Gradient-Based Join Ordering

Tim Schwabe and **Maribel Acosta**
Technical University of Munich
{tim.schwabe, maribel.acosta}@tum.de

Abstract

Join ordering is the NP-hard problem of selecting the most efficient order in which to evaluate joins (conjunctive, binary operators) in a database query. Because query execution performance critically depends on this choice, join ordering lies at the core of query optimization. Traditional approaches cast this problem as a discrete combinatorial search over binary trees guided by a cost model, but they have trade-offs between effectiveness and efficiency. We show that when the cost model is differentiable, query plans can be continuously relaxed into a soft adjacency matrix that represents a superposition of plans. This continuous relaxation, combined with differentiable constraints that enforce plan validity, enables a gradient-based search for low-cost plans within this relaxed space. Using a Graph Neural Network as the cost model, we demonstrate that this gradient-based approach can find comparable and even lower-cost plans compared to traditional discrete search methods on two different graph datasets. Furthermore, we empirically show that the runtime of this approach scales better than discrete search algorithms. We believe this first step towards gradient-based join ordering can lead to more effective and efficient query optimizers in the future.

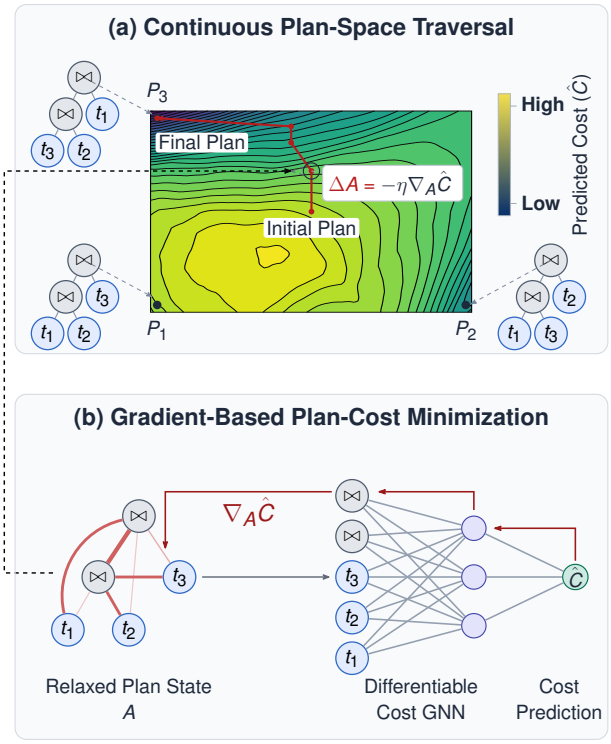


Figure 1: Join Ordering using a learned Cost Model. (a) Continuously relaxing the discrete Search Space of Plans allows traversing it using Gradient Descent. (b) The gradient of the cost with respect to the relaxed plan is backpropagated through the cost model.

1 Introduction

Query optimization, particularly join ordering, is one of the most critical tasks in any database system. It involves determining the optimal evaluation order for the joins in a query, such that the resulting execution order has a low query runtime. Efficiently ordering join operations can significantly impact query execution time, often resulting in either instantaneous responses or impractically long delays.[1]

At the core of join ordering is a *cost model*, which estimates the computational expense associated with executing a query or a part of it. Accurately estimating these costs is a challenging statistical estimation problem, primarily due to the complex data distributions and correlations within real-world datasets. But, finding a good join ordering does not only involve estimating one specific order. It constitutes a discrete search problem over the space of plans, which are represented as binary trees, and which grows super-exponentially. Classical approaches to navigating this space typically rely on discrete methods, such as dynamic programming, which systematically evaluates possible plans at an exponential cost, or heuristics, which choose locally optimal steps quickly but often find suboptimal solutions.

Recently, learned cost models based on neural networks have been proposed, demonstrating significant superiority in cost estimation compared to classical statistical methods [Zhu

[1] An extended version of our paper with additional details is available at https://arxiv.org/abs/2511.14482.

et al., 2024]. However, these learned cost models are either still paired with traditional discrete search methods or completely rely on Reinforcement Learning, which is sample-inefficient and has unstable convergence dynamics.

In this work, we present a fundamentally different approach: we can exploit the differentiability of learned cost models and reformulate join ordering as a continuous optimization problem by relaxing the discrete search space. As shown in Figure 1, we represent query plans as continuous, i.e., edges are not binary but weighted between 0 and 1. This enables direct gradient-based search guided by a differentiable cost model. The main contributions of this work are:

- We propose a novel gradient-based method for join ordering using learned cost models based on Graph Neural Networks, operating on a continuous relaxation of the discrete search space.
- We empirically demonstrate that this method generates solutions that match and sometimes surpass those found by discrete search methods.
- We show that the runtime of this approach is lower than typical randomized discrete baselines, and scales better than many of the classic discrete approaches.

2 Preliminaries

In this work, we assume a graph data model, where queries typically involve a large number of joins due to the fine-grained, triple-based representation of relationships. We assume basic familiarity with sets, graphs, and matrix notation.

2.1 Triple-Based Graph Queries

We assume a graph data model in which the data is stored as *triples*, i.e. labelled, directed edges

$$t = (s, p, o) \in \mathcal{E},$$

where each element belongs to a set $s, p, o \in \mathcal{I}$. A finite set of triples $\mathcal{E}$ forms an edge-labelled directed graph.

Triple patterns and conjunctive queries. Such graphs are queried with *triple patterns*, i.e., triples in which any position may be a variable from an infinite set $\mathcal{V}$ (s.t. $\mathcal{V} \cap \mathcal{I} = \emptyset$) of *variables*:

$$tp = (s, p, o) \in (\mathcal{I} \cup \mathcal{V})^3.$$

A triple pattern is the most basic query. Its answer is the set of solution mappings

$$\Omega_{tp} = \{\mu : vars(tp) \to \mathcal{I} \mid \mu(tp) \in \mathcal{E}\},$$

where $vars(tp)$ are the variables appearing in tp. The number of solution mappings is called the *cardinality* $|\Omega_{tp}|$ of the triple pattern. More general *conjunctive queries* combine several triple patterns $Q = \{tp_1, \ldots, tp_n\} \subseteq (\mathcal{I} \cup \mathcal{V})^3$.

Join Operators. The answer to a conjunctive query is obtained by iteratively *joining* the solution mappings. Let $\mathrm{dom}(\mu)$ denote the variables on which μ is defined. Mappings μ_1, μ_2 are *compatible*, $\mu_1 \sim \mu_2$ if they assign the same value to every shared variable, i.e. $\forall v \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2) : \mu_1(v) = \mu_2(v)$. Their union $\mu_1 \cup \mu_2$ is the mapping with domain $\mathrm{dom}(\mu_1) \cup \mathrm{dom}(\mu_2)$. The join of two solution sets is then

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1,\ \mu_2 \in \Omega_2,\ \mu_1 \sim \mu_2\},$$

i.e., a set of solution mappings with cardinality $|\Omega_1 \bowtie \Omega_2|$.

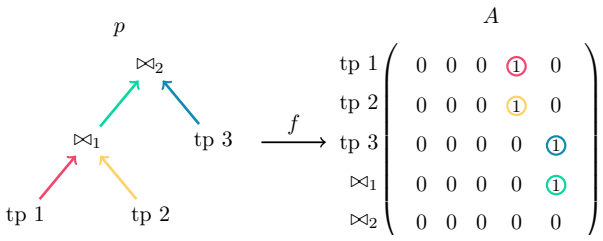


Figure 2: A join plan p is represented as a $(2n-1) \times (2n-1)$ dimensional matrix $f(p) = A$. The first n rows denote the outgoing edges of triple patterns, while the last $n-1$ rows are outgoing edges of join nodes. The root node always corresponds to the last row.

2.2 Query Plans

The final set of solution mappings is invariant to the order in which the triple patterns are joined. However, since the number of intermediate results can differ vastly, the order still matters. A *query plan* p for Q defines the order in which these joins occur. Formally, it is a binary tree whose leaves correspond exactly to the triple patterns in Q, and internal nodes represent join operators combining intermediate results from their children.

- A plan is *bushy* if no structural restrictions exist. Joins may freely combine the results of arbitrary subplans.
- A plan is *left-linear* if each join operator has at most one non-leaf child. Such plans represent sequential join orders.

Since the space of bushy plans is vast ($C_{n-1}n!$, where C_{n-1} is the Catalan Number), commercial optimizers often restrict the search space to left-linear plans (with size $n!$). While our proposed method is general and can search the space of bushy plans, we first focus on left-linear plans in this work. For a query with n triple patterns, we represent a plan p as an adjacency matrix $A \in \{0,1\}^{(2n-1)\times(2n-1)}$, where $A_{ij} = 1$ means node i is joined into node j. The first n indices in A correspond to triple patterns, and the last $n-1$ indices correspond to join nodes (see example in Figure 2).

2.3 Execution Cost and Cost Model

The goal of join ordering is to minimize the execution *cost* of a plan, which measures resources used, e.g., runtime, memory, disk access. Formally, the cost is defined as a function $C : \mathcal{P} \to \mathbb{R}_{\geq 0}$, mapping a plan $p \in \mathcal{P}$ to its cost.

During optimization, a *cost model* is employed to approximate the true cost:

$$\hat{C}_\theta : \mathcal{P} \to \mathbb{R}_{\geq 0},$$

with parameters θ. Simpler approaches rely on nonparametric statistical synopses, while recent research often models $\hat{C}_\theta$ using neural networks.

2.4 Cost Definition (C_{out})

There are many options to define the cost of a query plan and its internal join operations. A popular definition of cost that is widely used and adopted in this paper is the so-called C_{out} cost function [Cluet and Moerkotte, 1995]. It recursively defines the cost of a join v as its output cardinality plus the costs of the left (l) and right (r) sides of the join:

$$C_{\text{out}}(v) = |\Omega(v)| + C_{\text{out}}(l) + C_{\text{out}}(r),$$

The cost of a leaf node (a triple pattern) is defined to be 0:

$$C_{\text{out}}(tp_i) = 0, \quad \text{for all leaves } tp_i.$$

2.5 Discrete Search Algorithms for Join Ordering

In order to find a good join ordering, several search strategies are used. **Exhaustive Search (ES)** evaluates all possible plans and is guaranteed to find the optimum of the given cost model. However, its factorial complexity ($\mathcal{O}(n!)$ for left-linear plans) makes it impractical beyond small queries.

Dynamic Programming (DP) [Selinger *et al.*, 1979] reduces this to $\mathcal{O}(3^n)$ by applying Bellman's principle of optimality: the best plan for a set of joins can be built from optimal subplans. However, this only holds when the cost model satisfies this property and has no guarantees for arbitrary, non-monotonic cost models.

To circumvent the intractable time complexity of DP and ES, most optimizers employ heuristic search methods. **Greedy Search** [Fegaras, 1998] builds plans incrementally by choosing locally optimal moves. At every iteration, the triple pattern (or intermediate sub-plan) whose addition yields the lowest estimated cost according to the cost model $\hat{C}_\theta$ is appended to the join plan. Its downside is that it can find suboptimal local optima. However, it only requires $O(n^2)$ evaluations of the cost model, making it computationally efficient. Several randomized algorithms have been explored and are routinely used in database systems. **Iterative Improvement (II)** [Swami, 1989] starts from a random initial plan and explores a local neighborhood (e.g., pairwise swaps of join positions). At each iteration, all swap neighbors (or a random sample) are evaluated, and if any neighbor improves the cost, the search moves to the best one. This continues until no improving neighbor exists (a local optimum).

Genetic Query Optimization (GEQO) applies evolutionary algorithms to join ordering [Steinbrunn *et al.*, 1997]. It is implemented in PostgreSQL as the default optimization strategy for queries with more than 12 joins [PostgreSQL Global Development Group, 2025]. A population of candidate plans (represented as permutations) evolves over generations through selection (ranked by predicted cost), crossover (edge-recombination crossover [Whitley *et al.*, 1989]), and mutation operators (e.g., swapping two join positions, as in II).

3 Related Work

Learned Cost Models

Traditional query optimizers rely on formula-based cost estimators that combine histograms, independence assumptions, and hand-crafted heuristics or sampling-based approaches, and are inaccurate for larger joins. Recent research, hence, turned to learned models to estimate the cardinality or cost of a query. These models try to map from a suitable representation of a query or plan to its cardinality or cost.

For relational data, *MSCN* [Kipf *et al.*, 2019] models tables, joins, and predicates with separate subnetworks that are aggregated to produce a cardinality estimate. To capture the inherent tree-based structure of query plans, several works use architectures such as Tree-LSTM [Sun and Li, 2019], or Transformers [Zhao *et al.*, 2022] with tree-based attention.

For graph databases, several approaches have used Graph Neural Networks (GNNs) [Schwabe and Acosta, 2024; Zhao *et al.*, 2021] or Recurrent Neural Networks [Aytimur *et al.*, 2025] and pretrained embeddings to accurately represent queries and estimate their cardinality. Similar to approaches on relational databases, approaches for cost estimation explicitly represent the tree structure of the query plan [Casals *et al.*, 2023; Mohanaraj *et al.*, 2025; Qin and Acosta, 2025]. Building on this prior work, we use a GNN to estimate the cost of a plan.

Learned Query Optimization

The learned cardinality and cost estimators above can be used to search for optimal plans using discrete search methods. To combine the time efficiency of greedy approaches with the ability to find lower-cost plans of more exhaustive approaches, recent work has turned to reinforcement learning. The search for an optimal plan is cast as a Markov decision problem where the optimal next action is joining two subplans (where a subplan can be an unjoined table or triple pattern) such that the expected cost is minimized. Various approaches in relational and graph databases use common algorithms such as policy gradients [Marcus and Papaemmanouil, 2018] or Q-Learning [Chen *et al.*, 2022; Yu *et al.*, 2020]. Similar to learned cost models, those approaches use Tree-based representations [Yu *et al.*, 2020; Marcus *et al.*, 2019] and Graph Neural Networks [Chen *et al.*, 2022; Chen *et al.*, 2023] to represent the plans. Several surveys summarize the recent work in relational [Yan *et al.*, 2023; Zhu *et al.*, 2024] and graph databases [Acosta *et al.*, 2025].

Since RL approaches typically suffer from data-inefficient training, our approach focuses on supervised learning, followed by a novel way to search the space of plans.

Gradient Based Optimization of Discrete Structures

Using Gradient Descent to optimize structures that are discrete in nature has been explored in various previous works and domains. *DARTS* [Liu *et al.*, 2019] and *SNAS* [Xie *et al.*, 2019] reformulate Neural Architecture Search to a smooth loss and use gradient descent to directly optimize the network architecture. The approach of *DART* is transferred to differentiable program synthesis by Cui and Zhu. *NRI* [Kipf *et al.*, 2018] simultaneously learns the underlying discrete interaction graph and the dynamics model of a dynamical system using GNNs inside an autoencoder. Structure learning, the task of finding the graph that best explains observed data, has also been approached using gradient-based optimization [Zheng *et al.*, 2018; Ng *et al.*, 2020; Vowels *et al.*, 2023]. In the space of control theory using learned world models, V *et al.* perform gradient-based planning using a re-

current world model. In comparison to our work, they use gradient-based search to find a good order of actions to take, while we aim to find a good order of join operations. Another line of work has applied continuous relaxation to the discrete process of sorting [Grover *et al.*, 2019; Lecorvé *et al.*, 2022; Prillo and Eisenschlos, 2020]. Neural Sort [Grover *et al.*, 2019] proposes a differentiable relaxation of `argsort` with a soft permutation matrix.

Despite these diverse approaches to gradient-based optimization in discrete search spaces, their application to query optimization remains unexplored, a gap that we aim to address in this work.

4 Gradient-Based Join Ordering

We now present GBJO, our approach that reformulates join ordering using gradient descent. We first formalise the optimization objective, then show how a continuous relaxation of the discrete plan space enables end-to-end gradient descent.

4.1 Optimization Objective

Given the set $\mathcal{P}(Q)$ of possible plans for a query Q, i.e. $\mathcal{P}(Q) \subseteq \mathcal{P}$, and let $f : \mathcal{P}(Q) \to \{0,1\}^{N\times N}$ map a plan p to its binary adjacency matrix $A = f(p)$, where $N = 2n-1$ is the total number of plan nodes. Given a feature matrix $X \in \mathbb{R}^{N\times F}$ with F-dimensional features for the N leaves, and a differentiable cost model $\hat{C}_\theta$, join-ordering is the following discrete optimization problem:

$$\min_{p\in\mathcal{P}(Q)} \quad \hat{C}_\theta\big(X, A\big). \tag{1}$$

4.2 Continuous Relaxation of Plans

To transform the discrete objective in Equation (1) into a continuous optimization problem, we relax the binary constraints on the adjacency matrix $A \in \{0,1\}^{N\times N}$. We introduce a real-valued *logit matrix* $L \in \mathbb{R}^{N\times N}$ and derive edge probabilities by applying a softmax:

$$A^{\text{soft}}_{ij} = \frac{\exp(L_{ij}/\tau)}{\sum_k \exp(L_{ik}/\tau)}, \tag{2}$$

$\tau \in \mathbb{R}^+$ is a temperature parameter and controls the sharpness of the distribution. A^{soft} now represents a continuous superposition of discrete plans.

Since valid join plans are located at the binary vertices of the hypercube $[0,1]^{N\times N}$, the optimization must eventually drive each A^{soft}_{ij} towards either 0 or 1. This is achieved by annealing the temperature parameter τ from τ_0 to a minimum value $\tau_{\min}$. As $\tau \to 0$, the entries of A^{soft}_{ij} approach binary values. To balance free exploration in early iterations with convergence towards discrete solutions in later stages, we linearly anneal the temperature over I optimization steps like $\tau_t = \max\big(\tau_{\min},\, \tau_0 - \frac{t}{I}(\tau_0 - \tau_{\min})\big)$.

Relaxed Optimization Problem. Relaxing the plan adjacency to continuous values now gives rise to a smooth objective

$$\min_{L\in\mathbb{R}^{N\times N}} \hat{C}_\theta\big(X, A^{\text{soft}}\big), \tag{3}$$

which can be approached using gradient-based methods.

4.3 Structural Constraints

Optimizing Equation (3) alone empirically converges to degenerate minima, e.g., cyclic or disconnected graphs, that do not correspond to valid query plans. We therefore move to a constrained optimization problem by adding differentiable penalties that quantify the invalidity of a binary A and vanish *iff* it is a valid, binary join tree.

(i) Degree constraints. We want to enforce that triple-pattern nodes have exactly one outgoing edge, while join nodes have exactly two incoming edges and one outgoing edge. Furthermore, the (single) root join node has no outgoing edges. Without loss of generality, let $r = 2n-1$ be the index of the root node. With in-degree $d^{\text{in}}_v = \sum_i A^{\text{soft}}_{iv}$ and out-degree $d^{\text{out}}_v = \sum_j A^{\text{soft}}_{vj}$, we introduce quadratic penalties that are *zero iff* the degree conditions above are satisfied:

$$P_{\text{TO}} = \sum_{v\le n} \big(d^{\text{out}}_v - 1\big)^2, \quad P_{\text{JI}} = \sum_{v>n} \big(d^{\text{in}}_v - 2\big)^2,$$

$$P_{\text{JO}} = (d^{\text{out}}_r)^2 + \sum_{n<v<r} \big(d^{\text{out}}_v - 1\big)^2.$$

Note that we do not require a penalty for the in-degree of the triple pattern, as in a valid join-tree, they never have incoming connections. Instead, all connections from any node to a triple pattern are set to negative infinity in L and hence are 0 after applying the softmax. Similarly, since a valid root node can never have outgoing connections, we mask out all outgoing edges from it.

(ii) Left-linearity. Our approach supports any bushy tree. However, it can be enforced to produce left-linear plans. Let $J = \{n+1, \dots, 2n-1\}$ denote the ordered set of join nodes. We require, without loss of generality (i) the first join node j_0 (index $n+1$) to have two triple-pattern children and no join child, and (ii) every subsequent join node $v \in J \setminus \{j_0\}$ to have exactly one triple-pattern child and one join child. With the sum of incoming connections from triple patterns c^{tp}_v and from join nodes c^{jn}_v to v as

$$c^{\text{tp}}_v = \sum_{u\le n} A^{\text{soft}}_{uv}, \qquad c^{\text{jn}}_v = \sum_{u>n} A^{\text{soft}}_{uv},$$

the left-linear penalty becomes

$$P_{\text{LL}} = \big(c^{\text{tp}}_{j_0} - 2\big)^2 + \big(c^{\text{jn}}_{j_0}\big)^2 + \sum_{v\in J\setminus\{j_0\}} \big[(c^{\text{tp}}_v - 1)^2 + (c^{\text{jn}}_v - 1)^2\big].$$

The penalty is zero iff the plan is left-linear.

(iii) Acyclicity. Any valid join tree does not contain cycles. Zheng *et al.* proposed a differentiable objective to measure the DAG-ness of a graph and hence we use the following penalty to suppress cycles:

$$P_{\text{ACYC}} = \operatorname{tr}\big(e^{A^{\text{soft}}}\big) - N, \tag{4}$$

which is zero exactly if no cycles are present.
Together, these penalties characterize a valid plan:

Theorem 1 (Validity of the structural penalties[2])**.** *Let $n \ge 2$ and $A \in \{0,1\}^{N\times N}$ with $N = 2n-1$, subject to the masking*

[2]See the extended paper for a proof.

introduced above: $A_{iv} = 0$ *for all* $v \leq n$ *and* $A_{rj} = 0$ *for all* j*, with* $r = N$*. Then* $P_{\text{TO}} + P_{\text{JI}} + P_{\text{JO}} + P_{\text{LL}} + P_{\text{ACYC}} = 0$ *if and only if* A *is the adjacency matrix of a valid left-linear plan.*

4.4 Constrained Objective and Optimization

Let $P_{\text{STRUCT}} = \lambda_{\text{TO}} P_{\text{TO}} + \lambda_{\text{JI}} P_{\text{JI}} + \lambda_{\text{LL}} P_{\text{LL}} + \lambda_{\text{JO}} P_{\text{JO}} + \lambda_{\text{ACYC}} P_{\text{ACYC}}$ be the weighted sum of all structural penalties. The final time-dependent objective to be minimized is now given as

$$\mathcal{L}_t = \hat{C}_\theta\big(X, A^{\text{soft}}\big) + \lambda(t)\, P_{\text{STRUCT}}, \tag{5}$$

where $\lambda(t) = \lambda_{\max}(t/I)^q$ nonlinearly increases the influence of the constraints over the I optimization steps, up to a parameter $\lambda_{\max}$. Here, the exponent q is a hyperparameter. Similar to τ, this allows for largely unconstrained exploration early on and forces valid plans at the end of optimization, and was empirically found to be critical for convergence.

The differentiability of the cost model, penalties and relaxed adjacency matrix makes $\mathcal{L}_t$ amenable to standard gradient descent updates:

$$L \leftarrow L - \alpha \nabla_L \mathcal{L}_t. \tag{6}$$

We use GD with Nesterov momentum together with a One Cycle LR schedule for optimization. This enables fast convergence with a minimal number of gradient descent steps.

Projection to a Discrete Plan. To convert the best found soft adjacency A^{soft} to a valid left-linear plan $p^\star$, we perform a simple greedy search: with $T_{free} = \{1, \ldots, n\}, J_{free} = \{n+1, \ldots, 2n-1\}$, starting from the root join $c = 2n-1$ we iteratively select

$$(j^\star, k^\star) = \arg \max_{j \in J_{\text{free}},\, k \in T_{\text{free}}} \big(A^{\text{soft}}_{jc} + A^{\text{soft}}_{kc}\big),$$

and attach the edges $j^\star \to c$ and $k^\star \to c$ to $p^\star$, then set $c \leftarrow j^\star$ and $T_{free} = T_{free} \setminus k^\star, J_{free} = J_{free} \setminus j^\star$; the final join receives the two remaining triples. This projection is guaranteed to yield a valid plan. For the final returned solution, we dynamically retain the best discretized plan encountered so far during optimization. The full pseudocode is provided in the extended version.

5 Experimental Results

5.1 Datasets and Queries

We evaluate GBJO on two popular knowledge graphs: LUBM [Guo *et al.*, 2005] and a subset of Wikidata (Wikidata 5M [Wang *et al.*, 2021]). LUBM is a synthetic dataset used for Database Benchmarking, while Wikidata is one of the largest publicly available knowledge graphs. LUBM is a rather structured dataset, while Wikidata is complex and heterogeneous. For queries, we use the star- and path-shaped queries provided by Schwabe and Acosta and generate additional larger queries using their query generator. Star queries have a single central node that connects to various outer nodes, while a path query is a single chain of nodes. We selected these datasets and queries as they represent both smaller, simpler graphs as well as larger, more complex ones. The query shapes reflect those commonly studied in prior research and cover a wide range of cardinalities and costs, thus providing a representative range of typical query workloads.

5.2 Plan Generation

To train and evaluate the cost model, we need to obtain query plans for the used queries. A trivial way is to generate random join orders. However, this mostly generates plans with mediocre performance and does not provide plans with either very good or very poor performance. Hence, for each query, we generate three random, three good, and three bad plans. To generate good and bad plans, we use a bottom-up beam search with a beam width of three and direct execution on the database. For the good plans, we retain the three cheapest subplans (as per the true C_{out} cost) at each step of the search, while for the bad plans, we retain the three worst plans. [3]

To represent the feature matrix X of a plan, we again follow the scheme of Schwabe and Acosta and use Knowledge Graph Embeddings.

5.3 Cost Model Training

Our approach is agnostic of model architecture and only expects the model to be compatible with a soft adjacency matrix. We choose a simple GNN with 6 message-passing layers and two fully connected layers for graph-level readout, transforming the initial plan graph into a single cost estimate (architecture details are given in the extended version). As the message-passing function, we use *GIN Conv* [Xu *et al.*, 2019] and extend it with edge weights in order to be able to represent the soft adjacencies. The GNN is trained on valid plans with hard adjacencies only to predict the C_{out} cost.

The number of plans used for training and validation along with the median Q-error ($\max\big(\frac{\hat{C}}{C}, \frac{C}{\hat{C}}\big)$) between true ($C$) and predicted ($\hat{C}$) cost is shown in the table below.

	LUBM–Star	**LUBM–Path**	**WD–Star**	**WD–Path**
# Train	61224	37304	72000	35229
# Val	15311	9326	18000	8807
Q-error ↓	1.13	1.07	1.40	2.34

For both LUBM and Wikidata, and both query types, the models are accurately estimating the cost.

5.4 Join Order Optimization

We use the cost models trained in the previous step to compare GBJO against the baseline approaches presented in Section 2.5, as well as two approaches operating on the continuous plan space: NeuralSort [Grover *et al.*, 2019] and CMA-ES [Hansen and Ostermeier, 2001]. CMA-ES is a gradient-free evolutionary approach that optimizes continuous functions by adapting a multivariate normal search distribution over (soft) plans based on the covariance matrix of the more successful samples. All compared search methods use the same model. Most of the hyperparameters are rather robust to changes in their values (as determined using hyperparameter search). The most important hyperparameter was to gradually increase the penalties as opposed to them being applied completely from the beginning (Equation 5). For the approaches that require a set number of iterations, we

[3] All used and generated datasets, as well as the code with instructions on how to reproduce the results, are available on GitHub: https://github.com/TimEricSchwabe/GBJOv2

set: $I_{\text{GBJO}} = 10$; $I_{\text{II}} = 500$; $I_{\text{GEQO}} = 500$; $I_{\text{CMA-ES}} = 1500$; $I_{\text{NeuralSort}} = 10$.

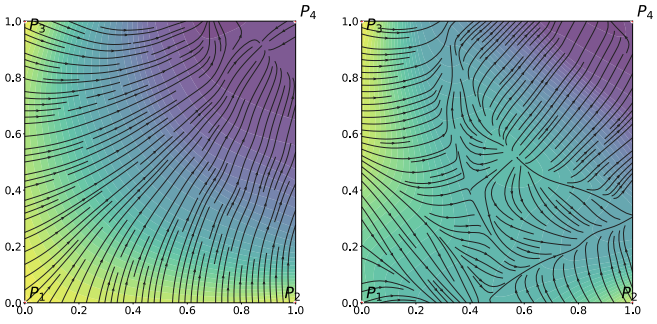
Visualizing the Cost Landscape

To verify that the learned cost model produces a well-behaved landscape outside the set of discrete plans on which it was trained, we visualise the predicted cost landscape between four randomly chosen plans in Figure 3. The plots show the predicted costs (yellow=high; purple=low) for a bilinear interpolation between the plans and the corresponding gradient flow. For the shown query with 8 triple patterns (Figure 3a), P_4 clearly has the lowest cost, and the relaxed landscape exhibits a rather monotonic gradient from all positions to a minimum near P_4. The query with 14 triple pattern in Figure 3b shows a more complex landscape. P_3 and P_2 have high cost, while P_1 now has medium cost. As a result, there exists a local maximum of cost in the center of the interpolation as well as (qualitatively) saddle points between P_1, P_2 and P_2, P_4. An optimizer with a small learning rate and without momentum may become trapped there. During the development of the approach, we indeed found that a higher learning rate and momentum significantly improve the quality of found plans. Since Figure 3 only shows a subspace of the full optimization space, we generated a more global view in Figure 4. For that, we sampled 10,000 different relaxed plans (with all incoming connections to triple patterns masked out) for a Wikidata star query with 5 triple patterns and represent each plan using the cost model's hidden representation of the plan (before the final graph-level MLP). We then generated 2D embeddings of those representations using UMAP. We show the same plans using a high (Figure 4a) and a low (Figure 4b) temperature. For $\tau = 5$ (corresponding to the early phase of optimization), the representations are smoothly distributed and most importantly, monotonically distributed by cost. For $\tau = 0.01$ (i.e., plans are approximately discrete), clusters that represent similar plans appear. However, there is still a clear trend from high to low cost.

Overall, these results indicate that, even though the model has never been exposed to interpolated plans during training, the relaxed landscape provides meaningful directional information for gradient-based optimization.

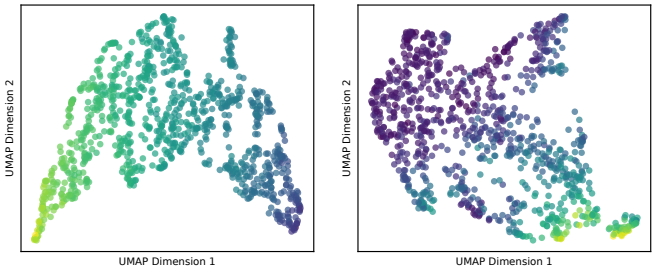
Comparison to Baseline Search Methods

Next, we evaluate how GBJO compares to other search methods. Figure 5 compares the median true costs of plans across increasing query sizes and for both star- and path-shaped queries on LUBM and Wikidata. The results show that GBJO is competitive with the other discrete search methods, and sometimes even surpasses them. It is also robust with respect to growing query sizes. Compared to the costs of random plans, the costs of plans found by GBJO are, for large query sizes, several orders of magnitude lower. Again, this is a surprising result: although the relaxed space between discrete plans does not correspond to physically possible plans, it resembles interpolations between those (and their costs) and is informative to find (using the gradient of the cost) low-cost plans. Furthermore, in comparison to the other two approaches operating on the relaxed space, Neural Sort and CMA-ES, GBJO finds significantly lower costs. This shows



(a) 8 tp (b) 14 tp

Figure 3: Contour plots of the cost landscape between four random plans for Wikidata star queries and the corresponding gradient flow. Even though the space between plans does not represent valid plans, the local gradient still contains directional information towards the better plan.

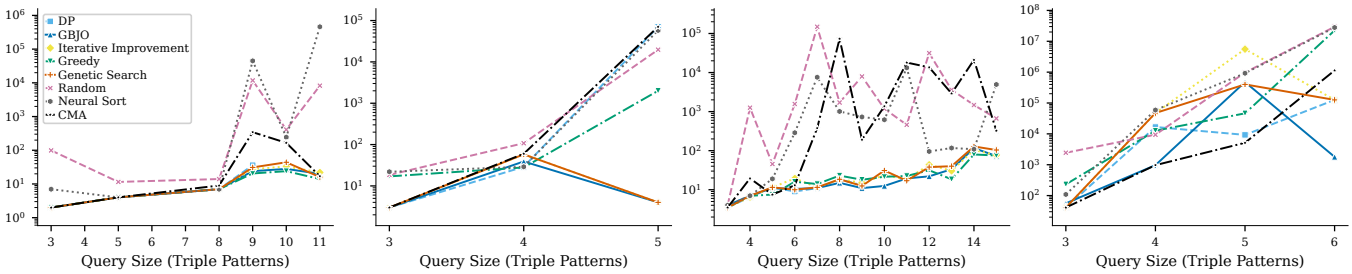


(a) $\tau = 5$ (b) $\tau = 0.01$

Figure 4: UMAP Projection of soft plans for a single query and their predicted costs using the hidden representation of the Cost Model for different temperatures. For both high and low temperatures, costs smoothly change in the representation space.

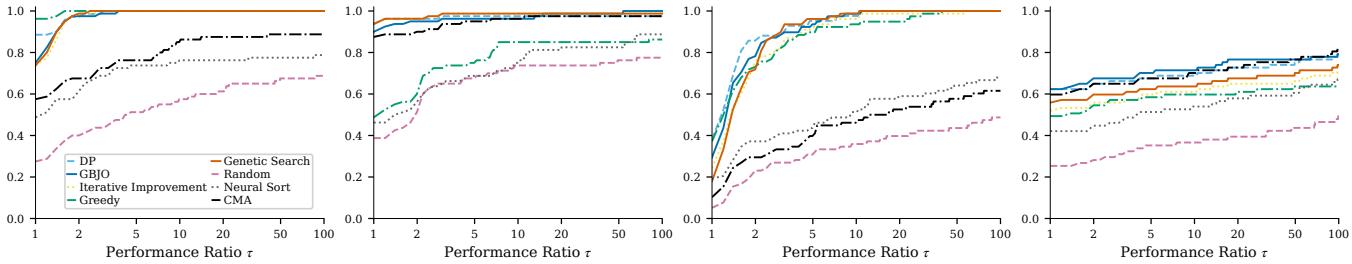
that the specific combination of model prediction, temperature annealing, and time-dependent penalties (compared to Neural Sort), as well as using gradients to traverse the continuous plan space (compared to CMA-ES), is required to reach good results. Note again that since the cost model does not fulfill the Bellman optimality principle, DP does not necessarily yield the best found solutions (it is, however, still a strong baseline as the results show).

Figure 6 displays the Dolan-Moré performance profiles for the different search methods on the four datasets. For each method, a Dolan-Moré profile shows the fraction of all queries for which this method is within a factor τ of the best-performing algorithm for each query. Hence, the y-intercept shows the percentage of cases where a given algorithm was the best among all others, while the tail to the right visualizes the robustness of an approach (the worst behavior compared to the best solution). In general, the higher and further to the left the curve of a given algorithm is, the better it performs. The performance plots show similar results compared to the median costs. GBJO has similar performance curves to the discrete baselines. Its robustness is also similar to that of the discrete baselines, showing that GBJO does not find plans significantly worse than the best found solution more often than the other approaches.



(a) LUBM - Star Queries (b) LUBM - Path Queries (c) Wikidata - Star Queries (d) Wikidata - Path Queries

Figure 5: Median true costs (y-axis) of plans with respect to query size found by different join-order algorithms for LUBM and Wikidata.



(a) LUBM - Star Queries (b) LUBM - Path Queries (c) Wikidata - Star Queries (d) Wikidata - Path Queries

Figure 6: Dolan-Moré performance profiles of the compared join ordering algorithms showing the probability (y-axis) that a given method is within a fraction τ of the best solution found.

Altogether, the results demonstrate that GBJO can match and even surpass commonly used discrete join ordering algorithms. GBJO can meaningfully traverse the continuous space of plan superpositions. This is intriguing since the cost model never saw plan superpositions during training, nor do they correspond to valid plans. Lastly, GBJO only uses $I = 10$ evaluations of the cost model to guide the search as opposed to several hundred for the other randomized approaches (and Greedy Search for larger queries).

5.5 Runtime Performance

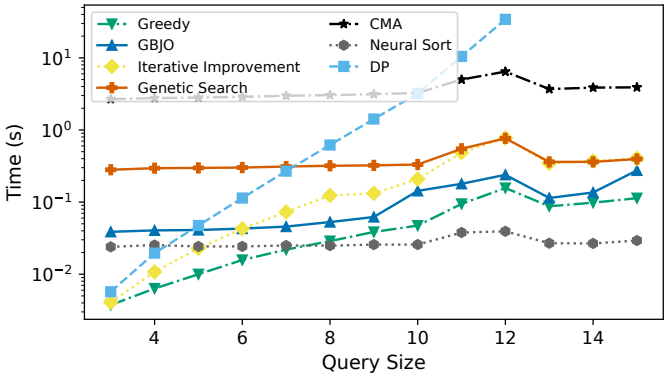


Figure 7: Search runtimes of the compared baselines. GBJO has low runtime and is only dominated by Greedy Search and Neural Sort.

Lastly, Figure 7 shows the wall-clock time w.r.t. query size for GBJO and the compared algorithms on consumer-grade CPU.

As initially stated, DP exhibits an exponential increase in runtime. CMA-ES and GEQO have a higher and rather constant runtime due to the higher and constant number of steps. Iterative Improvement initially has a low runtime (as it quickly finds local minima), but its runtime quickly grows towards that of GEQO. Greedy Search, as expected, starts off with a very small runtime but has a steeper, quadratic trend. Due to the small number of required steps, GBJO also has a small runtime that is only dominated by Greedy Search and Neural Sort. Importantly, it exhibits a more favorable trend compared to Greedy Search. The absolute runtime between 40 and 200 ms makes GBJO a practical option for real-time join order optimization. Furthermore, since the current implementation is not runtime-optimized, substantial improvements in absolute time are expected by compiling the forward and backward passes into static graphs and employing additional runtime optimization techniques.

6 Conclusion and Outlook

We demonstrated the feasibility of approaching join order optimization using gradient descent by continuously relaxing the discrete search over query plan trees. Our results indicate that gradient-based search can successfully find similar or better plans compared to discrete search methods, and do so with favourable absolute runtime and complexity.

For future work, the runtime can be further enhanced using model compilation and efficient kernels for the presented penalties.

References

[Acosta *et al.*, 2025] Maribel Acosta, Chang Qin, and Tim Schwabe. *Neuro-Symbolic Query Optimization in Knowledge Graphs*. IOS Press, March 2025.

[Aytimur *et al.*, 2025] Mehmet Aytimur, Theodoros Chondrogiannis, and Michael Grossniklaus. Space: Cardinality estimation for path queries using cardinality-aware sequence-based learning. *Proceedings of the ACM on Management of Data*, 3(3):1–26, 2025.

[Casals *et al.*, 2023] Daniel Casals, Carlos Buil-Aranda, and Carlos Valle. SPARQL query execution time prediction using deep learning. *Semantic Web*, 2023.

[Chen *et al.*, 2022] Jin Chen, Guanyu Ye, Yan Zhao, Shuncheng Liu, Liwei Deng, Xu Chen, Rui Zhou, and Kai Zheng. Efficient join order selection learning with graph-based representation. In Aidong Zhang and Huzefa Rangwala, editors, *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 97–107. ACM, 2022.

[Chen *et al.*, 2023] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. LOGER: A learned optimizer towards generating efficient and robust query execution plans. *Proc. VLDB Endow.*, 16(7):1777–1789, 2023.

[Cluet and Moerkotte, 1995] Sophie Cluet and Guido Moerkotte. On the complexity of generating optimal left-deep processing trees with cross products. In Georg Gottlob and Moshe Y. Vardi, editors, *Database Theory - ICDT'95, 5th International Conference, Prague, Czech Republic, January 11-13, 1995, Proceedings*, volume 893 of *Lecture Notes in Computer Science*, pages 54–67. Springer, 1995.

[Cui and Zhu, 2021] Guofeng Cui and He Zhu. Differentiable synthesis of program architectures. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 11123–11135, 2021.

[Fegaras, 1998] Leonidas Fegaras. A new heuristic for optimizing large queries. In Gerald Quirchmayr, Erich Schweighofer, and Trevor J. M. Bench-Capon, editors, *Database and Expert Systems Applications, 9th International Conference, DEXA '98, Vienna, Austria, August 24-28, 1998, Proceedings*, Lecture Notes in Computer Science, pages 726–735. Springer, 1998.

[Grover *et al.*, 2019] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Guo *et al.*, 2005] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Semant.*, 3(2-3):158–182, 2005.

[Hansen and Ostermeier, 2001] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, 2001.

[Kipf *et al.*, 2018] Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural relational inference for interacting systems. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2693–2702. PMLR, 2018.

[Kipf *et al.*, 2019] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org, 2019.

[Lecorvé *et al.*, 2022] Gwénolé Lecorvé, Morgan Veyret, Quentin Brabant, and Lina M. Rojas Barahona. SPARQL-to-text question generation for knowledge-based conversational applications. In Yulan He, Heng Ji, Sujian Li, Yang Liu, and Chua-Hui Chang, editors, *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 131–147, Online only, November 2022. Association for Computational Linguistics.

[Liu *et al.*, 2019] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Marcus and Papaemmanouil, 2018] Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In Rajesh Bordawekar and Oded Shmueli, editors, *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, pages 3:1–3:4. ACM, 2018.

[Marcus *et al.*, 2019] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, 2019.

[Mohanaraj *et al.*, 2025] Abiram Mohanaraj, Matteo Lissandrini, Katja Hose, et al. Planrgcn: Predicting sparql query performance. *PROCEEDINGS OF THE VLDB ENDOWMENT*, 18(6):1621–1634, 2025.

[Ng *et al.*, 2020] Ignavier Ng, AmirEmad Ghassami, and Kun Zhang. On the role of sparsity and DAG constraints for learning linear dags. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural*

Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[PostgreSQL Global Development Group, 2025] PostgreSQL Global Development Group. Postgresql 18.1 documentation: 61.3. genetic query optimization (geqo) in postgresql. https://www.postgresql.org/docs/18/geqo-pg-intro.html, 2025. Accessed: 2026-01-02.

[Prillo and Eisenschlos, 2020] Sebastian Prillo and Julian Martin Eisenschlos. Softsort: A continuous relaxation for the argsort operator. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7793–7802. PMLR, 2020.

[Qin and Acosta, 2025] Chang Qin and Maribel Acosta. Neuro-symbolic adaptive query processing over knowledge graphs. In *International Semantic Web Conference*, pages 253–270. Springer, 2025.

[Schwabe and Acosta, 2024] Tim Schwabe and Maribel Acosta. Cardinality estimation over knowledge graphs with embeddings and graph neural networks. *Proc. ACM Manag. Data*, 2(1):44:1–44:26, 2024.

[Selinger *et al.*, 1979] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In Philip A. Bernstein, editor, *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, USA, May 30 - June 1*, pages 23–34. ACM, 1979.

[Steinbrunn *et al.*, 1997] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB J.*, 6(3):191–208, 1997.

[Sun and Li, 2019] Ji Sun and Guoliang Li. An end-to-end learning-based cost estimator. *Proc. VLDB Endow.*, 13(3):307–319, 2019.

[Swami, 1989] Arun N. Swami. Optimization of large join queries: Combining heuristic and combinatorial techniques. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, USA, May 31 - June 2, 1989*, pages 367–376. ACM Press, 1989.

[V *et al.*, 2023] Jyothir S. V, Siddhartha Jalagam, Yann LeCun, and Vlad Sobal. Gradient-based planning with world models. *CoRR*, abs/2312.17227, 2023.

[Vowels *et al.*, 2023] Matthew J. Vowels, Necati Cihan Camgöz, and Richard Bowden. D'ya like dags? A survey on structure learning and causal discovery. *ACM Comput. Surv.*, 55(4):82:1–82:36, 2023.

[Wang *et al.*, 2021] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Trans. Assoc. Comput. Linguistics*, 9:176–194, 2021.

[Whitley *et al.*, 1989] L Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the 3rd international conference on genetic algorithms*, pages 133–140, 1989.

[Xie *et al.*, 2019] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Yan *et al.*, 2023] Zhengtong Yan, Valter Uotila, and Jiaheng Lu. Join order selection with deep reinforcement learning: Fundamentals, techniques, and challenges. *Proc. VLDB Endow.*, 16(12):3882–3885, 2023.

[Yu *et al.*, 2020] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. Reinforcement learning with tree-lstm for join order selection. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1297–1308. IEEE, 2020.

[Zhao *et al.*, 2021] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. A learned sketch for subgraph counting. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 2142–2155. ACM, 2021.

[Zhao *et al.*, 2022] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. Queryformer: A tree transformer model for query plan representation. *Proc. VLDB Endow.*, 15(8):1658–1670, 2022.

[Zheng *et al.*, 2018] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with NO TEARS: continuous optimization for structure learning. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9492–9503, 2018.

[Zhu *et al.*, 2024] Rong Zhu, Lianggui Weng, Bolin Ding, and Jingren Zhou. Learned query optimizer: What is new and what is next. In Pablo Barceló, Nayat Sánchez-Pi, Alexandra Meliou, and S. Sudarshan, editors, *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago, Chile, June 9-15, 2024*, pages 561–569. ACM, 2024.