq 20$, so we may drop p_3.

Suppose that there is only one plan p at stage 2 and $\Pr(\mathbf{r}(p) = 6 \text{ minutes}) = 1$. Then

- $\Pr(\mathbf{r}(p_1;p) = 11 \text{ minutes}) = 1.0$,
- $\Pr(\mathbf{r}(p_2;p) = 8 \text{ minutes}) = 0.5$,
- $\Pr(\mathbf{r}(p_2;p) = 17 \text{ minutes}) = 0.5$,
- $\Pr(\mathbf{r}(p_3;p) = 26 \text{ minutes}) = 0.9$, and
- $\Pr(\mathbf{r}(p_3;p) = 13 \text{ minutes}) = 0.1$.

So we have that

- $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1;p) = 21(1.0) = 21$,
- $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_2;p) = 8(0.5) + 27(0.5) = 17.5$, and
- $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_3;p) = 26(0.9) + 7(0.1) = 27.7$.

Note that we indeed have $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_1;p) \leq \mathbf{E}^{\Pr}_{\mathbf{c}}(p_3;p)$, so we can indeed prune p_3 (since this is a two-level join). However, $\mathbf{E}^{\Pr}_{\mathbf{c}}(p_2;p) < \mathbf{E}^{\Pr}_{\mathbf{c}}(p_1;p)$, so we cannot prune p_2 and still obtain a global LEC plan. ∎

We have seen how we can deal with non-additive plan-cost functions in this section. However, depending on the sizes of ζ^* and ζ_*, we might have to keep a lot of plans for each subset. If we have some information about a non-additive $\mathbf{c}$, can we do better? In the next section, we investigate the case of a cost function of particular interest, variance.

4.2 Variance Minimization

As we have argued, users might prefer plans with more stable behavior, so users might prefer plans with low variance (see Example 3.2). As we have seen, just taking the plan-cost function to be the variance of the running time leads to a preference ordering over plans that would almost certainly not satisfy most users. However, another approach to dealing with variance might be to consider a plan-cost function $\mathbf{c}$ that takes into account both the running time and the variance, for example $\mathbf{c}(p) = \alpha\mathbf{r}(p) + \beta\mathbf{V}(\mathbf{r}(p))$, where $\mathbf{V}(x)$ denotes the variance of random variable x. The user can then choose α and β to reflect her relative preference for running time vs. variance. If this is in fact the user's plan-cost function, then there is still the problem of choosing the LEC plan. In this section, we show how to exploit the properties of variance to get approximations to the LEC plan.

Recall that $\mathbf{V}(x) = \mathbf{E}(x^2) - \mathbf{E}(x)^2$. Let the *covariance* of p_1 and p_2 be denoted by $\mathbf{\Lambda}(p_1, p_2)$. Recall that $\mathbf{\Lambda}(x, y) = \mathbf{E}(xy) - \mathbf{E}(x)\mathbf{E}(y)$. Two random variables x and y are

- *uncorrelated* iff $\mathbf{\Lambda}(x, y) = 0$,
- *positively correlated* iff $\mathbf{\Lambda}(x, y) > 0$, and
- *negatively correlated* iff $\mathbf{\Lambda}(x, y) < 0$.

Note that $\mathbf{V}(x + y) = \mathbf{V}(x) + \mathbf{V}(y) + 2\mathbf{\Lambda}(x, y)$, so $\mathbf{V}(x + y) = \mathbf{V}(x) + \mathbf{V}(y)$ for uncorrelated random variables x and y. Let $\mathbf{E}^{\Pr}_{\mathbf{c}}\mathbf{c}_{\mathbf{V}}(p) = \mathbf{V}(\mathbf{r}(p))$ (which means that $\mathbf{c}_{\mathbf{V}}(p)$ is a constant function, independent of the parameter setting). The random variables that we encounter in doing query optimization (i.e., the running times of subplans of a particular plan p) are typically *non-negatively correlated* (i.e., $\mathbf{\Lambda}(x, y) \geq 0$). For example, at any particular stage, the size of the result and the running time both depend on the input sizes, and the bigger the input sizes, the longer the running time and the bigger the result. A bigger result means longer running time for the next stage, so the running time of the current stage and the next stage are also (non-negatively) correlated. For the rest of this section, we will assume that all random variables are pairwise non-negatively correlated.

Recall that Theorem 4.3 gives a bound on how non-optimal a DPA plan could be that depends on $\mathbf{E}^{\Pr}_{\Delta_{\mathbf{c}}}(p_1, p_2)$. As the following theorem shows, we can give a tight bound for $\mathbf{c}_{\mathbf{V}}$ that does not depend on $\Delta_{\mathbf{c}_{\mathbf{V}}}$.

Theorem 4.7: *If $\mathbf{E}^{\Pr}_{\mathbf{c}_{\mathbf{V}}}(p_1) \leq \mathbf{E}^{\Pr}_{\mathbf{c}_{\mathbf{V}}}(p_2)$, then*

$$\frac{\mathbf{E}^{\Pr}_{\mathbf{c}_{\mathbf{V}}}(p_1;p)}{\mathbf{E}^{\Pr}_{\mathbf{c}_{\mathbf{V}}}(p_2;p)} \leq 2$$

and the bound is tight.

Theorem 4.7 shows that the DPA is off by at most a factor of two for two-stage joins. The next theorem shows that the error grows only linearly with respect to the depth of the join.

Theorem 4.8: *If* $\mathbf{E}^{\Pr}_{\mathbf{cV}}(p_{i,j_i}) \leq \mathbf{E}^{\Pr}_{\mathbf{cV}}(p_{i,3-j_i})$ *for all* $1 \leq i \leq n$ *and* $1 \leq j \leq 2$*, then for all* $1 \leq k_1, \ldots, k_n \leq 2$*,*

$$\frac{\mathbf{E}^{\Pr}_{\mathbf{cV}}(p_{1,j_1}; \cdots; p_{n,j_n})}{\mathbf{E}^{\Pr}_{\mathbf{cV}}(p_{1,k_1}; \cdots; p_{n,k_n})} \leq n.$$

Essentially, Theorem 4.8 shows that the DPA plan is no more than a factor n away from any plan, so in particular, the DPA plan is no more than a factor n away from an LEC plan.

These bounds approximations are still nowhere near as good as we would like. We expect that in practice, things will be much better. After all, Just because there is some sequence join for which the DPA plan has variance a factor of n times that of the optimal plan does not mean that that is what will typically happen. Our initial experiments bear this out. We are currently trying to find some additional assumptions that guarantee that we will get a better approximation using the DPA.

5 Conclusions

We have investigated the extent to which we can use specific parameter settings (i.e., LSC optimization) to produce LEC plans. Somewhat surprisingly, we found that in many cases of interest, LSC optimization could produce LEC plans. However, we must be careful to choose the parameters and their settings appropriately. This may involve transforming a distribution on (say) user input values, to a distribution on selectivity, so that we can compute the expected selectivity with respect to this distribution. Current implementations of query optimizers do not seem to take advantage of probabilistic information, even when it is readily available, so that even in cases where there is a reasonable specific setting of the parameters that can be used, this is not the setting that is actually used in the computation. (For example, it seems that for the DBMS we tested, the setting it uses is the first one given.) In cases where one plan dominates all others no matter what the parameter value (as is the case in some of the examples in Section 2.1.1), then the specific value chosen does not matter. Otherwise, of course, it could make a big difference. We see one of the contributions of this paper as clarifying exactly when we can use LSC optimization (and what parameter setting to use in these cases).

On the other hand, particularly in the case where running time is not the appropriate cost measure, LEC optimization becomes particularly important. It can be used to capture things like deadlines, a preference for minimizing variance, and features unrelated to running time, like the issue of whether or not there is blocking [HHW97]. However, considering more general cost functions opens up a host of new issues. For one thing, it requires constructing an appropriate cost function, either from information provided by users (which may be difficult to get) or through an understanding of the application domain. Secondly, it requires designing algorithms that can take advantage of this information to produce high-quality plans. We are currently investigating both problems.

Acknowledgements

We would like to thank Ken Birman and Robbert van Renesse for discussions regarding transmission time in networks.

References

[CHS99] F. Chu, J. Y. Halpern, and P. Seshadri. Least expected cost query optimization: an exercise in utility. In *Proc. 18th ACM Symp. Principles of Database Systems*, pages 138–147, 1999.

[GBC98] G. Graefe, R. Bunker, and S. Cooper. Hash joins and hash teams in microsoft sql server. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98*, pages 86–97, 1998,

[HHW97] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. *Proc. SIGMOD 1997*, pages 171–182, 1997.

[HKWY97] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. *Proc. 23rd VLDB*, pages 276–285, 1997

[INSS92] Y. Ioannidis, R. Ng, K. Shim, and T.K. Sellis. Parametric Query Optimization. pages 103–114, 1992.

[LG98] P. Larson and G. Graefe. Memory management during run generation in external sorting. *Proc. SIGMOD 1998*, pages 472–483, 1998.

[ROH99] M. T. Roth, F. Özcan, and L. M. Haas. Cost models do matter: Providing cost information for diverse data sources in a federated system. *Proc. 25th VLDB*, pages 599–610, 1999.

[SAC$^+$79] P. G. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access Path Selection in a Relational Database Management System. *Proc. ACM SIGMOD 1979*, pages 23–34, 1979.

[TPPC99] Transaction Processing Performance Council. *TPC Benchmark™ H (Decision Support) Standard Specification Revision 1.2.1*. Transaction Processing Performance Council, 1999.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge Base Systems, Volume II: The New Technologies*. Computer Science Press, 1989.