Hypergraphs for Dummies: A Layman's Guide to Advanced Query Processing Using Worst-case Optimal Joins and Hypertree Decompositions

Galen Ballew
Univeristy of Washington
gballew@uw.edu

June 3, 2022

Abstract

In the database context, representing queries as hypergraphs creates potential opportunities for improved query processing. This paper provides an introductory overview of hypergraphs and their applications, including worst-case optimal join algorithms and hypertree decompositions.

1 Introduction

This paper is about hypergraphs and some of the ways they can be leveraged for query processing. There are many stages throughout the query life cycle. Therefore, there are many areas and opportunities for improvement. The hypergraph applications explored in this paper, *worst-case optimal join algorithms* and *hypertree decompositions*, focus on hypergraph (and thus query) cyclicity. Cyclic queries can be characterized by their growing intermediate results. However, current cardinality estimation techniques can wrong by orders of magnitude, especially for multi-join queries [8]. The lack of a unified framework for accurate cardinality estimation has impacted commercial RDBMS—state of the art systems like Microsoft CosmosDB use over 400 heuristic rules during query optimization [3]. While hypergraphs do not currently improve traditional cardinality estimation techniques, they do help to manage intermediate join results and can be used to simplify heuristics for join ordering.

Contribution Summary. This paper introduces hypergraphs, worst-case optimal joins and their related mathematical underpinnings, and hypertree decompositions. This paper does not seek to be exhaustive or investigative, but attempts to summarize advanced mathematics and query processing techniques in a succinct and approachable manner. It is geared towards beginners, undergraduates, and laymen in general. Readers are encouraged to leverage the cited materials for more robust definitions and additional context.

Outline of the Paper. We begin in Section 2 with a brief outline hypergraphs and how they can be leveraged to represent queries in a RDBMS context. In Section 3, we describe a mathematical bound for the output size of queries. In Section 4, we provide and overview of worst-case optimal join algorithms. In Section 5, we provide a similar overview for hypertree decompositions. In Section 6, we outline real-world examples of both worst-case optimal join algorithms and hypertree decompositions. Finally, we close with suggestions for additional reading for interested readers in Section 7.

2 Hypergraphs Explained

2.1 Much ado about the triangle

We begin with the triangle query.

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(C, A)$$

The above query is a great example of cyclicity and will serve as a useful example to illustrate numerous concepts throughout the paper. This query may be represented as an undirected hypergraph H where $H = (V, E)$, v is a set of vertices, and E is a set of non-empty subsets of V called *hyperedges* or *edges*. It is notable that hyperedges may contain ≥ 1 vertices while, in traditional graphs, edges contain exactly 2 vertices. In the context of queries, vertices correspond to *attributes* and edges correspond to *relations*.

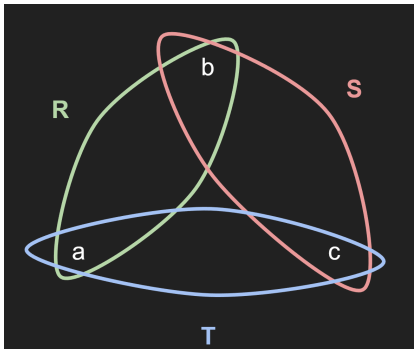

Figure 1: The triangle query represented as a hypergraph.

2.2 Edge cover

The *edge cover* of H is a subset of E such that each vertex in H is contained by the subset. In the triangle query, any combination of 2 of the 3 relations yields valid edge cover. For example, R and T together yields the vertices (A, B, C) which is equivalent to the entire set V. *Fractional edge cover* takes this concept one step further. Fractional edge cover is a set of numbers, one for each edge in the hypergraph, such that for every vertex v

$$\sum_{e:v \in e} u_e \geq 1$$

In layman's terms, a set of numbers serves as valid fractional edge cover so long as the sum of values for the edges containing any given vertex is ≥ 1.

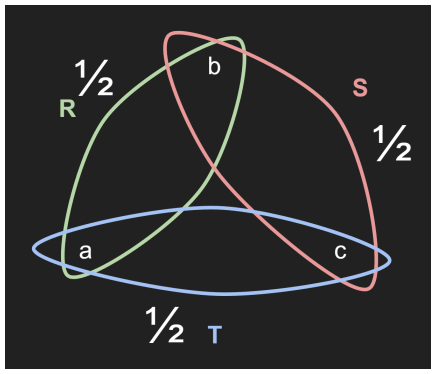

Figure 2: An example of valid fractional edge cover for the triangle query.

We can see that the values $\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\}$ are valid edge cover in Figure 2. For any given vertex in A, B, C, the sum of the 2 edges that contain that vertex is equal to 1. It is important to note that different edges may have different values assigned to them—they do not need to be uniform. Furthermore, for our purposes, it is highly desirable to seek the *minimal* valid edge cover. The reasons for which will be discussed in the next section.

3 The AGM Bound

3.1 Cardinality versus complexity

Continuing to work with our triangle query example, what are we able to say about the size of the expected output? It should follow intuitively that the upper bound corresponds to the situation where there are no matches between tuples for given join, resulting in $|Q_\triangle| \leq N^2$ for $|R|, |S|, |T| \leq N$. This is far from optimal however. In the case where a relation is very small, say $|R| = 1$, we will have an equally small finally result, $|Q_\triangle| = 1$, but will still have a time complexity of $\mathcal{O}(N^2)$ for the join.

3.2 The AGM bound explained

The AGM bound states

$$|Q_\triangle| \leq |R_1|^{u_1} \times ... \times |R_m|^{u_m}$$

where $u_1, u_2, ...u_m$ is fractional edge cover [13]. Applying this to our working example of the triangle query with fractional edge cover $\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\}$ yields

$$|Q_\triangle| \leq |R|^{\frac{1}{2}} \times |S|^{\frac{1}{2}} \times |T|^{\frac{1}{2}}$$

For $|R|, |S|, |T| \leq N$, this reduces to

$$|Q_\triangle| \leq N^{\frac{3}{2}}$$

3.3 Implications of AGM bound for query processing

Although the AGM bound is a purely mathematical phenomenon, it suggests that improved time complexity guarantees are possible for join algorithms. In other words, even in the case of the triangle query where $|R| = 1$, the AGM bound implies that $Q_\triangle$ can be computed in $\mathcal{O}(N^{\frac{3}{2}})$ rather than $\mathcal{O}(N^2)$. This may not appear to be dramatic at first glance, but the difference between $|Q_\triangle| \leq N^2$ and $|Q_\triangle| \leq N^{\frac{3}{2}}$ is best illustrated by an example: $(1,000)^2 = 1,000,000$ and $(1,000)^{\frac{3}{2}} = \lfloor 31,622 \rfloor$. The question then becomes whether or not we are able to design algorithms that match the implied runtimes promised by the AGM bound. As we will see in the next section, not only can we design these algorithms, but their design stems from hypergraphs the same way that the AGM bound does.

4 A User's Guide to Worst-case Optimal Join Algorithms

There are multiple flavors of worst-case optimal join algorithms, but they share common elements [14][12]. Foremost, they are built on top of hypergraphs. Figure 3 provides an example of the `Generic Join` algorithm. Conceptually, it performs a recursive backtracking search through the intersections of the edges (i.e., relations) of the query's hypergraph. By leveraging the same concept of fractional edge cover as the AGM bound, `Generic Join` achieves a log linear runtime of $\mathcal{O}\left(AGM\left(Q_\triangle\right)\log\left(N\right)\right)$.

Notably, in a break from orthodoxy, WCOJ algorithms are multi-way join algorithms. In our triangle query example, this means that all 3 relations R, S, T are simultaneously joined—there is no intermediate result. This also means that the variable ordering has a significantly smaller impact on the optimality of the algorithms—their runtime is guaranteed to be worst-case optimal regardless of variable ordering. However, in practice, variable order does still matter, but exactly how is poorly understood to date [13].

There are additional considerations for WCOJ algorithms when implementing them in a query engine. First, WCOJ algorithms are commonly slower than binary joins if the output cardinality is less than the maximum of the input cardinalities [1][5]. It follows that WCOJ are well suited for cyclic queries, which are characterized by growing intermediate results, but it may be beneficial for a query optimizer to use a heuristic-based system for deciding whether or not to use a WCOJ algorithm for a binary join. An example of such a query optimizer is described in [5]. Another serious consideration is that WCOJ algorithms require suitable indexes on all permutations of attributes that can partake in a join. This can entail an enormous storage and maintenance overhead and the requirement for pre-sorted relations or indexing must be addressed in a successful implementation. A novel hash-based WCOJ algorithm that relies only on data structures that can be built efficiently during query execution is described in [5].

ALGORITHM 1: Generic Worst-Case Optimal Join Algorithm

```
1   //Input: Hypergraph H=(V,E), and a tuple t.
2   Generic-Join(V,E,t):
3     if |V|=1 then return ∩_{e∈E} R_e[t].
4     Let I={v_1} // the first attribute.
5     Q ← ∅ // the return value
6     // Intersect all relations that contain v_1
7     // Only those tuples that agree with t.
8     for every t_v ∈ ∩_{e∈E:e∋v_1} π_I(R_e[t]) do
9       Q_t ← Generic-Join(V−I, E, t :: t_v)
10      Q ← Q ∪ {t_v} × Q_t
11    return Q
```

Figure 3: Psuedocode of the Generic Join algorithm [2].

5 Hypertree Decompositions

5.1 Impact of hypertree decompositions on query processing

WCOJ algorithms are an amazing development that are very useful for cyclic queries. Similarly, the `Yannakakis` algorithm limits the size of intermediate result for acyclic conjunctive queries for efficient query processing. Hypertree decompositions (HD) allow us to extend the benefits of acyclic conjunctive queries to *"mildly acyclic"* conjunctive queries and thereby leverage `Yannakakis` for efficient query processing over this broader class of conjunctive queries.

Additionally, many aspects of hypertree decompositions offer polynomial time complexity and other benefits:

— For a fixed constant k, it can be checked in polynomial time whether a conjunctive query Q is of hypertree width $\leq k$, and if so, a HD of width $hw(HQ)$ can be computed in polynomial time. The best upper bound currently known for this is $O(m^{2k}v^2)$ time, where m and v are the number of atoms and the number of variables in Q, respectively [6].

Hypertree width is defined and discussed in Section 5.2.1.

— Hypertree decompositions can be used effectively in systems that use multi-way joins (like WCOJ).

— Hypertree decompositions and their computation is highly parallelizable and can be done effectively in distributed systems.

— Hypertree decompositions can be used for *top-k* queries and queries with aggregation operators like `COUNT`.

5.2 A user's guide to hypertree decompositions

5.2.1 Generalizing query cyclicity

Formally a hypertree decomposition is defined as follows:

`DEFINITION` 1 ([7]). A generalized hypertree decomposition of a hypergraph $\mathcal{H}$ is a triple $HD = \langle T, \chi, \lambda \rangle$, where $\langle T, \chi \rangle$ is a tree decomposition of $\mathcal{H}$, and λ is a function labeling the vertices of T by sets of hyperedges of $\mathcal{H}$ such that, for each vertex p of T, $\chi(p) \subseteq h \in \lambda(v)$. That is, all nodes in the χ labeling are covered by hyperedges in the λ labeling.

In layman's terms, a hypertree decomposition a given hypergraph is a tree decomposition where each node contains hypergraph vertices and the hyperedges that cover those vertices. A hypergraph potentially has many possible hypertree decompositions. The measure of a hypertree decomposition's *width* is a critical property for determining the usefulness and application of the decomposition. The width of $\langle T, \chi, \lambda \rangle$ is defined as $\max_{p \in V} (|\lambda(p)|)$, or the maximum number of edges in any given node. From here, we can define the *hypertree width* of hypergraph $\mathcal{H}$, denoted $hw(\mathcal{H})$, as the minimum width over all its hypertree decompositions. The hypertree width of $\mathcal{H}$ is a very good measure of how cyclic $\mathcal{H}$ is. Specifically, hypergraphs where $hw(\mathcal{H}) = 1$ are strictly acyclic [7]. That is to say a hypergraph with $hw(\mathcal{H}) = 1$ is acyclic and a hypergraph with $hw(\mathcal{H}) = 3$ is "more cyclic" than a hypergraph with $hw(\mathcal{H}) = 2$.

5.2.2 Hypertree decompositions of queries

In the context of queries as hypergraphs, the computation times referenced in Section 5.1 are entirely practical. This is due to the fact that $hw(\mathcal{H})$ of at most 2 or 3 typically suffice to represent queries and the k factor in the exponent of $O(m^{2k}v^2)$ is manageable. Even in a very complex query where the number of atoms m were larger, it can still be worthwhile to compute the HD if the query will be run more than once on changing, underlying data. The upfront expense of the HD computation will likely be recuperated by running a highly optimized query using the HD over and over again [6]. This author suggests that a "repeatability" factor for queries may be interest for future work regarding query optimization and hypertree decompositions.

5.2.3 Answering a query using a hypertree decomposition

This section will describe, in plain terms, how the computed HD of a query hypergraph can be used to generate an optimized query plan. To begin, it may be help to think of hypertree decompositions themselves as query plans for joins. Specifically, the hypertree decomposition gives way to a concrete join ordering that, when executed, will convert a cyclic conjunctive query Q_c into an equivalent, acyclic query Q'_a. `Yannakakis` can then be used to efficiently process Q'_a. A simplified outline of the overall process is given below:

1. Consider each node on T. Nodes and their χ, λ can be thought of as subqueries. The conjunction of all nodes can then be thought of as the overall query.

2. Starting at the bottom of T, in the "leaves" of the tree, join the edges in λ.

   These joins may be done using any number of join algorithms. WCOJ algorithms can be used here. Ideal conditions for applying WCOJs are described in Section 4. WCOJ are obvious candidates for nodes where $|\lambda| \geq 3$ due to their multi-way join efficiency.

3. Moving up the tree, when the children nodes of a parent node have joined their λ edges, the children will ass the relations projected onto shared attributes "up" to the parent. Simply put, only the relations which are contained in the parent χ bag are passed.

4. Eventually, the root node will process the join of its edges. This join will result the final tuples, but they will be incomplete. The tuples will only contain the attributes that were shared *between* nodes.

5. Finally, traverse the graph "top-down" and append the required attributes to the final set of tuples.

In this manner, the cyclicity of the original query Q_c is "pushed" into the individual subqueries. The subqueries can then be processed by whatever means are most efficient for that particular subquery. The natural tree structure of the HD is acyclic and is ideal for processing via `Yannakakis`. An example using WCOJ *within nodes* and `Yannakakis` *between nodes* is provided in [2].

6 Real-world Examples

For interested readers, this section provides a non-exhaustive list of projects using worst-case optimal joins and/or hypertree decompositions in their query engines.

6.1 WCOJ in the wild

— EmptyHeaded

— Umbra

— LevelHeaded

— LogicBlox

— RelationalAI

6.2 Hypertree decompositions in the wild

— EmptyHeaded

— LogicBlox

— PostgreSQL*

Contains a special semijoin operator supporting some aggregation operators [6].

7 Next Steps

Interested readers are encouraged to explore this paper's references. The following topics are discussed in greater detail within their respective papers:

— The proof mechanics of the AGM bound [12][13][10].

— The mathematical history behind the AGM bound and the astounding convergence of a multitude of research areas [12].

— A definition of the *Descendant Condition* or *Special Condition* which distinguishes hypertree decompositions from *generalized hypertree decompositions* [7][6][4].

— Hypertree decompositions of hypergraphs of Boolean conjunctive queries. [6]

— Additional domains where hypertree decompositions can be applied outside of query optimization [4][6].

— The computation of hypertree decompositions, the Robbers and Marshalls game, and the algorithm `opt-k-decomp` that is derived from it [7][6].

— A brief summary of heuristic systems used to construct hypertree decompositions for very large values of k [6].

The concept of *fractional hypertree width* is briefly discussed in [7][6][2][1], but not thoroughly explored. Fractional hypertree width is a topic of current and active research. Curious readers may find the ties between fractional edge cover, the AGM bound, and hypertree width to be intellectually stimulating.

Acknowledgements

GB's work and understanding was made entirely possible Professor Dan Suciu. Suciu's consistent office hours, patient explanations, and support in exploring advanced topics made for an extremely rewarding experience—thank you!

References

[1] Aberger, C. R., Lamb, A., Olukotun, K., and Ré, C. Levelheaded: Making worst-case optimal joins work in the common case. *CoRR abs/1708.07859* (2017).

[2] Aberger, C. R., Lamb, A., Tu, S., Nötzli, A., Olukotun, K., and Ré, C. Emptyheaded: A relational engine for graph processing. *ACM Trans. Database Syst. 42*, 4 (oct 2017).

[3] Bruno, N., and Galindo-Legaria, C. A. The cascades framework for query optimization at microsoft. https://courses.cs.washington.edu/courses/csep590d/22sp/lectures/CascadesUW.pdf, Apr 2022.

[4] Fischl, W., Gottlob, G., and Pichler, R. General and fractional hypertree decompositions: Hard and easy cases. *CoRR abs/1611.01090* (2016).

[5] Freitag, M., Bandle, M., Schmidt, T., Kemper, A., and Neumann, T. Adopting worst-case optimal joins in relational database systems. *Proc. VLDB Endow. 13*, 12 (jul 2020), 1891–1904.

[6] Gottlob, G., Greco, G., Leone, N., and Scarcello, F. Hypertree decompositions: Questions and answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (New York, NY, USA, 2016), PODS '16, Association for Computing Machinery, p. 57–74.

[7] Gottlob, G., Grohe, M., Musliu, N., Samer, M., and Scarcello, F. Hypertree decompositions: Structure, algorithms, and applications. In *Graph-Theoretic Concepts in Computer Science* (Berlin, Heidelberg, 2005), D. Kratsch, Ed., Springer Berlin Heidelberg, pp. 1–15.

[8] Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., and Neumann, T. How good are query optimizers, really? *Proc. VLDB Endow. 9*, 3 (nov 2015), 204–215.

[9] Mhedhbi, A., and Salihoglu, S. Optimizing subgraph queries by combining binary and worst-case optimal joins. *Proc. VLDB Endow. 12*, 11 (jul 2019), 1692–1704.

[10] Ngo, H. Q. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (New York, NY, USA, 2018), SIGMOD/PODS '18, Association for Computing Machinery, p. 111–124.

[11] Ngo, H. Q., Porat, E., Ré, C., and Rudra, A. Worst-case optimal join algorithms. *J. ACM 65*, 3 (mar 2018).

[12] Ngo, H. Q., Ré, C., and Rudra, A. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec. 42*, 4 (feb 2014), 5–16.

[13] Suciu, D. Cse544 data management lectures 9-10 advanced query processing. https://courses.cs.washington.edu/courses/cse544/21wi/lectures/lecture09-10-agm-wcoj.pdf, 2021. Accessed: 2022–05-15.

[14] Veldhuizen, T. L. Leapfrog triejoin: a worst-case optimal join algorithm, 2012.