Auto-BI: Automatically Build BI-Models Leveraging Local Join Prediction and Global Schema Graph

Yiming Lin*
University of California, Irvine
yiminl18@uci.edu

Yeye He
Microsoft Research
yeyehe@microsoft.com

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

ABSTRACT

Business Intelligence (BI) is crucial in modern enterprises and billion-dollar business. Traditionally, technical experts like database administrators would manually prepare BI-models (e.g., in star or snowflake schemas) that join tables in data warehouses, before less-technical business users can run analytics using end-user dashboarding tools. However, the popularity of self-service BI (e.g., Tableau and Power-BI) in recent years creates a strong demand for less technical end-users to build BI-models themselves.

We develop an AUTO-BI system that can accurately predict BI models given a set of input tables, using a principled graph-based optimization problem we propose called *k-Min-Cost-Arborescence* (k-MCA), which holistically considers both local join prediction and global schema-graph structures, leveraging a graph-theoretical structure called *arborescence*. While we prove k-MCA is intractable and inapproximate in general, we develop novel algorithms that can solve k-MCA optimally, which is shown to be efficient in practice with sub-second latency and can scale to the largest BI-models we encounter (with close to 100 tables).

AUTO-BI is rigorously evaluated on a unique dataset with over 100K real BI models we harvested, as well as on 4 popular TPC benchmarks. It is shown to be both efficient and accurate, achieving over 0.9 F1-score on both real and synthetic benchmarks.

1 INTRODUCTION

Business Intelligence (BI) is increasingly important in modern enterprises for data-driven decision making, and has grown into a multi-billion dollar business [24]. In traditional BI settings, database administrators (DBAs) typically need to manually prepare BI-models (table schemas and join relationships) in data warehouses, so that less-technical business users can perform ad-hoc analysis using tools like dashboards [15].

In recent years, in a growing trend called "*self-service BI*" [25] that is popularized by vendors like Tableau [10] and Power-BI [8], less-technical business users are increasingly expected to set up

and perform BI analysis themselves, without relying on DBAs or central IT. The goal is to democratize BI, so that business users can make agile data-driven decisions themselves, without depending on technical users.

Building BI models: still a pain point. At a high level, there are two main steps in BI project: (1) building BI models, and (2) performing ad-hoc analysis by querying against BI models. While querying BI-models was made simple by vendors like Tableau and Power-BI (through intuitive user interfaces and dashboards) [27, 38], the first step of building "BI-models", a prerequisite for ad-hoc analysis, remains a key pain point for non-technical users.

In the context of self-service BI tools like Tableau and Power-BI, "BI-modeling" refers to the process of preparing raw data and establishing relationships between tables, where a central task is to establish join relationships for a given set of input tables[1]. This closely relates to foreign-key (FK) detection [17, 30, 58] but works specifically in the context of BI, where the resulting schema graphs from the modeling step frequently correspond to structures known as star-schema and snowflake-schema studied in data warehouses [15], like shown in Figure 1.

While self-service BI tools also attempt to improve the usability of the BI-modeling step through better GUI (e.g., allowing users to specify join columns using drag-and-drop) [9, 11], building BI models remains a key pain point. This is because when faced with a large number of tables, even experienced technical users like DBAs can find the task of identifying all possible join relationships challenging and time-consuming. For less-technical enterprise users who are not familiar with concepts like fact/dimension tables, building BI models from scratch can be a daunting challenge.

Foreign-key detection: not yet sufficient. It would clearly be useful, if the join relationships in BI models can be automatically predicted on given input tables (without requiring users to specify them manually). Since joins in BI models are often primary-key (PK) foreign-key (FK) joins, existing FK detection algorithms [17, 30, 58] would seem to apply.

To study this systematically, we harvested over 100K real BI models built using self-service BI tools, from public sources like GitHub and search engines. For each BI model file, we programmatically extract the input tables used in the model, as well as the ground-truth join relationships specified by users, thus creating a real BI benchmark for large-scale evaluation for the first time (prior work mostly use synthetic benchmarks for evaluation instead).

Our large-scale evaluation using these real BI datasets suggests that existing FK-detection algorithms are still insufficient for the

---

---

[1]We note that more advanced BI modeling can involve additional steps such as schema redesign and performance optimization, which is usually out of scope for non-technical users in self-service BI tools, and thus not considered in this work.
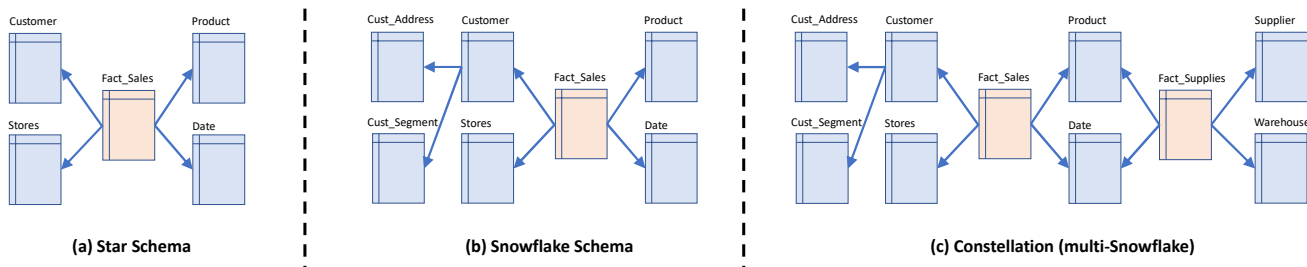
Figure 1: Types of common BI schemas. (a) Star schema, which can be seen as a simplified version of Snowflake. (b) Snowflake schema, where dimension tables like "Customers" can connect to additional dimension tables. (c) Constellation schema, which can be seen as multiple Snowflake schemas.

task, as they frequently produce incorrect results (with ~0.6 precision for all methods we tested). This is because real BI data in the wild tend to be considerably more challenging (e.g., column-headers can often be cryptic with generic names like "name" or "id", and column-values can often overlap accidentally). Crucially, most existing approaches are "local" in nature, that consider two tables at a time to make greedy local decisions, without a principled global optimization, thus unable to produce accurate predictions on challenging real BI test cases (Section 2).

Auto-BI: global optimization using schema graphs. A key insight we develop in Auto-BI, is that because we know users are finding join relationships specifically for building BI-models, and we know the typical structure that schema-graphs in BI-models should generally follow (e.g., variants of snowflake), this gives us a unique opportunity to leverage the *global graph structure* of the resulting schema-graph, to predict joins much more accurately.

We thus formulate the Auto-BI problem as a global optimization problem that holistically considers both the *local decision* of pair-wise joinability, as well as the *global decision* of graph structures.

Specifically, we show that the snowflake schema popular in BI-modeling corresponds to a graph-theoretical concept called *Arborescence* [47]. Leveraging this structure, we formulate a new graph problem called *k-MCA (k-Minimum-Cost Arborescence)*, which finds the most probable *k*-snowflakes, by considering both local join-ability and global graph structures, all using a precise probabilistic interpretation based on calibrated probabilities.

We prove that the new *k*-MCA problem and its variants are in general intractable (NP-hard) and inapproximable (EXP-APX-complete). Nevertheless, we develop novel algorithms based on branch-and-bound principles, which are surprisingly efficient on real test cases and can scale to the largest BI models we encounter, while still being provably optimal. Our extensive evaluations using real BI models and synthetic TPC benchmarks suggest that the proposed Auto-BI is substantially more accurate when compared to existing solutions in the literature.

Contributions. We make the following contributions:

- We are the first to harvest and leverage over 100K real BI models in the wild, for the problem of predicting BI models. This enables a data-driven algorithm design and makes it possible to rigorously evaluate different algorithms on real BI data.
- We are the first to exploit the snowflake-like schema structure in BI settings for our predictions, by mapping snowflakes to a less-known graph-theoretical concept called arboresence. We formulate a set of novel graph-based optimization problems that we call *k*-MCA, that have precise probabilistic foundations.
- We study the theoretical hardness of *k*-MCA variants, and propose efficient algorithms that are provably optimal. Extensive

evaluations show that our algorithms are both effective (with over 0.9 F1 scores), and efficient (scales to the largest BI models we encounter and runs in a few seconds).

2 RELATED WORKS

BI and dashboarding tools. There are a wide variety of BI and dashboarding tools that aim to help users perform ad-hoc data analysis, with Tableau [10] and Power-BI [8] being the leading vendors [24]. These tools use visual drag-and-drop interfaces [38] (without requiring users to write SQL), and are particularly popular among non-technical users.

Foreign key detection. Foreign key (FK) detection is an important problem in database settings, with many influential methods developed in the literature [17, 30, 48, 58].

MC-FK [58] is a pioneering effort to detect FK using an EMD-based randomness metric for distribution similarity between two columns, which is more reliable to predict true relationships when unrelated key-columns that can frequently have overlapping ranges.

Fast-FK [17] develops an efficient method that selects FKs with the best pre-defined scores until all input tables are connected.

HoPF [30] improves upon prior work by considering not only FK-scores but also PK-scores, which are combined using a prede-fined scoring function, making this also a global method in spirit. The algorithm enumerates all PK/FK combinations and returns the combination that has the highest total score.

ML-FK [48] proposes an ML-based classifier to predict FK, trained on known FKs. As we will show analytically and experimentally, without principled global optimizations, ML classification alone is still local in nature and not sufficient to achieve high accuracy.

While FK detection methods are clearly related to our problem (we experimentally compare them with these methods), there are also a few key distinctions that we would like to highlight.

First, while FK detection targets general database settings, we focus on predicting joins in BI models, which gives us a unique opportunity to exploit the likely graph structure (e.g., snowflake) to make more accurate predictions, which is a unique direction not considered in prior work.

Second, unlike FK-detection methods that typically consider the canonical PK/FK (1:N) joins, in Auto-BI the types of joins we consider are more general, because real BI models in the wild frequently employ 1:1 joins as well as joins that are not perfectly 1:N, making the prediction problem more challenging.

Lastly, with the exception of [48], most prior FK detection methods primarily rely on hand-crafted scoring functions to make local join predictions (pairwise between two tables). In comparison, we leverage the BI models harvested to first predict local-joinability in a data-driven way, which is then combined into a principled global

optimization for optimal global decisions (at the graph-level for all tables). This also makes our technique different from prior work.

Detect inclusion dependency. Inclusion dependency (IND) is closely related to FK, and there is an influential body of work focusing on *efficiently* enumerating inclusion dependency (IND) in large databases [12, 33, 34, 39, 43, 50, 51, 53]. The focus on efficiency and scalability of this line of work makes them orthogonal to typical FK-detection methods [17, 30, 58], where the focus is on accurately predicting meaningful FKs (from a large collection of IND candidates). And like prior FK-detection methods that employ efficient IND detection [30, 58], we also use IND-detection as a pre-processing step to enumerate possible FK candidates efficiently.

Complex non-equi joins. Beyond equi-joins, techniques to automatically detect and handle complex join relationships have also been studied, e.g., transformation-based join [42, 55, 60], fuzzy-join [16, 37, 52], search-join [36], semantic lookup-join [29], etc., which is an interesting area of future work in the context of BI.

3 PROBLEM STATEMENT

In this section, we first describe preliminaries and the real BI models we harvest, before introducing the Auto-BI problem.

3.1 Preliminary: BI models

Business Intelligence is closely related to topics like data warehousing and decision support system, and has been extensively studied. We provide a brief preliminary here and refer readers to surveys like [15, 32, 40] for more information.

Fact and dimension tables. In data warehousing and BI terminologies, tables involved in BI modeling can be categorized as two types: *fact tables* and *dimension tables* [32]. A *fact table* contains key metrics and measurements of business processes that one intends to analyze (e.g., the revenue of sales transactions). In addition, a fact table contains *foreign keys* that can reference multiple dimension tables, where each *dimension table* contains detailed information associated with the measurements from a unique facet (e.g., a "Product" dimension table contains details of products sold, whereas a "Date" dimension table has detailed day/month/year info of transactions, etc.). Figure 1 shows a few examples of the BI schemas, with fact and dimension tables in different colors.

Such a separation of fact/dimension tables has many benefits, such as storage/query efficiency, ease of maintenance, etc. [15, 32, 40]. The fact/dimension design is a de-facto standard in BI modeling.

Star, snowflake, and constellation schemas. In BI modeling, fact/dimension tables are frequently organized into what is known as *star/snowflake/constellation schemas* [32], like shown in Figure 1.

Star-schema refers to the cases where there is one fact table, whose foreign-key columns refer to primary-keys from one or more (non-hierarchical) dimension tables, as illustrated by Figure 1(a).

Snowflake-schema generalizes the star-schema, with dimension tables referring to each other in a hierarchical manner. For example, in Figure 1(b), the "Customer" dimension refers to a coarser-grained dimension "Cust-Segment". Similarly an "Address" dimension can refer to a coarser-grained "City", which in turn refers to "Country".

While there is only one fact table in star and snowflake schemas, *constellation-schema* generalizes to the cases with multiple fact tables, as shown in Figure 1(c).

We note that these three types of schemas are extensively studied in the literature [15, 32, 40] and widely adopted in practice.

3.2 Harvest Real BI Models

In order to understand real BI models created in the wild, and systematically evaluate the effectiveness of Auto-BI, we harvest over 100K real BI models from public sources that are created using a popular tool Power BI [8], whose model files have a suffix ".pbix".

We crawl these ".pbix" model files from two sources. The first is GitHub, where developers and data analysts upload their Power BI models for sharing and versioning. We crawl a total of 27K such model files from GitHub. As a second source, we go through the URLs crawled by a commercial search engine that can lead to ".pbix" model files. Most of these URLs are not crawled by the search engine so we crawl ourselves and obtain 86K model files.

Combining data from the two sources gives us a large collection of 100K+ real BI models, which cover diverse BI use cases, including financial reporting, inventory management, among many others.

These real BI models become a valuable dataset for rigorous evaluation – specifically, from each ".pbix" model file, we programmatically extract all tables used in the model, as well as the ground-truth BI model (join relationships) manually specified by users. This enables us to thoroughly evaluate different algorithms using real BI models in the wild, which turn out to be more challenging than synthetic TPC benchmarks used in prior work that tend to be clean and simple.

3.3 Problem Statement: Auto-BI

We now define the high-level Auto-BI problem as follows.

DEFINITION 1. [Auto-BI]. Given a set of input tables $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ used for BI modeling, where each table T_i consists of a list of columns $T_i = (c_{i1}, c_{i2}, \ldots, c_{im})$. Predict a desired BI model $M(\mathbf{T})$ that consists of a set of joins $M(\mathbf{T}) = \{J_{ij}, J_{pq}, \ldots\}$, where each join J_{ij} is a join between T_i and T_j, in the form of $J_{ij} = ((c_{ik}, c_{il}, \ldots), (c_{jr}, c_{js}, \ldots))$.

Note that the output $M(\mathbf{T})$ is restricted to equi-joins for now, which can be single-column joins, or multi-column joins in the form of $J_{ij} = ((c_{ik}, c_{il}, \ldots), (c_{jr}, c_{js}, \ldots))$. Also note that we only state the high-level problem so far, and will defer the exact graph-based formulation to Section 4.3.

In addition to Definition 1, a more general version of the problem goes beyond equi-joins to detect and handle more complex forms of joins, such as transformation-joins [60] and fuzzy-joins [37], which we will leave as future work.

4 AUTO-BI

We will start with an overview of our Auto-BI architecture, before discussing how we predict joins holistically on graphs.

4.1 Architecture Overview

Figure 2 shows the overall architecture of our Auto-BI system. It consists of an offline component and an online component, as depicted by the two boxes shown in the figure.

In the offline step, using real BI models we harvested and their ground-truth BI join models, we perform offline training to learn
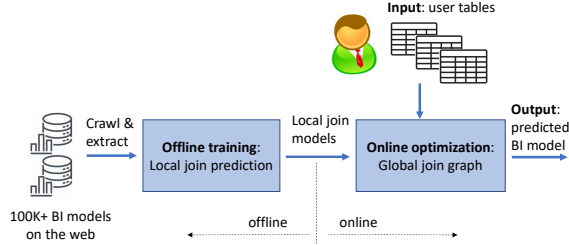
Figure 2: Architecture overview of Auto-BI

what we call "local join models" that can predict, for a given pair of table columns, whether the two are likely joinable. We use the term "*local*" here, because the predictions are pair-wise between two columns, which is in contrast to the "*global*" decision at the entire graph level (the focus of this work).

Since the problem of predicting local joinability using data has been studied in other contexts (e.g., [48, 56]), we do not regard this as our key contribution in Auto-BI, so we will only briefly highlight the important optimizations we make here (e.g., label transitivity, and splitting 1:1 and N:1 joins) in Section 4.2.

The online step is the key of Auto-BI. Here, given a set of tables (modeled as vertices in a graph), we leverage the local-join prediction models trained offline to "score" the joinability of each pair of columns/tables, using calibrated probabilities (which are then modeled as edges in the graph). We formulate Auto-BI on the resulting graph as a novel graph problem k-MCA, which finds the most probable sub-graph that maximizes the joint probability of all edges, subject to graph-structure constraints. We prove the hardness of the problem and develop efficient algorithms. We will describe this online part in Section 4.3.

4.2 Join Prediction: Train Local Classifier

In this step, given two lists of columns $C_i \subseteq T$, $C_j \subseteq T'$, we need to predict the probability that (C_i, C_j) is joinable. (Note that C_i, C_j can be single-columns, but are in general lists of columns for multi-column joins). This problem of predicting local joins has been studied in other contexts [48, 56]. We do not regard this as a new contribution, and will only briefly describe the overall process.

The prediction task. At a high-level, given any two candidate columns (C_i, C_j), our task is to predict the corresponding "join-ability" label, denoted by L_{ij}, where $L_{ij} = 1$ if (C_i, C_j) joins, and $L_{ij} = 0$ otherwise. Since we harvested large amounts of real BI models, each of which contains both data tables and ground-truth joins (programmed by human users), we can use this rich collection of data to produce training data of the form $\{(C_i, C_j), L_{ij}\}$, where L_{ij} corresponds to the actual joined vs. not-joined ground-truth between the two column, which can be programmatically extracted from the BI models we harvest.

This naturally leads to a supervised ML formulation, where we featurize (C_i, C_j) both at the schema-level (e.g., column-header similarity using standard string distance functions such as Jaccard and Edit, as well as pre-trained embedding-based similarity such as SentenceBERT [5]), and at the content-level (e.g., column-value overlap based on Jaccard and Containment), for a total of 21 features. In the interest of space, we leave detailed descriptions of these features, as well as two unique optimizations we develop in this work: (1) separate N:1/1:1 joins, and (2) apply label transitivity, to Appendix A and Appendix B.

Calibrate classifier scores into probabilities. After we train the feature-based model using data extracted from real BI models, given a new pair of columns (C_i, C_j), we can use the model to produce classifier-scores and predict the joinability of (C_i, C_j). However, the scores so produced are still heuristic in nature – e.g., a 0.5 classifier score does not necessarily corresponds to a true join-probability of 0.5, or a 50% chance of the join being correct.

In order to make a principled global decision at the graph level, we "calibrate" the classifier-scores into true probabilities, using calibration techniques from the ML literature [41]. For any column pair (C_i, C_j), the calibration step produces $P(C_i, C_j)$ that corresponds to the probability of the pair being joinable – e.g., $P(C_i, C_j) = 0.5$ really means that a join prediction between the two columns has 50% chance of being correct. As we will see, this gives a precise probabilistic interpretation, which is important when we reason about the most probable global join graph.

4.3 Auto-BI: Exploit Global Join Graph

We are now ready to solve the Auto-BI problem. We first introduce how we represent tables and candidate joins on a global join graph, before describing our graph formulation.

4.3.1 *Representing relationships in a global graph.*

Given a set of tables **T** and possible joins, we can construct a directed graph $G = (V, E)$ as follows. We represent each input table $T \in \mathbf{T}$ as a vertex $v(T) \in V$, and each possible join candidate between columns (C_i, C_j) as a weighted edge $e_{ij} \in E$, where the edge weight $w(e_{ij})$ is simply the calibrated join probability $P(C_i, C_j)$, produced by our local classifier (Section 4.2), which scores every pair of candidate columns whose containment is over a threshold. We follow the convention to use a directed edge e_{ij} to represent N:1 joins, which point from N-side (FK) columns C_i, to the 1-side (PK) columns C_j. We represent 1:1 joins as bi-directional edges.

EXAMPLE 1. Figure 3 shows a graph representation of the tables in Figure 1(b), where each vertex corresponds to a table. We mark the ground-truth joins in Figure 1(b) as solid green edges in Figure 3, while other candidate joins not in ground-truth as dotted red edges.

For instance, the dotted edge ($e_5 : 0.8$) represents an candidate join between the column "Customer-ID" (in table "Cust-Details"), and column "Customer-Segment-ID" (in table "Cust-Segments"). Note that the column pair ("Customer-ID", "Customer-Segment-ID") should not join because they refer to two semantically different types of IDs, which however may appear like a plausible join to Local-Classifier (because of high name-similarity and value-overlap), which leads to a high Local-Classifier score (0.8). A greedy method that focuses on promising edges locally can incorrectly predict this false-positive join, which is a mistake that global methods like Auto-BI can prevent.

4.3.2 *Precision Mode (k-MCA-CC).*

We are now introducing our formulation using the graph. At a high level, we operate in two steps: (1) a "precision-mode" stage where we focus on finding the salient snowflake-like structures that are the "backbones" of the underlying schema graph (which ensures high precision thanks to the graph-structure constraints it imposes); and (2) a "recall-mode" stage that complements the precision-mode,
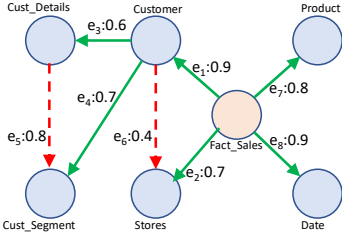
Figure 3: Solve 1-MCA on the graph representation of the tables in Figure 1(b), with join candidates as edges.

	known results	new results	
Problem	**1-MCA**	**k-MCA**	**k-MCA-CC**
Description	find the most probable 1-snowflake	find the most probable k-snowflakes	most probable k-snowflakes w/ constraints
Hardness	Poly-time solvable [18]	Poly-time solvable (Theorem 2)	EXP-APX-hard (Theorem 3)
Algorithm	Chu-Liu/Edmond's algorithm [18]	ours (Algorithm 2)	ours (Algorithm 3)

Table 1: Summary of results for MCA problem variants.

by finding additional joins beyond typical snowflakes. We will introduce both in turn below.

We first introduce our precision-mode formulation, referred to as k-MCA-CC. For ease of exposition, we will illustrate the thought process of arriving at k-MCA-CC in 3 steps: (1) We will start with a simplistic assumption that the schema graph has exactly one snowflake, which we model with a graph-theoretical problem called 1-MCA. (2) We then generalize this to arbitrary numbers of snowflakes, using a formulation we call k-MCA. (3) Finally, we generalize k-MCA to include additional graph-structure constraints to arrive at our final formulation, which we call k-MCA-CC. We summarize our key results in this section in Table 1 for reference.

(1) Exactly one snowflake: 1-MCA. To start with, we discuss the simplistic case where we know there is exactly one snowflake in the underlying graph (e.g., Figure 1(b)). Note that this is only for ease of illustration, and not an actual assumption that we rely on (we will show how this can be relaxed next).

Given a graph $G = (V, E)$ where candidate joins are marked as edges like in Figure 3. Since we know there is one snowflake schema, intuitively we want to select edges (joins) $J \subseteq E$, such that:

(a) The graph induced by J, $G' = (V, J)$, is a snowflake that connects all vertices in V;

(b) If more than one such snowflake-structure exists, find the most probable snowflake, based on the joint-probability of all joins selected in J, $P(J) = \prod_{e_{ij} \in J} P(C_i, C_j)$.

These two considerations can be written as the following optimization problem, which we refer to as 1-Most-Probable-Snowflake (1-MPS):

$$(\text{1-MPS}) \qquad \max_{J \subseteq E} \prod_{e_{ij} \in J} P(C_i, C_j) \qquad (1)$$

$$\text{s.t. } G' = (V, J) \text{ is a snowflake} \qquad (2)$$

Note that the constraint in Equation (2) of 1-MPS corresponds to requirement (a), while the objective function in Equation (1) maps to requirement (b).

Since snowflake is used informally in BI, to make it more formal we map it to a structure in graph theory called *arboresence* [22].

DEFINITION 2. [Arborescence]. A directed graph $G = (V, E)$ is called an *arborescence*, if there is a unique vertex $r \in V$ known as the root, such that there is exactly one directed path from r to every other $v \in V, v \neq r$. Equivalently, a directed graph G is an arborescence if all its vertices have in-degree of 1, except a unique root vertex $r \in V$ that has in-degree of 0 [22].
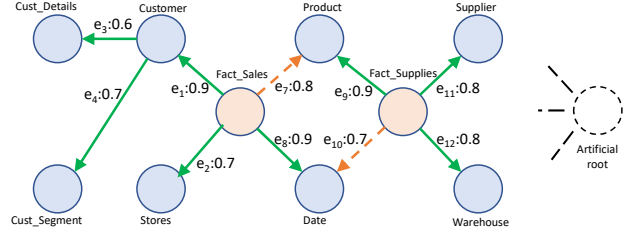

Figure 4: Solve k-MCA on the graph representation of the tables in Figure 1(c), with join candidates as edges.

Intuitively, arborescence is a directed rooted tree where all its vertices are pointing away from the root. We use an example to show the relationship between snowflakes and arborescence.

EXAMPLE 2. Consider the sub-graph G induced by all green edges in Figure 3, where each green edge would correspond to a ground-truth join in the snowflake schema of Figure 1(b). This sub-graph G is an arborescence, because if we take the vertex marked as "Fact_Sales" as the root r, then there is exactly one directed path from r to every other vertex in G. Equivalently, we can check that this root r has in-degree of 0, while all other vertices in G have in-degree of exactly 1, which also ensures that G is an arborescence.

Recall that we know there is exactly one snowflake in 1-MPS, which is equivalent to saying that the sub-graph induced by $J \subseteq E$ is an arborescence. We rewrite 1-MPS into 1-MPA (1-Most-Probable-Arborescence), defined as follows.

$$(\text{1-MPA}) \qquad \max_{J \subseteq E} \prod_{e_{ij} \in J} P(C_i, C_j) \qquad (3)$$

$$\text{s.t. } G' = (V, J) \text{ is an arborescence} \qquad (4)$$

EXAMPLE 3. We revisit the graph in Figure 3. Using the formulation of 1-MPA, the best arborescence with the highest joint probability (correspondingly, the most probable snowflake), is the set of solid edges marked in green, denoted by $J^* = \{e_1, e_2, e_3, e_4, e_7, e_8\}$, whose joint probability is $0.9 * 0.7 * 0.6 * 0.7 * 0.8 * 0.9$. It can be verified that J^* is the most probable arborescence among all arborescences (that span the entire vertex set), which corresponds to the ground-truth snowflake schema shown in Figure 1(b).

Consider an alternative arborescence, such as $J' = \{e_1, e_3, e_5, e_6, e_7, e_8\}$, which removes e_2, e_4 from J^* and adds e_5, e_6 (marked in dotted red lines). Observe that this new J' also forms a 1-arborescence based on Definition 2. However, J' has a lower joint probability than J^* (because e_5, e_6 in J' has a joint probability of $0.8 * 0.4 = 0.32$, while e_2, e_4 in J^* has a joint probability of $0.7 * 0.7 = 0.49$), which will thus not be selected based on the 1-MPA formulation above.

We note that in 1-MPA, J^* is selected based on a global decision of optimality, which avoids locally-greedy decisions – e.g., the incorrect e_5 will not be selected despite its high score like we mentioned in Example 1.

While we can now use arborescence to formalize 1-MPA, it still has a cross-product that is not amenable to optimization. We thus perform the following transformation: instead of assigning join-probability $P(C_i, C_j)$ as the edge weight for each edge e_{ij}, we perform a logarithmic transformation and set the edge weight of each e_{ij} as:

$$w(e_{ij}) = -\log(P(C_i, C_j)) \qquad (5)$$

Using the new transformed $w(e_{ij})$, for each instance of the 1-MPA problem, we construct a new minimization problem below that we

Algorithm 1: Construct graph with edge-weights

---

    input   : all input tables **T** in a BI-model
    output : Graph $G = (V, E)$ that represents **T**
1 $V \leftarrow \{v_T | T \in \mathbf{T}\}$, with v_T representing each $T \in \mathbf{T}$
2 $E \leftarrow \{\}$
3 **foreach** (C_i, C_j) *satisfying Inclusion-Dependency in* **T do**
4     $\big|$ $P(C_i, C_j) \leftarrow$ Local-Classifier(C_i, C_j)
5     $\big|$ $w(e_{ij}) \leftarrow -\log(P(C_i, C_j))$
6     $\big|$ $E \leftarrow E \cup \{e_{ij}\}$, with edge-weight $w(e_{ij})$
7 **return** $G(V, E)$

---

term 1-MCA, which uses summation in the objective function:

$$\text{(1-MCA)} \qquad \min_{J \subseteq E} \sum_{e_{ij} \in J} w(e_{ij}) \qquad\qquad (6)$$

$$\text{s.t. } G' = (V, J) \text{ is arboresence} \qquad (7)$$

It can be shown that solving 1-MPA is equivalent to solving the corresponding 1-MCA.

LEMMA 1. *A solution $J^* \subseteq E$ is an optimal solution to 1-MPA, if and only if J^* is an optimal solution to the corresponding 1-MCA.*

PROOF. First, observe that 1-MPA and 1-MCA have identical feasible regions, as they are subject to the same constraints.

Next, we show that an optimal solution J^* that maximizes the objective function $\prod_{e_{ij} \in J^*} P(C_i, C_j)$ in Equation (3) of 1-MPA, will simultaneously minimize the objective function $\sum_{e_{ij} \in J} w(e_{ij})$ in Equation (6) of 1-MCA. This is the case because the objective function of 1-MCA is: $\sum_{e_{ij} \in J^*} w(e_{ij}) = \sum_{e_{ij} \in J^*} -\log(P(C_i, C_j))$ $= -\log(\prod_{e_{ij} \in J^*} P(C_i, C_j))$, where the term inside $-\log()$ is exactly the objective function of 1-MPA, thus ensuring that 1-MCA is minimized if and only if 1-MPA is maximized, and completes the proof. $\qquad\square$

The reason we construct 1-MCA for each instance of 1-MPA, is that 1-MCA relates to a known problem in graph theory called "*Minimum-Cost-Arborescence*" (abbreviated as MCA[2]) [20], which finds a spanning arborescence (covering all vertices) in a directed graph G that has the smallest edge-weights. Note that in the simplistic setting where we know there is exactly 1 snowflake (arborescence), our 1-MCA problem directly translates to the MCA problem [20]. Since MCA is known in graph theory with polynomial-time solutions called the Chu–Liu/Edmonds' algorithm [18, 20], our construction allows us to solve 1-MPA efficiently by leveraging the same algorithms.

We use our running example to show the connection between 1-MPA and 1-MCA.

EXAMPLE 4. Continue with Example 3. To solve the 1-MPA problem for the graph in Figure 3, we use the transformation in Equation (5) and construct an instance of 1-MCA on the same graph, where all edge-weights are now $w(e_{ij}) = -\log(P(C_i, C_j))$. It can be verified that $J^* = \{e_1, e_2, e_3, e_4, e_7, e_8\}$ is the minimizer of Equation (6) in 1-MCA, with the smallest objective-value $-(\log(0.9) + \log(0.7) + \log(0.6) + \log(0.7) + \log(0.8) + \log(0.9))$, which can be

---

[2]Although MCA is not a well-known concept, we note that MCA is directly analogous to a better-known problem in graph theory called "*Minimum-Spanning-Tree* (MST)" [31], with the only difference that MST is defined on undirected graphs whereas MCA is on directed graphs.

efficiently solved using the Chu–Liu/Edmonds algorithm. Note that J^* is the same optimal solution to 1-MPA, as we see in Example 3.

Algorithm for 1-MCA. Given a set of input tables **T** (for which a BI-model needs to be built), the complete pseudo-code to construct a graph G for 1-MCA is shown in Algorithm 1. Since we are given tables **T**, in Line 3 we enumerate column-pairs (C_i, C_j) in **T** for which Inclusion-Dependencies (IND) [14] hold approximately, which are possible joins that we should consider (note that efficient IND-enumeration is a standard step [19], so we invoke existing techniques here). In Line 4, we "score" each (C_i, C_j) using our Local-classifier to obtain calibrated probabilities $P(C_i, C_j)$ (Section 4.2), which are transformed in Line 5 to become edge-weights $w(e_{ij})$ (Equation (5)).

Using the resulting graph G constructed from **T**, we can invoke the Chu–Liu/Edmonds' algorithm, which yields the optimal solution J^* to 1-MCA (and thus also 1-MPA).

We summarize our main result for 1-MCA in the first column of Table 1. We note that although we leverage known results from the graph theory here, to the best of our knowledge, we are the first to connect the popular concept of snowflakes in BI, with arborescence in graph theory.

In the following, we will extend 1-MCA to general cases, and develop new results not known in graph theory or databases.

(2) Arbitrary number of snowflakes: k-MCA. So far we use 1-MCA to solve the simplistic case where we know there is exactly 1 snowflake. In general, there can be multiple snowflakes, and we do not know its exact number beforehand (e.g., the ground-truth schema of Figure 1(c) has 2 snowflakes, which is unknown a priori).

In this section, we extend 1-MCA to the more general k-MCA, where the proposed k-MCA formulation will not only find the most probable k snowflakes like in 1-MCA, but also infer the right number of snowflakes k at the same time.

We first extend the concept of arborescence in graph-theory (Definition 2), to the more general k-arboresence below.

DEFINITION 3. [k-arboresence]. A directed graph $G = (V, E)$ is an k-arborescence, if its underlying undirected graph has a total of k joint connected-components, written as $\{G_i = (V_i, E_i) | i \in [k]\}$, such that $\bigcup_{i \in [k]} V_i = V$, $\bigcup_{i \in [k]} E_i = E$. Furthermore, each G_i is an arborescence, for all $i \in [k]$.

Note that when $k = 1$, the notion of 1-arborescence degenerates into arborescence described in Definition 2. We will henceforth use 1-arborescence and arborescence interchangeably when the context is clear.

EXAMPLE 5. Consider the graph in Figure 4, which is the graph representation of the constellation (multi-snowflake) schema in Figure 1(c). The sub-graph with green solid edges (ignoring the dotted edges for now), is a 2-arborescence. It is a 2-arboresence because both of its two connected-components are arboresences (Definition 2), rooted at "Fact_Sales" and "Fact_Supplies", respectively.

Intuitively, we use the k-arboresence structure to force the k underlying snowflakes to emerge, which reveals the most salient "backbones" of the underlying schema. It should be noted that in the precision-mode of AUTO-BI (this section), because we focus exclusively on finding snowflake/arborescence structures to ensure high precision, some desired joins may be missing in the k-arboresence

(e.g., the orange dotted edges in Figure 4), which is something that we will get to in the recall-mode of Auto-BI next (Section 4.3.3).

Given that we want to use k-arborescence to let the k snowflakes emerge, the next question is how to find the right k. Conceptually, we can iterate over all possible k, and pick the "best" k, using a suitable "goodness" function (e.g., revealing the right number of snowflakes).

First, recall that a k-arborescence on graph $G = (V, E)$ has exactly k connected components, and is bound to have exactly $(|V| - k)$ edges (because every vertex except the root has in-degree of 1, per Definition 2).

Given these, we can see that while the "sum of edge-weights" objective function in Equation (6) for 1-MCA is a suitable "goodness" function to compare two 1-arboresences (both with $k = 1$), it is no longer suitable to compare a k_1-arborescence and a k_2-arboresence (with $k_1 \neq k_2$), because the two will have different number of edges, making the comparison using the sum of edge-weights in Equation (6) "unfair". In fact, using Equation (6) and a flexible k is guaranteed to lead to $|V|$ disconnected vertices (a trivial $|V|$-arborescence) as the optimal, because it has no edges and thus no edge-weights.

To make the comparisons "fair" between two k-arboresences with different values of k, and prevent disconnected components from having artificial advantages, we modify the objective function as follows. Since a k-arboresences has exactly k connected components and $(|V| - k)$ edges, we imagine that there are $(k - 1)$ "virtual-edges", each with a parameterized edge-weight p, that connect the k connected components into one, such that a k-arboresences always has the same number of edges as 1-arboresences, regardless of k (because $(|V| - k) + (k - 1) = (|V| - 1)$). Accounting for the $(k - 1)$ new virtual edges in Equation (6) leads to Equation (8) below, which is new objective function we use for k-MCA:

$$(k\text{-MCA}) \quad \min_{J \subseteq E, \ k \leq |V|} \sum_{e_{ij} \in J} w(e_{ij}) + (k - 1) \cdot p \quad (8)$$

$$\text{s.t. } G' = (V, J) \text{ is a } k\text{-arboresence} \quad (9)$$

The parameter p we introduce effectively controls the number of snowflakes (e.g., a larger p would "penalize" having more disconnected snowflakes). The question is how to set p appropriately. Recall that we use calibrated join-probability $P(C_i, C_j)$ to produce edge weight $w(e_{ij}) = -\log(P(C_i, C_j))$, then naturally the virtual edges should be imagined as edges with a join-probability of exactly 0.5, which means a 50% chance of being joinable and thus a coin-toss that is ok to either include or exclude (which makes them "virtual").

As another way to look at this, consider when we drop a regular edge e_{ij} from a k-arborescence to create a $(k + 1)$-arborescence – the objective function of the latter in Equation (8) will have the edge-weight $w(e_{ij})$ removed, but incur a penalty cost of p from the additional virtual-edge we introduce. When the penalty-weight p on the virtual-edge corresponds to a join-probability of 0.5, it would discourage e_{ij} from being dropped if its $P(C_i, C_j)$ is over 0.5 (50% chance of being joinable), and encourage e_{ij} to be dropped if its $P(C_i, C_j)$ is under 0.5, which is exactly the right thing to do.

We, therefore, use $p = -\log(0.5)$ as our natural choice of penalty weight in k-MCA. Our experiments suggest that empirically 0.5 is

---

Algorithm 2: Solve k-MCA for constellation schema

> **input** : Graph $G = (V, E)$
> **output** : optimal k-MCA (k-snowflakes)
> 1 $V' \leftarrow V \cup \{r\}$
> 2 $E' \leftarrow E \cup \{e(r, v) | v \in V\}$, with $w(e(r, v)) = p$
> 3 $J_1^* \leftarrow$ solve 1-MCA on $G' = (V', E')$ with Chu-Liu/Edmonds' algo
> 4 $J_k^* = J_1^* \setminus \{e(r, v) | v \in V\}$
> 5   **return** J_k^*

---

indeed the right choice (Section 5), thanks to the fact that we use true calibrated probabilities (Section 4.2).

Observe that if we know $k = 1$ a priori, our k-MCA degenerates exactly into 1-MCA like before. When k is unknown however, the objective function in Equation (8) would help reveal the best k-snowflakes, as well as the right number of k. We show this intuitively using the example below.

EXAMPLE 6. We revisit Figure 4 where all edges are possible join candidates, and solve k-MCA on this example graph.

First, observe that there is no 1-arborescence in this graph. Let J^* be all the green edges, then the subgraph induced by J^* is a 2-arboresence (rooted at the two fact-tables). Since $k = 2$, the penalty term in this case is $(2 - 1)(-\log(0.5)) = 1$. It can be verified that J^* has the lowest cost in all 2-arboresences.

It is possible to have 3-arboresences here too – for example if we remove e_1 from J^*, then the remaining green-edges in $J' = J^* \setminus \{e_1\}$ induce a 3-arboresence. It can be verified that the cost of J' is higher than that of J^*, because J' removes e_1 from J^* and thus lowers its cost by $w(e_1) = -\log(0.9) = 0.13$, but incurs a higher penalty-cost of $(3 - 1)(-\log(0.5)) = 2$, which makes J' less desirable than J^*.

Algorithm for k-MCA. A naive way to solve k-MCA that builds on top of 1-MCA above, is to enumerate different values of k, and for each k, exhaustively enumerate all k-way partitions of G, then invoke 1-MCA on each resulting graph partition, to find the optimal k-MCA. This approach is straightforward but inefficient (because k can be as large as $|V|$, making it exponential in $|V|$).

We design an algorithm that solves k-MCA, using a graph-based construction that reduces any instance of the new k-MCA problem into *one* instance of 1-MCA (which admits efficient solutions [18]). Specifically, given a graph $G = (V, E)$ on which k-MCA needs to be solved, we introduce a new vertex r that is an "artificial root", and connects r with all $v \in V$ using edges $e(r, v)$, with edge-weight $w(e(r, v)) = p$. This leads to a new constructed graph $G' = (V', E')$ where $V' = V \cup \{r\}$, $E' = E \cup \{e(r, v) | v \in V\}$.

We solve 1-MCA on the new G', and let J_1^* be the optimal 1-MCA. Let $J_k^* = J_1^* \setminus \{e(r, v) | v \in V\}$ be edges in J_1^* not incident with the artificial-root r. We can show that J_k^* is the optimal solution to k-MCA on the original G. The complete steps of this algorithm are shown in Algorithm 2.

THEOREM 2. Algorithm 2 solves k-MCA optimally, in time polynomial to the input size.

PROOF. Given a graph $G = (V, E)$, let J_k^* be the optimal k-MCA of G, with objective function value $c_k(J_k^*)$, where c_k is the objective function defined in Equation (8). Because J_k^* is a k-arborescence, by Definition 3, J_k^* has exactly k disjoint components $\{G_1, G_2, \ldots G_k\}$,

all of which are 1-arboresence, with $R = \{r_1, r_2, \ldots r_k\}$ as their roots.

We show that on our constructed graph $G' = (V', E')$, the solution $J_1^* = J_k^* \cup \{(r, r_i) | r_i \in [k]\}$ must be an optimal solution to the 1-MCA problem on G'.

We show this by contradiction. Suppose J_1^* is not an optimal solution to G', which means that there is another J_1' with a smaller cost as defined in the 1-MCA objective function. Let the objective function in Equation (6) be c_1, we can write this as

$$c_1(J_1') < c_1(J_1^*) \tag{10}$$

In the solution J_1', let R' be the set of vertices incident with artificial root r. Removing from J_1' the set of edges incident with r produces $J_k' = J_1' \setminus \{e(r, r') | r' \in R'\}$. J_k' is an $|R'|$-arboresence, and thus a feasible solution to k-MCA on G. Similarly, removing from J_1^* the set of edges incident with r produces $J_k^* = J_1^* \setminus \{e(r, r') | r' \in R\}$, which is also a feasible solution to k-MCA on G.

Since J_k^* is an $|R|$-arboresence and J_k' is an $|R'|$-arboresence, by the definition of c_k in Equation (8), we know

$$c_k(J_k^*) = \sum_{e \in J_k^*} w(e) + (|R| - 1)p \tag{11}$$

$$c_k(J_k') = \sum_{e \in J_k'} w(e) + (|R'| - 1)p \tag{12}$$

Because we know J_k^* is an optimal solution to k-MCA on G, we know $c_k(J_k^*) \leq c_k(J_k')$. Combining with Equation (11) and (12), we get:

$$\sum_{e \in J_k^*} w(e) + (|R| - 1)p \leq \sum_{e \in J_k'} w(e) + (|R'| - 1)p \tag{13}$$

Adding p to both sides of Equation (13), we get $\sum_{e \in J_k^*} w(e) + (|R|)p \leq \sum_{e \in J_k'} w(e) + (|R'|)p$. Note that the left-hand-side is exactly the objective-value of J_1^* on G', or $c_1(J_1^*)$, while the right-hand-side is the objective-value of J_1' on G'. This gives $c_1(J_1^*) \leq c_1(J_1')$, contradicting with our assumption in Equation (10), thus proving that J_1^* must be an optimal solution to 1-MCA on G'.

Since J_1^* can be solved optimally in Line 3 of Algorithm 2 for 1-MCA in polynomial time (Line 3), and all other steps in Algorithm 2 also take time polynomial in the input size, we conclude that Algorithm 2 can solve k-MCA optimally, in polynomial time. □

EXAMPLE 7. We continue with Example 6. Using Algorithm 2, we would construct a new graph G' by adding an artificial root r (shown on the right of Figure 4), which connects to all existing vertices v_i with an edge $e(r, v_i)$ with the same penalty weight $w(e) = p$. Using Line 4 of Algorithm 2 to solve the 1-MCA on G' produces the optimal solution of J_1^* that consists of all solid green edges, plus two artificial-edges connecting r with the two fact-table, which can be verified is the optimal 1-MCA. Line 4 of Algorithm 2 would then produce J_k^* corresponding to of all green edges (removing the two artificial-edges connecting fact-tables to the artificial-root). It can be verified that J_k^* in this example is the optimal k-MCA, which is also the ground-truth schema in Figure 1(c).

(3) Arbitrary number of snowflakes with constraints: k-MCA-CC. The k-MCA formulation we considered so far can handle arbitrary numbers of snowflakes. This however, can be further improved by adding an additional constraint on the graph that we call "FK-once".[3]

FK-once refers to the property that the same FK column in a fact-table should likely not refer to two different PK columns in two dimension tables (a common property known in the database literature [17, 30]). On a graph, such a structure would correspond to two edges $(C_i, C_j), (C_i, C_m)$, pointing from the same C_i, to two separate C_j and C_m, which usually indicates that one of the joins is incorrect. For example, the same FK column "Customer-ID" in "Sales" table may appear joinable with both (1) the PK column "C-ID" of the "Customers" table, and (2) the PK "Customer-Segment-ID" of the "Customer-Segments" table (because both have high column-header similarity and value-overlap). However, we know that an FK should likely only join one PK (or otherwise we have two redundant dimension tables).

We add the FK-once constraint as a cardinality-constraint (CC) to k-MCA, which leads to k-MCA-CC below:

$$(\text{k-MCA-CC}) \min_{J \subseteq E, \, k \leq |V|} \sum_{e_{ij} \in J} w(e_{ij}) + (k - 1) \cdot p \tag{14}$$

$$\text{s.t. } G' = (V, J) \text{ is } k\text{-arboresence} \tag{15}$$

$$i \neq l, \forall e_{ij} \in J, e_{lm} \in J, j \neq m \tag{16}$$

Note that Equation (16) is the new FK-once constraint, which states that no two edges e_{ij}, e_{lm} in the selected edge-set J should share the same starting column-index, or $(i \neq l)$.[4]

THEOREM 3. k-MCA-CC is NP-hard. Furthermore, it is Exp-APX-complete, making it inapproximable in polynomial time.

We prove Theorem 3 using a reduction from non-metric min-TSP [21]. Proof of this can be found in Appendix D. It is interesting to see that adding one constraint in k-MCA-CC makes it considerably more difficult than k-MCA (which however is important in terms of result quality, as we will show in experiments).

Algorithm for k-MCA-CC. Despite the hardness, we develop a novel algorithm that solves k-MCA-CC optimally, leveraging the branch-and-bound principle [35], and the sparsity of join edges pointing from the same columns.

Algorithm 3 shows the pseudo-code of this recursive procedure. We first invoke Algorithm 2 to solve the unconstrained version k-MCA and obtain J (Line 1). We check J for constraint violations (Line 2) – if there is no violation, we are done and J is the optimal solution to k-MCA-CC. Alternatively, if J has constraint violations, but the cost of the unconstrained version $c(J)$ is already higher than a best solution to k-MCA-CC found so far (Line 4), we know adding the constraints will only degrade solution quality so we return null for this solution space. Otherwise, in Line 7, let $C_s = \{e_{sj}, e_{sk}, \ldots\} \subseteq J$ be one set of conflicting edges in J from the same column (thus violating the FK-once constraint). We partition C_s into $|C_s|$ number of subsets $C_s^1, C_s^2, \ldots, C_s^{|C_s|}$, each with exactly one edge in C_s (Line 8). We then construct $|C_s|$ number of k-MCA-CC problem instances, each with a new graph $G_i = (V_i, E_i)$, where $V_i = V, E_i = E \setminus C_s \cup C_s^i$. We recursively solve k-MCA-CC on each graph G_i to get J_i (Line 11). Let $c(J)$ be the objective function

---

[3]Other possible constraints that can benefit the inference, such as "no-cycles" of join paths in inferred schema graph, is implicit from the definition of arboresence.

[4]Note that the constraint is on column-index and at a column-level – one table can still have multiple different FK columns pointing from the same *table/vertex*, to different PK columns of different *tables/vertices*.

Algorithm 3: Solve k-MCA-CC

> **input** : Graph $G = (V, E)$
> **output** : optimal solution to k-MCA-CC (k-snowflakes)

1 $J \leftarrow$ Solve-k-MCA(G) using Algorithm 2
2 **if** J is a feasible solution to k-MCA-CC(G)) **then**
3   **return** J
4 **else if** cost $c(J)$ is higher than best k-MCA-CC found so far **then**
5   **return** null
6 **else**
7      $C_s \leftarrow$ edges in J with the same source column index s that violates FK-once constraint (Equation 16)
8      $C_s^1, C_s^2, \dots C_s^{|C_s|} \leftarrow$ disjoint subsets of C_s, each with exactly one edge from C_s
9      $E_i = E \setminus C_s \cup C_s^i, \forall i \in [|C_s|]$
10     $G_i = (V, E_i) \ \forall i \in [|C_s|]$
11     $J_i =$ Call Solve-k-MCA-CC(G_i) recursively, $\forall i \in [|C_s|]$
12     $J^* = \text{argmin}_{J_i, i \in [|C_s|]} \ c(J_i)$
13 **return** J^*

in Equation (14), the J_i that minimizes the cost function, $J^* = \text{argmin}_{J_i} c(J_i)$, is the optimal solution to the original k-MCA-cc problem on G (Line 12).

Note that our recursive call (Line 11) partitions and reduces the solution space into disjoint subsets without losing optimal solutions, while the pruning step (Line 4) quickly prunes away unpromising solution spaces leveraging efficient k-MCA algorithms. This two steps can conceptually be seen as a form of branch-and-bound used in the optimization literature [35], but is specifically tailored to our graph problem (leveraging the mutually-exclusive nature of conflicting edges based on the FK-once property).

THEOREM 4. Algorithm 3 solves k-MCA-CC optimally.

A proof of the theorem can be found in Appendix E. Intuitively, this algorithm ensures optimality, because at most one edge in C_s may appear in the optimal solution to k-MCA-CC (otherwise the FK-once constraint is violated), making edges in C_s mutually exclusive. This allows us to partition the conflicting edges in C_s, and recursively construct instances of the problem with smaller feasible regions, without losing the optimal solution in the process.

Given the hardness of k-MCA-CC, we clearly cannot hope Algorithm 3 to solve k-MCA-CC in polynomial time. In practice, however, our branch-and-bound partitioning exploits the sparseness of edges (there are few edges pointing from the same columns), which turns out to be highly efficient. In our experiments (Section 5), we observe the mean and median latency of Algorithm 3 on real-world BI schemas is 0.11 and 0.02 second, respectively, with the max being 11 seconds on a case with 88 data-tables (the largest case we encounter in the 100K+ real BI models harvested). We note that this is encouraging, and analogous to many classical combinatorial problems (e.g., Euclidean TSP and Knapsack) that are intractable in theory but reliably solvable in practice.

4.3.3 *Recall Mode (MaxEdge-CC).*

At this point, we did with the precision-mode of AUTO-BI, which is precision-oriented as we focus on finding the salient k-snowflakes to reveal the "backbone" of the underlying schema. However, not all desired joins are included in k-snowflakes/arborescence. For example, the green edges in Figure 4 correspond to the optimal solution to k-MCA-CC, but we are still missing two joins in the ground truth,

	Average	50-th p%	90-th p%	95-th p%
# of rows per table	1053.4	50.1	1925.4	13981.2
# of columns per table	8.1	4.2	11	17.1
# of tables (nodes) per case	3.2	2	6.9	9.1
# of relationships (edges) per case	3.9	3	8.2	11.6

Table 2: Characteristics of all BI models harvested

	Average	50-th p%	90-th p%	95-th p%
# of rows per table	7730	80	10992	33125
# of columns per table	9	5	20	27
# of tables (nodes) per case	12.9	9	25	32
# of relationships (edges) per case	11.7	8.6	19	29

Table 3: Characteristics of 1000-case REAL benchmark

	TPC-C	TPC-E	TPC-H	TPC-DS
average # of rows per table	288590	7850832	1082655	814889
average # of columns per table	10.2	5.8	7.6	17.7
# of tables (nodes)	9	32	8	24
# of relationships (edges)	10	45	8	107

Table 4: Characteristics of 4 TPC benchmarks

marked as dotted orange edges (these are joins that reference the same dimension-table, from the multiple fact tables).

We, therefore, introduce the "recall-mode" of AUTO-BI, where we "grow" additional edges on top of the "backbone" identified in the precision-mode.

Specifically, given the original graph $G = (V, E)$, let J^* be the optimal solution to k-MCA-CC. Let $R = \{e_{ij} | e_{ij} \in (E \setminus J^*), P(e_{ij}) \geq \tau\}$ be the remaining edges that are promising (meeting a precision threshold τ[5]) but not yet selected by J^*. For our recall-oriented solution of AUTO-BI, we want to select as many as edges $S \subseteq R$, subject to certain graph-structure constraints below. We write this as EMS (Edge-Maximizing Schema) below:

$$(\text{EMS}) \qquad \underset{S \subseteq R}{\text{argmax}} \ |S| \qquad\qquad (17)$$

$$\text{s.t. } i \neq l, \forall e_{ij}, e_{lm} \in S \cup J^*, j \neq m \qquad (18)$$

$$S \cup J^* \text{is cycel-free} \qquad\qquad (19)$$

The new constraint in Equation (19) ensures that there should be no cycles in the resulting graph induced by $S \cup J^*$, as circular joins between columns are uncommon at a schema level.

The EMS problem is NP-hard, and is 1/2-inapproximable using a reduction from Max-Sub-DAG [26]. However, because this step operates on top of the optimal solution J^* to k-MCA-CC, there is typically limited flexibility in selecting from R. We find different solutions have very similar results (Section 5), and we thus solve EMS greedily by picking the most confident edges (based on $P(e_{ij})$), without using more expensive solutions.

This now completes the overall AUTO-BI. To recap, we first solve k-MCA-CC optimally using Algorithm 3 in the precision-mode, to get J^* (the salient k-snowflakes). Then in the recall-mode, we solve EMS using J^* and obtain S^*. The sub-graph induced by $J^* \cup S^*$ is the final solution of AUTO-BI.

5 EXPERIMENTS

We perform large-scale evaluations to test AUTO-BI and related methods in the literature, in both quality and efficiency.

5.1 Evaluation Setup

Benchmarks. We use two benchmarks to evaluate AUTO-BI.

---

[5]By default, we threshold with τ= 0.5 here, since our $P(e_{ij})$ are calibrated probability, and 0.5 is a natural cutoff for joinability.

Table 5 header and data:

		Real (OLAP)				Synthetic (OLAP)						Synthetic (OLTP)					
	Benchmark	1000-case REAL benchmark				TPC-H			TPC-DS			TPC-C			TPC-E		
Method Category	Method	P_{edge}	R_{edge}	F_{edge}	P_{case}	P_{edge}	R_{edge}	F_{edge}	P_{edge}	R_{edge}	F_{edge}	P_{edge}	R_{edge}	F_{edge}	P_{edge}	R_{edge}	F_{edge}
Auto-BI	Auto-BI-P	**0.98**	0.664	0.752	**0.92**	1	0.88	0.93	**0.99**	0.28	0.43	1	0.6	0.75	1	0.4	0.58
	Auto-BI	**0.973**	**0.879**	**0.907**	0.853	1	1	1	0.96	**0.91**	**0.93**	1	**0.8**	**0.89**	0.96	**0.93**	**0.95**
	Auto-BI-S	0.951	0.848	0.861	0.779	1	1	1	0.92	0.89	0.91	1	0.7	0.82	0.93	**0.94**	**0.94**
Commercial	System-X	0.916	0.584	0.66	0.754	0	0	0	0	0	0	0	0	0	0	0	0
Baselines	MC-FK	0.604	0.616	0.503	0.289	1	1	1	0.73	0.65	0.68	0.46	**0.8**	0.63	0.57	0.79	0.48
	Fast-FK	0.647	0.585	0.594	0.259	0.71	0.88	0.79	0.62	0.35	0.44	0.62	0.57	0.6	0.73	0.84	0.78
	HoPF	0.684	0.714	0.67	0.301	0.86	0.75	0.8	0.87	0.51	0.65	0.75	0.7	0.72	0.71	0.91	0.81
	ML-FK	0.846	0.77	0.773	0.557	0.6	0.75	0.667	0.369	0.589	0.454	0.273	0.3	0.286	0.694	0.756	0.723
Language model	GPT-3.5	0.73	0.64	0.67	0.43	0.75	0.75	0.75	0	0	0	0.16	0.2	0.15	0.78	0.56	0.65

Table 5: Quality comparison on the 1000-case REAL benchmark and 4 TPC benchmarks.

		Denormalized (OLAP-like)												Normalized (OLTP-like)											
	Benchmark	Foodmart			Northwind			AdventureWork			WorldWideImporters			Foodmart			Northwind			AdventureWork			WorldWideImporters		
Method Category	Method	P_e	R_e	F_e	P_e	R_e	F_e	P_e	R_e	F_e	P_e	R_e	F_e	P_e	R_e	F_e	P_e	R_e	F_e	P_e	R_e	F_e	P_e	R_e	F_e
Auto-BI	Auto-BI-P	1	0.5	0.67	1	1	1	1	0.34	0.51	1	0.31	0.47	1	0.63	0.77	1	1	1	1	0.44	0.62	1	0.25	0.4
	Auto-BI	0.75	1	**0.86**	1	1	1	0.98	0.93	0.97	1	0.83	**0.91**	0.8	1	**0.89**	1	1	1	0.97	0.82	**0.89**	0.97	**0.86**	**0.91**
	Auto-BI-S	0.68	1	0.81	0.9	1	0.95	0.94	0.91	0.92	1	0.79	0.88	0.8	1	0.88	1	1	1	0.93	0.8	0.86	0.92	0.84	0.88
Commercial	System-X	0.75	0.6	0.67	0.85	0.79	0.82	0.97	0.66	0.78				0.75	0.6	0.67	0.9	1	0.95	0.9	0.34	0.5	1	0.09	0.17
Baselines	MC-FK	0.42	0.9	0.57	0.28	1	0.44	0.33	0.71	0.45	0.21	**0.86**	0.34	0.54	0.88	0.67	0.43	0.8	0.56	0.2	**0.87**	0.33	0.2	0.52	0.3
	Fast-FK	0.38	0.6	0.46	0.53	0.64	0.65	0.33	0.34	0.34	0.27	0.47	0.35	0.55	0.75	0.63	0.55	0.6	0.57	0.43	0.69	0.53	0.45	0.14	0.2
	HoPF	0.44	0.4	0.42	0.53	0.5	0.52	0.78	0.73	0.75	0.89	0.72	0.8	0.43	0.38	0.4	0.8	0.8	0.8	0.55	0.71	0.62	0.75	0.51	0.61
	ML-FK	0.43	0.8	0.56	0.35	0.86	0.5	0.84	0.8	0.82	0.89	0.75	0.81	0.57	0.88	0.69	0.5	0.9	0.64	0.95	0.83	**0.89**	0.9	0.5	0.64
Language model	GPT-3.5	1	0.6	0.75	1	0.71	0.83	1	0.32	0.48	0.77	0.69	0.73	1	0.75	0.86	1	1	1	0.87	0.54	0.67	0.88	0.61	0.72

Table 6: Quality comparison on FoodMart, NorthWind, AdventureWork and WorldWideImporters . $(P_e, R_e, F_e) = (P_{edge}, R_{edge}, F_{edge})$.

	# of tables	4	5	6	7	8	9	10	[11,15]	[16,20]	21+
Case-type statistics	(ST,SN,C,O)	(50,18,31,1)	(49,12,37,2)	(36,14,48,2)	(19,23,49,9)	(10,22,57,11)	(7,25,50,18)	(2,39,40,19)	(7,14,60,19)	(1,7,72,20)	(9,5,62,24)
Auto-BI	Auto-BI	0.97 (0.99,0.95)	0.97 (0.98,0.96)	**0.96** (0.99,0.94)	**0.95** (0.98,0.91)	**0.95** (0.99,0.92)	0.96 (1.00,0.93)	**0.94** (0.98,0.90)	**0.90** (0.97,0.85)	**0.84** (0.95,0.75)	**0.79** (0.94,0.69)
	Auto-BI-P	0.91 (1.00,0.84)	0.91 (0.99,0.85)	0.88 (1.00,0.79)	0.81 (0.98,0.69)	0.83 (0.98,0.71)	0.83 (1.00,0.70)	0.81 (0.99,0.69)	0.71 (0.98,0.56)	0.60 (0.96,0.44)	0.55 (0.95,0.39)
	Auto-BI-S	0.95 (0.98,0.93)	0.95 (0.96,0.94)	0.94 (0.97,0.92)	0.94 (0.97,0.90)	0.95 (0.96,0.91)	0.93 (0.97,0.89)	0.92 (0.96,0.88)	0.85 (0.92,0.79)	0.78 (0.90,0.70)	0.74 (0.89,0.63)
Commercial	System-X	0.76 (0.94,0.66)	0.67 (0.91,0.55)	0.76 (0.94,0.66)	0.76 (0.93,0.66)	0.75 (0.91,0.65)	0.78 (0.91,0.70)	0.77 (0.92,0.67)	0.74 (0.90,0.65)	0.65 (0.80,0.56)	0.66 (0.88,0.54)
Baselines	MC-FK	0.69 (0.88,0.57)	0.65 (0.93,0.49)	0.63 (0.70,0.58)	0.65 (0.67,0.63)	0.62 (0.70,0.56)	0.56 (0.46,0.72)	0.54 (0.48,0.63)	0.54 (0.48,0.63)	0.52 (0.42,0.69)	0.42 (0.30,0.68)
	Fast-FK	0.76 (0.79,0.72)	0.76 (0.79,0.74)	0.68 (0.69,0.66)	0.53 (0.53,0.52)	0.65 (0.67,0.63)	0.47 (0.46,0.47)	0.45 (0.48,0.42)	0.49 (0.53,0.46)	0.47 (0.49,0.46)	0.42 (0.40,0.44)
	HoPF	0.83 (0.86,0.81)	0.77 (0.77,0.76)	0.72 (0.73,0.71)	0.63 (0.58,0.70)	0.68 (0.67,0.70)	0.55 (0.49,0.64)	0.62 (0.55,0.70)	0.61 (0.58,0.64)	0.57 (0.55,0.60)	0.49 (0.44,0.56)
	ML-FK	0.87 (0.91,0.84)	0.86 (0.91,0.85)	0.8 (0.94,0.79)	0.83 (0.88,0.83)	0.84 (0.89,0.84)	0.8 (0.84,0.81)	0.8 (0.88,0.79)	0.72 (0.84,0.71)	0.64 (0.69,0.65)	0.56 (0.65,0.6)
Language model	GPT-3.5	0.79 (0.86,0.76)	0.75 (0.83,0.7)	0.8 (0.83,0.78)	0.74 (0.78,0.74)	0.72 (0.77,0.69)	0.69 (0.76,0.66)	0.69 (0.76,0.66)	0.56 (0.62,0.54)	0.49 (0.56,0.46)	0.39 (0.46,0.36)

Table 7: Edge-level quality reported as "F-1 (precision, recall)", by number of input tables in REAL benchmark. In the first row, we report "case type statistics", denoted as (ST, SN, C, O), which stand for the number of cases in (Star, Snowflake, Constellation, Others), respectively.

- **REAL**. We sample 1000 real BI models crawled in the wild (Section 3.2) as our first benchmark, henceforth referred to as REAL. In order to account for different levels of difficulty in predicting BI models, we perform stratified sampling as follows – we bucketize models into 10 groups based on the number of input tables as {4, 5, 6, 7, 8, 9, 10, [11-15], [16-20], 21+}, and randomly select 100 cases in each group, to create this 1000-case benchmark that covers the entire spectrum in terms of levels of difficulty, from easy (with only a few tables) to hard (with over 20 tables). Table 3 summarizes the characteristics of the 1000-case REAL benchmark. Note that these test cases are held out and never used when training of our local classifier, which is consistent with the standard practice of machine learning to avoid data leakage.

In comparison, Table 2 shows the characteristics of all BI models harvested in the entire collection, which are substantially simpler (e.g., with a small number of tables). This is not entirely surprising, when these BI models are programmed by non-technical business users using GUI tools, which is why we perform stratified sampling described above to create a more balanced and challenging test set.

- **SYNTHETIC**. In addition to evaluation on real BI models, we perform tests on 4 TPC benchmarks, henceforth referred to as SYNTHETIC. Specifically, TPC-H and TPC-DS are two popular benchmarks for BI/OLAP workloads. We evaluate predicted joins with the ground-truth relationships in TPC specifications. We further perform tests on TPC-C and TPC-E, two popular OLTP benchmarks. While Auto-BI is not designed for general OLTP databases, we perform the tests nevertheless, in order to understand Auto-BI's ability to detect foreign-keys in general (beyond the snowflake-like schemas that Auto-BI was initially designed for).

	# of tables	4	5	6	7	8	9	10	[11,15]	[16,20]	21+
Auto-BI	Auto-BI	**1.00**	0.96	0.95	0.89	0.95	0.97	0.85	0.78	0.64	0.55
	Auto-BI-P	**1.00**	**0.98**	**0.99**	**0.94**	**0.96**	**0.99**	**0.95**	**0.89**	**0.83**	**0.67**
	Auto-BI-S	0.99	0.95	0.93	0.93	0.95	0.80	0.76	0.69	0.49	0.31
Commercial	System-X	0.91	0.87	0.81	0.85	0.77	0.78	0.74	0.75	0.52	0.54
Baselines	MC-FK	0.76	0.68	0.41	0.34	0.19	0.14	0.1	0.13	0.09	0.05
	Fast-FK	0.68	0.65	0.39	0.14	0.32	0.08	0.09	0.12	0.09	0.03
	HoPF	0.78	0.57	0.42	0.21	0.32	0.06	0.18	0.19	0.18	0.11
	ML-FK	0.87	0.86	0.68	0.65	0.7	0.55	0.53	0.42	0.18	0.12
Language model	GPT-3.5	0.67	0.61	0.61	0.6	0.44	0.42	0.42	0.21	0.1	0.05

Table 8: Case-level precision, by number of tables.

We further perform tests on 4 commonly-used synthetic databases: FoodMart [3], Northwind [4], AdventureWorks [2] and WorldWide-Importers [6]. We use both of their OLAP and OLTP versions to stress test our algorithms, for a total of 8 additional test databases.

Metrics. We compare the predicted joins of different methods, against the ground truth (which in the case of REAL, are human-specified relationships that we programmatically extract from BI models we crawled). We evaluate prediction quality of algorithms both at the *edge-level* and *case-level*, defined as below:

Edge-level quality. For each BI test case C, we evaluate the fraction of predicted join relationships (edges on the graph) that is identical to ground-truth relationships. We use standard precision/recall/F metrics for the edge-level evaluation, defined as precision $P_{edge}(C) = \frac{\text{num-of-correctly-predicted-edges}}{\text{num-of-total-predicted-edges}}$, recall $R_{edge}(C) = \frac{\text{num-of-correctly-predicted-edges}}{\text{num-of-total-ground-truth-edges}}$. The F-1 score, denoted by $F_{edge}(C)$, is then the harmonic mean of precision and recall, or $\frac{2P_{edge}(C)R_{edge}(C)}{P_{edge}(C)+R_{edge}(C)}$.

We report precision/recall/F-1 for the entire REAL benchmark, as the average across all 1000 test cases.

Case-level quality. Since it is difficult for non-technical users to identify incorrectly-predicted relationships, high precision is crucial for Auto-BI (Section 4.3.2). We report *case-level* precision

for each test case C, denoted by $P_{case}(C)$, defined as:

$$P_{case}(C) = \begin{cases} 1, & \text{if } P_{edge}(C) = 1 \\ 0, & \text{otherwise} \end{cases} \qquad (20)$$

In other words, $P_{case}(C)$ is 1 only if no incorrect edge is predicted; and 0 even if a single false-positive edge is predicted (for it is unlikely that non-technical users can identify and correct the incorrect predictions, making us to "fail" in such a situation). The case-level precision on the entire benchmark is also the average across all 1000 cases. [6]

Latency. We report the latency of all methods using wall-clock time. All methods are implemented using Python 3.8 and tested on a Windows 11 machine with 64-core Intel CPU and 128 GB memory.

5.2 Methods compared

We compare Auto-BI with the following methods in the literature. Our code and data are available for reproducibility [1].

MC-FK [58]. MC-FK pioneered a method to accurately detect multi-column FKs, using an EMD-based randomness metric that measures the distribution similarity of two columns.

Fast-FK [17]. Fast-FK makes FK predictions based on a scoring function that combines column-value and column-name similarity. This method employs fast pruning and focuses on ensuring interactive predictions.

HoPF [30]. HoPF is a recent method that detects FKs and PKs together, in order to ensure better prediction accuracy. It employs hand-tuned scoring functions as well as structural constraints (e.g., no cycles) to improve accuracy.

ML-FK [48]. ML-FK is an ML-based approach to predict FK, which is similar in spirit to our local-classifiers, and we feed ML-FK with the same training data used by our local-classifiers.

System-X. System-X is a commercial system from a leading BI vendor that has a feature to detect joins. We anonymize the name of the system, in keeping with benchmarking traditions in the database literature [7, 23, 44, 49].

GPT-3.5. GPT is a family of language models [13] capable of performing many downstream tasks such as NL-to-SQL [45, 54] and schema-matching [59]. We use GPT as a baseline with few-shot learning [13], where we provide few-shot demonstrations of the join task in the prompt, and then ask the model to predict for new test cases. We use the latest publicly available version of GPT (GPT-3.5-turbo), [7] and in the prompt we provide both the table-schema (column names) as well as sample data rows. Note that because synthetic benchmarks (e.g., TPC) are heavily represented on the web, it is likely that GPT has seen such cases in its pre-training (e.g., TPC queries on the web), causing data leakage and inflated results. We report these numbers nevertheless, for reference purposes.

Auto-BI. This is our proposed method using global k-MCA optimization. We evaluate three variants of our method: (1): **Auto-BI-Precision (Auto-BI-P)** is the precision-mode of Auto-BI (Section 4.3.2) that focuses on high precision (by finding snowflake-like "backbones" from the schema). (2): **Auto-BI** is our full approach

with both precision and recall modes (Section 4.3.3). (3): Furthermore, we test a light-weight version of Auto-BI called **Auto-BI-Schema-Only (Auto-BI-S)**, which uses only schema-level metadata (table/column names) for local-classifiers (without using column-values). This method is thus efficient, with sub-second latency in most cases, while still being surprisingly accurate.

5.3 Quality comparisons

Overall comparison. Table 5 compares the overall quality of all methods, on both the 1000-case Real benchmark and 4 Synthetic TPC benchmarks. As we can see, Auto-BI-based methods consistently outperform alternatives, with Auto-BI-P always being the best in precision (very close to 1), while the full Auto-BI being the best overall in F-scores.

On the 1000-case Real benchmark, we note that the difference between Auto-BI and the best baselines is significant – the difference is 13 percentage points for edge-level precision (0.98 vs. 0.846), and over 10 percentage points for edge-level recall (0.879 vs. 0.77). The difference is even more pronounced at the case-level, which is over 35 percentage points (0.92 vs. 0.557), showing the strong benefit on quality when using a principled global optimization in our k-MCA.

Even the light-weight Auto-BI-S (with the same k-MCA but using schema-only features) is surprisingly accurate (losing only 2-3 percentage points in precision/recall). As we will see, it is much more efficient, however, making it a strong contender for practical use when latency becomes important.

For the Synthetic TPC benchmarks, we observe similar trends consistent with the Real benchmark. It is interesting to note that, not only is Auto-BI the best on OLAP benchmarks (TPC-H/TPC-DS), it turns out to be the best on OLTP benchmarks too (TPC-C/TPC-E). This is somewhat surprising, because OLTP databases do not usually follow snowflake-like schema, and are not the use cases that Auto-BI is originally designed for. We inspected TPC-C and TPC-E carefully to understand why Auto-BI is effective. It turns out that while these OLTP databases do not have snowflake designs, they nevertheless have clusters of tables (e.g., a cluster of tables on "Customers", and another cluster on "Market", etc., for TPC-E), where each cluster loosely follows a hub-and-spoke pattern with related tables joining through a few "central" tables (e.g., a cluster of tables relating to customers, such as "Customers-Account" and "Customers-Tax-Rate", all join through a central "Customers" table in TPC-E, like very nicely depicted in Figure 2 of prior work on schema summarization [57]). Similar results are also observed in additional synthetic benchmarks as shown in Table 6. Exploring the applicability of Auto-BI-like global optimization in FK-detection for general OLTP databases is an interesting area for future work.

Among all baselines, the commercial System-X produces high precision at the cost of a low recall. Among the four FK methods from the literature, ML-FK produces the best quality (since we feed it with the same training data harvested from real BI models), while the remaining three have comparable precision/recall.

Detailed breakdown. Table 7 and Table 8 show a more detailed comparison of edge-level and case-level quality, respectively, bucketized by the number of input tables. The overall trend is consistent with Table 5 – Auto-BI has the best F-1, while Auto-BI-P has the

---

[6] We note that it is possible that a predicted join graph may differ syntactically from the ground truth when the two are semantically equivalent. For example, the ground truth may have a fact-table F join a dimension-table A (N:1), which in turn 1:1 join with another dimension-table B. If it is predicted that F joins with B (N:1), and then B joins with A (1:1), while the ground truth is F-(N:1)-A-(1:1)-B, then the are in fact identical (except that the join order is different). We account for semantic equivalence in our evaluation and mark both as correct, which is fair for all methods tested.
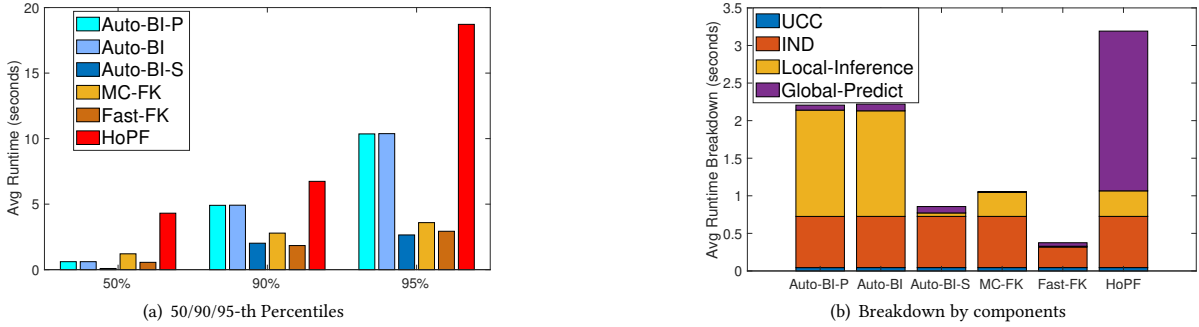
[7] https://platform.openai.com/docs/models/gpt-3-5

(a) 50/90/95-th Percentiles

(b) Breakdown by components

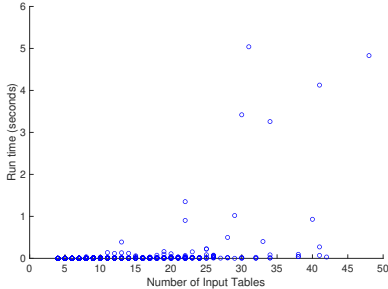**Figure 5: Comparison of end-to-end latency.**

Figure 6: Latency distribution of k-MCA-CC on 1000-case REAL. The 50/90/95-th p-tiles are 0.02/0.06/0.17 seconds, respectively.

best precision across the board. We note that AUTO-BI works reasonably well on the most challenging slice of the test cases (with 21+ tables), maintaining high precision (0.94) and reasonable recall.

We report additional experiments and analysis (comparing with baselines enhanced using components from AUTO-BI to better understand its component-wise benefits) in Appendix C.

5.4 Efficiency comparison

Overall comparison. We show the 50/90/95-th percentile latency of all methods in Figure 5(a), on the 1000-case REAL benchmark. Overall, AUTO-BI-S and Fast-FK are the fastest, taking 2-3 seconds even in the largest cases. AUTO-BI is 2-3x slower but the latency is still acceptable. HoPF takes the most time in comparison.

We also report the latency breakdown of all methods in Figure 5(b). Overall, there are 4 main components for all methods: (1) UCC: generate unique column combinations; (2) IND: generate inclusion dependencies; (3) Local-Inference: generate features and perform local-classifier inference for joins in AUTO-BI (while baselines use predefined scoring functions here); (4) Global-Predict: AUTO-BI uses k-MCA, whereas baselines use various other algorithms. For AUTO-BI, the most expensive part is Local-Inference (specifically, feature generation for inference). AUTO-BI-S is effective in reducing the latency of this part (by using metadata-only features), thus achieving significant speedup. IND-generation is also expensive for all methods, but it is a standard step and the latency is comparable across all methods. In Global-Predict, we see that AUTO-BI (using the k-MCA algorithm) is highly efficient here. In comparison, baselines like MC-FK and HoPF are very expensive.

Efficiency of k-MCA-CC. Our key innovation in AUTO-BI is the k-MCA-CC formulation and its efficient solutions. Despite the hardness of the problem, we solve it optimally (Algorithm 3), which is efficient as shown by the purple Global-Predict component in Figure 5(b). It is interesting to drill down and study the latency of this key step in detail.

Figure 6 shows the distribution of latency when we solve k-MCA-CC on the 1000-case REAL. We can see that the latency for the vast majority of cases is sub-second. In fact, the 50-th, 90-th and 95-th

percentile latency are 0.02, 0.06, and 0.17 seconds, respectively. This confirms that our Algorithm 3 is not only optimal but also real-time and practical. There are a few cases where the latency is over 1 second, which are mostly cases with more than 40 input tables, which we argue is acceptable considering the size of the data. We also report that the largest case we encounter among all 10K+ BI models has 88 input tables, which k-MCA-CC still solves optimally in about 11 seconds (not shown in the figure as it is not sampled in the 1000-case REAL). This confirms that our algorithm is practical in solving large real-world BI cases both optimally and efficiently.

Figure 7 shows the benefit of the two key optimization techniques we used in solving k-MCA-CC: (1) artificial root in k-MCA (Algorithm 2) and (2) branch-and-bound in k-MCA-CC (Algorithm 3). We compare the number of 1-MCA calls with and without the optimizations. (Note that we use the number of 1-MCA calls instead of wall-clock time, because the algorithm can timeout without our optimizations). We can see that the two optimization techniques give 5 and 4 orders of magnitude improvement in efficiency, respectively (the y-axis is in log-scale). Compared to a brute-force approach that solves k-MCA without optimization, the combined benefit is a staggering 10 orders of magnitude, again showing the importance of our algorithms.

5.5 Ablation Studies

We perform an ablation study to analyze the benefits of various ideas developed in AUTO-BI, using the 1000-case REAL benchmark. Our results are summarized in Figure 8.

The effect of FK-once constraint. The bar marked as "no-FK-once-constraint" in Figure 8 shows the result quality that removes the FK-once constraint (or using k-MCA instead of k-MCA-CC). There is a noticeable drop in precision/recall, showing the importance of this constraint in k-MCA-CC.

The effect of precision-mode. The "no-precision-mode" bar shows the result quality if we omit the precision-mode step (Section 4.3.2) and use the recall-mode directly (Section 4.3.3). We see a substantial drop in precision (6 and 13 percentage points at edge-level and case-level, respectively), again showing the importance of k-MCA-CC.

The effect of N-1/1-1 separation in local classifier. The "no-N-1/1-1-seperation" bar shows the results when we use one local classifier, instead of splitting N-1 and 1-1 joins into two prediction tasks (Section 4.2). The drop in quality is again evident.

The effect of label transitivity in local classifier. The bar for "no-label-transitivity" shows the drop in quality when we do not apply label transitivity in local classification (Section 4.2). The effect is noticeable but diluted across 1000 cases, as the benefit is more pronounced in large cases and less obvious in smaller ones.

The effect of no data-value features in local classifier. The "no-data-features" bar shows the effect of removing features relating
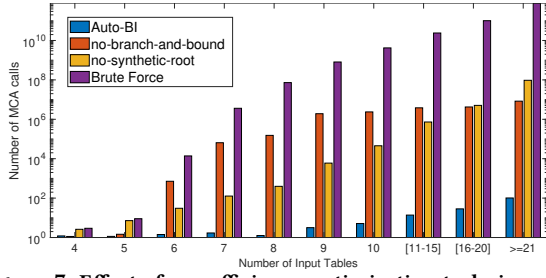
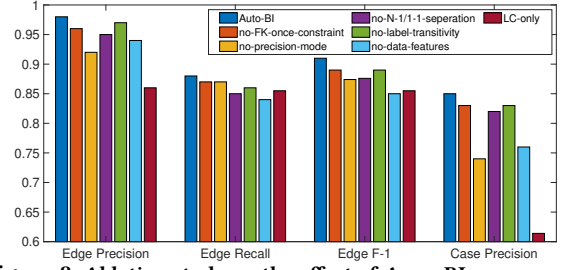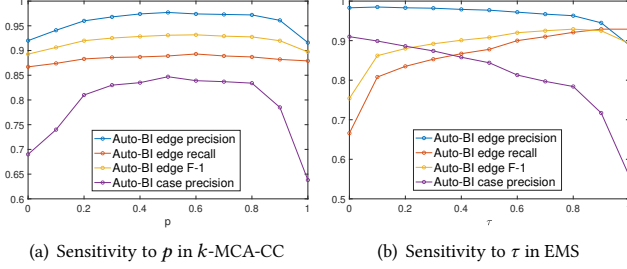**Figure 7: Effect of our efficiency optimization techniques, as measured by the number of 1-MCA invocations.**

Figure 8: Ablation study on the effect of Auto-BI components on result quality (average of 1000-case Real).

(a) Sensitivity to p in k-MCA-CC

(b) Sensitivity to τ in EMS

Figure 9: Sensitivity study on Real benchmark.

to data-value in our local classifier (we use meta-data features only instead). This directly corresponds to our lightweight Auto-BI-S (Section 5.3). The drop in quality is apparent but not too significant, which gives us a useful trade-off between quality and latency.

The effect of using local classifier only. Lastly, we perform an important ablation study, to understand the benefit of our graph-based k-MCA algorithm on top of local classifier results. The "LC-only" bar shows the performance if we employ the same local classifier at the edge-level (keeping edges that have calibrated probability over 0.5), without using a holistic k-MCA optimization. The difference in performance is significant (25 percentage points in case-precision), underscoring the importance of our graph-based k-MCA.

5.6 Sensitivity Analysis

Figure 9 shows a sensitivity analysis of the two parameters in Auto-BI, the penalty-term p in k-MCA (Section 4.3.2), and the edge-weight threshold τ in EMS (Section 4.3.3). Recall that because we use calibrated true probabilities, 0.5 is the natural choice in both cases. This is confirmed in our study. Figure 9(a) shows that when varying p from 0 to 1, the region around 0.5 indeed gives the best result. Figure 9(b) shows the sensitivity to τ, which is used to prune unpromising edges, where a lower τ naturally leads to better recall at the cost of precision. We see that $\tau = 0.5$ strikes a reasonable balance between precision/recall and produces high F-1, which is also our default setting in Auto-BI.

6 CONCLUSIONS AND FUTURE WORK

In this work we develop an Auto-BI system that can accurately predict join relationships in BI models, leveraging a novel graph-based formulation called k-MCA that exploits snowflake-like structures common in BI models.

Future directions include improving the end-to-end efficiency of Auto-BI, and extending the system to synthesize transformation steps, in order to automate BI model building end-to-end.

REFERENCES

[1] [n.d.]. Benchmark data will be released after an internal review:. https://github.com/yiminl18/Auto-BI.

[2] 2022. AdventureWorks Database. https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks.

[3] 2022. FoodMart Database. https://begincodingnow.com/microsofts-foodmart-database/.

[4] 2022. NorthWind Database. https://github.com/Microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs.

[5] 2022. SentenceTransformers. https://www.sbert.net/.

[6] 2022. WorldWide-Importers Database. https://github.com/Microsoft/sql-server-samples/releases/tag/wide-world-importers-v1.0.

[7] Retrieved in 2023-01. DeWitt Clause (retrieved 2022-09). https://en.wikipedia.org/wiki/David_DeWitt#DeWitt_Clause.

[8] Retrieved in 2023-01. Power BI. https://powerbi.microsoft.com/en-us/.

[9] Retrieved in 2023-01. Power BI: Create and manage relationships. https://learn.microsoft.com/en-us/power-bi/transform-model/desktop-create-and-manage-relationships.

[10] Retrieved in 2023-01. Tableau. https://www.tableau.com/.

[11] Retrieved in 2023-01. Tableau: Relate your data by drag-and-drop. https://help.tableau.com/current/pro/desktop/en-us/relate_tables.htm.

[12] Jana Bauckmann, Ulf Leser, Felix Naumann, and Véronique Tietz. 2007. Efficiently detecting inclusion dependencies. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 1448–1450.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[14] Marco A Casanova, Ronald Fagin, and Christos H Papadimitriou. 1982. Inclusion dependencies and their interaction with functional dependencies. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*. 171–176.

[15] Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. 2011. An overview of business intelligence technology. *Commun. ACM* 54, 8 (2011), 88–98.

[16] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. 2006. A primitive operator for similarity joins in data cleaning. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 5–5.

[17] Zhimin Chen, Vivek Narasayya, and Surajit Chaudhuri. 2014. Fast foreign-key detection in Microsoft SQL server PowerPivot for Excel. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1417–1428.

[18] Yoeng-Jin Chu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica* 14 (1965), 1396–1400.

[19] Fabien De Marchi, Stéphane Lopes, and Jean-Marc Petit. 2002. Efficient algorithms for mining inclusion dependencies. In *International Conference on Extending Database Technology*. Springer, 464–476.

[20] Jack Edmonds et al. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B* 71, 4 (1967), 233–240.

[21] Bruno Escoffier and Vangelis Th Paschos. 2006. Completeness in approximation classes beyond apx. *Theoretical computer science* 359, 1-3 (2006), 369–377.

[22] Jean-Claude Fournier. 2013. *Graphs theory and applications: with exercises and problems*. John Wiley & Sons.

[23] Florian Funke, Alfons Kemper, and Thomas Neumann. 2011. Benchmarking hybrid oltp&olap database systems. *Datenbanksysteme für Business, Technologie und Web (BTW)* (2011).

[24] Gartner. [n.d.]. The Gartner 2022 Analytics BI Platforms Magic Quadrant Highlights. ([n. d.]).

[25] Gartner. [n.d.]. How to Enable Self-Service Analytics. ([n. d.]).

[26] Venkatesan Guruswami, Johan Håstad, Rajsekar Manokaran, Prasad Raghavendra, and Moses Charikar. 2011. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM J. Comput.* 40, 3 (2011), 878–914.

[27] Pat Hanrahan. 2006. Vizql: a language for query, analysis and visualization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. 721–721.

[28] Juris Hartmanis. 1982. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review* 24, 1 (1982), 90.

[29] Yeye He, Kris Ganjam, and Xu Chu. 2015. Sema-join: joining semantically-related tables using big table corpora. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1358–1369.

[30] Lan Jiang and Felix Naumann. 2020. Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems* 54, 3 (2020), 439–461.

[31] David R Karger, Philip N Klein, and Robert E Tarjan. 1995. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)* 42, 2 (1995), 321–328.

[32] Ralph Kimball and Margy Ross. 2011. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons.

[33] Sebastian Kruse, Thorsten Papenbrock, Christian Dullweber, Moritz Finke, Manuel Hegner, Martin Zabel, Christian Zollner, and Felix Naumann. 2017. Fast approximate discovery of inclusion dependencies. *Datenbanksysteme für Business, Technologie und Web (BTW 2017)* (2017).

[34] Sebastian Kruse, Thorsten Papenbrock, and Felix Naumann. 2015. Scaling out the discovery of inclusion dependencies. *Datenbanksysteme für Business, Technologie und Web (BTW 2015)* (2015).

[35] Ailsa H Land and Alison G Doig. 2010. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*. Springer, 105–132.

[36] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. 2015. The mannheim search join engine. *Journal of Web Semantics* 35 (2015), 159–166.

[37] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples. In *Proceedings of the 2021 International Conference on Management of Data*. 1064–1076.

[38] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. 2007. Show me: Automatic presentation for visual analysis. *IEEE transactions on visualization and computer graphics* 13, 6 (2007), 1137–1144.

[39] Fabien De Marchi, Stéphane Lopes, and Jean-Marc Petit. 2009. Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems* 32, 1 (2009), 53–73.

[40] Solomon Negash and Paul Gray. 2008. Business intelligence. In *Handbook on decision support systems 2*. Springer, 175–193.

[41] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*. 625–632.

[42] Arash Dargahi Nobari and Davood Rafiei. 2022. Efficiently transforming tables for joinability. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1649–1661.

[43] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. 2015. Divide & conquer-based inclusion dependency discovery. *Proceedings of the VLDB Endowment* 8, 7 (2015), 774–785.

[44] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J DeWitt, Samuel Madden, and Michael Stonebraker. 2009. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 165–178.

[45] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498* (2022).

[46] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[47] Kenneth H Rosen. 2008. ITS APPLICATIONS. (2008).

[48] Alexandra Rostin, Oliver Albrecht, Jana Bauckmann, Felix Naumann, and Ulf Leser. 2009. A machine learning approach to foreign key discovery.. In *WebDB*.

[49] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J Carey, Ioana Manolescu, and Ralph Busse. 2002. XMark: A benchmark for XML data management. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 974–985.

[50] Nuhad Shaabani and Christoph Meinel. 2015. Scalable inclusion dependency discovery. In *International Conference on Database Systems for Advanced Applications*. Springer, 425–440.

[51] Nuhad Shaabani and Christoph Meinel. 2018. Improving the efficiency of inclusion dependency detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 207–216.

[52] Yasin N Silva, Walid G Aref, and Mohamed H Ali. 2010. The similarity join database operator. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 892–903.

[53] Fabian Tschirschnitz, Thorsten Papenbrock, and Felix Naumann. 2017. Detecting inclusion dependencies on very many tables. *ACM Transactions on Database Systems (TODS)* 42, 3 (2017), 1–29.

[54] Lihan Wang, Bowen Qin, Binyuan Hui, Bowen Li, Min Yang, Bailin Wang, Binhua Li, Jian Sun, Fei Huang, Luo Si, et al. 2022. Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1889–1898.

[55] Robert H Warren and Frank Wm Tompa. 2006. Multi-column substring matching for database schema translation. In *Proceedings of the 32nd international conference on Very large data bases*. 331–342.

[56] Cong Yan and Yeye He. 2020. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1539–1554.

[57] Xiaoyan Yang, Cecilia M Procopiuc, and Divesh Srivastava. 2009. Summarizing relational databases. *Proceedings of the VLDB Endowment* 2, 1 (2009), 634–645.

[58] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2010. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 805–814.

[59] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas Mueller, Dalitso Banda, Fotis Psallidas, and Jignesh Patel. 2023. Schema Matching using Pre-Trained Language Models. In *ICDE 2023*.

[60] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.

A LOCAL JOIN: TWO OPTIMIZATIONS

Separate N-1 and 1-1 classifiers. Unlike key/foreign-key that are typically N-1 joins, joins in real BI models can also be 1-1 joins, especially between related dimension tables describing the same logical entity (e.g., "employee-salaries" and "employee-details").

Initially, we use one generic classifier to predict joinability between (C_i, C_j) regardless of 1:1 or N:1. While performing an error analysis on incorrect predictions, we realized that while both 1:1 or N:1 are joins, they represent conceptually different relationships with disparate characteristics. For example, 1:1 joins tend to be between the primary-keys of two dimension tables on the same logical entity (e.g. "employees"), often with the same number of rows from both tables, producing perfect 1-to-1 matches (N:1 joins are, on the other hand, very different). Furthermore, because 1:1 joins tend to be between tables about the same logical entity with complementary information, the column-headers for both tables that do not join will also have high overlap (e.g., with the prefix "employees_" in the example of "employee-salaries" and "employee-details").

We thus treat N:1 and 1:1 as two separate classification tasks. This separation of is unique in Auto-BI and we will show the effect of this in our ablation studies.

Enhance labels with transitivity. Given that we deal with entire BI models that have multiple tables, using the pair-wise join/not-join labels is no longer sufficient, because joinability can be "transitive".

Consider, for instance, a table "Fact_Sales" has a FK column called "sales_emp_id", that N:1 joins with table "Dim_Employee" on its PK column "emp_id", which in turn 1:1 joins with table "Dim_Employee_Salaries" also on its PK "emp_id". In such cases, even though "Fact_Sales.sales_emp_id" does not directly join with "Dim_Employee_Salaries.emp_id", and the pair would ordinarily be considered a negative example (should not join), the two actually refer to the same semantic concept (employee_ids) and are logically joinable. Formally, given the joinability label $L_{ij} = 1$ for (C_i, C_j) and $L_{jk} = 1$ for (C_j, C_k), we mark $L_{ik} = 1$ for (C_i, C_k) in training, even if there is no explicit join between (C_i, C_k). Applying label transitivity overcomes incomplete ground-truth and makes it easier for ML models to fit the ground-truth, which leads to better quality end-to-end.

We note that the transitivity-based optimization is unique when we deal with general types of joins on a large graph, which is not needed when only pair-wise joins are considered, or when only PK-FK (N:1) joins are used as they are typically not transitive.

B LOCAL JOIN CLASSIFIER: DETAILS

We give details on the Local Classifier (LC) in Section 4.2, which takes two columns (C_i, C_j) and predicts the joinability label L_{ij}.

As discussed in Section 4.2, we separate the prediction problem for N:1 and 1:1 joins, since the two types of joins are different at a conceptual level. For each classifier, we featurize each column pair (C_i, C_j), which can be categorized as metadata-features (column-names, table-names) as well as actual data-features (e.g., data value overlap). In the following, we will describe the features of both the N:1 and 1:1 classifiers.

N:1 classifier. We design the following features for the N:1 classifier. We list the name of each feature, as well as its description.

Metadata-features. These are features relating to column-names and table-names. We pre-process all column/table names, by first standardizing all names into tokens, based on camel-casing and delimiters (e.g., dash or underscores).

- *Jaccard_similarity, Jaccard_containment, Edit_distance, Jaro_winkler*: These are standard string similarity between names. We compute these similarities between two column names (C_i, C_j). In addition, assuming C_i is the "1" side, we also compute the similarities between $(T_i + C_i, C_j)$, where T_i is the name of the table from the "1" side, where the observation is that while column-names from fact-tables are often fully descriptive (e.g., "Employee-ID"), sometimes column-names from dimension tables are simplified (e.g., "ID" and 'Name"), with the central entity (e.g., "Employees") only mentioned in table-names, thus making it necessary to piece together table-names and column-names to recover complete metadata. We use $max(sim(C_i, C_j), sim(T_i + C_i, C_j))$ as features for metadata similarity (where sim can be different forms of similarity functions mentioned above, for a total of 4 features).
- *Embedding_similarity*: In addition to string similarity, we also use embedding similarity, specifically SentenceBERT [5, 46] trained on top of all-mpnet-base-v2), to compute the column name similarity, using a setup similar to above.
- *Token_count, Char_count*: the number of tokens and characters in column names, for both C_i, C_j. These serve as auxiliary features on top of similarity scores above (e.g., longer column-names with high similarity may be a more reliable indicator of match, compared to shorter ones with the same similarity scores).
- *Col_frequency*: the frequency of the column name C_i (and C_j) in all BI-models we collect in the training set. Intuitively, matches of common names (e.g., Code and Index) are less reliable, so this feature works similarly to Inverse-Document-Frequency (IDF) in TF-IDF.
- *Col_position*: This feature keeps the positional index of C_i and C_j, counting from left (e.g., the 2nd column from left). This is based on the observation that columns on the left are more likely to join.
- *Col_relative_position*: This is the same feature as *Col_position* above, except that it is measured in relative terms, defined as: $\frac{Col\_position}{Num\_of\_total\_columns}$.
- *Unique_col_position*: Similar to *Col_position*, except here we count unique columns, where the observation is that the first few *unique* column from the left of a table is more likely to be PK for join.

Data-features. These features relate to the actual data content in columns, such as value overlap. Data-features provide complementary signals to metadata-features above, but are generally more expensive to process.

- *Left_containment, Right_containment, Max_containment*: value containment is an important signal for PK/FK joins [17, 30, 58]. we compute containment in both directions (left and right), and also the max of the two, for a total of three features.
- *Value_distinct_ratio*: This feature calculates the distinctness of columns, or the fraction of values in C_i and C_j that are distinct.

- *Range_overlap*: computes the overlap of the min/max value ranges, if both C_i and C_j are of numeric types.
- *EMD_score*: This is a feature based on distributional-similarity and proposed in [58], we use this to complement the overlap-based features above.
- *Value_length*: we cast all values in C_i and C_j to string type, and compute the average value length (longer values tend to produce more reliable matches).
- *Value_type*: We one-hot encode column value types, such as integer, float, string.
- *Row_cnt*: We use the number of rows in both tables as an auxiliary feature, where the observation is that fact table tends to have more rows compared to dimension tables.
- *Row_ratio, Col_ratio, Cell_ratio*: These use similar intuition as *Row_cnt* above, except that we featurize the ratio of the number of rows/cols/cells explicitly between two tables.

Feature importance. Our results suggest that, for the N:1 classifier, the following features are the most important (in that order, based on sklearn output): *Max_containment, Jaccard_similarity, Col_relative_position, Edit_distance, Jaro_winkler, Range_overlap, EMD_score, Embedding_similarity, Col_frequency*.

1:1 classifier. Our 1:1 classifier share many of the same features as the N:1 classifier (except the *Row_ratio, Col_ratio, Cell_ratio* features, which are applicable to N:1 fact-dimension joins, but would be as useful for 1:1 joins). We omit these identical features, and only describe features that are unique to the 1:1 classifier below.

Metadata-features. These are features based only on column-names and table-names, like in the N:1 classifier.

- *Table_embedding*: We measure the SentenceBERT embedding similarity of table names where T_i and T_j, based on the intuition that two tables with 1-1 join should likely refer to the same entity (e.g., Employees and Employee-Details, or Country and Country-Code, etc.), making high table-name similarity a useful signal.
- *Header_jaccard*: measures the jaccard similarity between all column-names of T_i and T_j. This is based on our observation that two overlapping fact tables with highly similar column names do not 1:1 join in BI models, since such joins produce mostly redundant information. Higher *Header_jaccard* is thus inversely correlated to joinability.

Data-features. These are features relating to column-values.

- *Min_containment*: Instead of using *Max_containment* between the *Left_containment* and *Right_containment* as in the N:1 classifer above, we use *Min_containment* for as the feature 1:1, based on the observation that 1:1 joins tend to join tuple-for-tuple between two tables, unlikely PK/FK joins.

Feature importance. Our results suggest that for the 1:1 classifier, the following feature are the most important (in that order, based on sklearn output): *Min_containment, Col_position, Jaccard_similarity, Col_relative_position, EMD_score, Header_jaccard, Col_frequency* and *Embedding_similarity*.

C ADDITIONAL EXPERIMENTS

We present additional experimental results in this section, mainly focusing on understanding the reason behind the quality advantage of Auto-BI over baselines.

	Methods	Average	50%tile	90%tile	95%tile
Auto-BI	Auto-BI-P	2.21	0.61	4.91	10.36
	Auto-BI	2.23	0.61	4.92	10.38
	Auto-BI-S	0.45	0.09	2.02	2.65
Original Baselines	MC-FK	1.21	0.33	2.79	3.59
	Fast-FK	0.56	0.08	1.84	2.93
	HoPF	4.31	0.25	6.74	18.72
Enhanced Baselines	MC-FK+LC	2.13	0.60	4.91	10.34
	Fast-FK+LC	2.15	0.61	4.92	10.36
	HoPF+LC	6.41	1.13	11.88	27.23

Table 9: Comparison of end-to-end latency. Enhanced baselines (+LC) pay the cost of classifiers, which have latency comparable to Auto-BI methods.

At a high level, the advantage of Auto-BI main comes from two sources: (1) the local-classifier step (Section 4.2), that uses data-driven ML scoring together with principled probability calibration, and (2) the graph-based k-MCA step for global optimization (Section 4). In order to understand the contributions of these two sources, we "enhance" existing baselines, by injecting our local-classifier scores from step (1) into their algorithm as follows.

Enhanced baselines: MC-FK+LC, Fast-FK+LC, HoPF+LC. Since MC-FK, Fast-FK and HoPF are not competitive on the Real benchmark, we inject our Local-Classifier (LC) scores for join-likelihood into these baselines, replacing their heuristic scores with our calibrated classifier scores. This leads to stronger baselines, and allows us to see the benefit attributable to better local classifer scores, vs. the global k-MCA algorithm. To differentiate from the original baselines, we use a "+LC" suffix for these enhanced baselines, which become MC-FK+LC, Fast-FK+LC, and HoPF+LC, respectively.

Quality Comparisons. Table 10 shows the additional comparison with MC-FK+LC, Fast-FK+LC, HoPF+LC, on top of the results reported in Table 5. As we can see, on the Real benchmark, the enhanced baselines improve substantially over the original baselines. However, all baselines still lag behind Auto-BI, especially in terms of precision: the edge-level error rate of Auto-BI is 5x smaller than the best baseline (2% vs. 10%), while the case-level error rate of Auto-BI is 4x smaller (8% vs. 34%). Because all the enhanced baselines use the same LC classifiers as Auto-BI, the precision benefit can be attributable to the global optimization in the k-MCA step, underscoring the importance of our graph-based formulation. On the Synthetic TPC benchmarks, we have similar observations consistent with the Real benchmark. With the help of the better scores from the local classifier, the enhanced baselines improve over the original baselines on both OLAP benchmarks (TPC-H and TPC-DS) and OLTP benchmarks. (TPC-C and TPC-E) But overall, Auto-BI still gains the best performance.

Table 11 and Table 12 report more detailed numbers, bucketized based on the number of input tables. Similar trends are observed here, as we see Auto-BI is still the best method overall in terms of quality, across the spectrum of large and small test cases. The advantage is the most significant in precision, especially on larger test cases, which tend to be more difficult to predict correctly.

Latency comparisons. As we can see from Table 9, introducing the LC classifier step into baselines increases their latency, though not substantially. The enhanced baselines have latency comparable to Auto-BI methods, with the exception of HoPF+LC, which is more expensive in terms of latency.

		Real (OLAP)				Synthetic (OLAP)						Synthetic (OLTP)					
	Benchmark	1000-case REAL benchmark				TPC-H			TPC-DS			TPC-C			TPC-E		
	Metric	P_{edge}	R_{edge}	F_{edge}	P_{case}	P_{edge}	R_{edge}	F_{edge}	P_{edge}	R_{edge}	F_{edge}	P_{edge}	R_{edge}	F_{edge}	P_{edge}	R_{edge}	F_{edge}
Auto-BI	Auto-BI-P	**0.98**	0.664	0.752	**0.92**	1	0.88	0.93	**0.99**	0.28	0.43	1	0.6	0.75	1	0.4	0.58
	Auto-BI	**0.973**	**0.879**	**0.907**	0.853	1	1	1	0.96	**0.91**	**0.93**	1	**0.8**	**0.89**	0.96	**0.93**	**0.95**
	Auto-BI-S	0.951	0.848	0.861	0.779	1	1	1	0.92	0.89	0.91	1	0.7	0.82	0.93	**0.94**	**0.94**
Commercial	System-X	0.916	0.584	0.66	0.754	0	0	0	0	0	0	0	0	0	0	0	0
Original Baselines	MC-FK	0.604	0.616	0.503	0.289	1	1	1	0.73	0.65	0.68	0.46	**0.8**	0.63	0.57	0.79	0.48
	Fast-FK	0.647	0.585	0.594	0.259	0.71	0.88	0.79	0.62	0.35	0.44	0.62	0.57	0.6	0.73	0.84	0.78
	HoPF	0.684	0.714	0.67	0.301	0.86	0.75	0.8	0.87	0.51	0.65	0.75	0.7	0.72	0.71	0.91	0.81
Enhanced Baselines	MC-FK+LC	0.903	**0.872**	0.887	0.636	1	1	1	0.89	0.87	0.88	0.56	**1**	0.57	0.92	0.83	0.88
	Fast-FK+LC	0.898	**0.879**	0.883	0.631	1	0.88	0.93	0.94	0.5	0.6	1	0.7	0.82	0.94	0.87	0.91
	HoPF+LC	0.738	0.765	0.726	0.524	1	0.88	0.93	0.93	0.53	0.68	1	0.7	0.82	0.91	0.88	0.9
	LC	0.885	0.864	0.87	0.631	1	0.88	0.93	0.85	**0.91**	0.88	1	0.6	0.75	1	0.6	0.75

Table 10: Quality comparison on the 1000-case REAL benchmark and 4 TPC benchmarks.

	# of tables	4	5	6	7	8	9	10	[11,15]	[16,20]	21+
Auto-BI	Auto-BI	**0.97 (0.99,0.95)**	**0.97 (0.98,0.96)**	**0.96 (0.99,0.94)**	**0.95 (0.98,0.91)**	**0.95 (0.99,0.92)**	**0.96 (1.00,0.93)**	**0.94 (0.98,0.90)**	**0.90 (0.97,0.85)**	**0.84 (0.95,0.75)**	**0.79 (0.94,0.69)**
	Auto-BI-P	0.91 (1.00,0.84)	0.91 (0.99,0.85)	0.88 (1.00,0.79)	0.81 (0.98,0.69)	0.83 (0.98,0.71)	0.83 (1.00,0.70)	0.81 (0.99,0.69)	0.71 (0.98,0.56)	0.60 (0.96,0.44)	0.55 (0.95,0.39)
	Auto-BI-S	0.95 (0.98,0.93)	0.95 (0.96,0.94)	0.94 (0.97,0.92)	0.94 (0.97,0.90)	0.95 (0.98,0.91)	0.93 (0.97,0.89)	0.92 (0.96,0.88)	0.85 (0.92,0.79)	0.78 (0.90,0.70)	0.74 (0.89,0.63)
Commercial	System-X	0.76 (0.94,0.66)	0.67 (0.91,0.55)	0.76 (0.94,0.66)	0.76 (0.93,0.66)	0.75 (0.91,0.65)	0.78 (0.91,0.70)	0.77 (0.92,0.67)	0.74 (0.90,0.65)	0.65 (0.80,0.56)	0.66 (0.88,0.54)
Baselines	MC-FK	0.69 (0.88,0.57)	0.65 (0.93,0.49)	0.63 (0.70,0.58)	0.65 (0.67,0.63)	0.62 (0.70,0.56)	0.56 (0.46,0.72)	0.54 (0.48,0.63)	0.54 (0.49,0.61)	0.52 (0.42,0.69)	0.42 (0.30,0.68)
	Fast-FK	0.76 (0.79,0.72)	0.76 (0.79,0.74)	0.68 (0.69,0.66)	0.53 (0.53,0.52)	0.65 (0.67,0.63)	0.47 (0.46,0.47)	0.45 (0.48,0.42)	0.49 (0.53,0.46)	0.47 (0.49,0.46)	0.42 (0.40,0.44)
	HoPF	0.83 (0.86,0.81)	0.77 (0.77,0.76)	0.72 (0.73,0.71)	0.63 (0.58,0.70)	0.68 (0.67,0.70)	0.55 (0.49,0.64)	0.62 (0.55,0.70)	0.61 (0.58,0.64)	0.57 (0.55,0.60)	0.49 (0.44,0.56)
Enhanced Baselines	MC-FK+LC	0.86 (0.90,0.83)	0.85 (0.91,0.80)	0.87 (0.87,0.86)	0.89 (0.87,0.90)	0.88 (0.85,0.91)	0.90 (0.87,0.92)	0.87 (0.83,0.91)	0.84 (0.81,0.87)	0.78 (0.79,0.77)	0.74 (0.73,0.74)
	Fast-FK+LC	0.90 (0.89,0.90)	0.91 (0.91,0.92)	0.87 (0.87,0.88)	0.85 (0.85,0.85)	0.87 (0.88,0.86)	0.86 (0.87,0.84)	0.85 (0.85,0.85)	0.83 (0.82,0.84)	0.76 (0.73,0.78)	0.71 (0.68,0.75)
	HoPF+LC	0.85 (0.87,0.83)	0.81 (0.82,0.81)	0.77 (0.78,0.75)	0.70 (0.65,0.75)	0.75 (0.74,0.76)	0.62 (0.57,0.67)	0.64 (0.58,0.72)	0.70 (0.67,0.72)	0.64 (0.60,0.69)	0.53 (0.46,0.62)
	LC	0.89 (0.90,0.88)	0.89 (0.90,0.88)	0.87 (0.87,0.87)	0.88 (0.89,0.87)	0.86 (0.87,0.86)	0.89 (0.91,0.87)	0.86 (0.87,0.84)	0.83 (0.84,0.82)	0.77 (0.79,0.75)	0.72 (0.72,0.71)

Table 11: Edge-level quality reported as "F-1 (precision, recall)", by number of input tables in REAL benchmark.

	# of tables	4	5	6	7	8	9	10	[11,15]	[16,20]	21+
Auto-BI	Auto-BI	**1.00**	0.96	0.95	0.89	0.95	0.97	0.85	0.78	0.64	0.55
	Auto-BI-P	**1.00**	**0.98**	**0.99**	**0.94**	**0.96**	**0.99**	**0.95**	**0.89**	**0.83**	**0.67**
	Auto-BI-S	0.99	0.95	0.93	0.93	0.95	0.80	0.76	0.69	0.49	0.31
Commercial	System-X	0.91	0.87	0.81	0.85	0.77	0.78	0.74	0.75	0.52	0.54
Baselines	MC-FK	0.76	0.68	0.41	0.34	0.19	0.14	0.1	0.13	0.09	0.05
	Fast-FK	0.68	0.65	0.39	0.14	0.32	0.08	0.09	0.12	0.09	0.03
	HoPF	0.78	0.57	0.42	0.21	0.32	0.06	0.18	0.19	0.18	0.11
Enhanced Baselines	MC-FK+LC	0.89	0.92	0.75	0.62	0.67	0.70	0.63	0.53	0.35	0.31
	Fast-FK+LC	0.88	0.92	0.77	0.64	0.78	0.73	0.67	0.49	0.22	0.21
	HoPF+LC	0.87	0.74	0.66	0.62	0.57	0.49	0.48	0.34	0.28	0.19
	LC	0.87	0.85	0.72	0.66	0.68	0.81	0.65	0.53	0.28	0.26

Table 12: Case-level precision, by the number of input tables in REAL benchmark.

D PROOF OF THEOREM 3

PROOF. We show the hardness and inapproximability of k-MCA-CC. We will first show the hardness of the problem using a reduction from Hamilton path [28]. We then show inapproximability using a reduction from non-metric min-TSP [21].

Recall that given a graph $G = (V, E)$, the Hamilton path problem looks for a path that visits each vertex exactly once. We reduce Hamilton path to k-MCA-CC. Given an instance of Hamilton path on $G = (V, E)$, we construct an instance of k-MCA-CC as follows. We construct a new graph $G' = (V, E')$ on the same set of vertices V, where for each $v_i \in V$ we construct a table T_i with a single column C_i in k-MCA-CC. For each directed edge $e(v_i, v_j) \in E$, we construct a possible N:1 FK/PK join between C_i of T_i and C_j of T_j in k-MCA-CC, which would correspondingly create an edge $e(v_i, v_j) \in E'$ in the graph representation for k-MCA-CC. Note that because we construct each table T_i to have a single column C_i, and given the FK-once constraint (Equation (16)), when there are multiple edges pointing away from a vertex v in G', a solution to k-MCA-CC is forced to pick only one such edge, ensuring that the k-MCA returned by the solution contains either single paths, or isolated vertices (which are trivial forms of single paths). Furthermore, in our constructed k-MCA-CC, we set all edge-weight to be unit weight, and set the penalty weight p to be a large constant $|V| + 1$. This large penalty weight ensures that if a 1-MCA exists on G', it is guaranteed to have a lower cost than any k-MCA with $k > 1$, because the penalty term $|V|$ is already larger than the cost of 1-MCA (which is at most $|V| - 1$).

We now show that by this construction, a solution P to Hamilton-path on G, will also be a solution to the k-MCA-CC we constructed. This is the case because if P is a Hamilton-path of G, it must be a feasible solution on G' to k-MCA-CC, because first of all, P is a single path, which is a trivial 1-MCA, satisfying Equation (15). Furthermore, since P passes through each vertex exactly once, it satisfies our FK-once constraint in Equation (16). Lastly, its cost in Equation (14) is exactly $|V| - 1$ (with $|V| - 1$ unit-weight edges in the path P), which is guaranteed to be lower than any k-MCA with $k > 1$ (whose penalty cost alone is $|V|$). All of these above guarantee that P is an optimal solution to the k-MCA-CC we constructed on G'.

Assume for a moment that we can solve k-MCA-CC efficiently in polynomial time, then by this construction, we can solve any instance of Hamilton path using the reduction above, also in polynomial time, which contradicts with existing complexity result of Hamilton path [28]. We can thus conclude that k-MCA-CC is also NP-hard.

We now show the inapproximability of k-MCA-CC using a reduction from non-metric min-TSP [21]. Recall that the non-metric min-TSP on a graph $G = (V, E)$ finds the min-cost cycle that visits each vertex exactly once, and returns to the origin, where cost is defined as the sum of edge-weights (which in the non-metric version of min-TSP, can be arbitrary numbers).

Recall that the Hamilton cycle problem (a special version of min-TSP) can be reduced to Hamilon path, by adding an artificial source and sink vertex s and t to the input graph, which is connected to a vertex v and its cleaved copy v' with the same neighborhood as v [28]. We perform a similar construction from non-metric TSP (cycle) to non-metric min-cost Hamilton path, using the same artificial source and sink vertex s and t, together with a cleaved v'. For any instance of the non-metric min-TSP problem, we can thus construct a min-cost Hamilton path problem. Because the edges $(s, v), (t, v')$ are constructed to have 0 cost, the reduction is value preserving as an optimal solution to min-TSP is also the optimal min-cost Hamilton-path (modulo s and t), and the two solutions have the same objective function values. Furthermore, using the same reduction from Hamilton path to k-MCA-CC above, we can

then reduce an instance of non-metric min-TSP first to min-cost Hamilton path, and then to k-MCA-CC, all with the same objective values for optimal solutions.

Given that the non-metric min-TSP is EXP-APX-complete [21], we know k-MCA-CC is also EXP-APX-complete (for otherwise given any instance of non-metric min-TSP, we can use the reduction above to solve the corresponding k-MCA-CC efficiently, and thus non-metric min-TSP, contradicting with the complexity result of non-metric min-TSP). □

E PROOF OF THEOREM 4

PROOF. We show that Algorithm 3 solves k-MCA-CC optimally. If the optimal solution to k-MCA, J, does not violate FK-once constraint (Equation (16), then in Line 2 we know J must be a feasible solution to k-MCA-CC on the same graph. Furthermore, because the feasible region of k-MCA is strictly no smaller than that of k-MCA-CC, we know if J is an optimal solution to k-MCA and it is feasible for k-MCA-CC, it must also be an optimal solution to k-MCA-CC. Thus we return J in Line 3 and we are done.

If instead, the optimal solution to k-MCA, J, does violate FK-once constraint in Line 2, because there are at least one edge set

$C_s = \{e_{sj}, e_{sk}, \ldots\} \subseteq J$ that causes violations, where all edges in C_s point from the same vertex with the same column-index s. We partition C_s into $|C_s|$ number of subsets $C_s^1, C_s^2, \ldots, C_s^{|C_s|}$, each with exactly one edge from C_s, and then construct $|C_s|$ number of k-MCA-CC problem instances, each with a new graph $G_i = (V, E_i)$, where $V_i = V, E_i = E \backslash C_s \cup C_s^i$, which we then recurse and solve the k-MCA-CC on each graph G_i in Line 11. To show that $J^* = \mathrm{argmin}_{J_i} c(J_i)$ is the optimal solution to the original k-MCA-cc problem on G, we need to show that the optimal solution to the original k-MCA-cc problem on G, J^+, is still not pruned when we split edges in C_s and create $|C_s|$ number of smaller problems with G_i. In order to see this, notice that the optimal solution to the original k-MCA-cc problem on G, J^+, will have at most one edge in C_s (otherwise it violates FK-once constraint and would not have been a feasible solution). If J^+ has no edge in C_s, then it has not been pruned away when we partition C_s and reduce the solution space, as it is still a feasible solution in each G_i. Alternatively, J^+ has one edge in C_s, and let it be the edge in C_s^l, then all the edges in J^+ is still in the graph G_l (which only pruned edges in $C_s \backslash C_s^l$). As such, J^+ is still a feasible solution in G_l, and will be returned in step Line 12. □