Calibrated Junction Hypertree: Data Structure for Exploratory Queries over Join Result

Zezhou Huang
zh2408@columbia.edu
Columbia University

Eugene Wu
ewu@cs.columbia.edu
DSI, Columbia University

ABSTRACT

...

ACM Reference Format:
Zezhou Huang and Eugene Wu. 2021. Calibrated Junction Hypertree: Data Structure for Exploratory Queries over Join Result. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern enterprise analytics tasks usually involve heterogeneous relation tables from different sources managed by data warehouse [7]. While there are a large number of tools [9, 13, 17] available to support big-data analytics and machine learning, they are constrained to receive one single table as input, and materialize the join of normalized table can cause blow-up in storage and performance. To bypass full materialization, recent factorized data management systems [5, 16, 20, 25] decompose analytics tasks into semiring aggregates over join, and push down aggregates before join to avoid blow-up.

While previous factorized data management systems efficiently execute single query, modern analytics tasks are commonly exploratory [19], where multiple queries are needed to analyze data from different perspectives. To execute multiple queries efficiently, factorized query engine like LMFAO [25] clusters queries in one batch and relies on heuristics to conduct multi-query optimizations. However, exploratory queries are usually incremental and interactive [12], where future queries are based on previous query results, which makes it hard to batch all potential queries in advance. These queries also have huge overlap with the same aggregates or similar join graph. Therefore, they are large work sharing opportunities. We illustrate these exploratory analytics tasks with the following examples:

Exploratory Data Analysis: First, Exploratory Data Analysis [2, 21, 28] is an essential step for data science. For traditional OLAP operations like drill down and roll up, users execute the similar queries over the same aggregation function and join graph, but with different group-by attributes. Predicting the queries in advance and pre-compute them through batch optimization is complex and unreliable. Instead, data cube [14] has been widely used in industry to exploit the similarity between OLAP queries. Naively building the base cuboid for all possible group-by attributes over the join graph could cause blow-up in storage and performance. Calibrated Junction Hypertree could be considered as data cube for complex join graph. As shown later, data cube could be considered as calibrated junction hypertree over one multidimensional table.

What-if Analysis and Streaming: Second, what-if analysis [24] and stream query processing [1] performs the same query multiple times over the same relation tables where tuples are appended or intervened (e.g. deletion propogation). While recent work [20] applies incremental view maintenance to factorization for single query, the redundancy between queries remains unexploited. Calibrated junction hypertree can naturally utilize incremental view maintenance to maintain data structure, and further improve performance by exploiting work sharing opportunities.

Augmented Machine Learning: Third, data scientists usually want to augment machine learning with potential datasets from data warehouse or markets [8, 11]. There could be a substantial number of candidate datasets to join, and data scientists want to understand how would each augmentation improve model performance. While previous factorized data management systems have successfully decomposed machine learning model into aggregates [16, 26], they have to retrain each augmented model from scratch, disregarding the overlap among join graphs. Calibrated junction hypertree can share the computations between model training and efficiently augment new datasets.

To this end, we introduce calibrated junction hypertree, a data structure that is easy to build with low overhead (up to a constant factor compared to a single query), exploits the work sharing opportunities and executes exploratory queries efficiently. The idea of junction tree originates from probabilistic graphical model [27] to share the computations between posterior distribution queries, and has been widely applied to areas including artifical intelligence [6], computer vision [10], civil engineering [29], medical diagnosis [22], genetics [18], control engineering [23], etc. In database area, We find that junction trees have a wide range of applications for data exploration over semiring aggregations. To the best of our knowledge, we are the first to apply junction tree beyond probalistic tables. We illustrate the core idea behind calibrated junction hypertree with a simple example in Figure 1:

Example 1. *Consider relations in Figure 1a and an example aggregation query Q: total count of join result grouped by A and D. The standard solution is to materialize join result and then apply aggregation. The materialized join result is in Figure 1b. Factorized data management systems decomposes aggregation as marginalization and join, and pushes marginalization before join. For example, for R[A,B] and R[A,C], only the A is used for join. We can marginalize B*

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

early by partitioning R[A,B] by A and sum count for each partition. The marginalized result is sent as a message to R[A,C], where C is similarly marginalized and pass message to future nodes. The whole message-passing process is illustrated in Figure 1c. Finally, we have calibrated junction hypertree in Figure 1d. The key insight is that calibrated junction hypertree passes messages bidirectionally, and materialize messages as part of data structure. For example, after T[A,D] receives message from previous nodes, it pass messages back to R[A,B] and S[A,C]. Calibrated junction hypertree takes only about twice as much time of a single query to build because of the bi direction, and can be built offline. The benefit is that three nodes are now autonomous and can answer the similar queries. Suppose users want to query the count grouped by attribute B (or C), node R[A,B] (or S[A,C]) can serve query independently without re-passing messages. Only a subset of junction tree nodes are involved to execute the query.

To analyze the complexity, The materialized join result in Figure 1b is of size $|D|^{\rho^*(Q)}$ [4] where $|D|$ is relation size and $\rho^*(Q)$ is fractional edge cover number of Q. The intermediate result for factorized query in Figure 1c is of size $|D|^{\text{fhtw}(Q)}$ [3, 15] where fhtw(Q) is the fractional hypertree width of Q. Finally, under the assumptions that junction hypertree nodes are dangling tuples free, the Calibrated junction hypertree φ also takes $|D|^{\text{fhtw}(Q)}$ to build. However, given a future query Q', the intermediate result is $|D|^{\text{fshtw}_\varphi(Q')}$, where $\text{fshtw}_\varphi(Q')$ is the fractional hypersubtree width of Q' given calibrated junction hypertree φ. Intuitively, the more similar φ and Q' are, the smaller $\text{fshtw}_\varphi(Q')$ is. We have $1 \le \text{fshtw}_\varphi(Q') \le \text{fhtw}(Q') \le \rho^*(Q')$, and the gap between any two could be infinitely large. We will provide formal definition of fractional hypersubtree width, and extends it to situation where junction nodes are not dangling tuple free.

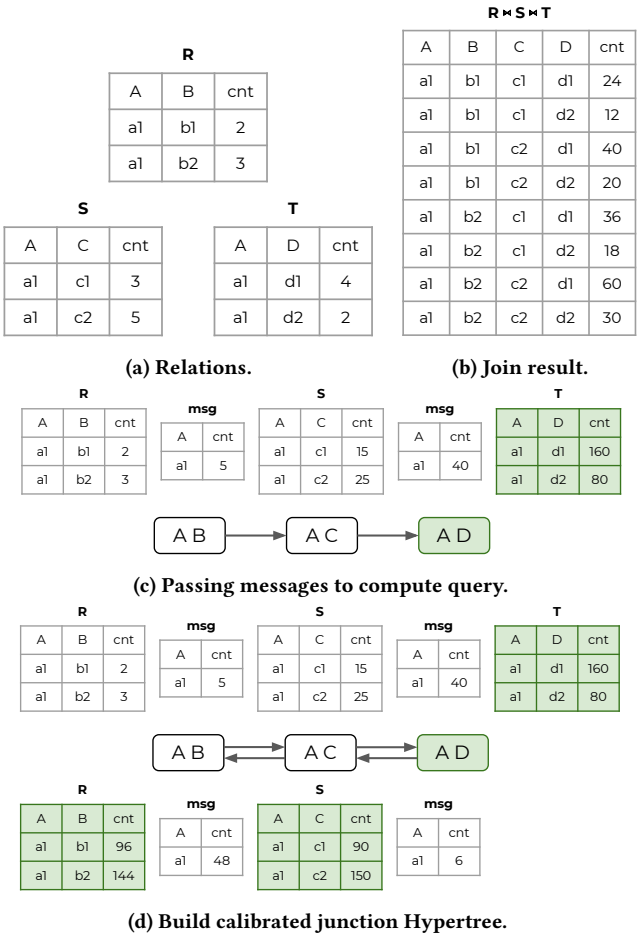

Figure 1: junction tree

REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al. The design of the borealis stream processing engine. In *Cidr*, volume 5, pages 277–289, 2005.

[2] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *The VLDB Journal*, 24(4):557–581, 2015.

[3] M. Abo Khamis, H. Q. Ngo, and A. Rudra. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 13–28, 2016.

[4] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 739–748. IEEE, 2008.

[5] N. Bakibayev, T. Kočiskỳ, D. Olteanu, and J. Závodnỳ. Aggregation and ordering in factorised databases. *arXiv preprint arXiv:1307.0441*, 2013.

[6] T. Braun and R. Möller. Lifted junction tree algorithm. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 30–42. Springer, 2016.

[7] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[8] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger. Arda: automatic relational data augmentation for machine learning. *arXiv preprint arXiv:2003.09758*, 2020.

[9] S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, and J. McPherson. Ricardo: integrating r and hadoop. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 987–998, 2010.

[10] J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. Large-scale object classification using label relation graphs. In *European conference on computer vision*, pages 48–64. Springer, 2014.

[11] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1001–1012. IEEE, 2018.

[12] D. Fisher, S. M. Drucker, and A. C. König. Exploratory visualization involving incremental, approximate database queries and uncertainty. *IEEE computer graphics and applications*, 32(4):55–62, 2012.

[13] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *2011 IEEE 27th International Conference on Data Engineering*, pages 231–242. IEEE, 2011.

[14] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[15] M. Joglekar, R. Puttagunta, and C. Ré. Aggregations over generalized hypertree decompositions. *arXiv preprint arXiv:1508.07532*, 2015.

[16] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. Ac/dc: In-database learning thunderstruck. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, pages 1–10, 2018.

[17] A. Kumar, F. Niu, and C. Ré. Hazy: making it easier to build and maintain big-data analytics. *Communications of the ACM*, 56(3):40–49, 2013.

[18] S. L. Lauritzen and N. A. Sheehan. Graphical models for genetic analyses. *Statistical Science*, pages 489–514, 2003.

[19] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. New trends on exploratory methods for data analytics. *Proceedings of the VLDB Endowment*, 10(12):1977–1980, 2017.

[20] M. Nikolic and D. Olteanu. Incremental view maintenance with triple lock factorization benefits. In *Proceedings of the 2018 International Conference on Management of Data*, pages 365–380, 2018.

[21] J. Peng, W. Wu, B. Lockhart, S. Bian, J. N. Yan, L. Xu, Z. Chi, J. M. Rzeszotarski, and J. Wang. Dataprep. eda: Task-centric exploratory data analysis for statistical modeling in python. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2271–2280, 2021.

[22] A. L. Pineda and V. Gopalakrishnan. Novel application of junction trees to the interpretation of epigenetic differences among lung cancer subtypes. *AMIA Summits on Translational Science Proceedings*, 2015:31, 2015.

[23] J. C. Ramirez, G. Munoz, and L. Gutierrez. Fault diagnosis in an industrial process using bayesian networks: Application of the junction tree algorithm. In *2009 Electronics, Robotics and Automotive Mechanics Conference (CERMA)*, pages 301–306. IEEE, 2009.

[24] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1579–1590, 2014.

[25] M. Schleich, D. Olteanu, M. Abo Khamis, H. Q. Ngo, and X. Nguyen. A layered aggregate engine for analytics workloads. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1642–1659, 2019.

[26] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, pages 3–18, 2016.

[27] G. R. Shafer and P. P. Shenoy. Probability propagation. *Annals of mathematics and Artificial Intelligence*, 2(1):327–351, 1990.

[28] M. Staniak and P. Biecek. The landscape of r packages for automated exploratory data analysis. *arXiv preprint arXiv:1904.02101*, 2019.

[29] F. Zhu, H. A. Aziz, X. Qian, and S. V. Ukkusuri. A junction-tree based learning algorithm to optimize network wide traffic control: A coordinated multi-agent framework. *Transportation Research Part C: Emerging Technologies*, 58:487–501, 2015.