Pit Fender

Efficient Memoization Algorithms for Query Optimization

Top-Down Join Enumeration through Memoization on the Basis of Hypergraphs

Anchor Academic Publishing

disseminate knowledge

Fender, Pit: Efficient Memoization Algorithms for Query Optimization: Top-Down Join Enumeration through Memoization on the Basis of Hypergraphs, Hamburg, Anchor Academic Publishing 2015

Buch-ISBN: 978-3-95489-336-2
PDF-eBook-ISBN: 978-3-95489-836-7
Druck/Herstellung: Anchor Academic Publishing, Hamburg, 2015

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Bibliographical Information of the German National Library:
The German National Library lists this publication in the German National Bibliography. Detailed bibliographic data can be found at: http://dnb.d-nb.de

Hermannstal 119k, 22119 Hamburg
http://www.diplomica-verlag.de, Hamburg 2015
Printed in Germany

Zusammenfassung

Für ein Datenbankmanagementsystem, das Unterstützung für deklarative Anfragesprachen wie SQL bietet, ist der Anfrageoptimierer eine unerlässliche Softwarekomponente. Oft können deklarative Anfragen in verschiedene äquivalente Ausführungspläne übersetzt werden. Der Übersetzungsprozess, welcher unter allen äquivalenten Alternativen einen geeigneten Ausführungsplan auswählt, wird Anfrage-Optimierung genannt. Der Auswahlprozess beruht auf einem Kostenmodell und auf Verteilungsstatistiken über die zugrunde liegenden Daten. Für die geschätzten Ausführungskosten des Ausführungsplans ist die Reihenfolge der Join-Operationen ausschlaggebend. Dabei kann der Laufzeitunterschied zwischen verschiedenen Ausführungsplänen mit unterschiedlichen Ausführungsreihenfolgen ihrer Join-Operationen mehrere Größenordnungen betragen. Eine vollständige Suche unter allen äquivalenten Operatorbäumen ist oft zu berechnungsintensiv. Daher muss die Komplexität der Suche eingeschränkt werden, indem die Größe des Suchraumes reduziert wird. Dazu wird eine weitverbreitete Heuristik angewendet: Es werden nur die Operatorbäume erzeugt und in ihren Kosten miteinander verglichen, welche frei von Kreuzprodukten sind.

Für die Suche nach dem optimalen und damit billigsten Ausführungsplan gibt es zwei mögliche Herangehensweisen: Top-Down Join Enumeration und Bottom-Up Join Enumeration. Dabei hat Top-Down Join Enumeration einen wesentlichen Vorteil: Durch die bedarfsgesteuerte Aufzählungsreihenfolge können Branch-and-Bound-Pruning-Strategien verwendet werden. Durch Branch-and-Bound kann die Übersetzungszeit der Anfrage um mehrere Größenordnungen reduziert werden. Trotz merklicher Verkürzung der Übersetzungszeit wird in jedem Fall der optimale und somit kostengünstigste Plan erzeugt. Falls es nach dem jeweilig verwendeten Kostenmodell mehrere optimale Pläne gibt, wird einer dieser Kandidaten erzeugt.

Die vorliegende Arbeit widmet sich dem Top-Down-Join-Enumeration-Prozess. Im ersten Teil der Arbeit werden zwei gleich effiziente Partitionierungsalgorithmen für Graphen vorgestellt, die für Top-Down Join Enumeration von Relevanz sind. Jedoch gibt es bei den im ersten Teil vorgestellten Strategien zwei erhebliche Einschränkungen: (1) Die Algorithmen eignen sich nur für Anfragen mit einfachen (binären) Join-Prädikaten. (2) Anfragen, die neben inneren Join-Operationen auch auf äußere Join-Operationen zurückgreifen, können nicht verarbeitet werden.

Im zweiten Teil dieser Arbeit werden diese Einschränkungen aufgehoben. Dazu wird zunächst eine von zwei Partitionierungsstrategien für Graphen angepasst und erweitert. Anschließend wird ein generisches Framework vorgestellt, das jeden Partitionierungsalgorithmus für Graphen derart umrüstet, dass auch Anfragen mit komplexen Join-Prädikaten und äußeren Join-Operationen übersetzt werden können. Wie sich zeigen wird, ist das generische Framework effizienter als der modifizierte und erweiterte Partitionierungsalgorithmus für Graphen.

Der dritte Teil dieser Arbeit beschäftigt sich mit Branch-and-Bound-Pruning-Strategien. Als erstes werden zwei bereits bekannte Pruning-Strategien erläutert

und klassifiziert. Im Weiteren wird erklärt, wie beide Strategien vereint werden können. Darauf aufbauend werden sieben Verbesserungen vorgeschlagen. Der daraus resultierende neue Branch-and-Bound-Pruning-Algorithmus verbessert (1) die Effizienz von Pruning, macht (2) Branch-And-Bound Pruning robuster und verhindert (3) Szenarien, bei denen die Übersetzungszeit durch Pruning um mehrere Größenordnungen verlangsamt wird.

Die vorliegende Arbeit evaluiert mit Hilfe verschiedener Experimente, inwieweit Laufzeitverbessungen durch die vorgestellten neuen Algorithmen erreicht werden können. Dabei werden die Anfragen der TPC-H, TPC-DS und SQLite Test Suite Benchmarks übersetzt und die Laufzeit der Optimierungsphase gemessen. Unsere Ergebnisse zeigen, dass sich die Übersetzungszeit bei Verwendung der hier vorgestellten Algorithmen für die Benchmark-Anfragen um 100% verbessert. Bei Verwendung synthetischer Workloads können sogar noch größere Laufzeitverbesserungen erreicht werden.

Abstract

For a DBMS that provides support for a declarative query language like SQL, the query optimizer is a crucial piece of software. The declarative nature of a query allows it to be translated into many equivalent evaluation plans. The process of choosing a suitable plan from all alternatives is known as query optimization. The basis of this choice are a cost model and statistics over the data. Essential for the costs of a plan is the execution order of join operations in its operator tree, since the runtime of plans with different join orders can vary by several orders of magnitude. An exhaustive search for an optimal solution over all possible operator trees is computationally infeasible. To decrease complexity, the search space must be restricted. Therefore, a well-accepted heuristic is applied: All possible bushy join trees are considered, while cross products are excluded from the search.

There are two efficient approaches to identify the best plan: bottom-up and top-down join enumeration. But only the top-down approach allows for branch-and-bound pruning, which can improve compile time by several orders of magnitude, while still preserving optimality.

Hence, this book focuses on the top-down join enumeration. In the first part, we present two efficient graph-partitioning algorithms suitable for top-down join enumeration. However, as we will see, there are two severe limitations: The proposed algorithms can handle only (1) simple (binary) join predicates and (2) inner joins. Therefore, the second part adopts one of the proposed partitioning strategies to overcome those limitations. Furthermore, we propose a more generic partitioning framework that enables every graph-partitioning algorithm to handle join predicates involving more than two relations, and outer joins as well as other non-inner joins. As we will see, our framework is more efficient than the adopted graph-partitioning algorithm. The third part of this book discusses the two branch-and-bound pruning strategies that can be found in the literature. We present seven advancements to the combined strategy that improve pruning (1) in terms of effectiveness, (2) in terms of robustness and (3), most importantly, avoid the worst-case behavior otherwise observed.

Different experiments evaluate the performance improvements of our proposed methods. We use the TPC-H, TPC-DS and SQLite test suite benchmarks to evaluate our joined contributions. As we show, the average compile time improvement in those settings is 100% when compared with the state of the art in bottom-up join enumeration. Our synthetic workloads show even higher improvement factors.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Queries against DBMSs are often formulated in declarative languages. Prominent examples are SQL, OQL, XPath and XQuery. Writing such a declarative query has two advantages: (1) The querist does not need to decide upon the actual algorithms and execution order to access and combine the data, which in turn (2) leaves the DBMS with several degrees of freedom to choose the best evaluation and execution strategy in order to answer the query. This is a shift of complexity: from formulating the query towards how to answer it in a most efficient way. We refer to the process of transforming the declarative query in an imperative execution plan as plan generation, and we call the component in the DBMS which deals with the complexity of choosing a suitable plan from all alternatives the plan generator.

Today's plan generators are cost-based. This means that they rely on a cost model and statistics over the data in order to select from all equivalent plans the one with the lowest costs. Essential for the costs of a plan is the execution order of join operations in its operator tree, since the runtime of plans with different join orders can vary by several orders of magnitude. An exhaustive search for an optimal solution over all possible operator trees is computationally infeasible. To decrease complexity, the search space must be restricted. For the optimization problem discussed in this book, the well-accepted connectivity heuristic is applied: We consider all possible bushy join trees, but exclude cross products from the search, presuming that all considered queries span a connected query graph. Thereby, a query graph is an undirected graph where join predicates span the edges between the relations referenced in the SQL query, i.e., a graph edge between R_1 and R_2 is introduced if there exists a join predicate involving attributes of R_1 and R_2.

When designing a plan generator, there are two strategies to find an optimal join order: bottom-up join enumeration via dynamic programming, and top-down join enumeration through memoization.

Both plan generation approaches rely on Bellman's Principle of Optimality[1]: They generate an optimal join tree for a set of relations S by considering optimal subjoin trees only. This means that non-optimal, i.e., more expensive, subjoin trees can be discarded, which curtails the search space enormously[2]. Moreover, since the connectivity heuristic is applied[3], the optimal (sub) join tree needs to be constructed only for those

[1] The presence of properties requires additional care. For reasons of simplicity properties are ignored here.

[2] The search space is reduced from $|V|!\,\mathcal{C}(|V|-1)$ number of plans to $\frac{3^{|V|}-2^{|V|+1}+1}{2}$ where $|V|$ is the number of relations referenced in the query and $\mathcal{C}$ are the Catalan Numbers with $\mathcal{C}(n) = \frac{1}{n+1}\binom{2n}{n}$ [4]. $|V|!\,\mathcal{C}(|V|-1)$ can be simplified to $\frac{(2|V|-2)!}{(|V|-1)!}$ [13, 19, 31]

[3] Which can reduce the search space further depending on the query graph down to $\frac{|V|^3-|V|}{3}$ number of plans.

subsets of relations S that can be joined without the need of applying cross products. In other words, the subset S of relations referenced in the SQL query has to induce a connected subgraph of the original query graph.

In order to determine the best join tree for a given subset S of relations, the top-down/bottom-up plan generator must enumerate all partitions (S_1, S_2) of S such that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$ holds. Furthermore, since we exclude cross products, S_1 and S_2 must induce connected subgraphs, and there must be two relations $R_1 \in S_1$ and $R_2 \in S_2$ such that they are connected by a graph edge. Let us call such a partition (S_1, S_2) a `ccp`. Denote by T_i the optimal plan for S_i. Then the query optimizer has to consider the plans $T_1 \bowtie T_2$ for all *ccp*s (S_1, S_2) in order to compute the optimal join tree for the relations contained in S.

Thus, both the bottom-up and the top-down join enumeration face the same challenge: to efficiently compute the *ccp*s. There has been an ongoing race between top-down and bottom-up join enumeration concerning this challenge. Traditionally, all partitioning strategies have been generate-and-test based. But depending on the shape of the query graph, most of the generated partitions are not valid *ccp*s, i.e., are filtered out by the tests for connectivity. That is why those approaches are suboptimal and can have an exponential overhead[4].

In bottom-up join enumeration, all the connected subsets for a given set are already generated. Therefore, a partitioning strategy for dynamic programming that is not generate-and-test based should be easier to design. Moerkotte and Neumann [22] presented a dynamic programming variant called DPccp, producing all partitions in constant time $O(1)$ per valid *ccp*.

For top-down join enumeration via memoization, no such equally efficient solution is known yet. Finding an analogous variant to DPccp for memoization is very appealing, not only for the outcome of the race but also because the nature of top-down processing can leverage the benefits of branch-and-bound pruning. The beauty of those pruning strategies is that they are risk-free: They can speed up processing by several orders of magnitude, while at the same time they preserve optimality of the final join tree.

DeHaan and Tompa took up the challenge and proposed with MinCutLazy [5] a minimal graph cut partitioning algorithm for memoization. Nevertheless, TDMcL, which is the generic top-down join enumeration algorithm instantiated with MinCutLazy, cannot compete with DPccp. The first contribution of this work are two partitioning algorithms for top-down join enumeration that close the performance gap to DPccp.

However, the proposed algorithms DPccp and TDMcL are not ready to be used in real-world scenarios yet because there exist severe limitations: First, as has been argued in several places, hypergraphs must be handled by any plan generator [2, 27, 35]. Second, plan generators have to deal with outer joins and antijoins [14, 27]. In general, these operators are not freely reorderable, i.e., there might exist different orderings, which produce different results. Because it has been shown that the non-inner join reordering problem can be reduced to hypergraphs, it remains the top goal of

[4] In case of a chain query for example, the naive generate-and-test based approach for top-down join enumeration generates $2^{|V|+2} - |V|^2 - 3 * |V| - 4$ partitions but only $\frac{|V|^3 - |V|}{3}$ are valid *ccp*s [26].

any plan generator to deal with hypergraphs [2, 21, 27, 20]. Consequently, Moerkotte and Neumann [21] extended DPCCP to DPHYP to handle hypergraphs.

The second main contribution of this work is a generic partitioning framework that transforms hypergraphs to restrictive graphs and applies some further modifications. The advantage of this approach is that existing, well performing graph-partitioning algorithms can be reused in order to efficiently handle hypergraphs.

As the third and last contribution of this book, we present advancements to the known branch-and-bound pruning strategies.

As will be shown, all combined contributions of this book result in a performance advantage of 100% over DPHYP by considering the TPC-H [34], TPC-DS [33] and the SQLite test suite [29] benchmarks. For syntactic workloads we present average runtime improvements by orders of magnitude.

The detailed contributions together with the outline of this book are described in the following section.

1.2. Contribution

1.2.1. Graph-Aware Join Enumeration Algorithms

In Chapter 2, we give a general introduction to top-down join enumeration. We explain a naive approach and give a complexity analysis that motivates three new graph-partitioning strategies. For the last partitioning algorithm, we show that it has a complexity in $O(1)$ per emitted *ccp* for acyclic and standard query graphs. A performance evaluation concludes this chapter, showing that two of the three partitioning algorithms are competitive with DPCCP. The following publications contributed to this chapter:

[9] Pit Fender and Guido Moerkotte. Reassessing top-down join enumeration. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1803–1818, 2012

[12] Pit Fender, Guido Moerkotte, Thomas Neumann, and Viktor Leis. Effective and robust pruning for top-down join enumeration algorithms. In *Proceedings of the 28th International Conference on Data Engineering*, pages 414–425, 2012

[8] Pit Fender and Guido Moerkotte. A new, highly efficient, and easy to implement top-down join enumeration algorithm. In *Proceedings of the 27th International Conference on Data Engineering*, pages 864–875, 2011

1.2.2. Hypergraph-Aware Join Enumeration Algorithms

We start Chapter 3 by motivating why the handling of hypergraphs is indispensable. After that, we adjust the naive top-down join enumeration algorithm of Chapter 2 and explain the necessary changes. We continue with a description of the first hypergraph-aware partitioning algorithm. Then we present our main contribution: a generic partitioning framework that enables graph-aware partitioning algorithms to cope with hypergraphs. We show how the partitioning strategies of Chapter 2 can be reused. Then we conclude with a performance evaluation that includes the runtime results of the

TPC-H, TPC-DS and the SQLite test suite benchmarks. The hypergraph-aware partitioning algorithm and the generic framework have already been published:

[11] Pit Fender and Guido Moerkotte. Top-down plan generation: From theory to practice. In *Proceedings of the 29th International Conference on Data Engineering*, pages 1105–1116, 2013

[10] Pit Fender and Guido Moerkotte. Counter strike: Generic top-down join enumeration for hypergraphs. *Proceedings of the VLDB Endowment*, 6(14):1822–1833, September 2013

1.2.3. Branch-and-Bound Pruning

The main advantage of top-down join enumeration over bottom-up join enumeration is that it allows for branch-and-bound pruning. Chapter 4 starts with an introduction to branch-and-bound pruning. Then, we follow with seven advancements that improve pruning (1) in terms of effectiveness, (2) in terms of robustness and (3) by avoiding its potential worst case behavior otherwise observed. At the end of Chapter 4, we give an in-depth performance evaluation. Furthermore, we give results for the TPC-H, TPC-DS and the SQLite test suite benchmarks. We have published advancements and results as follows:

[12] Pit Fender, Guido Moerkotte, Thomas Neumann, and Viktor Leis. Effective and robust pruning for top-down join enumeration algorithms. In *Proceedings of the 28th International Conference on Data Engineering*, pages 414–425, 2012

1.2.4. Conclusion and Appendix

We conclude this book in Chapter 5. Appendix A gives a complexity analysis of the work of DeHaan and Tompa [5]. Furthermore, we include the C++ Code of two partitioning algorithms in Appendix B.1 and B.2.