***Science* Manuscript Template**

**General Instructions on using this template and submitting a manuscript to *Science*:** Thank you for preparing a manuscript for submission to *Science*. Using this template, or following the guidelines below, will help us in processing your paper. Our goal is to be able to identify each section of your manuscript so that we can accurately record the title, authors, abstract, etc. and also be able to enrich it by including reference links and an accurate layout.

Please use the actual template, which starts on page 2. When you are ready to submit, please delete the text on this cover page.

You can submit your paper at <https://cts.sciencemag.org>

Additional instructions are available at  [http://www.sciencemag.org/authors/science-information-authors](http://www.sciencemag.org/site/feature/contribinfo/index.xhtml)

If you are using LaTeX , please convert your paper into a Word docx file if possible. If this is not possible, please use our LaTeX template and upload a PDF version of your paper. Some conversion approaches are available here: <http://www.tug.org/utilities/texconv/textopc.html>

So that we can identify the parts of your paper, and even if you do not use our template, please begin each section with the specific key words listed below, some of which are followed by a colon. Please do not use paragraph breaks in the title, author list, or abstract.

**Title:**

**Authors:**

**Affiliations:**

**One Sentence Summary:**

**Abstract:**

**Main Text:**

**References and Notes** (followed by a numbered list)

**Acknowledgements:**

**Fig. #:** (Begin each figure caption with a label, “**Fig. 1**.” for example, as a new paragraph.).

**Table #:** (Begin each table caption with a label “**Table 1.”**, etc.) as a new paragraph.

**Supplementary Materials:** a list of the supplementary materials, followed by the actual text of the Supplementary Materials.

Several of these headings are optional, for example, not all papers will include a one-sentence summary, figures or tables, or supplementary material.

Please use the .docx format (all version after Word 2007).

More specific formatting instructions are provided in the actual template, which follows.

Title: How to Format a Science Paper (replace with your real title)

**Authors:** The author list should be one single paragraph (no breaks). Authors should be listed by first name or initial followed by last name and separated by commas. Use superscript numbers to link affiliations, and symbols \*†‡ for author notes. For example, X. Jones1\*, P. Smith1,2

**Affiliations:**

1Affiliations should be preceded by superscript numbers corresponding to the author list, and each affiliation should end with a period.

2Each affiliation should be a separate paragraph.

3You can include group authors, but please include a list of the actual authors in the Supplementary Materials.

\*Correspondence to: include the email addresses of the corresponding author(s). Please use the asterisk (\*) symbol for the corresponding author information.

†Additional author notes should be indicated with symbols (for example, for current addresses).

**Abstract**: The abstract should be about 100-150 words, and organized in this structure: An opening sentence that sets the question that you address and is comprehensible to the general reader, background content specific to this study, results, and a concluding sentence. It should be one paragraph only.

**One Sentence Summary:** A brief summary of the main result of your paper, without excessive jargon and less than 150 characters.

**Main Text:** In general, this should include a brief (1-2 paragraph) introduction, followed by a statement of the specific scope of the study, followed by results and then interpretations. Please avoid statements of future work or claims of priority, and avoid repeating the conclusions at the end. All of the Figures and Tables should be cited in order, including those in the supporting online material (which should be cited as, for example, figure S1, and table S1). You can include page breaks if you would like to place the figures within the text.

**Subheads**. These can be included in Research Articles or Reviews and should be brief.

References should be cited in parentheses with an italic number (*1*). Multiple reference citations are separated by commas (2, 3) or if a series, dashes (4-6). References are cited in order by where they first are called out, through the text, then the notes, captions, then through the supplementary material.

Equations can be included. We do not recommend using the native Word 2007, 2008, 2010 or 2011 equation editor. This can in some cases produce less reliable MathML, the online markup language we use, which may result in display errors. Instead, use the legacy equation editor in word (Insert menu; select insert object; select word equation) or use Mathtype (recommended).

References and Notes:

1. There is only one reference list spanning the text, figure captions and supplementary materials. Do not include a second reference list in the supplementary materials section. Reference only cited in the supplementary materials section are not counted toward length guidelines.
2. Each reference should be on a separate line ending in a period. For a style guide, see [http://www.sciencemag.org/authors/instructions-preparing-initial-manuscript](http://www.sciencemag.org/authors/instructions-preparing-initial-manuscript%20)
3. You can include titles in references and full page ranges. Titles are not counted toward length guidelines
4. Please include the above heading “References and Notes:”
5. You can use a numbered list in Word.
6. Each reference should have a separate number.
7. Please do not mix in references with explanatory notes.

**Acknowledgments:** The acknowledgments should include a statement about where the data reported in the paper are presented, archived, or available (for example, in the Supplementary Materials or in a community archive). If in an archive, include the accession number or a placeholder for it. Please also include relevant funding information such as grant numbers and funding agencies. You can also include a statement of author contributions here or in the Supplementary Materials.

The acknowledgements can be multiple paragraphs. Acknowledgments follow the references and notes but are not numbered.

**Fig. 1**. The figure caption should begin with an overall descriptive statement of the figure followed by additional text. They should be immediately after each figure. Figure parts are indicated with capital letters (**A**). If you prefer, you can place both the actual figures and captions logically through the text near where they are cited rather than at the end of the file (but not both). If a paragraph in the main text begins with the name of a figure, write out “Figure” in full (e.g., <para>“Figure 2 shows….”)

**Fig. 2.** You can place graphics above each equation as part of this file.

**Table 1.** Start this caption with a short description of your table. Format tables using the Word Table commands and structures. Do not create tables using spaces or tabs characters.

Supplementary Materials:

Materials and Methods

Figures S1-S#

Tables S1-S#

Movies S1-S#

Audio Files S1-S#

External Databases S1-S#

References (*##-##*)

Supplementary Materials:

This section includes the actual text of the Supplementary Materials, which can include any or all of the preceding items, and figure captions and tables that can easily be incorporated into one supplementary material file. Please edit the list above as appropriate and include it at the end of your main paper. If there are additional files that cannot be easily accommodates (e.g., movies or large tables), please include captions here.

**Materials and Methods:** Can include the Materials and Methods here. Additional references should be cited here and included in the main reference list.

**Any Additional Author notes:** For example, author contributions or a list of group authors.

**Title:** **A Fundamental Limit Within Cost-Based Database Query Optimizers**

**Authors:** Richard Snodgrass1\*, Sabah Currim1, and Young-Kyoon Suh1\*†

**Affiliations:**

1University of Arizona, Tucson, AZ 85721 USA.

\*Correspondence to: {[rts,yksuh}@cs.arizona.edu](mailto:rts,yksuh%7d@cs.arizona.edu)

†Current address: Korea Inst. of Sci. and Tech. Info. (KISTI), Daejeon, Rep. of Korea 34141.

**One Sentence Summary:** We identify a fundamental limit on the number of query evaluation operators and show that database management systems may have reached that limit.

**Abstract**: Implementers of database management systems (DBMSes) add query evaluation operators to increase the performance of these systems. We investigate the conventional cost-based query optimization phase within a DBMS, which ostensibly finds the fastest query execution plan from a potentially large set of enumerated plans, all of which correctly execute the specified query. Suboptimality is indicated by the existence of a query plan that performs more efficiently than the DBMS's chosen plan, for the same query. Through a novel experiment that examines the plans for thousands of queries run on one hundred thousand query/cardinality combinations on four popular DBMSes, we present evidence for a previously-unknown upper bound on the number of operators (which we previously demonstrated to be a major source of suboptimality) a DBMS may be able to support. We show that this upper bound may have already been reached by one or more extant DBMSes.

**Main Text:** Database management systems (DBMSes) underlie information systems and hence optimizing their performance is of critical importance. The relational DBMS's query optimizer plays an important role ostensibly finding the fastest query execution plan from a potentially large set of enumerated plans, all of which correctly compute the specified query. But what if the optimizer *doesn't*: what if it selects the wrong plan?

A possible implication of a previously-validated model is that as query evaluation operators are added to the DBMS, suboptimality will increase. Given the (negative) influence of having more operators on the suboptimality that our model predicts, is there actually a *limit* to the number of query operators beyond which the performance of the DBMS begins to degrade? We present empirical evidence, drawn from 13,000 hours of query executions, of a previously-unknown upper bound on the number of database query operators that a DBMS is able to support and demonstrate that this upper bound may have already been reached by one or more extant DBMSes.

There has been extensive work in query optimization over the last 40 years (*1-2*), during which a particular quite effective paradigm had taken hold, in both open-source and proprietary DBMSes. In this over-arching paradigm, query optimization and evaluation proceeds in several general steps (*3*). First, the query is translated into alternative query evaluation plans expressed in the relational algebra via *query enumeration*. The cost of each plan is then estimated and the plan with the lowest estimated cost is chosen. These steps comprise query optimization, specifically *cost-based query optimization* (*4, 6*). The selected query plan is then evaluated by the query execution engine which implements a set of physical operators, often several for each logical operator (*5*).

To determine the best query access plan, the cost model estimates the execution time of each plan. There is a vibrant literature on this subject (*7-8*), including proposals for histograms, sampling, and parametric methods. Again, most of these papers are engineering studies, providing new techniques that improve on the state-of-the-art through increased accuracy or performance. There have also been a few mathematical results, such as “the task of estimating distinct values is *provably* error prone, i.e., for any estimation scheme, there exists a database where the error is significant” (*6*).

However, even with great effort over decades, optimizers as a general class are still poorly understood. As has been observed, “query optimization has acquired the dubious reputation of being something of a black art'” (*9*). DeWitt has gone farther, stating that `”query optimizers [do] a terrible job of producing reliable, good plans [for complex queries] without a lot of hand tuning” (*10*, page 59).

We previously articulated and tested a predictive causal model of the aforementioned suboptimality that put forth four independent constructs: optimizer complexity, schema complexity, query complexity, and data complexity, one intervening construct: plan space complexity, and one dependent construct: query suboptimality, and showed through correlational and regression analysis across four DBMSes and thousands of queries that this model is strongly supported (*11*). The finding of most relevance here is that the number of operators in the DBMS explains 9.21% of the variance of query suboptimality.

In order to increase query performance, DBMSes are extended over time with new query operators. Also over time, DBMSes are extended with new storage and indexing structures, which themselves elicit new query operators. Each subsequent generation of the DBMS thus supports an ever-expanding collection of operators, with the current incarnation the most recent within a series of DBMS *generations*.

Consider a Gedanken experiment that examines the plan selected for each query at each cardinality (termed a *Q@C*), *for each DBMS generation*. In early generations, there will be few plan changes for a given query as the cardinality varies, simply because there are few operators available. For subsequent generations, some of the Q@Cs will be associated with different plans enabled by the new operators that were added.

In many (hopefully most) cases, a new plan selected by a generation will be more efficient than the plan selected by the immediate previous generation. After all, that is the very reason the new operator(s) were added to that subsequent generation. However, in the presence of suboptimality, sometimes the new plan at that Q@C is *not* preferable, as that DBMS generation's query optimizer selected the wrong plan. Indeed, the query optimizer also evolves and improves with each new generation, in part to minimize the chance of selecting the wrong plan.

As a refinement, let's assume each subsequent generation adds a single operator. Starting with a fixed set of queries, for each DBMS, we run each generation on each Q@C and then sum up the query times to compute a *per-generation (total) time* for that DBMS. We can also sum over the four DBMSes, to see how this *overall per-generation time* (a single number for each generation number) varies with generation.

The underlying question then becomes, does an additional operator made available in a subsequent generation of the DBMS actually help or hurt? More fully, is the predicted increase in suboptimality originating from that added operator compensated for by the increased performance afforded by that operator? Our causal model predicts that suboptimality will increase with generation number. Given that each new operator will improve a *shrinking* subset of Q@Cs for any given query, while the suboptimality occurs in a portion over an *expanding* subset of Q@Cs, our causal model also implies that a curve plotting performance across generation will fall (improve) with each successive generation, but then start to level off as suboptimality becomes more prevalent. Eventually, the per-generation time will either asymptotically approach a horizontal line (with the first derivative approaching 0 from below), or worse: the first derivative will change to positive, with the per-generation time over the queries we are studying actually *increasing* with DBMS generation. In either case, the causal model predicts a point where the *increase in execution time due to suboptimality obviates the decrease enabled by the new operator*.

While we do not have access to prior generations of our DBMSes (which is why the previous discussion was in the form of a Gedanken experiment), we can approximate this with an experiment using data collected on the current version available for each DBMS, to *simulate* the prior generations, each successively having a fewer number of operators, and thus a smaller set of realizable query plans. (The effect will probably be smaller in our simulation, as we are nonetheless using the most recent query optimizer in all of the simulated generations; only the available simulated operators in a DBMS will vary across generations.)

In our study, we randomly generated queries in the SQL standard query language (*12*), each a select-project-join (possibly with an aggregate) query, with a few attributes in the SELECT clause, a few relations referenced in the FROM clause, a few equality predicates in the WHERE clause, and zero or one aggregate functions in the SELECT clause. For some of the queries, we ensured that there was at least one primary key attribute in one of the comparisons in the WHERE condition.

We generated two sets of tables, one without any secondary indexes and one with a key specified for each relation on the non-key (that is, other than the first) attribute, for each of four relations per database. We then varied the cardinality of one of the relations to generate 200 versions of this *variable* relation: from 2M tuples to 10K tuples in steps of 10K tuples (for one DBMS, from 60K to 300 rows in steps of 300 tuples). We generated four successive data sets: (i) one with no primary key, no duplicate rows, and no skew, (ii) one adding small skew (0.1% duplicates), (iii) one adding a primary key of the first column of each relation, and (iv) one adding the specification that the other three columns should be associated with a secondary index, only on that one column.

We extracted the query plan selected by the optimizer at each cardinality, identifying the adjacent cardinalities when the plan changed. We then ran the query on both cardinalities (Q@Cs), in each identified pair, termed a *change point*.

Given the set of plans from the change points for each DBMS, and from each plan a set of operators, we assign each DBMS operator to a separate DBMS generation based on a systematic ordering of the operators.

The question then becomes, does a plan change by a subsequent generation (enabled by the additional operator(s) in that generation) represent a win (runs faster) or a loss (runs slower, because of suboptimality)? More broadly, do the Q@Cs in the aggregate enabled by each succeeding generation continue to overcome the increasing burden of suboptimality?

Define *ops(p)* for a given plan *p* to be the set of operators present in that plan, with some operators perhaps repeated in that plan. A generation (also a set of operators) is *applicable* to a plan *p* if . By definition, if a generation is applicable to a given plan, it is applicable to all subsequent generations. The smallest such generation is termed the *minimally applicable generation*, or *mingen*. As each change point consists of a pair of adjacent Q@Cs, we have two generations to consider. We extrapolate the measured query time of the lower generation to the cardinality of the higher generation to determine whether the plan at the higher generation is suboptimal, by computing a *relative delta*, defined as the measured time of the plan at the higher generation minus the measured time of the plan at the lower generation extrapolated to that same cardinality, divided by the measured time of the plan at the higher generation. The relative delta is thus scaled, and has a range of -1 (indicating maximum non-suboptimality, where the plan at the higher generation is faster than that of the lower generation) to 1 (indicating maximum suboptimality, where the plan at the higher generation is slower). (In the former case of a negative relative delta, we can't state unequivocally that the associated plan is optimal, because there may be yet another plan involving the operators within that generation that is even faster.)

This analysis is for *a single pair of adjacent Q@Cs for a single query running on a specific DBMS*, providing a *relative delta* for the minimally applicable generation. A positive relative delta indicates that the later generation chose a suboptimal plan; a negative relative delta indicates the later generation did not. The relative delta is a percentage difference, and so is not affected by the absolute magnitude of the run time nor by the cardinality in question. Indeed, because it is a percentage difference, the relative delta is not affected by the query nor even which DBMS is involved. We associate this relative delta with the later generation, for it is that generation which had the choice between the two plans for that query.

Out of 112,118 Q@Cs examined, a total of 66,769 pairs/change points were identified and timed via our timing protocol (*13*). For the great majority (94%), both Q@Cs were of the same generation. We identified 1378 pairs with a non-suboptimal plan at the later generation and 1701 pairs with a suboptimal plan at the later generation that satisfied the requirements listed in the Supplementary Materials, including having a different generation number for the lower and upper plans.

**Trends Across DBMS Generations**. Consider how the *average* relative delta, computed across queries and DBMSes, and then computed for each generation, might behave across successive generations. The average relative delta is a characterization of the aggregate impact of that generation, providing a quantitative estimate of the benefit of adding that operator. (We use average so that each point is not impacted by the number of pairs over which that point is computed.)

Our causal model asserts that as the number of operators increases in subsequent generations, suboptimality will also increase. Our Gedanken experiment takes this behavior and predicts that as the generations contain successively greater numbers of operators, suboptimality will increase. If we plot the performance of the DBMS on the *y*-axis, say as the total time for a workload consisting of a set of queries over a prescribed data set, for a sequence of *DBMS generations* arranged on the *x*-axis, the average relative delta is in some way a characterization of the *slope* of this relationship. A negative average relative delta implies that the indicated generation is doing a good job, with less suboptimality, and so the total workload execution time will go down; a positive average relative delta implies that the suboptimal decisions are dominating, indicated by the total workload execution time going up for that generation.

We expect that the average relative delta for the first few generations will be negative, reflecting new operators that improves some plans, a natural result of the efforts of DBMS developers to increase performance over successive generations of their DBMS. With a negative slope, the

performance plot will decrease over successive DBMS generations.

However, each new operator is applicable to a successively smaller portion of the queries, and perhaps over a successively smaller portion of the cardinality space. As already noted, our causal model predicts that the prevalence of suboptimality would increase as the number of operators available increased. It seems that even with DBMS implementers doing smart things, these two considerations predict that the average relative delta (itself a slope of the performance curve) will *increase*, as suboptimality (a positive relative delta) becomes more prevalent.

This analysis suggests then that the performance curve will thus start to level off. That raises the possibility of the performance curve either asymptotically approaching a horizontal line (the first derivative approaching 0), or worse: the first derivative (the average relative delta) changing to positive, with the average time over the queries we are studying actually *increasing* with DBMS generation.

Table 1 aggregates the results over the four DBMSes, which had generations ranging from five to thirteen. This is the first study we are aware of that compares the generational trends of DBMSes over quite disparate code bases, thus getting at fundamental trends. The second column states the number of change points added by that generation. The next three columns show the *cumulative relative delta* for that generation, defined as the sum of the average relative delta for those change points associated with plans having only operators in its generation, divided by the number of change points. This is the relevant number for a generation, as it includes all plans that would have been emitted by that generation, using the operators at that generation's disposal.

The next-to-last column brings the non-suboptimal and suboptimal together, stating the *net cumulative relative delta,* gathering the change points with plans that would have been generated by that DBMS generation together. We see that the net starts out at -0.24, indicating that the new operators are *decreasing* the query time for the workload. Unfortunately, this happy situation starts to deteriorate: with each successive generation, the improvement is less. This is as predicted: the optimizer is struggling with both more options (plans over the available operators) to select from and a diminished opportunity to make a significant improvement.

We also see that at Generation 10, the net actually becomes very slightly positive: meaning that the performance curve has hit a minimum and is now on its way up, towards slower performance. This trend increases in later generations.

**Have Modern DBMSes Hit the Wall?** The final column integrates the net cumulative relative delta to produce a unit-less *relative performance*, a simulation of how the four DBMSes together would have performed (say, in total execution time) on the experiment workload of the experiment, or rather the 1378 change points selected by the criteria discussed above. (We emphasize that our experiment estimates just the first derivative of the performance graph, the net cumulative relative delta, and then only for a small number of Q@Cs, and then only as a relative measure, between -1 and 1. It is thus only very roughly indicative of the *shape* of the performance graph.) The relative performance gets better (faster, indicated by a falling value), but that trend again stops at generation 10, after which the relative performance actually gets worse (slower, indicated by a rising value).

Figure 1 tells the story graphically. Figure 1(A) plots the fourth column of Table 1 as the top line (suboptimal cumulative relative data), the third column of Table 1 as the bottom line (non-suboptimal cumulative relative delta), and the fifth column of Table 1 as the middle line (net cumulative relative delta) in the center. This middle line has a least squares slope of 0.024, which is consistent with the transition from a generation being net beneficial to actually being slightly suboptimal around generation 10 and more so at later generations. Figure 1(B) plots the last column of Table 1, the relative performance across generation. This illustrates visually that the performance time falls (gets better) until around generation 10, after which adding an operator might have an overall detrimental impact, actually slowing down the query.



(**A**) Cumulative Average Relative Delta over Generations (**B**) Performance over Generations

**Fig. 1.** Trend of Performance Improvement over Generations

The engineering perspective predicts that the overall per-generation time would monotonically decrease as the generations add query evaluation operators. The scientific perspective, using the prior causal model, reaches the opposite conclusion: the per-generation time will either asymptotically approach a horizontal line (with the first derivative approaching 0 from below), or worse: the first derivative will change to positive, with the per-generation time over the queries we are studying actually increasing with DBMS generation. In either case, the causal model predicts a point where the increase in execution time due to suboptimality obviates the decrease enabled by the new operator. The results of this study clearly support the latter (cf. Figure 1(B)).

This highly aggregated result, extracting 3000-odd change-point pairs drawn from over 100,000 Q@Cs implies that these four particular DBMSes, taken together, might be close to or have gotten to the point where adding an operator actually slows down the average query.

It is important to emphasize that we can't say whether any of these DBMSes have actually transitioned to where, for this class of queries, the errors of the suboptimal change points overwhelm the benefits of the non-suboptimal change points. Presumably, the DBMS vendors have performed extensive tests to ensure that the operator added to each generation did in fact effect a speedup on the representative workloads that they use in evaluating their optimizer enhancements. However, we *can* say that (a) the trends observed strongly point to a decreasing benefit and an increasing cost, as predicted by the simple arguments made above, and (b) if current DBMSes haven't yet reached the point of diminishing returns, that possibility exists.

We emphasize that in this experiment, we are looking at quite simple queries, over a quite limited range of data, with only a small percentage (3%) of change points identified in our experimental protocol. On the other hand, simpler queries are thought to be easier to generate good plans than more complex queries. Also, relational data is generally much less uniform in its (i) data types (we used only integer data), (ii) values (the values in our relations are quite evenly distributed), (iii) schemas (the schemas of our relations are identical and extremely simple), and (iv) range of relation cardinalities (the smallest relation is 1/200 of the largest relation). The simplicity of queries, schemas, and relations in this study should independently, and certainly in concert, minimize the suboptimality observed.

**References**

1. Y. Ioannidis, Query Optimization. *ACM Comp. Surv.* **23**, 121-123 (1996).

2. M. Jarke and J. Koch, Query Optimization in Database Systems. *ACM Comp. Surv.***16**, 111-152 (1984).

3. R. Ramakrishnan and J. Gehrke, *Database Management Systems*, Third Edition (McGraw-Hill, 2003).

4. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lori, and T. G. Price, Access Path Selection in a Relational Database System. In *Proc. ACM SIGMOD Conf.,* 23-34 (1979).

5. G. Graefe, Query Evaluation Techniques for Large Databases. *ACM Computing Surveys* **25,** 73-170 (1993).

6. S. Chaudhuri, An Overview of Query Optimization in Relational Systems. In *Proc. ACM PODS Conf.,* 34-43 (1998).

7. Y. Ioannidis, The History of Histograms (abridged). In *Proc. VLDB Conf.* 19-30 (2003).

8. M V. Mannio, P. Chu, and T. Sagar, Statistical Profile Estimation in Database Systems. *ACM Comp. Surv.* 20, 192-221 (1988).

9. B. Babcock and S. Chaudhuri, Towards a Robust Query Optimizer: A Principled and Practical Approach. In *Proc, ACM SIGMOD Conf.*, 119-130(2005).

10. M. Winslett, David DeWitt Speaks Out. *ACM SIGMOD Record* 31, 50-62 (2002).

11. R. T. Snodgrass, S. Currim, and Y.-K. Suh, A Causal Model of DBMS Suboptimality (Tech. Rep., Dept. Comp. Sci., Univ. of Arizona, 2017).

12. J. Melton (editor), ISO/IEC 9075, **Database Language SQL:2011 Part 2: SQL/Foundation**, 2011.

13. S. Currim, R. T. Snodgrass, Y.-K. Suh, and R. Zhang, DBMS Metrology: Measuring Query time. *ACM TODS* 42 (1):3:1-42 (2016).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Generation* | *Total*  *Number of*  *Change Points* | *Non-Suboptimal*  *Cumulative*  *Relative Delta* | *Suboptimal*  *Cumulative*  *Relative Delta* | *Net Cumulative*  *Relative Delta* | *Relative*  *“Performance”* |
| 1 | — | 0.00 | 0.00 | 0.00 | — |
| 2 | — | 0.00 | 0.00 | 0.00 | — |
| 3 | — | 0.00 | 0.00 | 0.00 | — |
| 4 | 26 | -0.28 | 0.2 | -0.24 | -0.24 |
| 5 | 493 | -0.130 | 0.077 | -0.045 | -0.285 |
| 6 | 395 | -0.203 | 0.102 | -0.103 | -0.388 |
| 7 | 599 | -0.243 | 0.196 | -0.0800 | -0.468 |
| 8 | 316 | -0.241 | 0.231 | -0.0256 | -0.493 |
| 9 | 7 | -0.241 | 0.230 | -0.0257 | -0.519 |
| 10 | 185 | -0.241 | 0.245 | 0.008 | -0.512 |
| 11 | 996 | -0.225 | 0.242 | 0.0296 | -0.482 |
| 12 | 59 | -0.225 | 0.245 | 0.0346 | -0.448 |
| 13 | 12 | -0.225 | 0.246 | 0.0360 | -0.412 |
| Cumulative | 3088 | -0.225 | 0.246 | 0.0360 | — |

**Table 1.** Non-Suboptimal and Suboptimal Change Points

**Acknowledgments:** This research was supported in part by NSF Grants IIS-1639306 and IIS-1016205. The data for this study are tabulated in the Supplementary Materials and are available from the authors.

**Supplementary Material: Materials and Methods**

**DBMS Query Suboptimality.** We and many others have found that the optimizer sometimes chooses a suboptimal (slower) plan for a given query. An example from our experiments is shown in Figure S1. The upper graph in Figure S1 represents the plans chosen by a common DBMS as the cardinality of a relation referenced by that query decreases from 2M to 10K tuples by steps of 10K. The lower graph indicates the query times executed at adjacent cardinalities when the plan changed termed the “query-at-cardinality” (Q@C) time. For some transitions, emphasized in red ovals, the Q@C time at the larger cardinality was rather smaller, such as the transition from plan P1 at 1,650,000 to P0 at 1,640,000 tuples. Such pairs identify suboptimal plans.



**Fig. S1.** An Example of Query Suboptimality

Every DBMS subject that we have examined exhibited this phenomenon, even when optimizing very simple queries.

**Simulating DBMS Generations.** Associate with eachDBMS a series of generations, with each generation having one more operator. (We'll explain shortly how we characterize the generations of each DBMS.) So generation 1 of the DBMS has just one operator, generation 2 has two operators, etc.

Our data consists of 8,840 query instances and their 112,118 Q@Cs. Specifically, we focus on pairs of adjacent Q@Cs (let's refer to them as the *lower* Q@C and the *upper* Q@C, and thus we also have a *lower plan* and an *upper plan*) for the same query at the *lower* and *upper* cardinalities, that are adjacent (that is, separated by the minimum cardinality, either 10,000 or 300 rows for one DBMS that was much slower), each with an actual execution time. All Q@Cs for that query and for that DBMS having cardinalities between the upper Q@C of one pair and the lower Q@C of the next pair are associated with that same plan. This will be the upper plan, guaranteed by the process in which we chose those Q@Cs for analysis (recall that we start from the highest cardinality, that of 2M tuples, looking for changes in the query plan).

**Fig S2.** An Example of a Negative Backward Delta

For this generational experiment, we associate with each Q@C a generation (a positive integer) that is the earliest generation of the DBMS that contains all the operators in the plan associated with that Q@C. We also only consider Q@C pairs (with adjacent Q@Cs) where the lower plan has a generation distinct from that of the upper plan. Say the upper generation is earlier than that of the lower plan, as illustrated in Figure S2. In this figure, as we scan from right to left in decreasing cardinality, we first encounter Plan A from generation 2 at the highest cardinality of 2M rows, then at 1320 rows (this is the measured Q@C that has Plan A that is closest to the measurement at 910K rows), then a pair of Q@Cs with Plan A at 910K rows and Plan B at the adjacent 900K rows, then later Plan A at 400K rows and then again at 30K rows.

In this case, the thinking goes that, had we been in the DBMS generation 2, the optimizer would have selected Plan A throughout, because Plan B simply wasn't possible (as it involves an operator not present in generation 2). Then when generation 3 was created by adding an operator, the optimizer chose Plan B for 900K. The reason Plan B was chosen is that it is faster than Plan A at that cardinality, shown in the figure by extrapolating the time down from 910K down to 900K (we will revisit this extrapolation shortly).

In this particular case, as just mentioned, Plan B is more appropriate (faster) than Plan A, but that is not the only possibility. Sometimes the later generation with more operators available chooses the *wrong plan*, one that is slower than the plan chosen by the earlier generation, due to the suboptimality we've observed. (We'll examine an example shortly.)

**Characterizing DBMS Generations.** To which generation do we assign each DBMS operator?

Note that we don't actually know the specific order in which the operators were added to each DBMS. But even if we did, that order was somewhat arbitrary, with a host of considerations going into those decisions over the years. Given that we are using the same optimizer for each such defined generation, we'll adopt a more systematic ordering of the operators. We start by gathering all *single-operator* plans, all *two-operator* plans, and so forth, and order the generation by the prevalence of their appearance in these query plans.

Specifically, for each DBMS we designate the first generation to contain the single operator that maximizes the number of plans at change points, that is, maximizing the number of Q@Cs, containing just that operator. We then examine the plans containing exactly two operators. Each subsequent generation adds that operator that maximizes the number of plans that operator will eventual enable.

**Number of Change Points Per Generation.** As an illustrative example of how we can look at DBMS query optimizer performance through the lens of change points, let's consider the *maximum number of change points per query,* or *maximum CPQ*, from the perspective of

DBMS generations.

For each query, we count the number of Q@Cs at each generation, so for instance a query might have five Q@Cs at generation 1, seven Q@Cs at generation 2, and 17 Q@Cs at generation 5. We then then take the maximum CPQ over the queries (so perhaps 17 is the maximum for generation 5 over all queries). Our hypothesis is that as the generation increases, the maximum CPQ will increase. The results are shown in Table S1.

|  |  |
| --- | --- |
| *Generation* | *Maximum Change Points*  *Per Query* |
| 1 | 1 |
| 2 | 66 |
| 3 | 115 |
| 4 | 136 |
| 5 | 132 |
| 6 | 141 |
| 7 | 42 |
| 8 | 172 |
| 9 | 2 |
| 10 | 12 |
| 11 | 55 |
| 12 | 29 |
| 13 | 1 |
| *Cumulative* | 172 |

**Table S1.** Max Change Points Per Query, Per Generation

This analysis indicates that there is something concerning that seems to get critical around generation 8. And indeed, for this data set, we'll see later that the efficacy of the query optimizers bottom out around that generation.

**Using Change Points.** We now consider how to use data already collected, that is, the *change points*: adjacent cardinalities (10K apart; 300 for one DBMS) with different plans. Specifically, how can change points be used to evaluate the effectiveness of different generations of a DBMS?

We extrapolate the measured time for the plan at the lower generation to the cardinality of the adjacent higher generation. An example is shown in Figure S3. In this case, the set of operators in Plan A for the query at cardinality 900K requires at least generation 3, whereas the set of operators for Plan B for the query at cardinality 910K requires but generation 2. (Note that the generations in this example are consecutive, but that is not required. Sometimes the generations within a pair are wildly different.)

To examine the wisdom of picking Plan A for 900K (whose time was measured as the *higher* point shown (the one above the dashed line), we find the closest Q@C also having Plan A. There are two illustrated here, at cardinalities 30K and 2000K, respectively, with the closer one at 30K. (If there is no such Q@C, we can't do the extrapolation, and simply remove that change point from further consideration.)

We use the slope of the dashed line between the measured query time of Plan B at cardinalities 30K and 910K to get an estimated query time at 900k. This realizes an estimate of how fast the query using Plan B would have run on the database having the variable relation with a cardinality of 900K. In this particular case, Plan B looks like it would have run *faster* (in fact, was faster even at 910K in this particular example).

**Fig. S3.** An Example of a Positive Backward Delta

Note that plans can be discontinuous, with its time increasing linearly as cardinality increases, but then jumping in time at a cardinality when the number of passes increases by one. However, given that we are extrapolating from the actual run time at a very close cardinality, our extrapolated estimate should be similarly close.

We term this situation a *backward extrapolation*, because we are extrapolating backward from a cardinality of 910K to one of 900K.

With this extrapolation, we can compute the *relative delta,* defined as the measured time of Plan A (the chosen one) at the lower cardinality (here, 900K) minus the extrapolated time of Plan B at that same cardinality, divided by the measured time of Plan A. The relative delta is scaled by original measured time at that granularity, and thus has a maximum possible value of 1.

In this case, the relative delta is positive (the measured time for the plan associated with the higher generation is greater than the extrapolated time), which indicates that the optimizer chose the *wrong plan*: here, Plan B should have been chosen.

On the other hand, a *negative* relative delta (where the measured time for the plan associated with the higher generation is lower than the extrapolated time, such as that illustrated in Figure S2) implies that the additional operator(s) available to the minimally applicable generation of the upper plan were indeed beneficial, in that that plan was faster. (In such cases, we again divide by the larger value, so the minimum possible value is -1.)

Summarizing, a positive relative delta (extrapolated in either the forward or backward direction) implies the presence of suboptimality: the wrong plan was chosen by the DBMS generation, perhaps due to the greater number of operators available. A negative relative delta (forward or backward) implies the *absence* of suboptimality: the chosen plan was faster at that cardinality than the one chosen at the adjacent cardinality.

There are six orthogonal possibilities for each change point (that is, a pair of adjacent Q@Cs), a total of 66,769 pairs/change points from the experiment. (There is also the case of a query instance containing a lone Q@C, which we don't consider further.)

1. The pair of Q@Cs shares the same generation: *mingen*(*lower*) = *mingen*(*upper*) (62,903 Q@C pairs, 94.2%), which we don't consider further.
2. The extrapolation yielded a computed query time that was negative, which we also drop (10 pairs, 0.01%).
3. The extrapolation from above and indicating no suboptimality, termed a *negative backward relative delta*, as exemplified in Figure S2, examined earlier (492 pairs, 0.7%).
4. The extrapolation, termed a *positive backward relative delta* and exemplified in Figure S3, indicates a suboptimal plan (583 pairs, 0.9%).
5. The extrapolation was a *negative forward relative delta*, indicating no suboptimality (1218 pairs, 1.8%).
6. An extrapolation was from below (consider Figure S2 but with Plan A from generation 4; we would then need to extrapolate from the closest Plan B, which is *at a smaller cardinality*), in a *forward direction*, to compute a *positive forward relative delta*, indicating a suboptimal plan (795 pairs, 1.2%).

From these six possibilities, we thus retain (3) and (5), which indicate a suboptimal plan at the higher generation (1710 pairs), and (4) and (6), which indicate a non-suboptimal plan at the higher generation (1378 pairs).

**Non-Suboptimal and Suboptimal Change Points.** We partition the relevant four sets of change points discussed above into two groups: (i) those with a *positive* relative delta (either forward or backward), denoting a *suboptimal decision* by the query optimizer at the later generation

(corresponding to a higher generation number) and (ii) those with a *negative* relative delta (either forward or backward) relative delta, indicating a *non-suboptimal* *decision* by the query optimizer at the later generation.

Let's first examine those change points for which the query optimizer made a good decision, the non-suboptimal change points; see Table S2. The third column gives the average relative delta across just that generation, while the last column aggregates over that and prior generations; thus the cumulative is equal to that of generation 13.

The averages jump around quite a bit, especially for the first few generations. The thing to focus on is the last column, where the cumulative relative delta starts off at a low of -0.28 (recall that low negative value is good, as it indicates the additional operator was effective at lowering the execution time) and slowly increases to -0.225: more negative numbers trending to less negative numbers, indicating decreasing benefits of optimization.

This general behavior matches our prediction arising from the structural causal model that it gets harder to squeeze out performance gains as operators are added to the DBMS over successive generations.

|  |  |  |  |
| --- | --- | --- | --- |
| *Generation* | *Number of*  *Change Points* | *Average*  *Relative Delta* | *Cumulative*  *Relative Delta* |
| 1 | — | — | — |
| 2 | — | — | — |
| 3 | — | 0.0 | 0.0 |
| 4 | 24 | -0.28 | -0.28 |
| 5 | 282 | -0.118 | -0.130 |
| 6 | 309 | -0.274 | -0.203 |
| 7 | 336 | -0.317 | -0.243 |
| 8 | 43 | -0.20 | -0.241 |
| 9 | 4 | -0.1 | -0.241 |
| 10 | 17 | -0.29 | -0.241 |
| 11 | 357 | -0.178 | -0.225 |
| 12 | 6 | -0.2 | -0.225 |
| 13 | 0 | — | -0.225 |
| *Cumulative* | 1378 | — | -0.225 |

**Table S2.** Non-Suboptimal Change Points

Table S3 provides the same information across the DBMS generations for the suboptimal change points: those for which the relative deltas are positive, indicating a poor decision, as the query time increased. While there are still only a small number of change points, there are a greater number of *suboptimal* change points, with a relatively greater number showing up at more recent generations. But more strikingly, the cumulative relative delta has the opposite behavior to the non-suboptimal change points: it *increases* over the generations (indicating greater suboptimality), starting at 0.077 and ending at 0.246, a value *higher* than that of the non-suboptimal change points.

|  |  |  |  |
| --- | --- | --- | --- |
| *Generation* | *Number of*  *Change Points* | *Average*  *Relative Delta* | *Cumulative*  *Relative Delta* |
| 1 | — | — | — |
| 2 | — | — | — |
| 3 | — | 0.0 | 0.0 |
| 4 | 2 | 0.2 | 0.2 |
| 5 | 211 | 0.076 | 0.077 |
| 6 | 86 | 0.16 | 0.102 |
| 7 | 263 | 0.303 | 0.196 |
| 8 | 273 | 0.303 | 0.231 |
| 9 | 3 | 0.04 | 0.230 |
| 10 | 168 | 0.319 | 0.245 |
| 11 | 639 | 0.237 | 0.242 |
| 12 | 53 | 0.35 | 0.245 |
| 13 | 12 | 0.40 | 0.246 |
| *Cumulative* | 1710 | — | 0.246 |

**Table S3.** Suboptimal Change Points