GHD-optimal join queries in compact space

1 Introduction

In relational databases, the join operation combines data from multiple tables by pairing tuples from participating relations if they store the same value for common attributes. Traditional query plans used pair-wise joins to retrieve the results of a join query. This approach transforms the query into a series of intermediate join operations between two tables at a time. Most query plans that involve doing a pair-wise join are sub-optimal, so much research and development has been made to find new processing strategies [7].

This led to the definition of the Atserias, Grohe and Marx (AGM) bound, which gives bounds to the output size of a join query in terms of the sizes of the participating relations [3]. Worst Case Optimal (WCO) join algorithms are algorithms that have a worst-case runtime in compliance with the AGM bound. In other words, if the worst-case output size is Q^* results, then a join algorithm is WCO if it runs in time $O(Q^*)$.

Graph databases are information systems in which data is represented in the form of nodes and the relationships between them. Due to the growth in the amount of graph data from sources such as social networks, bioinformatics, and even data structuring initiatives like Wikidata[1], there is a need to efficiently process and store this information.

Although there have been many substantial advances in the field of join query evaluations, there is still room for improvement, particularly in regards to the space used by the process [2]. This thesis aims to significantly reduce the space required by WCO join algorithms in graph databases, allowing processes to be stored in memory higher up in the memory hierarchy and thus optimizing query times. This space reduction would also enable querying large datasets.

2 Related Work

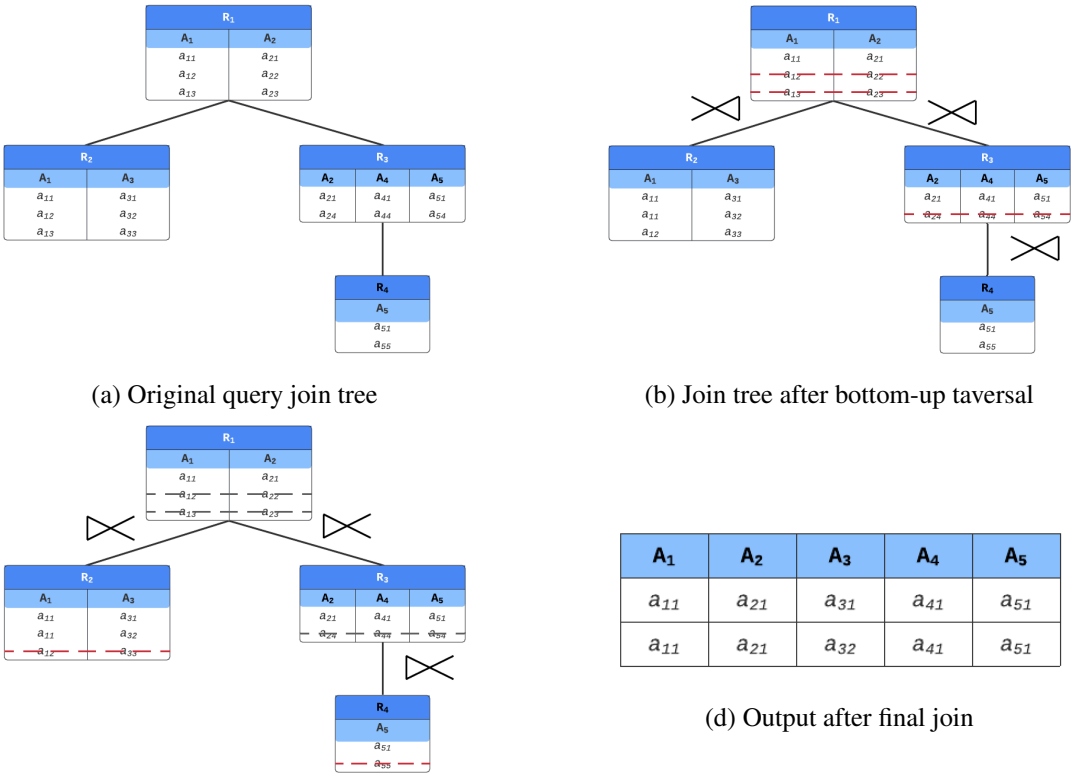
2.1 Yannakakis Algorithm

If a conjunctive query is acyclic, then it can be solved using Yannakakis algorithm in time $O(|input| + |output|)$ [8]. For a query to be acyclic, it needs to have a *join tree*, in which each node is a participating relation and two nodes are connected by an edge if they share an attribute.

The general idea of the Yannakakis algorithm is to do two sweeps of the join tree, first from the leaves to root, and then from root to the leaves. In each sweep a node will constrain the next node (either its parent or child depending on the sweep) by doing a semijoin between the two relations, eliminating the tuples that don't have a match. A final sweep is made by doing a join on the reduced tables. Figure 1 shows this process step by step using a basic example.

Since the first two sweeps are proportional to input size, and the final sweep is proportional to the output size, this algorithm's runtime is $O(|input| + |output|)$.

[1] More information about Wikidata can be found here `https://www.wikidata.org/wiki/Wikidata:Main_Page`



(a) Original query join tree

(b) Join tree after bottom-up taversal

(c) Join tree after top-down traversal

(d) Output after final join

Figure 1: Example of Yannakakis algorithm for acyclic join queries.

2.2 EmptyHeaded

A query can be represented as a hypergraph, in which each attribute is expressed as a node, and relations are represented by hyperedges [4]. A generalized hypertree decompositions (GHD) of a hypergraph is a pair (T, λ), in which T is a tree and λ is a function mapping a set of hyperedges of the hypergraph to each node of T. A GHD node contains a subset of attributes such that for every relation in the query, its attributes are contained in one or more nodes. In the tree, nodes that share a common attribute must form a connected subtree. If each tree node contained the attributes of a single relation then it would be a join tree, as described in subsection 2.1.

EmptyHeaded is a graph processing engine that proposed a novel join algorithm and query planner, based on GHD and tries [1]. Given a join query and its hypergraph representation, EmptyHeaded uses an heuristic to select a GHD for the query. This GHD is effectively a query plan, with each node in the GHD representing a multiway join that must be calculated. A worst-case optimal join algorithm is used on each node of the GHD, simplifying the query into a tree with the intermediate results as nodes. Yannakakis algorithm is then performed on the resulting acyclic query. This algorithm runs in time $O(N^w + |out put|)$, where w is the width of the GHD used. This means the performance of EmptyHeaded is strongly tied to the selection of a query's GHD. Regardless of this, EmptyHeaded's algorithm complies with the AGM bound, and in some cases it can outperform this bound. In this work we will refer to this algorithm as being "GHD-optimal".

EmptyHeaded stores triples as tries in main memory, and a recent set of experiments showed that it

requires more space for indexing (in terms of bytes per triple) than almost all of the other alternatives it was compared to [2]. Also, even though EmptyHeaded offers fast runtimes, the experimental evidence suggests that its space requirements prevent it from performing well in real-world scenarios.

2.3 Qdag

A Qdag is a data structure that builds upon compact quadtrees to represent relations and, more importantly, evaluate multijoin queries in WCO time. This structure is composed of a quadtree representing a relation and a constant-space mapping function that allows for the virtual traversal of the generalization of said quadtree to a higher dimension [6]. With this, we can extend a relation with new attributes.

The WCO join algorithm goes as follows. The Qdag representations of the participating relations are extended to all the attributes appearing in the query. The intersection of all the Qdags is computed, generating a quadtree that contains the values present in all the relations, and thus, the results of the join query. This is done by simulating a synchronized traversal along all the extended Qdags.

Recent experiments show that Qdags can even compress the database representation [6]. They also conclude that the Qdag index uses less space than other WCO algorithms. In particular, Qdags require around 250 times less space than EmptyHeaded. This gives Qdags the advantage of fitting in memory higher up on the memory hierarchy, and thus being faster in some cases. Another benefit of the Qdag algorithm is that the output of a join query is obtained in its compressed representation, so further processing can be done to it, like using it for another query.

In regards to experimental runtime, Qdags are competitive in queries with low dimensionality. When producing relations of five or more attributes, they fare worse than other WCO alorithms.

3 Problem

The aim of this project is to research if it's possible to develop a more efficient GHD-optimal join algorithm using succinct data structures. Our intention is to propose a method of solving cyclic join queries that arise in the GHD decomposition, requiring less resources to store intermediate results than the current standard, while guaranteeing worst-case optimality.

Our proposed strategy is to, like in EmptyHeaded, use a GHD of the initial query as a query plan. That is, transform the cyclic query graph into a tree in which the nodes represent subqueries, including cyclic joins. The plan is to use Qdags to solve these subqueries and materialize the results. Then we would use Yannakakis algorithm for acyclic join queries on the resulting tables. For this step, Qdags will have to be adapted to perform semijoins. Qdags are a promising option because each subquery will only encompass a few attributes, and Qdags perform best timewise when dealing with a small number of attributes. A graphic explanation of our proposed strategy is shown in Figure 2.

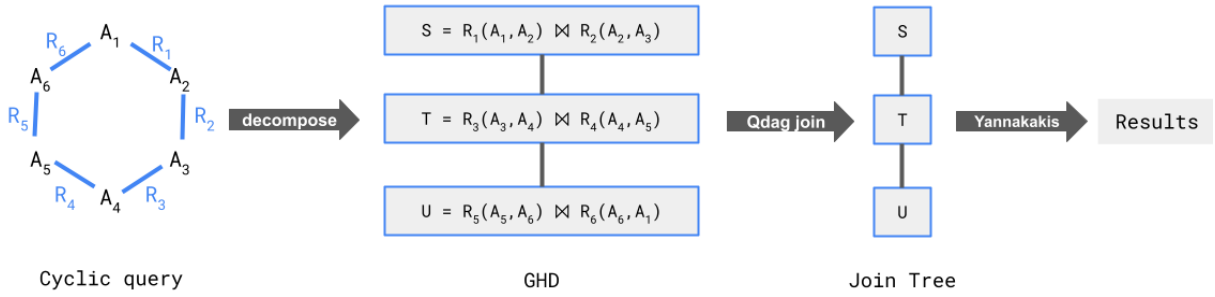


Figure 2: Diagram depicting our proposed strategy in action. In this example the query is a join between 6 relations, R_1 has the attributes A_1 and A_2, R_2 the attributes A_2 and A_3, and so on. The cyclic query is decomposed into a GHD with three nodes, each representing a join between two relations. These joins are solved using Qdag's algorithm, obtaining a join tree with three nodes. Finally, Yannakakis algorithm is used to get the final results.

4 Research questions

The research questions that will be guiding this work are:

1. Can the use of compact data structures reduce the space needed in join queries while complying with GHD optimality?
2. Will the amount of dimensional extensions needed to be done in Qdags join algorithm be significantly reduced in practice? Would this affect the query time in a significant manner?
3. Is it possible to stop the traversal of a Qdag before reaching the leaves when computing a semijoin?
4. How dependent is our proposed strategy on the GHD generated from the query graph?
5. Will the whole scheme be efficient in practice? Will it compete with EmptyHeaded and with the WCO algorithm based on plain Qdags?

5 Hypothesis

The maximum space used when calculating join queries can be reduced by implementing a generalized version of Yannakakis' algorithm, using Qdags to solve the cyclic components of the query graph.
Our proposed algorithm will fare better than the original Qdags when computing queries with many attributes.

6 Objectives

6.1 General Objective

The general objective of this thesis is to develop an efficient GHD-optimal join algorithm using compact data structures for indexing data.

6.2 Specific objectives

- Design and develop a new GHD-optimal join algorithm based on combining Qdags with EmptyHeaded's algorithm.
- Compare our solution to EmptyHeaded's algorithm and Qdags in terms of index space and query time.
- Prove or disprove that our algorithm is GHD-optimal.

7 Methodology

The following are the steps considered to develop this thesis.

- Research and evaluation of different options for generating a GHD of a cyclic join query, including the possibility of using EmptyHeaded's method.
- Adaptation of Qdags to support semijoins. This will be done by extending both relations to Qdags and then simultaneously traversing the Qdags. Instead of generating an output quadtree, we will only have to mark each intersecting tuple in the original quadtree, using an additional bitmap.
- Implementation of Yannakakis algorithm using Qdags and the GHD. In each step of the bottom-up traversal of the tree we will perform a semijoin between a node and each of its children, marking its bitmap accordingly. The tuples that are present in all semijoins will continue in the process. The ones that are not marked are ignored for the remainder of the algorithm, consequently "trimming" the Qdag. This will be repeated in the top-down traversal, now marking a node's children. Finally, the trimmed Qdags will be intersected using the join operation.
- Implementation of the entire algorithm. This includes the generation of a query graph's GHD, its traversal and the computation of the intermediate cyclic joins using Qdags. Then we will use Yannakakis algorithm to obtain the output.
- Testing of the algorithm's effectiveness. We will evaluate the algorithm's correctness using various test cases.
- Develop proof of our proposed algorithm's GHD-optimality.
- Experimental comparison of the new proposed algorithm to EmptyHeaded and Qdags in regards to space and query time, using the Wikidata Graph Pattern Benchmark [5].

The software will be developed using C++ since it will be an extension of the existing implementation of Qdags [2].

8 Expected results

We expect to obtain a new GHD-optimal join algorithm that requires less resources than existing solutions while providing support for cyclic join queries. Our algorithm's source code will be available in the public domain, allowing for further research in this field. We also expect to publish our results and methodology in an academic journal.

[2]The code can be found in the following repository https://github.com/darroyue/qdags

References

[1] Christopher R. Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun & Christopher Ré (2017): *EmptyHeaded*. *ACM Transactions on Database Systems* 42(4), p. 1–44, doi:10.1145/3129246.

[2] Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma & Adrián Soto (2021): *Worst-Case Optimal Graph Joins in Almost No Space*. In: *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD/PODS '21, Association for Computing Machinery, New York, NY, USA, p. 102–114, doi:10.1145/3448016.3457256. Available at `https://doi.org/10.1145/3448016.3457256`.

[3] Albert Atserias, Martin Grohe & Dániel Marx (2008): *Size Bounds and Query Plans for Relational Joins*. In: *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, IEEE Computer Society, pp. 739–748, doi:10.1109/FOCS.2008.43.

[4] Georg Gottlob, Martin Grohe, Nysret Musliu, Marko Samer & Francesco Scarcello (2005): *Hypertree Decompositions: Structure, Algorithms, and Applications*. In: *IN PROC. OF WG'05*.

[5] Aidan Hogan, Cristian Riveros, Carlos Rojas & Adrián Soto (2019): *A Worst-Case Optimal Join Algorithm for SPARQL*. *Lecture Notes in Computer Science The Semantic Web – ISWC 2019*, p. 258–275, doi:10.1007/978-3-030-30793-$6_1$5.

[6] G. Navarro, J. Reutter & J. Rojas (2020): *Optimal Joins using Compact Data Structures*. In: *Proc. 23rd International Conference on Database Theory (ICDT)*, pp. 21:1–21:21.

[7] Hung Q Ngo, Christopher Ré & Atri Rudra (2014): *Skew Strikes Back: New Developments in the Theory of Join Algorithms*. *SIGMOD Rec.* 42(4), p. 5–16, doi:10.1145/2590989.2590991. Available at `https://doi.org/10.1145/2590989.2590991`.

[8] Mihalis Yannakakis (1981): *Algorithms for Acyclic Database Schemes*. In: *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, IEEE Computer Society, pp. 82–94. Available at `db/conf/vldb/Yannakakis81.html`.