Exploring Multi-Table Retrieval Through Iterative Search

Allaa Boutaleb, Bernd Amann, Rafael Angarita and Hubert Naacke
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
{firstname.lastname}@lip6.fr

Abstract

Open-domain question answering over datalakes requires retrieving and composing information from multiple tables, a challenging subtask that demands semantic relevance and structural coherence (e.g., joinability). While exact optimization methods like Mixed-Integer Programming (MIP) can ensure coherence, their computational complexity is often prohibitive. Conversely, simpler greedy heuristics that optimize for query coverage alone often fail to find these coherent, joinable sets. This paper frames multi-table retrieval as an iterative search process, arguing this approach offers advantages in scalability, interpretability, and flexibility. We propose a general framework and a concrete instantiation: a fast, effective Greedy Join-Aware Retrieval algorithm that holistically balances relevance, coverage, and joinability. Experiments across 5 NL2SQL benchmarks demonstrate that our iterative method achieves competitive retrieval performance compared to the MIP-based approach while being 4-400x faster depending on the benchmark and search space settings. This work highlights the potential of iterative heuristics for practical, scalable, and composition-aware retrieval.

1 Introduction

Large Language Models (LLMs) increasingly leverage Retrieval-Augmented Generation (RAG) pipelines for natural language interfaces to structured databases [Li et al., 2023, Wang et al., 2024]. In this paradigm, the accuracy and coherence of an LLM's answer is critically dependent on the retrieval phase: if the retrieved tables do not collectively contain the necessary information or cannot be coherently composed (e.g., via joins), the LLM's ability to generate a correct answer is severely compromised. While previous work on table retrieval has primarily focused on retrieving individual tables from large corpora [Herzig et al., 2021], complex real-world queries often require retrieving and composing information from multiple tables—a challenge known as **multi-table retrieval**. This task introduces significant challenges beyond simple relevance ranking. Consider a query like: *"Show names and total order values for New York customers who ordered 'Laptop' and any customers who ordered 'Smartphone' after 2024."* Answering this requires tables like Customers, Orders, and Products, but the retrieval process faces several issues. Due to data redundancy or different valid reasoning paths, **multiple valid sets** of tables might answer the query. Furthermore, **path dependency** can arise in iterative methods, where the order of table selection influences the final set. This inherent ambiguity suggests that multi-table retrieval is not always about finding a single, pre-defined optimal set, but often involves exploring multiple plausible evidence paths.

Traditional retrieval methods, which assess tables independently, struggle with structural composability. Chen et al. [2024b] addressed this by proposing **composition-aware** retrieval focused on joinability, reframing the task as selecting a coherent sub-graph. Their Join-Aware Retrieval (JAR) method performs neural retrieval followed by a Mixed-Integer Program (MIP) to find the provably optimal joinable subset. Although highly effective, this one-shot NP-hard optimization suffers in terms of scalability, as acknowledged by the authors. JAR has inspired follow-ups com-

Workshop: AI for Tabular Data workshop at EurIPS 2025.

bining solver logic with LLMs [Chen et al., 2025] and agentic, multi-hop reasoning approaches like MURRE [Zhang et al., 2025]. Other related works like CRUSH4SQL [Kothyari et al., 2023] use a greedy decomposition strategy, but lack the explicit join-aware component central to JAR. Recent works such as DBCopilot [Wang et al., 2025] also explore iterative retrieval via schema routing for massive databases. Complementary to these efforts, recent work has also proposed retrieval-augmented methods that enable LLMs to interact with structured data via iterative and adaptive search frameworks [Wang et al., 2024].

This paper proposes framing multi-table retrieval as an **iterative and explorative search process**. To stimulate discussion on practical, scalable architectures for this complex retrieval task, we posit that constructing the table set step-by-step offers benefits in scalability (via heuristics), interpretability, and flexibility. Our contributions are threefold: (1) We propose a general, flexible iterative framework for multi-table retrieval. (2) We detail a fast and effective **Greedy Join-Aware Retrieval** algorithm as a concrete instantiation of this framework. (3) We demonstrate its empirical viability on standard benchmarks (Yu et al. [2018], Li et al. [2023]) as well as more complex enterprise benchmarks (Sen et al. [2020], Chen et al. [2024a]), showing it achieves competitive performance to the MIP-based JAR approach while being over 4-400x faster[1].

2 An Iterative Framework for Multi-Table Search

We propose framing multi-table retrieval as a sequential decision-making process where an algorithm iteratively expands a set of selected tables based on a dynamic **context**. This approach offers several advantages over one-shot global optimization:

- **Interpretability:** Each selection step provides a checkpoint for analyzing the reasoning process, potentially enabling human-in-the-loop guidance.
- **Extensibility:** The framework is modular and the selection logic can be dynamically adapted to prioritize different operators (e.g., JOIN vs. UNION) or evolving objectives.
- **Heuristic Potential:** While global optimization is often NP-hard for this task, step-by-step construction lends itself to efficient polynomial-time heuristics, addressing scalability.

Representing Query Coverage A central challenge in multi-table retrieval is tracking the capacity to answer the initial query by the already selected tables. To illustrate, consider the query *"For movies with the keyword of 'civil war', calculate the average revenue generated by these movies"*. To estimate what information is needed, methods process this query using an LLM in different ways. For instance, Kothyari et al. [2023] use the LLM to "hallucinate" a minimal, potential schema (e.g., `movies(title, revenue)`) that could answer it. In contrast, our method (following Chen et al. [2024b]) decomposes the query into a set of fine-grained concepts or sub-queries $\{q_j\}$, such as {`movies:keyword`, `movies:revenue`}. This general idea of estimating the query's requirements is known as *query coverage*. In our framework, query coverage tracks how well each concept q_j is addressed by the tables selected so far.

Abstract Formulation Let $\mathcal{T}$ be the set of all available candidate tables. The search process evolves at each step k a **context** $\mathcal{C}_k = (G_k, \mathbf{Q}_k)$ where:

- $G_k = (S_k, E_k)$ is a graph of k selected tables $S_k \subseteq \mathcal{T}$ and their discovered relationships E_k (e.g., potential joins).
- $\mathbf{Q}_k$ represents the query coverage state for the query concepts $\{q_j\}$. For example, in a simple setting we can represent $\mathbf{Q}_k$ by a vector $\mathbf{q}_k$ whose size equals the number of concepts, where element $q_{k,j}$ quantifies coverage for concept q_j.

The process starts with the empty context $\mathcal{C}_0 = ((\emptyset, \emptyset), \mathbf{0})$ with no selected tables and no coverage. At each step k, a **selection function** Φ chooses the next table T_{k+1} from the remaining candidates $\mathcal{T} \setminus S_k$ by maximizing a *context-dependent utility function* U: $T_{k+1} = \Phi(\mathcal{C}_k, \mathcal{T} \setminus S_k) = \arg\max_{T_i \in \mathcal{T} \setminus S_k} U(T_i, \mathcal{C}_k)$. An **update function** Ψ then transitions the system to the next state $\mathcal{C}_{k+1}$ by incorporating T_{k+1}: $\mathcal{C}_{k+1} = \Psi(\mathcal{C}_k, T_{k+1}) = (G_{k+1}, \mathbf{Q}_{k+1})$. This process repeats until a stopping criterion is met (e.g., $k = K$ or $\min_j q_{k,j} \geq \theta$). We illustrate a concrete instantiation of the selection and update functions in the next section.

[1] Code available at: `https://github.com/Allaa-boutaleb/iterative-jar/`

3 Case Study: Greedy Join-Aware Retrieval

In this section we present a simple instantiation of the iterative framework focused on *join-aware* multi-table retrieval. Our algorithm, like JAR [Chen et al., 2024b], operates on pre-computed scores quantifying relevance and compatibility for a query Q and candidate tables $\mathcal{T}$. These include:

- **Coarse-grained Relevance (r_i)**, the overall Q-to-table T_i semantic similarity via dense retriever embeddings [Izacard et al., 2022], where $\text{emb}(\cdot)$ is the embedding function:
$$r_i = \cos(\text{emb}(Q), \text{emb}(T_i)) \in [-1, 1] \tag{1}$$
- **Fine-grained Relevance (F_{ji})**, which measures how well T_i addresses a specific sub-query q_j (decomposed from Q via an LLM) by finding the maximum similarity between q_j and any column c in T_i: $F_{ji} = \max_{c \in \text{cols}(T_i)} \cos(\text{emb}(q_j), \text{emb}(c)) \in [-1, 1]$.
- **Join Compatibility (ω_{il})**, a score in $[0, 1]$ quantifying the join likelihood between T_i and T_l by combining schema, value overlap, and uniqueness signals to approximate a PK-FK link, following Chen et al. [2024b].

The **context** is $\mathcal{C}_k = (G_k, \mathbf{Q}_k)$ where $G_k = (S_k, E_k)$ is the graph of selected tables S_k and their join paths E_k. The **coverage state** $\mathbf{Q}_k$ is implemented as the vector $\mathbf{q}_k$ storing the maximum F_{ji} score seen for each sub-query q_j. The algorithm's **selection function** Φ works by maximizing a utility function U that is a weighted sum of three marginal gain components. These gains are calculated for a candidate table T_i relative to the prior context $\mathcal{C}_{k-1} = (G_{k-1}, \mathbf{Q}_{k-1})$:

- **Coarse Relevance Gain**, $G_{\text{coarse}}(T_i) = r_i$, which is intrinsic to the table;
- **Marginal Coverage Gain**, which depends on the prior coverage vector $\mathbf{q}_{k-1}$:
$$G_{\text{cov}}(T_i|\mathcal{C}_{k-1}) = \sum_j \max(0, F_{ji} - (\mathbf{q}_{k-1})_j) \tag{2}$$
- **Marginal Join Gain**, $G_{\text{join}}(T_i|\mathcal{C}_{k-1}) = \sum_{T_l \in S_{k-1}} \omega_{il}$, which depends on the nodes S_{k-1} of the prior graph G_{k-1}. For the main iterative step ($k > 1$), the utility function $U(T_i, \mathcal{C}_{k-1})$ combines these gains:
$$T_k = \arg \max_{T_i \in \mathcal{T} \setminus S_{k-1}} [\lambda_{\text{coarse}} G_{\text{coarse}}(T_i) + \lambda_{\text{cov}} G_{\text{cov}}(T_i|\mathcal{C}_{k-1}) + \lambda_{\text{join}} G_{\text{join}}(T_i|\mathcal{C}_{k-1})] \tag{3}$$

The seed selection ($k = 1$) is a special case. Starting from $\mathcal{C}_0 = ((\emptyset, \emptyset), \mathbf{0})$, the G_{join} term is undefined and G_{cov} simplifies (as $\mathbf{q}_0 = \mathbf{0}$), so the utility function reduces to selecting based on individual merit:

$$T_1 = \arg \max_{T_i \in \mathcal{T}} \left[\lambda_{\text{coarse}} \cdot r_i + \lambda_{\text{cov}} \cdot \sum_j F_{ji} \right] \tag{4}$$

Finally, the **update function** Ψ transitions the state to $\mathcal{C}_k$ by updating the graph $G_k = (S_{k-1} \cup \{T_k\}, E_{k-1} \cup E_{\text{new}})$ and the coverage vector $\mathbf{q}_k = \max(\mathbf{q}_{k-1}, \mathbf{F}_k)$, where $\mathbf{F}_k$ is the vector of fine-grained scores for T_k.

4 Experiments and Analysis

Experimental Setup. We compare our iterative greedy algorithm against three baselines: a **Dense Retrieval (Contriever)** baseline [Izacard et al., 2022], the official **JAR**$_{MIP}$ implementation[2] [Chen et al., 2024b], and **CRUSH** [Kothyari et al., 2023] (using Contriever as the embedding model). For all methods, we use a consistent table schema serialization. The Contriever baseline embeds this serialized schema, the query, and its concepts within the same embedding space to retrieve an initial ranked list of tables. For JAR (MIP) and our iterative method, we re-rank this initial candidate set: the **top-20** tables for SPIDER, BIRD, and BEAVER, and the **top-30** for FIBEN. Both methods use the identical set of pre-computed relevance (r_i, F_{ji}) and join compatibility (ω_{il}) scores, ensuring a fair comparison of the selection algorithms. For CRUSH, we follow the default settings from its implementation[3], using an initial candidate cutoff of 100 tables and a selection budget of 20.

[2] `https://github.com/peterbaile/jar`

[3] `https://github.com/tshu-w/DBCopilot/blob/master/scripts/crush4sql.py`

We evaluate on the multi-table queries from SPIDER [Yu et al., 2018], BIRD [Li et al., 2023], FIBEN [Sen et al., 2020], and the complex BEAVER [Chen et al., 2024a] benchmark (see Table 1 for statistics). BEAVER is composed of two distinct enterprise data warehouses: BEAVER-DW (university physical administration) and BEAVER-NW (virtual machine and network infrastructure). As our study focuses on multi-table retrieval, we exclude all single-table queries from our evaluation. A detailed breakdown of the multi-table query distribution for each benchmark is available in Appendix C.

For our greedy method, we use weights $\lambda_{\text{cov}} = 2.0, \lambda_{\text{join}} = 1.0, \lambda_{\text{coarse}} = 4.0$. These weights were determined empirically via an ablation study (Appendix A), which indicated that coarse relevance is the strongest signal, followed by coverage and join compatibility. We report **Recall (R)** and **Complete Recall (CR)** for K $\in \{2, 3, 5, 10\}$, where CR is a binary set-level metric indicating if all ground-truth tables were retrieved. We use a **1 hour** per-query timeout for the MIP solver to allow it ample time to find optimal solutions.

Table 1: Retrieval performance on multi-table queries. K is the number of tables retrieved. Contriever shows absolute R/CR scores (%) and base retrieval time (s). All other methods show relative R/CR gain (+) or loss (-) and total re-ranking time (+ s). † indicates prohibitive runtime. **Highest** and Second-Highest scores are marked per column within each benchmark.

Benchmark	Method	K=2		K=3		K=5		K=10		Time (seconds)
		Recall (R)	CR	Recall (R)	CR	Recall (R)	CR	Recall (R)	CR	
SPIDER	Contriever	81.3	59.9	93.8	85.6	98.9	97.6	**99.7**	**99.3**	15
Num. of DBs: 20 Num. of Tables: 81 Avg. Table Width: 5.4 Num. of Queries: 459	JAR (MIP)	+4.1	+7.2	**+2.6**	**+5.7**	**+0.5**	**+1.1**	†	†	+ (325 / 290 / 6360 / †)
	JAR$_{\text{iterative}}$ (Ours)	**+4.2**	**+8.1**	+1.8	+4.4	-0.9	-2.0	-1.4	-2.8	+ ~20
	CRUSH$_{\text{Contriever}}$	-17.6	-25.3	-21.4	-38.1	-21.5	-42.0	-21.9	-43.3	+ ~38
BIRD	Contriever	63.5	34.8	76.2	56.2	85.2	71.3	95.1	89.9	23
Num. of DBs: 11 Num. of Tables: 75 Avg. Table Width: 10.6 Num. of Queries: 1172	JAR (MIP)	**+10.9**	**+16.9**	**+9.4**	**+15.7**	+5.2	+10.4	†	†	+ (8501 / 13679 / 40756 / †)
	JAR$_{\text{iterative}}$ (Ours)	+10.7	+16.7	+8.5	+14.3	**+5.8**	**+10.8**	**+1.1**	**+2.7**	+ ~101
	CRUSH$_{\text{Contriever}}$	-12.8	-12.8	-13.5	-17.3	-12.4	-16.9	-20.6	-32.8	+ ~116
FIBEN	Contriever	22.8	0.7	26.1	1.1	32.0	1.8	41.4	5.4	13
Num. of DBs: 1 Num. of Tables: 152 Avg. Table Width: 2.5 Num. of Queries: 279	JAR (MIP)	**+7.9**	**+3.6**	**+12.2**	**+6.1**	+16.4	+8.2	+17.2	+8.6	+ (45 / 55 / 2206 / 11340)
	JAR$_{\text{iterative}}$ (Ours)	+6.4	**+3.6**	+10.7	+5.7	**+19.8**	**+9.0**	**+21.2**	**+9.3**	+ ~10
BEAVER-DW	Contriever	**29.3**	1.7	37.5	9.2	48.5	17.5	63.5	30.8	13
Num. of DBs: 1 Num. of Tables: 97 Avg. Table Width: 15.8 Num. of Queries: 120	JAR (MIP)	-21.1	-1.7	-13.3	-0.9	†	†	†	†	+ (18659 / 34756 / †/ †)
	JAR$_{\text{iterative}}$ (Ours)	-0.3	**+0.8**	**+4.2**	**+3.3**	**+4.1**	**+6.7**	**+1.5**	**+2.5**	+ ~82
BEAVER-NW	Contriever	23.6	1.2	30.2	1.2	38.5	2.3	48.4	9.3	19
Num. of DBs: 5 Num. of Tables: 366 Avg. Table Width: 7.4 Num. of Queries: 86	JAR (MIP)	**+10.3**	0.0	**+9.4**	+3.5	+10.7	-2.3	-0.4	-9.3	+ (210 / 572 / 7297 / 10449)
	JAR$_{\text{iterative}}$ (Ours)	+3.0	0.0	+6.5	**+4.6**	**+11.1**	**+8.2**	**+6.1**	**+3.5**	+ ~30

Analysis of Performance and Scalability. As shown in Table 1, our greedy method achieves highly competitive, and often superior, retrieval performance compared to the JAR$_{MIP}$ baseline, while being dramatically faster. For example, on BIRD at K=3, our method achieves 99% of JAR's CR score in only 0.7% of the time (101s vs 13679s, a >135x speedup). This efficiency advantage becomes an enabling factor on the complex BEAVER benchmarks. Our iterative method was able to complete all BEAVER-DW runs, while JAR (MIP) timed out on K$\geq$5. On BEAVER-NW (K=5), our method was >240x faster (30s vs 7297s) while achieving a +10.5 point higher CR.

It is also worth noting that even when JAR (MIP) finds a 100% optimal solution according to its objective function (e.g., on SPIDER at K=2, see Appendix B), our iterative method can still achieve slightly higher performance (see Table 7). This suggests the MIP's objective is not a perfect proxy for the end retrieval metrics, further motivating the exploration of alternative heuristic formulations.

CRUSH's lower performance stems from two core design differences: it is not explicitly join-aware, and it employs an aggressive coverage-first greedy logic, unlike our holistic gain-based approach. As analyzed in Appendix D, this heuristic is sensitive to noise and can fail to select optimal tables. Given the performance drop on SPIDER and BIRD, we skipped evaluating CRUSH on FIBEN and BEAVER benchmarks.

Finally, the slight performance drop of our method on SPIDER at K=5 and K=10 (relative to the Contriever baseline) can be attributed to the fixed candidate set. As K increases, the re-ranker is forced to include lower-ranked tables from the initial top-20 set, which are more likely to be irrelevant and can create confusion for the selection algorithm on certain queries.

5 Limitations and Future Work

This paper proposed reframing multi-table retrieval as an **iterative, explorative search process**, an alternative to one-shot optimization with inherent advantages in scalability, interpretability, and extensibility. Our **Greedy Iterative Join-Aware Multi-Table Retrieval algorithm** demonstrated this viability, achieving strong empirical results on standard and complex enterprise benchmarks at a fraction of the computational cost of the MIP-based JAR method [Chen et al., 2024b].While effective, our greedy approach is sensitive to its initial seed selection; a poor start can lead to a suboptimal final set.

A crucial direction for future work is to enhance the robustness of the iterative search. The framework's step-by-step nature lends itself well to incorporating **backtracking mechanisms**. Exploring strategies where the algorithm can revisit earlier decisions if subsequent steps yield low utility could significantly mitigate the risks associated with greedy choices.Furthermore, the framework's true potential lies in its **extensibility beyond joins**, such as incorporating the UNION operator. This involves modifying the selection utility $U(T_i, \mathcal{C}_k)$ to reward schema compatibility and row diversity. The iterative context $\mathcal{C}_k$ allows for dynamic strategies, like prioritizing joins then exploring unions. Exploring such adaptive strategies, potentially learning operator priority, is a rich research avenue.

Future work must also investigate the framework's robustness to the quality of the initial LLM-based query decomposition and test on more diverse, open-domain datasets with even more complex join paths. Hybrid models, using a fast greedy search by default but triggering a complex solver for specific patterns, also warrant investigation. To provide a broader comparative study, we also plan to evaluate our iterative framework against other recent LLM-Based and schema-routing approaches, such as MURRE [Zhang et al., 2025] and DBCopilot [Wang et al., 2025]. We advocate for continued research into practical, scalable, and flexible architectures for multi-table retrieval.

References

Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. Beaver: an enterprise benchmark for text-to-sql. *arXiv preprint arXiv:2409.02038*, 2024a.

Peter Baile Chen, Yi Zhang, and Dan Roth. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2687–2699, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.148. URL https://aclanthology.org/2024.acl-long.148.

Peter Baile Chen, Yi Zhang, Mike Cafarella, and Dan Roth. Can we retrieve everything all at once? ARM: An alignment-oriented LLM-based retrieval method. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30298–30317, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1463. URL https://aclanthology.org/2025.acl-long.1463.

Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. Open domain question answering over tables via dense retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 512–519, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.43. URL https://aclanthology.org/2021.naacl-main.43.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=jKN1pXi7b0.

Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. CRUSH4SQL: Collective retrieval using schema hallucination for Text2SQL. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14054–14066, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.868. URL https://aclanthology.org/2023.emnlp-main.868/.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQL. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. Athena++: natural language querying for complex nested sql queries. *Proc. VLDB Endow.*, 13(12):2747–2759, July 2020. ISSN 2150-8097. doi: 10.14778/3407790.3407858. URL https://doi.org/10.14778/3407790.3407858.

Mingzhu Wang, Yuzhe Zhang, Qihang Zhao, Junyi Yang, and Hong Zhang. Redefining information retrieval of structured database via large language models. *arXiv preprint arXiv:2405.05508*, 2024. doi: 10.48550/arXiv.2405.05508. URL https://arxiv.org/abs/2405.05508.

Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. Dbcopilot: Natural language querying over massive databases via schema routing. In Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic, editors, *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 707–721. OpenProceedings.org, 2025. doi: 10.48786/EDBT.2025.57. URL https://doi.org/10.48786/edbt.2025.57.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. MURRE: Multi-hop table retrieval with removal for open-domain text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5789–5806, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.386.

A Ablation Study on Coefficients

Table 2 details the ablation study on the coefficients for the utility function (Eq. 3) of our iterative greedy algorithm. We tested isolating each component ($\lambda = 1$ for one, 0 for others) and removing each component ($\lambda = 1$ for two, 0 for one). These results informed our Custom Configuration ($\lambda_{\text{coarse}} = 4, \lambda_{\text{cov}} = 2, \lambda_{\text{join}} = 1$), which consistently performs well. The study confirms that coarse relevance is the strongest signal, followed by coverage, and then join compatibility.

Table 2: Ablation study on utility function coefficients. All runs use the top-20 candidates from Contriever, except for FIBEN which uses top-30. Metrics are Recall (R) and Complete Recall (CR) at K. The **bold** values indicate the best score for each metric within each benchmark, and underlined values indicate the second-best.

Benchmark	Setting ($\lambda_{\text{cov}}, \lambda_{\text{join}}, \lambda_{\text{coarse}}$)	R@2 / CR@2		R@3 / CR@3		R@5 / CR@5		R@10 / CR@10	
		R	CR	R	CR	R	CR	R	CR
SPIDER (k=20)	Only Coverage (1,0,0)	74.2	50.3	78.6	57.7	78.8	57.7	78.9	58.0
	Only Join (0,1,0)	71.5	41.8	84.1	67.8	86.1	72.3	89.6	79.1
	Only Coarse (0,0,1)	81.3	59.9	93.8	85.6	98.9	97.6	**99.7**	**99.3**
	No Coarse (1,1,0)	71.6	49.2	78.7	62.3	82.3	69.5	85.9	75.8
	No Join (1,0,1)	83.0	64.3	94.8	87.1	**99.0**	**97.8**	**99.7**	**99.3**
	No Coverage (0,1,1)	74.8	47.7	85.8	71.0	87.2	74.5	90.0	80.2
	Custom Config (2,1,4)	**85.5**	**68.0**	**95.6**	**90.0**	98.0	95.6	98.3	96.5
BIRD (k=20)	Only Coverage (1,0,0)	62.5	32.2	67.5	39.9	68.8	41.8	69.4	42.6
	Only Join (0,1,0)	56.8	26.4	66.6	42.1	79.4	62.8	90.2	81.7
	Only Coarse (0,0,1)	63.5	34.8	76.2	56.2	85.2	71.3	95.1	89.9
	No Coarse (1,1,0)	66.1	39.3	75.7	55.5	84.2	71.2	92.7	86.7
	No Join (1,0,1)	72.2	47.5	84.2	69.5	**91.1**	**82.1**	**96.2**	**92.6**
	No Coverage (0,1,1)	59.9	31.7	73.4	56.3	83.4	71.3	93.9	89.6
	Custom Config (2,1,4)	**74.2**	**51.5**	**84.7**	**70.5**	91.0	**82.1**	**96.2**	**92.6**
FIBEN (k=30)	Only Coverage (1,0,0)	13.6	0.7	16.2	1.4	18.1	1.4	20.5	1.4
	Only Join (0,1,0)	24.0	1.4	33.1	4.3	**53.0**	**11.8**	62.0	13.6
	Only Coarse (0,0,1)	22.8	0.7	26.1	1.1	32.0	1.8	41.4	5.4
	No Coarse (1,1,0)	17.4	2.5	29.6	7.2	48.8	11.1	**62.6**	**14.7**
	No Join (1,0,1)	23.7	1.1	28.9	2.5	34.6	5.0	44.9	7.9
	No Coverage (0,1,1)	26.2	1.8	32.8	4.3	51.6	9.0	62.0	13.6
	Custom Config (2,1,4)	**29.2**	**4.3**	**36.8**	**6.8**	51.8	10.8	**62.6**	**14.7**
BEAVER-DW (k=20)	Only Coverage (1,0,0)	24.4	**2.5**	31.5	4.2	37.7	7.5	42.8	8.3
	Only Join (0,1,0)	25.6	**2.5**	37.7	7.5	44.4	13.3	56.7	22.5
	Only Coarse (0,0,1)	**29.3**	1.7	37.5	9.2	48.5	17.5	63.5	30.8
	No Coarse (1,1,0)	25.6	**2.5**	38.0	11.7	46.4	20.0	61.0	29.2
	No Join (1,0,1)	26.0	**2.5**	37.4	9.2	50.8	20.0	**66.3**	**34.2**
	No Coverage (0,1,1)	27.4	**2.5**	39.8	9.2	48.4	18.3	59.2	26.7
	Custom Config (2,1,4)	29.0	**2.5**	**41.7**	**12.5**	**52.6**	**24.2**	65.0	33.3
BEAVER-NW (k=20)	Only Coverage (1,0,0)	27.9	0.0	32.3	1.2	36.4	1.2	40.8	5.8
	Only Join (0,1,0)	27.3	**2.3**	33.8	**5.8**	41.2	8.1	50.8	11.6
	Only Coarse (0,0,1)	23.6	1.2	30.2	1.2	38.5	2.3	48.4	9.3
	No Coarse (1,1,0)	26.2	**2.3**	34.6	4.7	45.6	7.0	53.7	10.5
	No Join (1,0,1)	**28.2**	0.0	36.4	1.2	44.1	2.3	50.4	8.1
	No Coverage (0,1,1)	26.7	1.2	33.3	**5.8**	40.7	8.1	50.2	11.6
	Custom Config (2,1,4)	26.6	1.2	**36.7**	**5.8**	**49.6**	**10.5**	**54.5**	**12.8**

B MIP Solver Status

Table 3 provides a detailed breakdown of the JAR (MIP) solver's termination status. The experiments were run on a server equipped with Intel(R) Xeon(R) Gold 6330 CPUs @ 2.00GHz, using an allocated resource quota of 4 CPU cores (8 threads) and 64 GB of RAM for this task. We used the Python-MIP library, similar to Chen et al. [2024b], and the statuses correspond to its official numeric codes[4]. The

[4] `https://python-mip.readthedocs.io/en/latest/classes.html`

data shows that while most simple queries (SPIDER K=2, BIRD K=2) were solved to optimality, timeouts become common as complexity increases. This results in either a suboptimal `Feasible` solution (one was found, but not proven optimal) or `No Solution` (no solution was found within the 1-hour limit). Timeouts were prevalent on BIRD (K=5), FIBEN (K=10), BEAVER-NW (K=5, K=10), and most significantly on BEAVER-DW (K=2), which saw a high percentage of timeouts before any solution could be found.

Table 3: JAR (MIP) solver status breakdown for multi-table queries (3600s timeout/query). Percentages relative to total multi-table queries (SPIDER: 459, BIRD: 1172, FIBEN: 279, BEAVER-NW: 86, BEAVER-DW: 120).

Benchmark	K	Optimal (%)	Feasible (%)	No Solution (%)	Infeasible (%)
SPIDER	K=2	459 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=3	459 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=5	459 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
BIRD	K=2	1172 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=3	1172 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=5	1144 (97.6%)	24 (2.0%)	4 (0.3%)	0 (0.0%)
FIBEN	K=2	279 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=3	279 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=5	279 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=10	240 (86.0%)	39 (14.0%)	0 (0.0%)	0 (0.0%)
BEAVER-NW	K=2	86 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=3	86 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	K=5	84 (97.7%)	2 (2.3%)	0 (0.0%)	0 (0.0%)
	K=10	50 (58.1%)	28 (32.6%)	8 (9.3%)	0 (0.0%)
BEAVER-DW	K=2	32 (26.7%)	4 (3.3%)	84 (70.0%)	0 (0.0%)
BEAVER-DW	K=3	45 (37.5%)	28 (23.3%)	47 (39.2%)	0 (0.0%)

The status codes reported in the table are defined as follows:

- **Optimal:** The MIP solver found and proved the globally optimal solution.
- **Feasible:** The solver found a solution but ran out of time before proving optimality.
- **No Solution Found:** The solver exhausted its time limit before finding any solution.
- **Infeasible:** The constraints are contradictory; no solution can satisfy all of them.

C Distribution of Multi-Table Queries

The following table details the distribution of multi-table queries within each benchmark, categorized by the number of tables required to answer the query. These values were calculated based on the list of tables that were mentioned in the corresponding gold SQL queries. For example, in BIRD, one query requires 8 tables to be answered, so it is counted under the "8" column.

Table 4: Distribution of multi-table queries based on the number of tables required by the gold SQL query.

# Tables in Gold Query	Benchmark					Total
	SPIDER	BIRD	FIBEN	BEAVER-DW	BEAVER-NW	
2	393	936	77	5	13	1424
3	60	197	26	44	18	345
4	6	34	98	37	24	199
5	0	1	63	23	4	91
6	0	1	5	8	4	18
7	0	2	2	3	4	11
8	0	1	6	0	1	8
9	0	0	0	0	2	2
10	0	0	0	0	7	7
11	0	0	2	0	1	3
12	0	0	0	0	8	8
Total	**459**	**1172**	**279**	**120**	**86**	**2116**

D Case Study: Analysis of CRUSH Re-ranking Failure

To illustrate the performance dip of CRUSH mentioned in Section 4, we analyze a query from the SPIDER benchmark. This case study highlights how CRUSH's greedy, segment-coverage-first logic can prioritize covering diverse segments with irrelevant tables over selecting a set of relevant, coherent tables.

For the query regarding high schooler friendships, the baseline Contriever successfully retrieves the two correct tables in the top-2. CRUSH, however, selects one correct table and one incorrect table, failing the Complete Recall (CR) metric. Table 5 summarizes the results.

Table 5: Retrieval results for the example query. The baseline (Contriever) succeeds, while CRUSH fails.

Item	Tables	
Gold Tables	`network_1.highschooler`, `network_1.friend`	
Baseline Top-2	`network_1.friend` (0.6121), `network_1.highschooler` (0.5861)	**[CR@2: True]**
CRUSH Top-2	`network_1.friend` (0.4032), `network_2.personfriend` (0.3899)	**[CR@2: False]**

The query was decomposed into four segments. The similarity breakdown for the relevant tables against these segments is shown in Table 6. Note that the baseline's scores (e.g., 0.5861) are the maximum similarity from this table, which CRUSH uses as input.

Table 6: Similarity breakdown for candidate tables against query segments. Segments are: **(S1)** `high_schooler.name`, **(S2)** `high_schooler.student_id`, **(S3)** `friendship.student_id`, **(S4)** `friendship.friend_id`.

Table Name	(S1) ...name	(S2) ...student_id	(S3) ...student_id	(S4) ...friend_id
`network_1.highschooler`	0.5313	**0.5861**	0.5490	0.5344
`network_1.friend`	0.5297	**0.6121**	0.6065	0.5844
`network_2.personfriend`	0.5291	0.5377	0.5500	0.5597

Analysis of Greedy Selection The failure is a direct result of CRUSH's greedy, segment-coverage algorithm, which does not re-evaluate aggregate relevance but selects tables one-by-one to "check off" segments.

- **Iteration 1: Selecting the First Table**
  The algorithm looks for the *single best (table, segment) pair* across all candidate tables and all 4 segments.
  - `network_1.highschooler`'s best score is 0.5861 (on S2).
  - `network_1.friend`'s best score is **0.6121** (on S2).
  - `network_2.personfriend`'s best score is 0.5597 (on S4).

  Decision 1: The highest score overall is 0.6121. The table `network_1.friend` is selected, and segment S2 (`high_schooler.student_id`) is marked as "covered".
- **Iteration 2: Selecting the Second Table**
  The algorithm now seeks to cover one of the 3 *remaining* segments (S1, S3, S4). It ignores segment S2 and any table already selected (`network_1.friend`). It compares the best scores from *unselected* tables on *uncovered* segments:
  - For `network_1.highschooler` (Gold): Its best remaining score is 0.5490 (on S3).
  - For `network_2.personfriend` (Wrong): Its best remaining score is **0.5597** (on S4).

  Decision 2: Because 0.5597 > 0.5490, CRUSH greedily selects the incorrect table `network_2.personfriend` to cover segment S4.

Conclusion The baseline's #2 table, `network_1.highschooler`, was dropped. Its highest relevance score (0.5861 on S2) was "used up" by the first table selected, which had an even higher score (0.6121) on that same segment. CRUSH's logic then forced it to pick `network_2.personfriend` because its score on an *uncovered* segment was marginally higher than `network_1.highschooler`'s. This case study exemplifies the risk of a purely coverage-based heuristic, which, as noted in Section 4, can be sensitive to noise and lacks the robust, holistic scoring of our iterative, join-aware approach.

E Absolute Performance Scores

Table 7 presents the absolute retrieval performance scores (Recall and Complete Recall) for all methods, corresponding to the relative delta values shown in Table 1 in the main paper. For each benchmark and metric, the highest value is in **bold** and the second-highest is underlined.

Table 7: Absolute retrieval performance (R/CR %) on multi-table queries. K is the number of tables retrieved. All re-ranking methods (JAR, CRUSH, Ours) operate on the same top-20 Contriever candidate set. † indicates prohibitive runtime or timeout. **Highest** and Second-Highest scores are marked per column within each benchmark.

Benchmark	Method	K=2		K=3		K=5		K=10		Time (seconds)
		Recall (R)	CR	Recall (R)	CR	Recall (R)	CR	Recall (R)	CR	
SPIDER	Contriever	81.3	59.9	93.8	85.6	98.9	97.6	**99.7**	**99.3**	15
Num. of DBs: 20 Num. of Tables: 81 Avg. Table Width: 5.4 Num. of Queries: 459	JAR (MIP)	85.4	67.1	**96.4**	**91.3**	**99.4**	**98.7**	†	†	+ (325 / 290 / 6360 / †)
	JAR$_{\text{iterative}}$ (Ours)	**85.5**	**68.0**	95.6	90.0	98.0	95.6	98.3	96.5	+ ~20
	CRUSH$_{\text{Contriever}}$	63.7	34.6	72.4	47.5	77.4	55.6	77.8	56.0	+ ~38
BIRD	Contriever	63.5	34.8	76.2	56.2	85.2	71.3	95.1	89.9	23
Num. of DBs: 11 Num. of Tables: 75 Avg. Table Width: 10.6 Num. of Queries: 1172	JAR (MIP)	**74.4**	**51.7**	**85.6**	**71.9**	90.4	81.7	†	†	+ (8501 / 13679 / 40756 / †)
	JAR$_{\text{iterative}}$ (Ours)	74.2	51.5	84.7	70.5	**91.0**	**82.1**	**96.2**	**92.6**	+ ~101
	CRUSH$_{\text{Contriever}}$	50.7	22.0	62.7	38.9	72.8	54.4	74.5	57.1	+ ~116
FIBEN	Contriever	22.8	0.7	26.1	1.1	32.0	1.8	41.4	5.4	13
Num. of DBs: 1 Num. of Tables: 152 Avg. Table Width: 2.5 Num. of Queries: 279	JAR (MIP)	**30.7**	**4.3**	**38.3**	**7.2**	48.4	10.0	58.6	14.0	+ (45 / 55 / 2206 / 11340)
	JAR$_{\text{iterative}}$ (Ours)	29.2	**4.3**	36.8	6.8	**51.8**	**10.8**	**62.6**	**14.7**	+ ~10
BEAVER-DW	Contriever	**29.3**	1.7	37.5	9.2	48.5	17.5	63.5	30.8	13
Num. of DBs: 1 Num. of Tables: 97 Avg. Table Width: 15.8 Num. of Queries: 120	JAR (MIP)	8.2	0.0	24.2	8.3	†	†	†	†	+ (18659 / 34756 / †/ †)
	JAR$_{\text{iterative}}$ (Ours)	29.0	**2.5**	**41.7**	**12.5**	**52.6**	**24.2**	**65.0**	**33.3**	+ ~82
BEAVER-NW	Contriever	23.6	1.2	30.2	1.2	38.5	2.3	48.4	9.3	19
Num. of DBs: 5 Num. of Tables: 366 Avg. Table Width: 7.4 Num. of Queries: 86	JAR (MIP)	**33.9**	1.2	**39.6**	4.7	49.2	0.0	48.0	0.0	+ (210 / 572 / 7297 / 10449)
	JAR$_{\text{iterative}}$ (Ours)	26.6	1.2	36.7	**5.8**	**49.6**	**10.5**	**54.5**	**12.8**	+ ~30