A Demonstration of Q^{2}O: Quantum-augmented Query Optimizer

Hanwen Liu
hanwen_liu@usc.edu
Department of Computer Science
University of Southern California
Los Angeles, California, USA

Federico Spedalieri
fspedali@isi.edu
Information Sciences Institute
University of Southern California
Los Angeles, California, USA

Ibrahim Sabek
sabek@usc.edu
Department of Computer Science
University of Southern California
Los Angeles, California, USA

ABSTRACT

The join order (JO) optimization problem is a key challenge in query optimization. Classical approaches can compute the optimal solution for smaller queries. For larger queries, some heuristic methods trade off plan quality to reduce the exponential search space. Recently, quantum-based methods have been proposed to leverage quantum mechanisms to accelerate exploration; however, encoding problem-specific constraints as penalty terms introduces extra overhead. Moreover, quantum-inspired methods on classical hardware do not harness the true advantages of quantum computation. Furthermore, these methods remain at the simulation stage.

In this demonstration, we present the first Quantum-augmented Query Optimizer (Q^2O) that integrates hybrid quantum-classical approach to solve JO problem in real database setup. This demonstration allows conference attendees to interact directly with Q^2O by executing queries and viewing detailed execution results (Scenario 1). Users can easily compare plan quality between Q^2O and PostgreSQL (Scenario 2). Additionally, they can experiment with quantum parameters to observe their impact (Scenario 3). Experimental results show that the query plan generated by our approach achieves up to a 13x speedup in query execution. The video corresponding to this demonstration is available at this link.

PVLDB Reference Format:
Hanwen Liu, Federico Spedalieri, and Ibrahim Sabek. A Demonstration of Q^{2}O: Quantum-augmented Query Optimizer. PVLDB, 18(1): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ihanwen99/Q2O.

1 INTRODUCTION

Query optimizers are critical components of database systems, and one of the key challenges they address is the Join Order (JO) optimization. The JO problem arises during query planning, and its resolution directly impacts the performance of query execution. The objective is to determine the most efficient sequence for joining multiple relations within a query. Given that the overall search space grows exponentially with the number of relations, the JO problem is recognized as NP-hard [3]. Dynamic programming (DP)-based approaches (e.g., [21]) can compute optimal results by exhaustively enumerating all possible join sequences for queries with relatively few relations. However, due to the exponentially expanding search space, DP becomes impractical for larger queries. To address this, various heuristic and classical methods (e.g., [9]) have been proposed to balance solution quality and computational efficiency in such cases. Instead of always pursuing an optimal plan, it is sometimes acceptable to adopt a sub-optimal plan to reduce search time.

Quantum computing holds significant promise for solving combinatorial optimization problems such as JO by leveraging its ability to explore multiple potential solutions simultaneously, unlike the sequential nature of classical hardware. A recent line of work (e.g., [14, 15, 19]), referred to as *quantum-based JO*, focuses on representing the JO problem as a Quadratic Unconstrained Binary Optimization (QUBO) problem [13] and solving it using quantum annealers (e.g., D-Wave [10]). While promising, current quantum annealers are constrained by bounded qubit number and sparse connectivity (each qubit interacts with only a limited number of neighbors), restricting the size and structure of the problems that can be represented. Additionally, problem-specific constraints (e.g., enforcing left-deep join tree shape) must be encoded as penalty terms, resulting in additional resource overhead. Similarly, quantum circuit models on NISQ (Noisy, Intermediate Scale, Quantum) devices face challenges, including a lack of efficient error correction mechanisms and the high computational cost of parameter optimization [2], making them impractical for applications requiring rapid solution generation. As a trial to address these limitations, another research direction (e.g., [20]), referred to as *quantum-inspired JO*, has explored the use of digital annealing hardware which is inspired by quantum processing units (QPUs) to encode discrete optimization problems. However, quantum-inspired hardware does not utilize actual quantum phenomena, relying entirely on classical hardware and hence foregoing any potential advantage from quantum superposition and entanglement. Importantly, all existing quantum and quantum-inspired JO approaches remain at the simulation stage. None have been integrated into real-world query optimizers or evaluated within practical database systems, leaving their effectiveness in real deployments an open question.

In this demonstration, we present Q^2O, the *first* real Quantum-augmented Query Optimizer that employs a hybrid quantum-classical approach to solve the JO optimization problem and alleviate the limitations of existing solutions. Hybrid quantum-classical solvers [8, 16] have recently gained traction in various real-world applications (e.g., production scheduling [1], power network optimization [5]) by combining the advantages of classical and quantum computing paradigms together. In these solvers, a classical algorithm preprocesses the problem and iteratively generates QUBO instances that can be executed natively by the quantum annealer.

Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

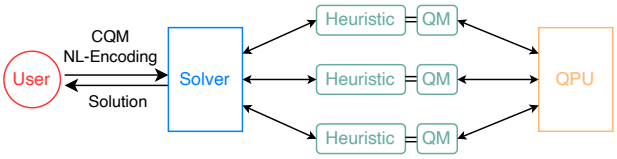

Figure 1: The Workflow of D-Wave Hybrid Solvers.

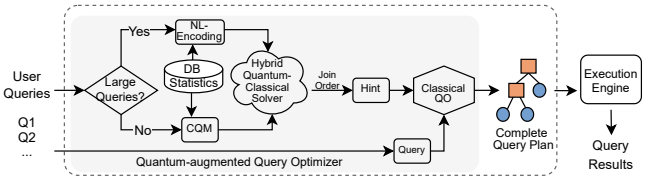

Figure 2: Our Proposed Q^2O Framework Workflow.

This approach circumvents hardware constraints and improves solution efficiency for large and complex problems. Specifically, Q^2O builds upon our initial work in [18] and introduces a novel hybrid quantum-classical solution for the JO problem that *operates in a real database system setup*. Q^2O explores the usage of two recent state-of-the-art D-Wave hybrid solvers: the Constrained Quadratic Model Solver (CQM-Solver) [8] and the Nonlinear-Program Solver (NL-Solver) [16]. Such solvers simplify the process of representing optimization problems and encodes problem-specific constraints without requiring a deep background in physics or quantum mechanics. Q^2O is fully integrated within PostgreSQL [17]. During the query planning phase, the JO of any join query is computed using Q^2O and passed to the existing query optimizer in PostgreSQL as a "plan hint". From there, the rest of the query planning and execution steps continue to produce the query output. Our experiments with the JOB workload [12] shows that running queries with Q^2O's plans can attain a 13x execution speedup over PostgreSQL.

This demonstration will enable conference attendees to interact directly with Q^2O. Users can execute queries of their choice, with the system providing detailed query execution results, including metrics related to quantum solver invocation (Scenario 1). Additionally, attendees can compare query execution performance and plan quality between Q^2O and PostgreSQL (Scenario 2). Finally, users can experiment with quantum parameters to observe how these adjustments influence the behavior of the hybrid quantum-classical solution and impact the overall performance of Q^2O (Scenario 3). A recorded video of the demonstration is available at this link.

2 HYBRID QUANTUM-CLASSICAL SOLVERS

Hybrid quantum-classical solvers combine classical heuristics with quantum annealing [11] (QA) to improve scalability and solution quality. In these solvers, classical algorithms are used to guide the search process—e.g., by pre-processing, decomposing, or refining problem instances—while QA is applied to solve subproblems or explore solution spaces that are otherwise hard to optimize efficiently using classical methods alone. QA shares similarities with simulated annealing (SA) in exploring the searching space by proposing random changes to the state configuration. In SA, a proposed change is accepted if it decreases the value of the cost function; if the cost increases, acceptance is decided by a certain probability. QA explores all states simultaneously through quantum superposition, enabling a highly parallelized search of the configuration space. Moreover, QA leverages quantum tunneling to facilitate exploration, thereby reducing the likelihood of becoming trapped in local minima.

QA is used to solve problems that can be naturally formulated as Quadratic Unconstrained Binary Optimization (QUBO) problems, which correspond to minimizing an energy function over binary variables representing qubit states [18]:

$$E(\vec{s}) = \sum_{ij} J_{ij}\, s_i\, s_j + \sum_i h_i\, s_i.$$

where s_i is a binary variable representing the qubit's state, h_i denotes the local bias on qubit, and J_{ij} represents the interaction strengths between qubits. The Quantum Processing Unit (QPU) uses superconducting loops to represent individual qubits. During quantum annealing, the initial superposition is transformed into the state that minimizes the system's energy, which, in turn, encodes the solution to the combinatorial optimization problem.

This demonstration utilizes two D-Wave's hybrid quantum-classical solvers: CQM-Solver [8] and NL-Solver [16]. Figure 1 provides an overview of their workflow. The front end accepts user-defined problems, represented using Constrained Quadratic Model (CQM) or Non-linear Encoding (NL-Encoding) (detailed in Sections 3.1 and 3.2), as input. The solver then initiates multiple parallel threads, each incorporating both a classical heuristic module and a quantum module (QM). The classical heuristic module iteratively refines the problem's search space while the QM submits QUBOs to the quantum processing unit (QPU). The iterative feedback from the QPU guides the heuristic module toward more promising regions of the solution space. Finally, the best solutions are returned to the user after either a predefined time limit or a user-specified duration.

3 SYSTEM OVERVIEW

Figure 2 illustrates the workflow of Q^2O. Upon receiving user queries, Q^2O selects the encoding method based on the number of relations in the queries. If the number of relations is below a specific number[1], CQM is applied to encode the JO problem; otherwise, NL-Encoding is used. This is because of two reasons: (1) CQM allows for a more faithful representation of the JO problem constraints (e.g., left-deep join tree) but with increased overhead [18], making it better suited for smaller queries, and (2) NL-Encoding enables the direct representation of valid JO permutations (details in Section 3.2), eliminating the need to handle these problem constraints explicitly by encoding them as penalties that need extra qubits to represent them. This feature allows it to process larger queries that CQM encoding cannot handle. Both encoding methods utilize the typical cardinality and predicate selectivity information from the DBMS engine to build the JO optimization cost function. After encoding the JO problem using either CQM or NL-Encoding, it is submitted to the corresponding hybrid solver via the D-Wave Leap platform [6]. The solver returns a join order, which is subsequently translated into a plan hint. This hint is used to guide a classical query optimizer (PostgreSQL's optimizer in our case) in generating

[1]We set 15 relations as the threshold based on our experiments.

a complete query plan. Finally, the execution engine follows the generated query plan to fetch and deliver the query results.

3.1 Constraint Quadratic Model

The Constraint Quadratic Model (CQM) requires a description of the optimization variables, an objective function (cost function), and constraints (if necessary). To formulate the JO optimization problem with CQM, we define two binary variables, following our previous work [18]. $\text{roj}_{r,j}$ is associated with relation r and the join at step j in the query, indicating whether relation r is selected to participate in the join at step j. Similarly, $\text{paj}_{p,j}$ specifies whether predicate p is applicable at join step j.

Considering that CQM only accepts a QUBO-form cost function, we cannot directly use the classical cost model in traditional database engines (e.g., PostgreSQL's cost model). Instead, we adapt a variation of the *quadratic cost approximation* method proposed in [20, 22], which has two steps. First, it formulates the *logarithmic intermediate cardinality* $\text{LogIntCard}(j)$ for join j as $\sum_{r=1}^{\text{R}} \text{LogCard}(r) \cdot \text{roj}_{rj} + \sum_{p=1}^{\text{P}} \text{LogPredSel}(p) \cdot \text{paj}_{pj}$, where $\text{LogCard}(r)$ is the logarithmic cardinality for relation $r \in R$ and $\text{LogPredSel}(p)$ is the logarithmic selectivity for predicate $p \in P$, respectively. Second, it approximates a final cost *QCost* as a function of $\text{LogIntCard}(j)$ using quadratic operations to get rid of inequality constraints [20].

To ensure that CQM generates valid join orders while using the *QCost* function, we introduce three constraints on the variables $\text{roj}_{r,j}$ and $\text{poj}_{p,j}$ that are defined in $\text{LogIntCard}(j)$. Constraint 1: $\sum_{r=1}^{R} \text{roj}_{r,j} = j + 1$, which enforces a left-deep join tree. At each join step j, the total number of relations selected to participate in the joins so far must be exactly $j + 1$. Constraint 2: $\text{roj}_{r,j} - \text{roj}_{r,j+1} \leq 0$, which ensures the continuity of relation selection across the sequence of join operations. Specifically, if a relation r is selected to participate in the join at step j, it must also be selected at step $j + 1$. Constraint 3: $\text{paj}_{p_{(r_1,r_2)},j} \leq \text{roj}_{r_1,j}$ and $\text{paj}_{p_{(r_1,r_2)},j} \leq \text{roj}_{r_2,j}$, which ensure that a predicate is applied to a join only if both relations involved in the predicate are included in the join at step j.

3.2 Non-linear Encoding

Nonlinear-Program Hybrid Solver (NL-Solver) [16] is the latest hybrid solver from D-Wave, which redefines several classical data structures to better manage the input of the problem. Specifically, NL-Solver's *list(n)* represents any ordered permutation of a user-defined n. In the non-linear (NL) encoding of the JO problem, we set n as the number of relations, so that $s = list(n)$ represents any possible join order sequence. For example, if a query contains three relations $\{A, B, C\}$, then s can represent any permutation of their indices, such as [0, 1, 2], [1, 2, 0] to specify a valid join order. Owing to this feature, we do not need to explicitly define additional constraints to ensure JO correctness. For example, in CQM, additional constraints (such as Constraint 1 in Section 3.1) must be explicitly defined to prevent invalid or incomplete join plans at any join step j, such as selecting only $\{A\}$ or $\{B, C\}$ when the full join of A, B, and C is required. In contrast to CQM, NL-Encoding inherently ensures that the three relations are included in this join step without defining constraints.

Additionally, NL-Solvers allow users to directly define the JO cost function in the classical form (as in [4]). We first define the cost

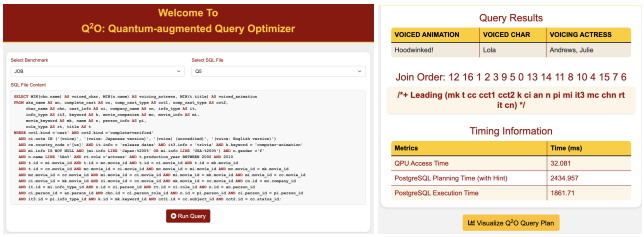

Figure 3: Scenario 1: Query Execution in Q^2O.

of a join between two relations r_i and r_j: $C_{\text{out}}(n_i, n_j) = f_{ij} n_i n_j$, where n_i and n_j represent the cardinalities of the relations, and f_{ij} denotes the join selectivity. To compute the cost for an entire join tree, we extend C_{out} to the join order sequence s as follows:

$$C(s) = \sum_{i=2}^{n} C_{\text{out}}(|s_1 \dots s_{i-1}|, |s_i|), \tag{2}$$

where $|s_1 \dots s_{i-1}|$ represents the size of the intermediate result obtained after joining $s_1, \dots, s_{i-1}$. During execution, NL-Solver internally represents the computational flow of $C(s)$ as a directed acyclic graph (DAG). By leveraging quantum phenomena, NL-Solver evaluates multiple join orders concurrently. Finally, NL-Solver returns the best solution found based on the defined optimization target.

4 DEMONSTRATION SCENARIOS

The demonstration contains three scenarios. In Scenario 1, we show the execution details of Q^2O. In Scenario 2, we compare Q^2O with PostgreSQL. In Scenario 3, we introduce users to directly interact with the control variable in the quantum execution workflow.

Scenario 1: Q^2O Planning and Execution Details.

When users enter our demonstration, they can select a benchmark and an SQL query to execute. Once a query is selected, it is automatically loaded and displayed in a text box with syntax highlighting for better readability. Figure 3 illustrates this user interface. After the user clicks the "Run Query" button, the query planning is initiated. As shown in the workflow diagram in Figure 2, the JO problem is submitted to Q^2O, which first returns a join order hint. This hint is then used to guide the construction of the query plan. Subsequently, the system executes the query based on the generated plan and retrieves the results for the user.

In this scenario, we first present the query results from the database system. Subsequently, we demonstrate internal details: **Join Order**: Derived from the quantum annealer's output; **Hint**: Generated from the join order to guide the PostgreSQL optimizer; **QPU Access Time**: Duration of QPU computation; **PostgreSQL Planning Time (with Hint)**: Time taken by the PostgreSQL optimizer to generate a complete query plan using the provided hint (join order); and **PostgreSQL Execution Time**: Time taken to execute the quantum-augmented query plan and retrieve results.

Scenario 2: Plans Comparison and Visualization.

In this scenario, we enable users to easily compare Q^2O with PostgreSQL's query optimizer. We first present PostgreSQL's query execution details (as Q^2 in Scenario 1) to directly compare fetched

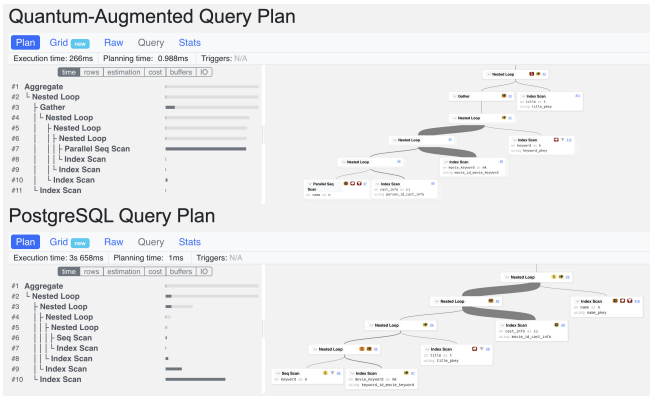

Figure 4: Scenario 2: Plan Comparison and Visualization.

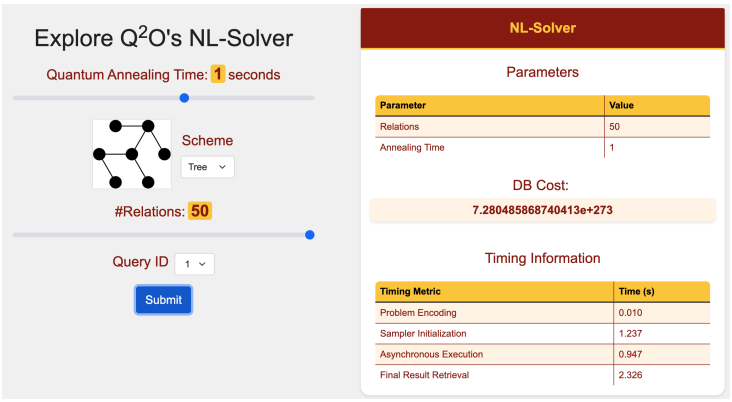

Figure 5: Scenario 3: Fine-grained Control Over Q^2O.

results and query execution time. Then, a visual comparison of the generated query plans is provided using an open-source plan visualization tool [7]. Users can click the "Visualize Plan" button corresponding to each optimizer to trigger a visualization panel that offers a detailed breakdown of the execution, allowing users to view the plan construction details step by step. Figure 4 shows the comparison and plan visualization. The plan generated by Q^2O demonstrates significant improvements in query execution time, achieving a 13x speedup from 3s658ms to 266ms (Q21 in JOB). The left side of the visualization tool provides a detailed breakdown of each operator's contribution to the execution time. The right side visualizes the plan construction process. This scenario helps users better identify the differences between the optimizers.

Scenario 3: Fine-grained Control Over Q^2O.

NL-Solver offers fine-grained control through adjustable parameters. One key parameter is the "Quantum Annealing Time," which specifies the maximum runtime allocated for the QPU to solve the problem. As shown in Figure 5, users can select a quantum annealing time within a range of 0.01 seconds to 5 seconds, enabling them to observe how varying the runtime impacts the hybrid solver's performance. With limited runtime, the solver may produce suboptimal results due to insufficient iterations; conversely, with sufficient runtime, it can generate more stable and higher-quality solutions. We allow users to explore the effect of this parameter on four large-scale synthetic workloads [20] featuring different query graph topologies, including Chain, Cycle, Star, and Tree. Users can select queries that include between 18 and 50 relations. We provide 10 predefined queries for each query topology, which users can select by modifying the "Query ID." After submitting their queries, the system processes the request, executes the query, and returns execution details, including a cost computed using Equation 2, which serves as an indicator of join order quality. Additionally, we present timing metrics that break down the complete NL-Solver workflow.

REFERENCES

[1] Abhishek Awasthi, Nico Kraus, et al. 2024. Real World Application of Quantum-Classical Optimization for Production Scheduling. In 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), Vol. 02. 239–244. https://doi.org/10.1109/QCE60285.2024.10285

[2] Kishor Bharti, Alba Cervera-Lierta, et al. 2022. Noisy intermediate-scale quantum algorithms. Rev. Mod. Phys. 94 (Feb 2022), 015004. Issue 1. https://doi.org/10.1103/RevModPhys.94.015004

[3] Sophie Cluet and Guido Moerkotte. 1995. On the complexity of generating optimal left-deep processing trees with cross products. In International Conference on Database Theory. Springer, 54–67.

[4] Sophie Cluet and Guido Moerkotte. 1995. On the Complexity of Generating Optimal Left-Deep Processing Trees with Cross Products. In Proceedings of the 5th International Conference on Database Theory (ICDT '95). Springer-Verlag, Berlin, Heidelberg, 54–67.

[5] Giuseppe Colucci et al. 2023. Power Network Optimization: A Quantum Approach. IEEE Access 11 (2023). https://doi.org/10.1109/ACCESS.2023.3312997

[6] D-Wave Leap's Hybrid Solvers. 2024. https://docs.dwavesys.com/docs/latest/doc_leap_hybrid.html.

[7] Dalibo. 2025. Postgres Explain Visualizer 2. https://github.com/dalibo/pev2

[8] Hybrid Solver for Constrained Quadratic Models [WhitePaper]. 2021.

[9] Toshihide Ibaraki and Tiko Kameda. 1984. On the optimal nesting order for computing N-relational joins. ACM Trans. Database Syst. 9, 3 (Sept. 1984), 482–502. https://doi.org/10.1145/1270.1498

[10] M. W. Johnson, M. H. S. Amin, et al. 2011. Quantum annealing with manufactured spins. Nature 473, 7346 (2011), 194–198. https://doi.org/10.1038/nature10012

[11] Tadashi Kadowaki and Hidetoshi Nishimori. 1998. Quantum annealing in the transverse Ising model. Phys. Rev. E 58 (Nov 1998), 5355–5363. Issue 5. https://doi.org/10.1103/PhysRevE.58.5355

[12] Viktor Leis et al. 2015. How Good Are Query Optimizers, Really? Proc. VLDB Endow. 9, 3 (2015), 204–215. https://doi.org/10.14778/2850583.2850594

[13] Andrew Lucas. 2014. Ising formulations of many NP problems. Frontiers in Physics 2 (2014). https://doi.org/10.3389/fphy.2014.00005

[14] Nitin Nayak et al. 2024. Quantum Join Ordering by Splitting the Search Space of QUBO Problems. Datenbank-Spektrum 24, 1 (2024), 21–32.

[15] Nitin Nayak, Jan Rehfeld, et al. 2023. Constructing Optimal Bushy Join Trees by Solving QUBO Problems on Quantum Hardware and Simulators. In Proceedings of the International Workshop on Big Data in Emergent Distributed Environments (BiDEDE '23). Article 7. https://doi.org/10.1145/3579142.3594298

[16] Eneko Osaba and Pablo Miranda-Rodriguez. 2024. D-Wave's Nonlinear-Program Hybrid Solver: Description and Performance Analysis. arXiv:2410.07980 [cs.ET]

[17] PostgreSQL. [n. d.]. https://www.postgresql.org/.

[18] Pranshi Saxena, Ibrahim Sabek, and Federico Spedalieri. 2024. Constrained Quadratic Model for Optimizing Join Orders. In Proc. 1st Workshop Quantum Comput. Quantum-Inspired Technol. Data-Intensive Syst. Appl. (Santiago, AA, Chile) (Q-Data '24). 38–44. https://doi.org/10.1145/3665225.3665447

[19] Manuel Schönberger, Stefanie Scherzinger, et al. 2023. Ready to Leap (by Co-Design)? Join Order Optimisation on Quantum Hardware. Proc. ACM Manag. Data 1, 1, Article 92 (may 2023), 27 pages. https://doi.org/10.1145/3588946

[20] Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2023. Quantum-Inspired Digital Annealing for Join Ordering. In VLDB.

[21] P. Griffiths Selinger, M. M. Astrahan, et al. 1979. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD '79). 23–34. https://doi.org/10.1145/582095.582099

[22] Immanuel Trummer and Christoph Koch. 2017. Solving the Join Ordering Problem via Mixed Integer Linear Programming. In Proceedings of the 2017 ACM International Conference on Management of Data (Chicago, Illinois, USA) (SIGMOD '17). Association for Computing Machinery, New York, NY, USA, 1025–1040. https://doi.org/10.1145/3035918.3064039