Query Batching Optimization in Database Systems

Mehrad Eslami[a], Vahid Mahmoodian[b], Iman Dayarian[c], Hadi Charkhgard[b,*], Yicheng Tu[a]

[a] *Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA*
[b] *Department of Industrial and Management Systems Engineering, University of South Florida, FL 33620, USA*
[c] *Culverhouse College of Business, The University of Alabama, Tuscaloosa, AL 35487, USA*

Abstract

Techniques based on sharing data and computation among queries have been an active research topic in database systems. While work in this area developed algorithms and systems that are shown to be effective, there is a lack of rigorous modeling and theoretical study for query processing and optimization. Query batching in database systems has strong resemblance to order batching in the warehousing context. While the latter is a well-studied problem, the literature on optimization techniques for query batching problem is quite scarce/nonexistent. In this study, we develop a Mixed Binary Quadratic Program (MBQP) to optimize query-batching in a database system. This model uses the coefficients of a linear regression on the query retrieval time, trained by a large set of randomly generated query sets. We also propose two heuristics, the so-called restricted-cardinality search methods I and II, for solving the proposed MBQP. To demonstrate the effectiveness of our proposed techniques, we conduct a comprehensive computational study over randomly generated instances of three well-known database benchmarks. The computational results show that when the proposed MBQP is solved using the designed heuristics, an improvement of up to 61.8% in retrieving time is achieved.

Keywords: batching problem, database systems, mixed binary quadratic programming, linear regression, restricted-cardinality search method

1. Introduction

Despite being barely 50 years old, database research has had a profound impact on the economy and society, creating an industry sector valued between US$37-US$50 billion annually [16]. The history of database research over the past 40 years has demonstrated a tremendous productivity that has led to the database system becoming arguably the most important development in the field of software engineering. The database system software is the foundation of today's Information Technology (IT) infrastructure and has fundamentally changed the way many organizations operate. In particular, the developments in this technology over the last few years have produced systems that are more powerful and more intuitive to use [29, 30]. This development has resulted in increasing availability of database systems for a wider variety of users.

Traditional database management systems process user-issued tasks (called *queries*) one by one, often by the order they arrive in the system. However, as a result of ongoing increases in

*Corresponding author
Email address: hcharkhgard@usf.edu (Hadi Charkhgard)

both the number of users/requests and data size, as well as improvement in computer hardware, the last fifteen years have seen a shift from *one-query-at-a-time* approaches towards shared work systems where queries are executed in batches. A batch of query includes one or more queries on the same database that are executed together. Typically, there is a high likelihood for the queries on a database to share computing resources such as IO (reading/writing data from/to disk), CPU (processing data by processor), memory, etc. Therefore, executing queries as batches could introduce savings in terms of computing resource usage and processing time. To process a batch of queries, two main questions must be answered:

1. How should a given set of queries be "optimally" partitioned into one or multiple batches of possibly different sizes? and,

2. How should a given single batch of queries be "optimally" processed?

Most previous studies have focused on the second question and have presented different methods to process a batch of queries by efficiently retrieving required data by the queries of a batch from a database. The proposed methods are different ways of sharing the computation resources or taking advantage of query overlaps [26]. As a result of that stream of research, many single-batch processing databases including PsiDB [6], SharedDB [8], Datapath [1], MQO [25], Qpipe [10] have been developed. As for the query batching optimization (the first question above), however, even though it has been stated in many of these articles as a means to improve the performance and a line of potential future study, to the best of our knowledge, no prior article has thoroughly investigated that field.

The main goal of the query batching problem is to partition a set of queries into different batches in order to minimize the total query processing time. In that sense, the query batching problem has a close resemblance to the order batching problem [31, 32]. The order batching problem is an optimization problem, which involves the operation of retrieval of goods in a warehouse [21, 23]. Specifically, the order batching consists of grouping a set of orders together, forming a batch, which is then assigned to a picker whose job is to retrieve all the orders within the same batch on a single tour through the warehouse. A common objective of order batching problem is to reduce picker travel time. In order batching problem, the retrieving time of a given batch of orders can be pre-calculated by solving a traveling salesman problem [18]. In the query batching problem, however, the processing time of a given batch is unknown until it is actually processed/executed. This is because of several reasons including (1) the size of the data that will be retrieved to process a query batch is unknown in advance; (2) defining the similarities between queries is not a trivial task; and (3) the processing time depends highly on the methods used for processing the batch.

As mentioned above, the processing time of queries depends on the database that is used. In this study, we mainly focus on the so-called relational database management systems (RDBMS). Such a commonly used database system software stores data in the form of tables, using columns and rows and follows the database normalized structure introduced by Codd [2]. Most of the well-known industrial database systems such as Microsoft SQL Server, Oracle, MySql, Postgres are for relational databases. The single-batch processing databases that have been mentioned before, are a type of RDBMS. Although the approach that we develop for generating the batches is independent from the methods used for processing each batch, our focus is mainly on PsiDB for processing each batch in this study (because its implementation/code was available to us). PsiDB is a single-batch processing database system, which is newly developed and has shown to perform better than

retrieving queries one by one, by a factor of almost 30x on workloads with large number of queries [6].

Our main contributions in this paper can be summarized as follows.

1. An effective query batching requires an understanding of the processing time of each potential batch of queries, before that batch is formed and executed. We therefore, develop a quadratic function based on the resulting coefficients of a linear regression model to predict the processing time of a given batch using PsiDB.

2. Building upon our batch processing time predictor, we develop a Mixed Binary Quadratic Program (MBQP) to efficiently partition any set of queries into batches (of possibly different sizes). We also propose an effective symmetry breaking technique to strengthen the proposed MBQP.

3. Since the main goal of the proposed algorithm is to minimize the query processing time, the solution time of the MBQP becomes critical. We first prove that the proposed MBQP is in fact an NP-hard problem. Consequently, one cannot afford to solve the MBQP for large-scale problems using exact approaches, as it could be too time demanding. Therefore, we develop two heuristics, the so-called Restricted-Cardinality Search Methods I and II (RCSA-I and RCSA-II), to generate high-quality solutions for the proposed MBQP in a relatively short amount of time.

4. We conduct a comprehensive computational study on three well-known benchmark database systems: TPC-H [28], TPC-DS [**?**], and Join-Order Benchmark (JOB) [15]. We consider instances of TPC-H with 5 tables and up to 10 GB of data, instances of TPC-DS with 10 tables and up to 5 GB of data, and instances of JOB with 23 tables and up to 4 GB of data. We generate a total of 600 random instances based on these database systems, each containing between 32 to 4096 queries. The computational results show that the proposed heuristics can generate optimal solutions to the MBQP for the instances for which checking the optimality was possible (instances with up to 512 queries). More importantly, we show that the total processing time of the queries by batching generated by the MBQP is up to 61.8% smaller than using PsiDB directly, i.e., processing all queries as a single batch. Moreover, we show that the proposed quadratic time prediction function has a high accuracy, i.e., R-squared value between 0.86 and 0.98.

The rest of this paper is organized as follows. In Section 2, we provide preliminaries on database systems. In Section 3, we describe the problem in hand in details. In Section 4, a detailed description of the proposed solution approach is given. In Section 5, we conduct a comprehensive computational study. Finally, in Section 6 some concluding remarks are given.

2. Preliminaries

In this section, we provide a high-level description of a well-known type of database systems, Relational Database Management System (RDBMS), and its main components [2, 4, 7]. Due to the significant number of applications of this class of database systems, it is the main focus of this research. One of the critical components of a RDBMS is the so-called *query optimizer*, which has raised a lot of research questions in the last four decades. In this section, we briefly review

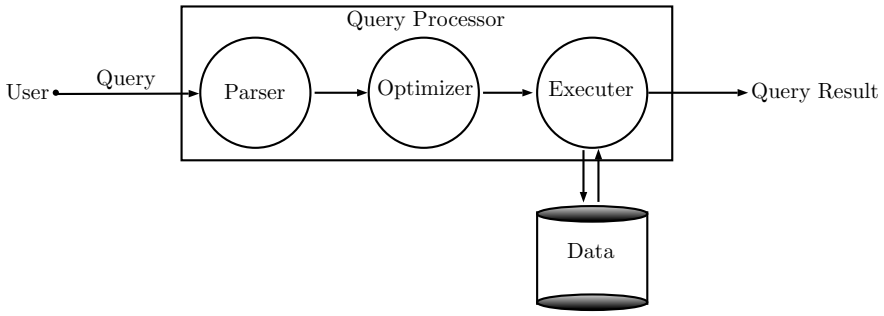


Figure 1: A relational database management system for one-query-at-a-time processing

the existing body of literature on query optimizer and explain a research gap, which is the main motivation of this study. In addition to the traditional RDBMS, we also cover a few new database systems based on RDBMS, developed for specific needs such as single-batch processing database systems.

2.1. A Relational Database Management System

Since in this paper we mainly focus on relational databases, it is essential to have fundamental knowledge about databases. Hence, we survey a few key components of relational databases in this section. The system software that handles the database functionality is called a Database Management System (DBMS). An RDBMS is a type of DBMS which follows certain rules designed by [2] using the relational model. In the following, we explain the structure of data in relational databases and the methods of retrieving data efficiently from such a database. Figure 1 shows the main components of an RDBMS. We briefly explain each component in this section.

Data Structure: Data values in an RDBMS are stored in the form of *tables*. A table is a fixed data structure that consists of *columns* (also called *attributes*) and *rows* (Tuples). A table has a specified number of columns, but can have any number of rows. Each row is a record stored in the database and each column is a type of data whose domain (e.g., data type, length, range of value) is pre-described. In designing a database table, certain attribute(s) of a table are set to be unique identifiers of the entire row, and such attribute(s) is called a *key*. There can be multiple keys in a table, e.g., Social Security Number (SSN) and Student ID are both keys in a University Student table as they both can be viewed as IDs of students (rows). Typically there are two kinds of key: *Primary key*, which is chosen among all keys in a table by the database designer, and *foreign keys*, referring to another table's primary key. The computations in such systems are described by an algebraic system named *Relational Algebra* (RA). Among the RA operators, concatenating rows from different tables is called *join*.

Queries: A query is a language expression that describes data to be retrieved from a database based on specific criteria. All the queries in an RDBMS are in Structured Query Language (SQL) form. The SQL language has changed through the time and deviated in several ways from its theoretical foundation. Our focus in this paper is on standard query expression of SQL. Such expressions are done in a *declarative* manner, which focuses on descriptions of the query results. A simple SQL statement (i.e. `SELECT` $a_1, ..., a_w$ `FROM` $t_1, ..., t_m$ `WHERE` $c_1, ..., c_f$) has the following parts:

- `SELECT`: Specifies the columns in the result

- `FROM`: The table(s) which data is stored in
- `WHERE`: A set of conditions which specifies criteria for the retrieved rows

The scope of this study is on the queries of the form `SELECT..FROM..WHERE` (SFW). Such queries are known to be one-to-one mappings to core operators in relational algebra, which is the query language used to described actual computations required to process the queries.

SQL Parser: SQL parser handles two tasks: Syntax validations and query transformation. Any query that is being executed, needs to be translated to the relational algebraic statement first. We have the same main components in the relational algebra:

- Projection: The set of columns/attributes written as $\Pi_{a_1,...,a_w}$;
- Join: The tables needed to execute the queries;
- Selection: The specific conditions on the attributes $\sigma_{c_1,...,c_f}$.

A SFW query in SQL language, translates to $\Pi_{a_1,...,a_w}\sigma_{c_1,...,c_f}(t_1 \times ... \times t_m)$ in algebraic form. After this transformation, the query is ready to be optimized and then executed.

Query Optimizer: The query optimizer is a main component in terms of database performance enhancement [29]. The query optimizer attempts to determine the most efficient way to execute a query by creating multiple execution plans and choose the optimal one. This happens right after parsing the query and before accessing the data. A query plan determines the query execution plan and how to access the data. Two different plans for the same query would have the same results but the execution time and resource consumption for each of them can be different.

SQL Executor: Given a query execution plan, the query processor fetches the data in the tables and generates the results. There are many factors that play major roles in a query execution such as buffering or sequential reading, which are outside the scope of this paper.

2.2. A Research Gap

As mentioned above, query optimizer plays a key role in any database system. Query optimizer techniques have been an active research topic in the last four decades. Up to this day, there are three main lines/streams of research:

1. How to efficiently process a given single query and retrieve the results?

   In traditional database systems, each query is expected to be executed independently [3, 17, 20, 22]. Figure 1 shows all the components of an RDBMS for executing a single query. A request from the user translates into a query, and the query will go through the parser, optimizer and executor to get the data from the storage and send the results to the user [5, 7]. The underlying idea of this line of research is to compute optimal execution plans for a single query. As a result of this line of research, several query optimizer techniques such as indexing and sequential scanning are developed [12, 13, 14, 24].

2. How to efficiently process a given set of queries as a single batch?

   The landscape of data management systems has changed dramatically over the last decade, and traditional RDBMS are frequently criticized for not meeting requirements of modern

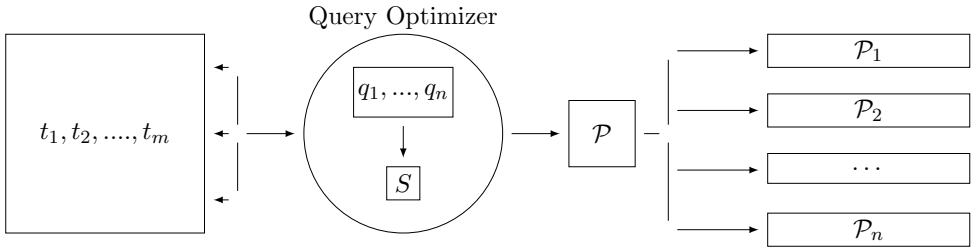


Figure 2: A single batch-processing, query optimizer model. Here $t_1, t_2, \ldots, t_m$ are Tables, $q_1, \ldots, q_n$ are the queries and $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ are output data for each query

data-intensive applications. Today's database systems often need to serve a multitude of user queries in a short period of time. For example, Alibaba, a leading Chinese online retailer, needs to process a workload at the rate of 10,000 query a second. Under such high demand, the one-query-at-a-time model falls short in meeting performance requirements because it can easily lead to resource contentions. On the other hand, the database community observed that queries in a workload may have a significant overlap in the data access paths and intermediate query results. Plausible efforts to harness computation and data access sharing among processing multiple queries have been reported [1, 6, 8, 10, 25].

In light of the above, the database community developed several database systems that combine a set of queries into a single batch and attempts to efficiently process the batch. Examples include: relational database management systems for batch processing i.e., PsiDB and SharedDB designed for workloads with large number of users. These databases are capable of handling a set of queries instead of one single query. The idea of this line of research is demonstrated in Figure 2. The query optimizer in Figure 2 gets a set of queries $q_1, \ldots, q_n$ and combines them into one query S so it can be processed by the database. After executing the combined query, $\mathcal{P}$ is the table containing all information required by the set of queries, which in the next step is distributed among them.

3. How to efficiently partition a given set of queries into batches and process each batch individually?

   While there are many studies for the first and second lines of research, there are no studies for this line of research (to the best of our knowledge). Therefore, the goal of our research is to study whether partitioning the queries into batches can improve the batch-processing database systems performance.

   The idea of such a database system is demonstrated in Figure 3. In this database system, a set of queries $q_1, \ldots, q_n$ are partitioned into multiple batches $S_1, \ldots, S_v$. Associated with each batch S_i, there is a result $\mathcal{P}_i$, leading into a set of results: $\mathcal{P}_1, \ldots, \mathcal{P}_v$. Each $\mathcal{P}_i$ is then distributed among the queries in batch S_i.

3. Problem Description

Batching is one of the main functions of modern databases due to how directly it affects the performance of data retrieval. On a PsiDB, the performance is impacted by the way we batch a set

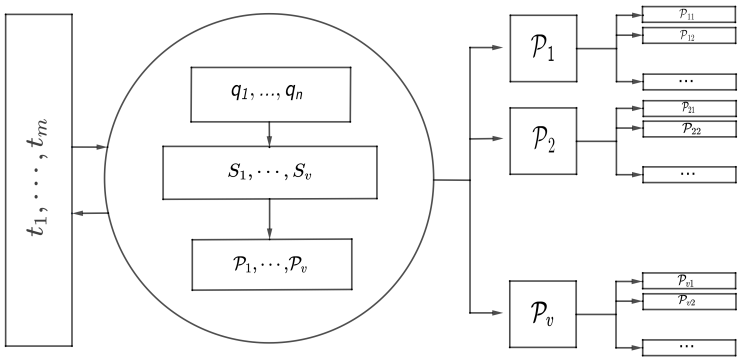


Figure 3: A batch-processing model with partitioned query. Here $t_1, t_2, \ldots, t_m$ are Tables, $q_1, \ldots, q_n$ are the queries, $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_v$ are batches of queries and $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_v$ are output data for each batch

of queries because it impacts the required run time to retrieve the data. In this research, the main goal of the query batching problem (QBP) is to find the best way of partitioning a given set of queries into batches in order to minimize the total processing time. For example, let $Q = \{q_1, q_2, q_3\}$ be a set of queries that needs to be processed. In this case, Q can be processed in 5 different ways by partitioning it into 1, 2, or 3 batches as follows: $\{(q_1), (q_2), (q_3)\}$, $\{(q_1, q_2), (q_3)\}$, $\{(q_1, q_3), (q_2)\}$, $\{(q_1), (q_2, q_3)\}$, and $\{(q_1, q_2, q_3)\}$. The total processing time of each solution of the query batching problem may vary due to the correlation between the queries. The correlation between queries can be established by comparing the data that each query is attempting to retrieve. For example, if q_1, q_2, and q_3 request similar data from similar tables, it will probably be faster to process them as a single batch.

Overall, in order to solve the query batching problem, it is important to know how long the processing time of any potential batch would be. However, computing such a processing time is not a trivial task. This is because to process a batch, a database system typically takes two steps and each one requires some computational efforts [6, 9]. In the first step, it combines/converts all the queries in the batch into a single global query containing all the attributes and records of each query in the batch. The database system will then process the global query and retrieve the entire data requested by the global query. It is also worth mentioning that processing the global query in an efficient manner is not a trivial task by itself because it requires operations such as joining tables and filtering. More importantly, the amount of data retrieved as a result of processing the global query is unknown in advance. In the second step, some additional computational efforts should be made for searching the retrieved data (that its size is unknown in advance) and distributing it among queries in the batch. Due to the existence of unknown parameters in these two steps, computing the exact total processing time of a given batch (in advance) is a challenging task that depends highly on the single-batch-processing method used.

In light of the above, if one can generate a function for predicting the processing time of any possible batch, then a mathematical program for obtaining an optimal solution of the QBP can be developed. Specifically, let $Q := \{q_1, \ldots, q_n\}$ be the set of queries. Observe that since the total number of queries is n, the maximum number of possible batches would be n, i.e., each batch will contain exactly one query. We denote batch $j \in \{1, \ldots, n\}$ by $S_j \subseteq Q$ and its predicted processing

time by $P(S_j)$. By using these notations, the QBP can be stated as follows:

$$\min_{S_1,\ldots,S_n \subseteq Q} \Big\{ \sum_{j=1}^{n} P(S_j) : \cup_{i=1}^{n} S_j = Q \text{ and } S_i \cap S_j = \emptyset \;\; \forall i, j \in \{1,\ldots,n\}, i \neq j \Big\}.$$

Note that due to the significant variability of processing times in practice, it would be hard to predict the processing times accurately. So, instead of predicting processing times, we predict the value of an indicator of processing times. In other words, in the remaining of this paper, $P(S_j)$ basically represents the predicted value for the processing-time indicator of batch S_j. So, for the convenience, whenever we say the 'predicted processing time', we mean the 'predicted value for the processing-time indicator' in the remaining of this paper.

In the next section, we will explain that the processing-time indicator of batch S_j is defined as the square of $\log_2$-transformation of the processing time of the batch S_j in this paper (see Eq. (3) for details). Hence, $P(S_j)$ attempts to predict the value of the indicator for the batch S_j. We observe that the indicator consists of a 'square' function and a 'log-transformation' function. We use the square function because of two main reasons including: (1) linear functions cannot capture the impact of the number of queries allocated to batches when being used in our developed optimization model for the QBP, see Section 4.2 for details; and (2) square function can be handled effectively by (commercial) mixed integer programming solvers such as Gurobi or CPLEX since it is convex. Also, we use the log-transformation because that is a common heuristic way to reduce the variability of data. Through this article, we use base 2 for the log-transformation function because it is a natural choice given that computers use binary encoded systems. However, during the course of this study, we observed that our proposed method is not sensitive to the base value.

Finally, we assume that queries are not empty. In other words, we assume that each query in Q should involve (i.e., search) at least one of the tables in the database system under consideration. This assumption is needed only for the purpose of defining valid instances when proving that the QBP is NP-hard (see the Appendix).

4. The Proposed Approach

In this section, we explain our proposed approach for solving the query batching optimization problem. Our approach contains three phases as follows.

1. Developing the batch processing time prediction function
2. Developing an optimization model to effectively partition the queries into batches
3. Developing effective solution methods to solve the optimization model

In the remaining of this section, we will provide further details about each phase.

4.1. The Batch Processing Time Prediction Function

In this section, we develop a function $P(S')$ that serves to predict batch processing time for any arbitrary query batch S' of a given database. An ideal function $P(S')$ has the two following desired characteristics: (1) The function has high accuracy; and (2) The function is simple as the complexity of the optimization model for the QBP and its solution approaches highly depend on it. The use of the QBP is justified only if the sum of the solution time of the QBP and the processing

time of the batches prescribed by the QBP is dominated by the processing time of the same set of queries when the QBP is not employed. Accordingly, if one cannot solve the QBP quickly then the benefit of batching may not be significant.

As mentioned in Section 3, for a given query batch S', its predicted (indicator of) processing time, $P(S')$, depends on the size of the data that must be retrieved for processing batch S'. However, such information is not known in advance and becomes available only while executing the batch. Therefore, in our approach we benefit from some other attributes (or features) of S' that are either available in advance or easy to compute to predict the processing time of batch S'.

Let $T := \{1, \ldots, m\}$ denote the index set of all tables in the database (where m is the total number of tables). To predict the processing time of a batch S', we introduce $|T^{S'}| + 1$ attributes associated with batch S', where $T^{S'} \subseteq T$ denotes the index set of all tables required by the queries in batch S'. The first $|T^{S'}|$ attributes are indicators of the size of tables required for batch S'. Specifically, we introduce the attribute $\log_2(s_t)$ for each $t \in T^{S'}$ with s_t being the number of records in table t. Note that the size of tables in a database can be significantly different and hence the proposed $\log_2$-transformation can be helpful in normalizing the size of the tables in such cases. We also introduce one additional attribute, indicating the number of queries in batch S', denoted by n'. Using the proposed attributes, we construct a function that can predict the processing time of any given batch. This function takes the following form.

$$P(S') := \big[\beta_0 + \beta_1 n' + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y'_t\big]^2, \tag{1}$$

where $y'_t \in \{0, 1\}$ is a parameter indicating whether table t is required to process queries in batch S'. Also, $\beta_0, \beta_1, \ldots, \beta_{|T|+1} \in \mathbb{R}$ are the coefficients that should be estimated. In Section 5, we will numerically show that the proposed function has high prediction accuracy on the three different database benchmarks that we consider. This implies that the proposed function addresses the first required characteristic, mentioned at the beginning of this section. Additionally, we observe that the proposed $P(S')$ is a quadratic function of n' and $y'_1, \ldots, y'_m$. Despite being a quadratic function, $P(S')$ can still be handled easily by commercial solvers such as CPLEX and Gurobi when used in the optimization model that we develop for the QBP, which satisfies the second desired characteristic mentioned above. Note that we do not use a linear function of the form,

$$\beta_0 + \beta_1 n' + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y'_t \tag{2}$$

instead of Eq. (1), since Eq. (2) has a significant weakness when being employed in the QBP. In Section 4.2, we will explain in detail why such a linear function is not suitable.

We estimate coefficients $\beta_0, \beta_1, \ldots, \beta_{|T|}$ through a training procedure over a large set of randomly generated batches of queries. In Section 5, we discuss in details how a random batch is generated. For each batch S', one can compute the value of each attribute and then actually process the batch to compute its real processing time, denoted by RealTime. The goal is to estimate the coefficients of the proposed processing time prediction function $P(S')$ such that,

$$P(S') \approx [\log_2(\text{RealTime})]^2. \tag{3}$$

This implies that,

$$\sqrt{P(S')} = \beta_0 + \beta_1 n' + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y'_t \approx \log_2(\text{RealTime}).$$

This itself implies that in order to estimate the coefficients of the proposed function, one can simply fit a linear regression function. Specifically, $\log_2(\text{RealTime})$ can serve as the dependent variable and the proposed attributes can serve as independent variables. In this study, we employed the package *linear_model* from *sklearn* in Python to compute the coefficients based on Ridge linear regression [11].

4.2. The Optimization Model

In this section, we develop a MBQP to solve the QBP using the proposed processing time prediction function. In the remaining of this section, it is assumed that coefficients $\beta_0, \beta_1, \ldots, \beta_{m+1}$ are already estimated and are available as parameters. However, the assignment of the queries to batches, and consequently the number of queries in each batch and the set of tables required for processing each batch are unknown and are to be determined by the optimization model. Let $Q_t \subseteq \{1, \ldots, n\}$ be the set of queries that requires table $t \in T$. Note that, by assumptions, each query requires at least one table. Let k be a user-defined parameter denoting the maximum number of possible batches that are allowed to be formed. Note that in the worst case the number of batches equals the number of queries; one batch per query. Hence, one can set k to n if no information about the value of k is available. Let x_{ij} be a binary decision variable that equals 1 if query $i \in \{1, \ldots, n\}$ is assigned to batch $j \in \{1, \ldots, k\}$, and 0 otherwise. Also, let y_{jt} be a binary decision variable indicating whether table $t \in T$ is used by at least one query in batch $j \in \{1, \ldots, k\}$. For each batch $j \in \{1, \ldots, k\}$, we introduce the continuous variable n_j for indicating the number of queries in batch j. As an aside, although n_j is defined as continuous decision variable, it will naturally take integer values in the proposed formulation for each batch $j \in \{1, \ldots, k\}$. So, one can define n_j as an integer variable in practice for each batch $j \in \{1, \ldots, k\}$. Finally, for each batch $j \in \{1, \ldots, n\}$, we introduce the binary decision variable z_j that takes the value of 1 if at least one query is assigned to batch j, i.e., batch j is not empty. Using these notations, the query batching problem can be formulated as the following MBQP,

$$\min \sum_{j=1}^{k} \left[\beta_0 z_j + \beta_1 n_j + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y_{jt}\right]^2 \tag{4}$$

$$\text{s.t.} \sum_{j=1}^{k} x_{ij} = 1 \qquad \forall i \in \{1, 2, \ldots, n\} \tag{5}$$

$$n_j = \sum_{i=1}^{n} x_{ij} \qquad \forall j \in \{1, 2, \ldots, k\} \tag{6}$$

$$x_{ij} \leq y_{jt} \qquad \forall t \in T,\ \forall i \in Q_t,\ \forall j \in \{1, 2, \ldots, k\} \tag{7}$$

$$y_{jt} \leq z_j \qquad \forall t \in T,\ \forall j \in \{1, 2, \ldots, k\} \tag{8}$$

$$y_{jt} \leq \sum_{i \in Q_t} x_{ij} \qquad \forall j \in \{1, 2, \ldots, k\},\ \forall t \in T \tag{9}$$

$$z_j \leq n_j \qquad \forall j \in \{1, 2, \ldots, k\} \tag{10}$$

$$n_j \geq 0 \qquad \forall j \in \{1, 2, \ldots, k\} \tag{11}$$

$$x_{ij}, y_{jt}, z_j \in \{0, 1\} \qquad \forall j \in \{1, 2, \ldots, k\},\ \forall i \in \{1, \ldots, n\},\ \forall t \in T, \tag{12}$$

where the objective function (4) measures the total predicted processing time of non-empty batches. Constraint (5) ensures that each query is assigned to exactly one batch. Constraint (6) computes

the value of n_j for any batch $j \in \{1, \ldots, k\}$. Constraint (7) ensures that if table $t \in T$ is required by at least one query in batch j, then $y_{jt} = 1$. Constraint (8) guarantees that if batch j needs some tables to be processed, i.e., $y_{jt} = 1$ for some $t \in T$, then $z_j = 1$ because the batch j must be non-empty in that case. Constraint (9) ensures that if for any given batch $j \in \{1, \ldots, k\}$ table $t \in T$ is not needed, i.e., $\sum_{i \in Q_t} x_{ij} = 0$, then we must have that $y_{jt} = 0$. Finally, Constraint (10) guarantees that if no query is assigned to a given batch $j \in \{1, \ldots, k\}$, i.e., $n_j = 0$, then we must have that $z_j = 0$. We note that Constraints (9) and (10) are not necessary for solving the QBP if $\beta_0, \beta_1, \ldots, \beta_{m+1} \geq 0$. However, for the purpose of studying the computational complexity of the QBP (see Section 4.2.1) we include them in the formulation. These two additional inequalities combined with the assumption that each query should involve at least one table guarantee that the (predicted) processing time of each batch is computed properly even if there exists $l \in \{0, 1, \ldots, m+1\}$ such that $\beta_l < 0$. Moreover, these two constraints ensure that if a batch is empty and/or do not need to explore any table then its associated predicted processing time is zero. Also, if a batch does not need some of the tables then those will not impact the processing time of that batch. In the remaining of this paper, we denote the formulation (4)-(12) by MBQP(k) where k is the input parameter set by users.

Observe that if $k = n$ then the objective function of the proposed formulation simply captures $\sum_{j=1}^{n} P(S_j)$ where $P(S_j)$ is the quadratic function obtained in Section 4.1. So, it is natural to ask why cannot $P(S_j)$ be a linear function, i.e., Eq. (2)? Note that the objective of the QBP is to minimize the total predicted processing time of a set of queries by regrouping them into batches. A linear function may have high accuracy for predicting the processing time of each batch individually but when used in the optimization framework for minimizing the total predicted processing time, it will perform poorly. Specifically, if one uses a linear processing time prediction function, the objective function of the proposed formulation (when $k = n$) will change to

$$\sum_{j=1}^{n} \beta_0 z_j + \beta_1 n_j + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y_{jt} = \beta_1 n + \sum_{j=1}^{n} \beta_0 z_j + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y_{jt}.$$

This implies that the number of queries in the batches will not play any role in the optimization process because $\beta_1 n$ is a constant. In that case, by assuming that $\beta_0, \beta_1, \ldots, \beta_{m+1} \geq 0$ (which is likely to be the case in practice), the optimal solution for the QBP is to create only one batch containing all queries. Therefore, such linear functions are not suitable for showing the effectiveness of batching. Hence, the main purpose of using the proposed quadratic function is that it allows to capture the importance of the number of queries assigned to different batches. We next present a technique to strengthen the proposed formulation.

The proposed model can be strengthen by reducing the level of existing symmetry in the formulation. As a result of the existing symmetry, one can expect that commercial solvers such as CPLEX and Gurobi struggle to solve the proposed MBQP without applying symmetry breaking techniques [27]. To break the symmetry, we propose to add the following constraints to the model,

$$x_{ij} = 0 \qquad \forall i \in \{1, \ldots, n\} \text{ and } \forall j \in \{i+1, \ldots, k\}. \tag{13}$$

Observe that without using the proposed symmetry breaking constraints, a query $i \in \{1, \ldots, n\}$ can be assigned to any batch $j \in \{1, \ldots, k\}$ in the optimization model. This is in fact one of the main reasons that symmetry exists in our proposed formulation. Hence, our proposed symmetry breaking technique limits this flexibility by forcing the optimization model to assign query $i \in \{1, \ldots, n\}$ to

only a batch $j \in \{1, \ldots, k\}$ with $j \leq i$. This implies that query 1 is only allowed to be assigned to batch 1, query 2 is only allowed to be assigned to either batch 1 or 2, query 3 is only allowed to be assigned to either batch 1 or 2 or 3, and so on. This process obviously does not remove any feasible solution of the QBP and will (partially) break the symmetry in the proposed formulation. Finally, in terms of implementation, instead of explicitly adding the proposed symmetry breaking constraints, it would be computationally better not to generate x_{ij} for all $i \in \{1, \ldots, n\}$ and $j \in \{i+1, \ldots, k\}$ when creating the model. This efficient implementation is used in this study.

4.2.1. Computational Complexity

We now explore the computational complexity of the QBP when its objective function is the proposed quadratic function. We again note that although Constraints (9) and (10) are not necessary for computing optimal solutions for the QBP when $\beta_0, \beta_1, \ldots, \beta_{m+1} \geq 0$, we assume that they are included in the formulation for the purpose of studying the computational complexity of the QBP. This is because for identifying the computational complexity of the QBP, the decision problem of the QBP should be explored. The decision problem of the QBP (when employing the proposed quadratic function), denoted by `QBP`, can be stated as follows: does there exist a solution that can satisfy the following set of constraints,

$$\sum_{j=1}^{k} \left[\beta_0 z_j + \beta_1 n_j + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y_{jt}\right]^2 \leq U$$
$$(5) - (12)$$

where U is a given parameter. The `QBP` is basically a feasibility problem in which the objective function of the proposed MBQP has changed to a constraint. Therefore, in order to make an accurate analysis on the computational complexity of the `QBP`, the objective value of each feasible solution must be captured accurately in the `QBP`. That is the main reason that Constraints (9) and (10) are included in the `QBP`.

Theorem 1. *The query batching problem when employing the proposed processing time prediction function* (1) *is NP-hard.*

Proof. The proof is rather lengthy and is provided in the appendix. □

4.3. Heuristic Solution Methods

Observe that if $k = n$ then the size of the proposed formulation, i.e., MBQP(n), is $O(mn^4)$ because the number of constraints is $O(mn^2)$ and the number of variables is $O(n^2)$. This combined with Theorem 1 implies that there is little hope that the proposed MBQP can be solved quickly enough for practical-sized instances with possibly thousands of queries and multiple tables. In Section 5, we will show that instances with up to 500 queries on a database benchmark with 5 tables and around 10 GB of data can be solved to optimality using commercial solvers within a few hours (on a typical computing node). Obviously, for larger instances, the solution time for the MBQP is expected to increase significantly. As a result, even if optimal or near optimal batching can offer some savings in terms of processing time of a set of queries, these savings can be dominated by long solution time of the MBQP. As a consequence, in this section, we propose two simple heuristic algorithms permitting us to quickly generate high-quality feasible solutions to the proposed MBQP.

We refer to the first algorithm as the Restricted-Cardinality Search Method I (RCSA-I). The underlying idea of RCSA-I is to set the value of k to values smaller than n. Observe that for such values the size of MBQP(k) is significantly smaller than the MBQP(n). Hence, one can expect to solve the MBQP(k) faster than MBQP(n) using commercial solvers when k is smaller than n. To improve the solution time even further, RCSA-I restricts the formulation even more by adding the following set of constraints to the MBQP(k),

$$\sum_{j=1}^{k} y_{jt} \leq 1 \qquad \forall t \in T. \tag{14}$$

Constraint (14) forces all queries requiring a given table t to be assigned to the same batch. The addition of Constraint (14) results in a restricted variant of MBQP(k) that we refer to as the MBQP-I(k) and it plays a key role in RSCA-I. Starting from $k = 1$, RCSA-I iteratively increases k with a stepsize one and solves the corresponding MBQP-I(k) in each iteration considering a time limit. As an aside, we note that the solution for $k = 1$ is trivial and hence MBQP-I(k) does not need to be solved for $k = 1$. If in a given iteration there is an improvement in the obtained objective value of MBQP-I(k) compared to the previous iteration, the algorithm increases k by one and starts a new iteration. Otherwise, the algorithm terminates and returns the best solution found and the corresponding number of batches, k^*. It is worth mentioning that for implementing RCSA-I, three points should be considered. First, the solution of each iteration can be used as a warm-start for the next iteration. Second, the proposed symmetry breaking technique developed for strengthening MBQP(k) can be used for MBQP-I(k). Third, because of Constraint (14), the maximum number of iterations of RCSA-I is m.

We refer to the second algorithm as the Restricted-Cardinality Search Method II (RCSA-II). RCSA-II is a two-phase method and its underlying idea is to improve the solution obtained by RCSA-I further (if possible). Hence, RCSA-II first calls RCSA-I in its first phase. Let k^* be the value of k in the last iteration of RCSA-I. For the second phase, the algorithm initializes k by k^* and then iteratively solves the MBQP(k), instead of MBQP-I(k). Similar to the first phase, the algorithm increases the value of k by one as long as some improvements in the obtained objective value of MBQP(k) is observed (compared to its last iteration). Otherwise, the algorithm terminates and returns the best solution found. It is worth mentioning that the implementation points discussed for RCSA-I should also be considered for the RCSA-II.

5. A Computational Study

In this section, we investigate the performance of our proposed approach on three database benchmarks. For conducing the experiments, we employ SQL Server 2017 and Julia programming Lagrange 0.6.4. Also, to solve optimization models, we use CPLEX 12.9 and we employ PsiDB for processing each batch of queries in database. All the computational experiments are conducted on a workstation with Intel quad-core 3.6GHz i7-7700 processor, 32GB of DDR4-2400 memory, a 256GB SSD system disk, and a 2TB 7200 RPM hard drive for database storage. Also, in this study we impose a time limit on each iteration of RCSA-I and RCSA-II. The proposed time limit is 300 seconds for each iteration. All the data involved are stored on the same hard drive to ensure consistent Input/Output (I/O) rate across different experimental runs. The workstation runs on Windows 10 Enterprise Version 1709.

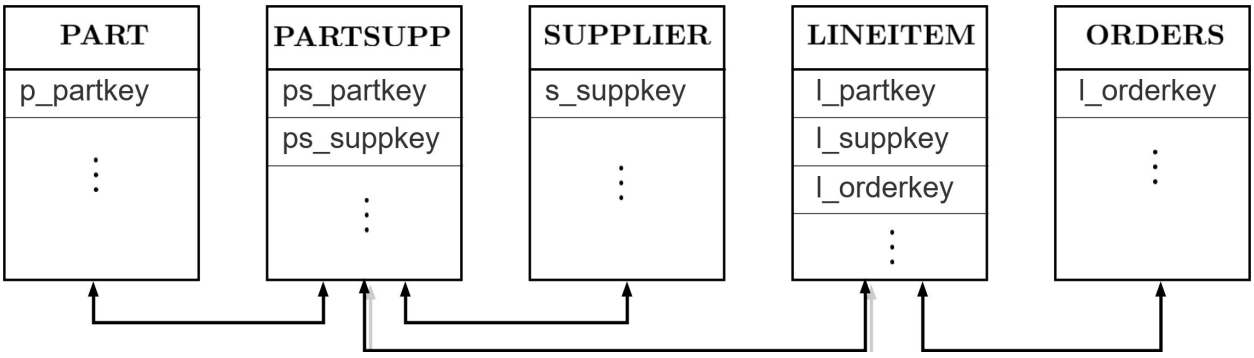


Figure 4: The structure of a TPC-H database

5.1. Database Benchmark TPC-H

In this section, we employ the well-known TPC-H database benchmark for showing the effectiveness of the QBP and our proposed heuristics [19, 28]. TPC-H database represents a customer-supplier system. It contains information of the provided parts by the suppliers and the orders by the customers.

This database benchmark takes a parameter, the so-called Scale Factor (SF), as an input. This parameter indicates the approximate size (in gigabytes) of the random TPC-H database that will be generated. In this section, we consider three different values for the SF, i.e., $\{1, 5, 10\}$. That is, we generate three random TPC-H database replicas with the size of approximately 1, 5, and 10 GB, respectively. The generated databases have 5 tables with established relationships between some of them as shown in Figure 4. The relationships are defined by the so-called *foreign keys* and shown by arrows in Figure 4. These relationships are important because PsiDB uses them when processing a batch of queries. For example, if there is a batch of queries on only tables PART and SUPPLIER, PsiDB has to also use table PARTSUPP (when joining tables) as it contains both foreign keys of tables PART and SUPPLIER. As an aside, we note that since there are only 5 tables, in theory, there should be $2^5 - 1 = 31$ scenarios for a query to involve the tables. However, because of the relationships between tables, only 17 scenarios are feasible. The number of records in each table for different values of SF are reported in Table 1.

Table 1: The number of records in each table of TPC-H for three values of SF

Table Name	**SF=1**	**SF=5**	**SF=10**
LINEITEM	6,001,215	30,006,075	60,012,150
ORDERS	1,500,000	7,500,000	15,000,000
PARTSUPP	800,000	4,000,000	8,000,000
PART	200,000	1,000,000	2,000,000
SUPPLIER	10,000	50,000	100,000

We first attempt to estimate the coefficients of the batch processing time prediction function, i.e., $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{m+1})$. Since each SF basically defines a new database, the coefficients should be estimated for each one independently. Hence, for each SF, we randomly generate 2754 data points, i.e., batches of queries. These data points are divided over 81 classes, each containing 34 batches, based on the number of queries that they involve. Specifically, the number of queries in each class of batches is taken from the set $\{50, 75, 100, 125, \ldots, 2050\}$. To create the batches of each

class, we first generate two random scenarios for showing the percentage of the data that should be retrieved as the result of processing a batch. Each scenario can take a value from the interval $(0\%, 5\%]$. For each scenario, we create 17 batches. For example, suppose that the first scenario for the class 50 is 1% and the second scenario is 5%. So, we create 17 (random) batches that each has 50 queries and requires to return 1% of the total size of the database for processing its queries. Similarly, we create 17 (random) batches that each has 50 queries and requires to return 5% of the total size of the database for processing its queries. The reason that we create 17 random batches for each scenario is that, as mentioned earlier, there are 17 possible ways of utilizing tables in (a batch of) queries for TPC-H databases. So, for each one, we randomly generate one batch of queries such that the amount of data that should be retrieved for it matches its associated scenario. For example, tables PART, PARTSUPP, and SUPPLIER construct one valid combination. So, for the class 50 and the scenario 1% that we mentioned earlier, one random batch will only utilize tables PART, PARTSUPP, and SUPPLIER. In other words, to process (all 50 queries of) that batch, 1% of the total size of the database will be returned from only tables PART, PARTSUPP, and SUPPLIER.

After creating random batches for each SF, we execute them individually and record their associated values for the dependent and independent variables of linear regression (see Section 4.1). We randomly pick 70% of data as the training set and use the remaining 30% as the test set. Overall, the results show that

$$\boldsymbol{\beta} = (1.6032, 0.0023, 0.1165, 1.7646, 1.0794, 2.4355, 0.9743,),$$

$$\boldsymbol{\beta} = (10.8716, 0.0028, 0.1977, 3.0068, 1.9187, 4.0763, 1.6231),$$

and

$$\boldsymbol{\beta} = (17.1205, 0.0028, 0.2480, 3.7526, 2.4400, 5.1369, 2.0199)$$

for SF=1, SF=5, and SF=10, respectively. Table 2 shows the accuracy of the regression model on this database for different values of SF where 'MSE' refers to Mean Squared Error and 'MAPE' refers to Mean Absolute Percentage Error.

Table 2: The statistics of the regression model for different values of SF on TPC-H databases

SF	**Training set**			**Test set**		
	R-squared	**MSE**	**MAPE(%)**	**R-squared**	**MSE**	**MAPE(%)**
1	0.94	0.04	3.44	0.94	0.05	3.46
5	0.95	0.04	2.15	0.95	0.04	2.13
10	0.95	0.09	1.83	0.96	0.04	1.82

In order to show the effectiveness of solving the query batching problem, a total of 120 instances are generated for each SF. An instance includes a specific number of queries on SF 1, 5, or 10 of the TPC-H database. With SF taking three values 1, 5, and 10, a total of 360 instances are generated in this experiment. The instances associated with each SF are regrouped into three classes of small, medium, and large instances. For the small-sized instances (denoted by S), the number of queries is taken form the set $\{32, 64\}$. For the medium-sized instances (denoted by M), the number of queries is taken form the set $\{256, 512\}$. Finally, for the large-sized instances (denoted by L), the number of queries is taken form the set $\{2048, 4096\}$. Therefore, each class of instances contains two sub-classes. For example, for Class S, the first subclass has instances with 32 queries each

and the second subclass has instances with 64 queries each. Within each instance subclass, 20 random instances are generated. We note that since TPC-H simulates a customer-supplier system database in the real world, it comes with some suggested scenarios for generating a set of queries to make sense in practice (see the details in [19, 28]). Hence, when creating an instance, we follow the suggested scenarios. Specifically, to generate each instance, we first randomly select a subset of all 17 valid combinations of employing tables and then generate all randomly queries based on the selected combinations. Table 3 summarizes the performance of RCSA-I and RCSA-II on all 360 instances of this experiment. Numbers reported in this table for each instance subclass are averages over 20 instances.

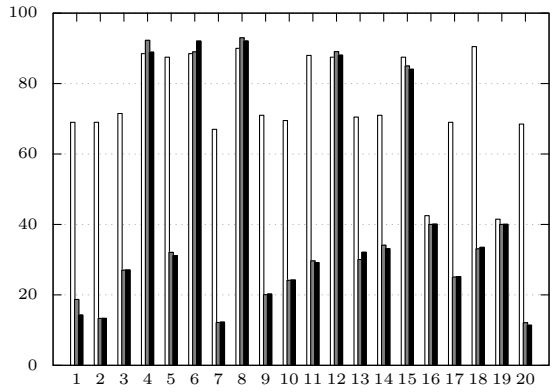
Table 3: Performance of RCSA-I and RCSA-II on TPC-H databases

SF	n	Default	RCSA-I			RCSA-II			Imp (%)	
		Total Time (s)	# of batches	Opt Time (s)	Total Time (s)	# of batches	Opt Time (s)	Total Time (s)	RCSA-I	RCSA-II
1	32	75.0	2.4	0.6	41.4	2.4	0.3	41.3	44.8	45.0
	64	130.9	2.3	0.2	83.1	2.3	0.7	83.1	36.5	36.5
	256	306.5	2.6	0.2	182.6	2.8	1.0	182.1	40.4	40.6
	512	414.6	1.9	0.2	319.9	2.3	2.1	317.8	22.9	23.4
	2048	756.7	2.2	4.3	550.2	2.7	40.1	550.6	27.3	27.2
	4096	964.4	2.1	13.1	745.1	2.6	353.3	761.8	22.7	21.0
5	32	453.5	2.1	0.2	268.1	2.1	0.2	272.3	40.9	40.0
	64	710.2	2.1	0.1	453.7	2.1	0.2	453.7	36.1	36.1
	256	1548.1	2.4	0.4	964.3	2.4	1.1	954.5	37.7	38.3
	512	2335.7	2.0	0.4	1822.2	2.0	2.9	1822.2	22.0	22.0
	2048	3772.3	2.1	1.9	2835.8	2.4	32.3	2873.4	24.8	23.8
	4096	5176.5	2.3	9.4	3751.3	2.6	217.1	3787.3	27.5	26.8
10	32	1120.8	2.1	0.1	702.6	2.1	0.1	703.2	37.3	37.3
	64	1638.4	2.3	0.1	969.5	2.3	0.2	969.5	40.8	40.8
	256	3259.4	2.4	0.2	2057.7	2.4	1.0	2054.2	36.9	37.0
	512	4479.9	2.2	0.4	3192.3	2.2	2.8	3192.3	28.7	28.7
	2048	8165.0	2.3	2.6	5484.4	2.5	51.0	5460.7	32.8	33.1
	4096	10502.2	2.3	10.1	7505.9	2.3	186.4	7507.4	28.5	28.5

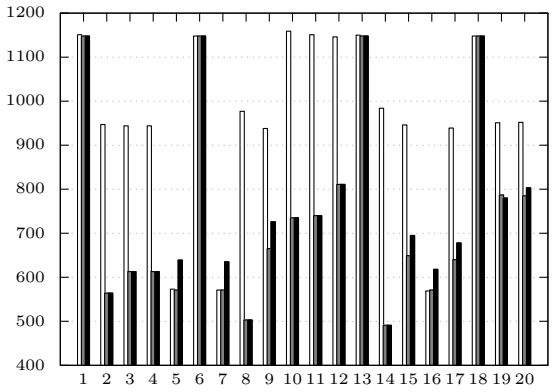
In Table 3, the columns labeled '# of batches' report the number of batches obtained by using a heuristic; the columns labeled 'Opt Time (s)' show the solution time of a heuristic in seconds; the columns labeled 'Total Time (s)' show the total processing time of an instance (in seconds) which include the solution time of a heuristic (if used any) and the processing time of the prescribed batches by the model; the column labeled 'Default' contains the results for the default setting of PsiDB, which is a single batch containing all queries; Finally, the columns labeled 'Imp (%)' report the percentage improvement in the total time obtained by using a heuristic to solve the QBP. As an aside, we note that PsiDB is shown to be faster than processing queries one by one by a factor of around 30 [6]. Consequently, we do not report any results for processing queries individually.

The last two columns of Table 3 reveal an improvement ranging from 21.0% to 45.0% in the query processing time, based on the proposed heuristic methods when compared to the default setting. Another interesting outcome from comparing the results of the two heuristics for large instances is that their total times are close. This implies that the time that RCSA-II's solution saves by a better batching is compromised by the additional time it spends in the optimization phase. For example, the last row of Table 3 shows that the processing time reduction achieved by potentially improved batching of RCSA-II can barely compensate the 176.3 seconds of extra computational time that it needs for optimization compared to RCSA-I.

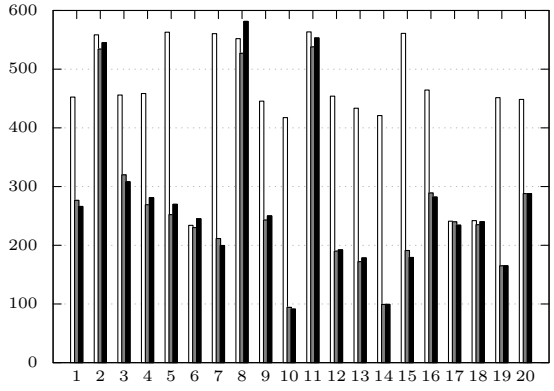
Figure 5 shows the total time of all instances of the subclasses $n = 31$ and $n = 4096$ for different SF values. In the Figure, white bars, gray bars, and black bars represent the total time (in seconds)



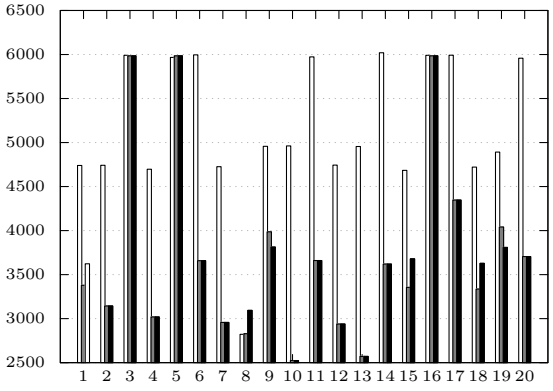
(a) SF = 1, $n = 32$



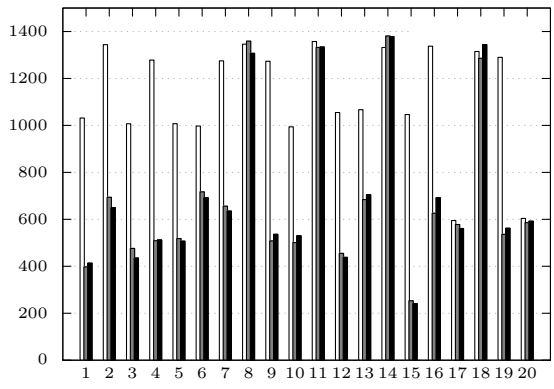
(b) SF = 1, $n = 4096$



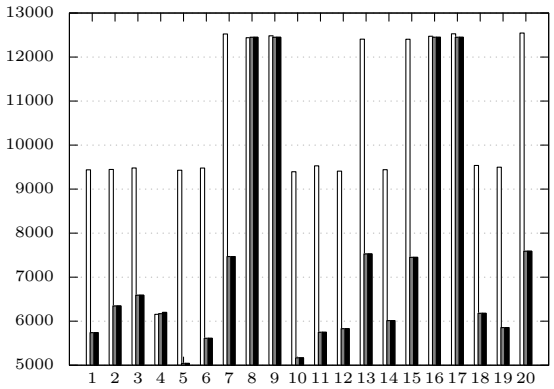
(c) SF = 5, $n = 32$



(d) SF = 5, $n = 4096$



(e) SF = 10, $n = 32$



(f) SF = 10, $n = 4096$

Figure 5: Total time (in seconds) of all 20 instances of the smallest and largest subclasses on TPC-H databases (white bars: default setting, gray bars: RCSA-I, black bars: RCSA-II)

of the default setting, RCSA-I, and RCSA-II, respectively. Observe that batching the queries using the proposed heuristics reduces the total time except for the cases in which final recommendation of the optimization model coincides the default setting (assigning all queries to one single batch).

Table 4 shows the (relative) optimality gap between the objective values reported by the heuristic methods and the global dual bounds reported by CPLEX when solving the exact MBQP formulation, i.e., MBQP(n), on instances with $n \leq 512$ over a two-hour time limit. In this table, the column labelled 'Exact Opt Time (s)' shows the solution time of CPLEX for solving the exact MBQP. Again, numbers reported in this table for each instance subclass are averages over 20 instances. CPLEX was able to solve all instances with $n \leq 256$ and most instances with $n = 512$ to optimality within the imposed time limit. Among the instances with $n = 512$, only two, three, and two of them were not solved to optimality for SF$= 1$, SF $= 5$, and SF $= 10$, respectively. From the table, we observe that RCSA-II was able to generate an optimal solution for all instances that their optimal solutions were known, i.e., solved within 2 hours time limit by CPLEX. Even for those few large instances that CPLEX was not able to prove optimality within 2 hours, the RCSA-II relative optimality gap from the global dual bound is below 5%. This provide evidences that RCSA-II can at least provide near optimal solutions for such instances. Note that the optimality gaps of RCSA-I are similar to the optimality gaps of RCSA-II but slightly worse for instances with $n = 512$. This explains the observation that we made earlier about the similarity of the total time of RCSA-I and RCSA-II for large instances in Table 3.

Table 4: Comparing the quality of the solutions obtained by the proposed heuristics with optimal solutions on the TPC-H

SF	n	Exact Opt Time (s)	RCSA-I gap (%)	RCSA-II gap (%)
1	32	0.4	0.00	0.00
	64	1.2	0.00	0.00
	256	324.1	0.02	0.00
	512	2463.0	2.54	2.45
5	32	0.6	0.00	0.00
	64	0.9	0.00	0.00
	256	309.3	0.00	0.00
	512	2887.8	3.37	3.37
10	32	0.5	0.00	0.00
	64	1.0	0.00	0.00
	256	392.9	0.00	0.00
	512	2143.2	2.14	2.14

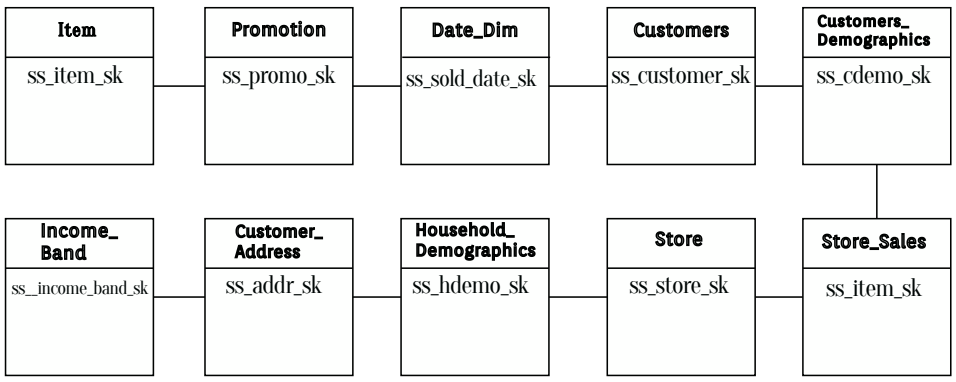


Figure 6: The structure of the TPC-DS database

5.2. Database Benchmark TPC-DS

In this section, we employ another database benchmark, i.e., TPC-DS [**?** 28], that consists of 10 tables and 5GB of data for showing the effectiveness of the QBP and our proposed heuristics. The database describes a retail product supplier. The supporting schema contains vital business information, such as customer, order, and product data. All the tables in this database and their relationships are shown in Figure 6. Note that while there were only 17 valid combinations for involving tables in the TPC-H benchmark, the TPC-DS benchmark involves a significantly larger number of valid combinations. Specifically, the total number of potential combinations is $2^{10} - 1$ but because of the relationships between the tables not all combinations are valid. In this section, we only select 55 valid combinations and employ them for generating instances. These 55 valid combinations are obtained based on the suggested scenarios that come with the TPC-DS benchmark to ensure the generated queries make sense in practice (see the details in [**?**]).

We again start by estimating the coefficients of the batch processing time prediction function, i.e., $\boldsymbol{\beta}$. We randomly generate 4,455 data points, i.e., batches of queries. We generate the data points for the TPC-DS benchmark following a similar procedure that we used for the TPC-H benchmark. Specifically, the data points are divided over 81 classes, each containing 55 batches, based on the number of queries that they involve. The number of queries in each class of batches is again taken from the set $\{50, 75, 100, \ldots, 2050\}$. For each class, we generate a single scenario for showing the percentage of the data that should be retrieved as the result of processing a batch. Each scenario can take a value from the interval $(0\%, 5\%]$. For each scenario, we created 55 batches. Specifically, there are 55 selected valid ways of including tables in (a batch of) queries for the TPC-DS databases. So, for each one, we randomly generate one batch of queries such that the amount of data that should be retrieved for it matches its associated scenario.

After creating random batches, we execute them individually and record their associated values for the dependent and independent variables of linear regression (see Section 4.1). We randomly pick 70% of data as the training set and use the remaining 30% as the test set. Overall, the results show an R-squared of around 0.86 (see Table 5) can be obtained by setting,

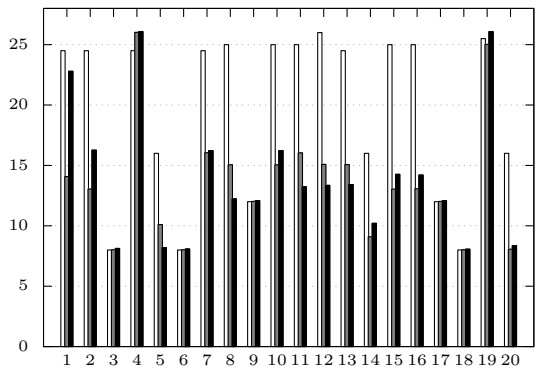
$$\boldsymbol{\beta} = (0.3267, 0.0033, 0.0026, 0.0042, 0.0011, 0.0021, 0.0022, 0.0040, 0.0012, 0.0041, -0.0004, 0.0037).$$

Observe that β_{11} is negative and close to zero, i.e., the coefficient corresponding to the ninth table is negative. This is mainly because of the size of this table (which is small) and the structure of the database.

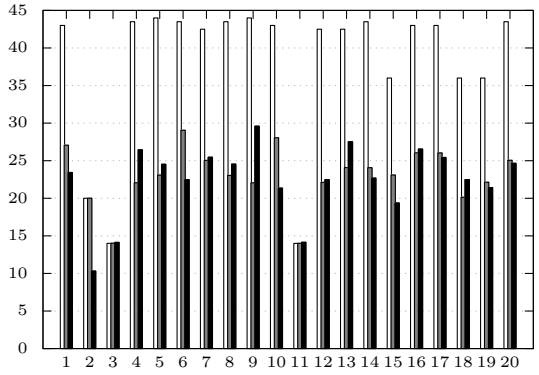
Table 5: The statistics of the regression model on the TPC-DS

Training set			**Test set**		
R-squared	**MSE**	**MAPE(%)**	**R-squared**	**MSE**	**MAPE(%)**
0.87	0.48	10.46	0.86	0.55	10.43

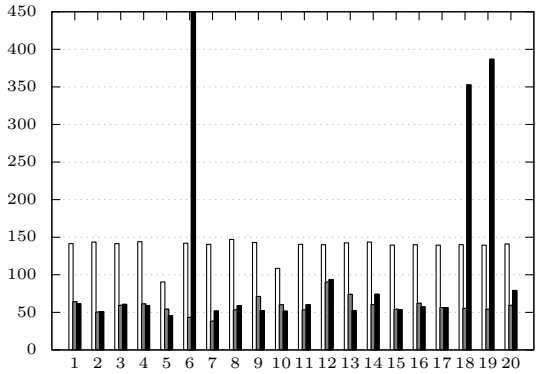
In order to show the effectiveness of solving the query batching problem on the TPC-DS database, a total of 120 instances are generated. Similar to what we did for the TPC-H database, the instances are divided over three classes of S, M, and L (based on their number of queries). Each class contains two subclasses, each with 20 instances. Table 6 shows the results of the proposed heuristics on the TPC-DS database. Numbers reported in this table for each instance subclass are averages over 20 instances. From columns 'Opt Time(s)', we observe that the solution time of our proposed heuristics has increased compared to the previous database, i.e., TPC-H. This can be



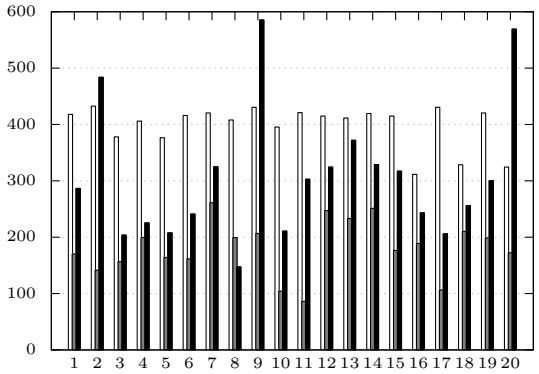
(a) $n = 32$



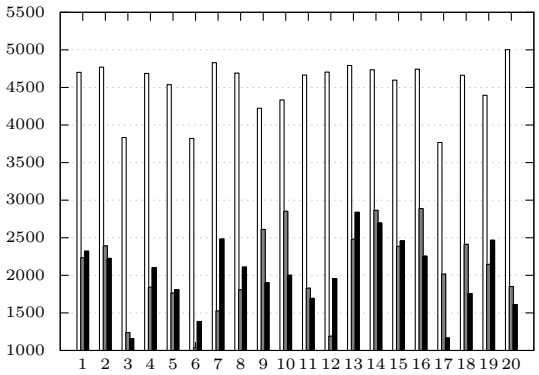
(b) $n = 64$



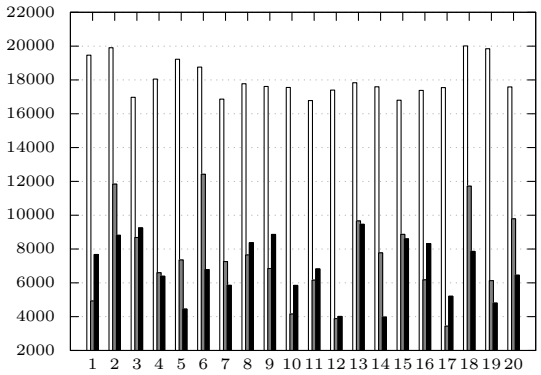
(c) $n = 256$



(d) $n = 512$



(e) $n = 2048$



(f) $n = 4096$

Figure 7: Total time (in seconds) of all 20 instances of different subclasses on the TPC-DS (white bars: default setting, gray bars: RCSA-I, black bars: RCSA-II)

Table 6: Performance of RCSA-I and RCSA-II on the TPS-DS

n	Default	RCSA-I			RCSA-II			Imp (%)	
	Total Time (s)	# of batches	Opt Time (s)	Total Time (s)	# of batches	Opt Time (s)	Total Time (s)	RCSA-I	RCSA-II
32	19.8	1.7	0.1	13.6	1.7	0.5	14.0	31.2	29.2
64	38.1	1.9	0.1	23.0	1.9	0.4	22.5	39.5	41.0
256	137.4	3.0	0.3	58.8	3.4	61.8	112.3	57.2	18.3
512	399.0	3.2	0.6	181.7	4.5	188.4	306.8	54.4	23.1
2048	4524.8	3.3	4.5	2068.4	5.0	668.3	2019.8	54.3	55.4
4096	18049.2	3.8	27.3	7566.9	4.1	532.1	6889.1	58.1	61.8

because more batches are constructed (on average) using our heuristics on the TPC-DS database as indicated in columns '# of batches'. We also observe from the table that our proposed algorithms can result in processing time reduction of between 18.3% to 61.8% on average. This is much larger than the improvement that we observed for the TPC-H and can be because of the structure of the TPC-DS database. Finally, we observe that RCSA-I and RCSA-II are competitive. In some subclasses of instances RCSA-I is better and in other subclasses of instances RCSA-II is better. The main reason that RCSA-I is better than RCSA-II in some cases is that the second phase of RCSA-II has been very time consuming, as shown in columns 'Opt Time(s)'. Figure 7 highlights this observation further because it shows the total time of all instances of all subclasses under default setting, RCSA-I, and RCSA-II. From the figure, we observe that RCSA-II has performed poorly for the Class M, i.e., instances with $n = 256$ and $n = 512$, which is again because the second phase of RCSA-II has been time consuming. Note that the imposed time limit for each iteration of RCSA-I/RCSA-II is 300 seconds in our study. Hence, one can make the results of RCSA-II close to RCAS-I by significantly decreasing the time limit imposed on each iteration of the second phase of RCSA-II.

Table 7: Comparing the quality of the solutions obtained by the proposed heuristics with optimal solutions on the TPC-DS

n	Exact Opt Time (s)	RCSA-I gap (%)	RCSA-II gap (%)
32	9.8	0.24	0
64	1144.3	4.53	3.84

Table 7 shows the (relative) optimality gap between the objective values reported by the heuristic methods and the global dual bounds reported by CPLEX when solving the exact MBQP formulation, i.e., MBQP(n), on instances with $n \leq 64$ over a two-hour time limit. Note that we did not report any result for larger instances, e.g., instances with $n = 256$ or $n = 512$, because CPLEX was unable to handle them as their optimality gap was close to 100% in 2 hours (and even hours beyond that). This observation is not surprising because as we mentioned earlier, even the second phase of RCSA-II was very time consuming for instances with $n = 256$ or $n = 512$. We note that in Table 7, 39 out of 40 instances were optimally solved within two hours using CPLEX and the optimality gap of only one instance was 46.53% after two hours. With this mind, we observe that both RCSA-I and RCSA-II have performed well on generating optimal solutions. Specifically, the optimality gap of RCSA-II is zero for all 39 instances (that we know an optimal solution) and only for one instance is non-zero.

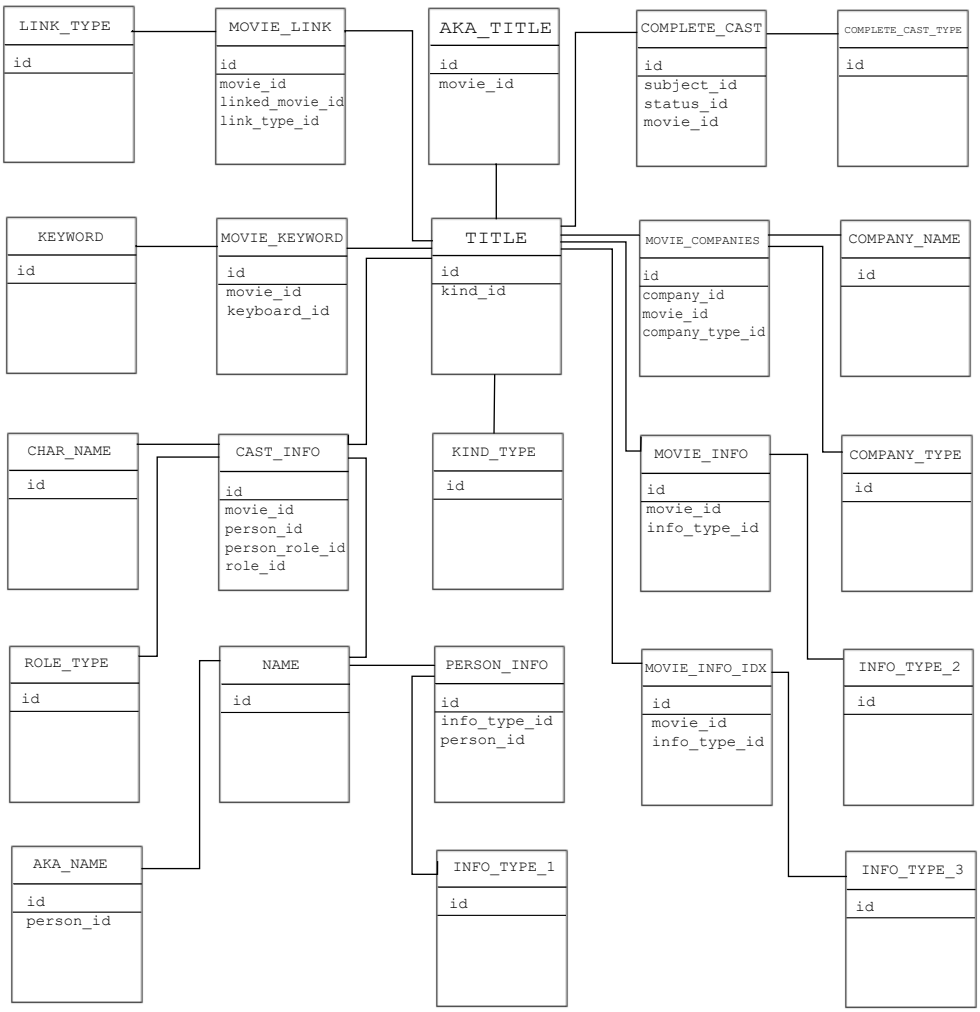


Figure 8: The structure of the JOB database

5.3. Database Benchmark JOB

The last benchmark we use to test the effectiveness of the QBP and our proposed heuristics is the Join-Order Benchmark (JOB) [15]. JOB is a fixed-sized database with almost 4GB of data which is created based on the well-known Internet Movie Database (IMDB). It contains detailed information about movies and related facts about actors, directors, production companies, etc. This database contains 23 tables, and relations among them. All the tables in this database and their relationships are shown in Figure 8. Again note that while there were only 17 valid combinations for involving tables in the TPC-H benchmark, the JOB benchmark involves a significantly larger number of valid combinations. Specifically, the total number of potential combinations is $2^{23} - 1$ but because of the relationships between the tables not all combinations are valid. In this section, we only select 165 valid combinations and employ them for generating instances. These 165 valid combinations are obtained based on the suggested scenarios that come with the JOB benchmark to ensure the generated queries make sense in practice (see the details in [15]).

Table 8: Performance of RCSA-I and RCSA-II on the JOB

n	**Default**	**RCSA-I**			**RCSA-II**			**Imp (%)**	
	Total Time (s)	**# of batches**	**Opt Time (s)**	**Total Time (s)**	**# of batches**	**Opt Time (s)**	**Total Time (s)**	**RCSA-I**	**RCSA-II**
32	145.1	2.4	0.2	115.3	1.9	0.3	111.2	20.6	23.3
64	325.4	2.2	0.1	263.9	1.9	0.3	257.4	18.9	20.9
256	1031.0	2.4	0.3	836.5	2.2	1.0	805.7	18.9	21.8
512	1757.6	1.9	0.4	1408.3	1.7	2.8	1377.0	19.9	21.7
2048	3694.3	1.8	1.6	3094.9	1.8	74.5	3083.2	16.2	16.5
4096	4502.8	2.3	14.6	3394.0	2.1	293.8	3765.2	24.6	16.4

Again, we first start by estimating the coefficients of the batch processing time prediction function, i.e., $\boldsymbol{\beta}$. We randomly generate 13,365 data points, i.e., batches of queries. We generate the data points for the JOB benchmark following a similar procedure that we used for the TPC-H benchmark. Specifically, the data points are divided over 81 classes, each containing 165 batches, based on the number of queries that they involve. The number of queries in each class of batches is again taken from the set $\{50, 75, 100, \ldots, 2050\}$. For each class, we generate a single scenario for showing the percentage of the data that should be retrieved as the result of processing a batch. Each scenario can take a value from the interval $(0\%, 5\%]$. For each scenario, we created 165 batches. Specifically, there are 165 selected valid ways of including tables in (a batch of) queries for the JOB databases. So, for each one, we randomly generate one batch of queries such that the amount of data that should be retrieved for it matches its associated scenario.

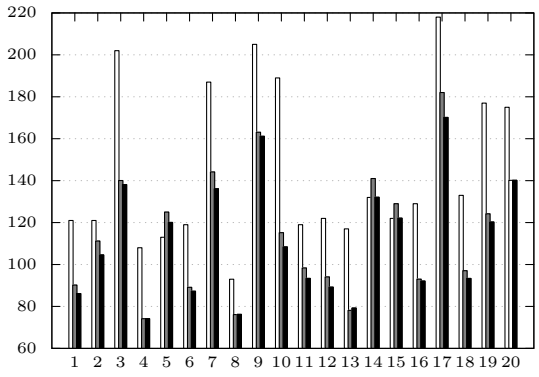
After creating random batches, we execute them individually and record their associated values for the dependent and independent variables of linear regression (see Section 4.1). We randomly pick 70% of data as the training set and use the remaining 30% as the test set. Overall, the results show an R-squared of around 0.98 (see Table 9) can be obtained by setting,

$$\begin{aligned}\boldsymbol{\beta} = (&2.3028, 0.0004, 0.0278, 0.3033, 0.2566, 0.2659, 0.2682, 0.2899, 0.2718, 0.2552,\\ &0.2516, 0.2580, 0.2620, 0.2466, 0.2462, 0.2602, 0.2625, 0.2661, 0.2879, 0.3121,\\ &0.3717, 0.3307, 0.3476, 0.2571, 0.2639).\end{aligned}$$

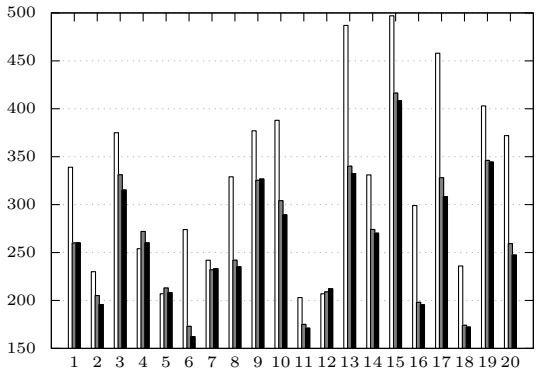
In order to show the effectiveness of solving the query batching problem on the JOB database, a total of 120 instances are generated. Similar to what we did for TPC-H database, the instances are

Table 9: The statistics of the regression model on the JOB

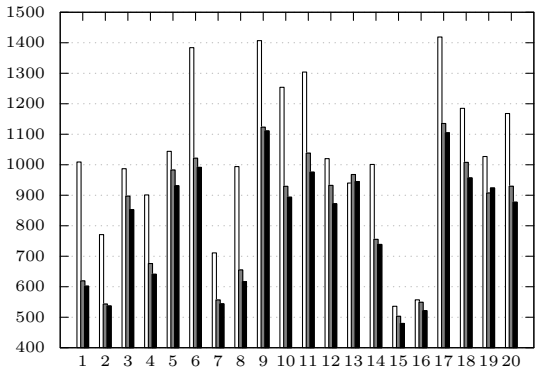
Training set			Test set		
R-squared	**MSE**	**MAPE(%)**	**R-squared**	**MSE**	**MAPE(%)**
0.98	0.04	2.18	0.98	0.04	2.17



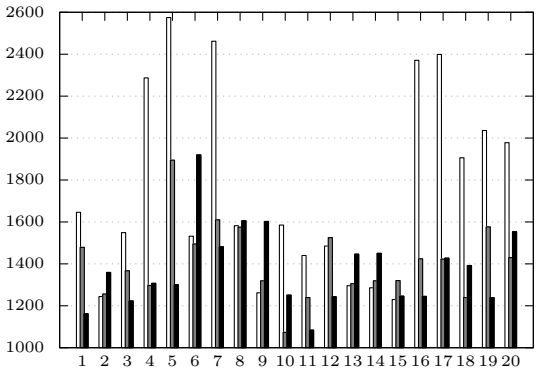
(a) $n = 32$



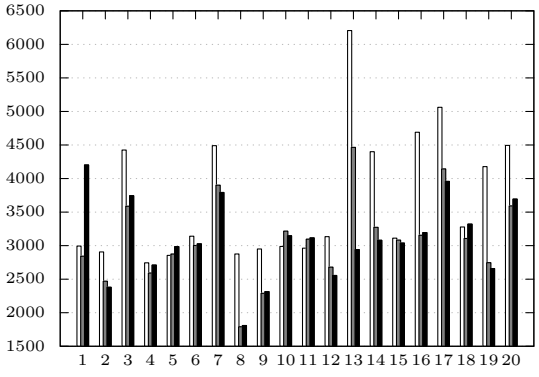
(b) $n = 64$



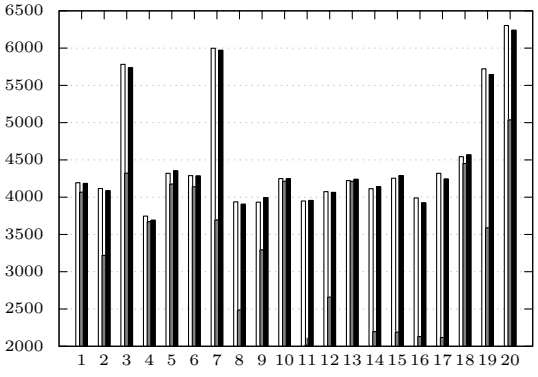
(c) $n = 256$



(d) $n = 512$



(e) $n = 2048$



(f) $n = 4096$

Figure 9: Total time (in seconds) of all 20 instances of different subclasses on the JOB (white bars: default setting, gray bars: RCSA-I, black bars: RCSA-II)

divided over three classes of S, M, and L (based on their number of queries). Each class contains two subclasses, each with 20 instances. Table 8 shows the results of the proposed heuristics on the JOB database. Numbers reported in this table for each instance subclass are averages over 20 instances. The last two columns demonstrate that the proposed heuristics reduce the total time up to 24.6%. Observe that since the JOB database contains more tables compared to the TPC-H and TPC-DS databases, the difference between the performances of RCSA-I and RCSA-II is more noticeable. Although, RCSA-II performs better in small instances (in terms of the total time), RCSA-I outperforms RCSA-II by around 8.2% for the largest subclass of instances. This can be mainly because of the significant increase in the optimization time of RCSA-II. Figure 9 highlights this observation further because (similar to Figure 5) it shows the total time of all instances of all subclasses under default setting, RCSA-I, and RCSA-II. Another interesting point from Table 8 is that, unlike the TPC-H databases, the number of batches for RCSA-II is smaller than the number of batches for RCSA-I. That is, RCSA-II sometimes assigns queries requiring the same table(s) to different batches, which can potentially result in a smaller number of batches.

Table 10 shows the (relative) optimality gap between the objective values reported by the heuristic methods and the global dual bounds reported by CPLEX when solving the exact MBQP formulation on instances with $n \leq 256$ over a two-hour time limit. CPLEX was able to solve all instances with $n = 32$ and most instances with $n = 64$ and $n = 256$ to optimality within the imposed time limit. Among the instances with $n = 64$, only 2 instances and among the instances with $n = 256$, 8 instances were not solved to optimality. From the table, we observe that both RCSA-I and RCSA-II performed similarly. They were both able to generate an optimal solution for all instances that their optimal solutions were known, i.e., solved within 2 hours time limit by CPLEX. Even for those few large instances that CPLEX was not able to prove the optimality within 2 hours, both heuristics relative optimality gap from the global dual bound is below 11%, while their required computational time is a small fraction of the time required by CPLEX.

Table 10: Comparing the quality of the solutions obtained by the proposed heuristics with optimal solutions on the JOB

n	Exact Opt Time (s)	RCSA-I gap (%)	RCSA-II gap (%)
32	5.9	0.00	0.00
64	726.9	0.93	0.93
256	3938.2	10.10	10.10

6. Final Remarks

In this paper, for the first time (to the best of our knowledge), we studied the query batching problem. This problem aims at partitioning a given set of queries into some batches before retrieving them from a database system in order to minimize the total retrieving/processing time. This optimization problem is challenging because predicting the time required for processing a given batch of queries is not a trivial task. We developed a generic three-phase approach for solving the problem for any given database system. In the first phase, using a quadratic function, our approach attempts to predict the processing time of any batch of queries for the given database. In the second phase, our approach uses the obtained quadratic function and develops a mixed binary quadratic programming formulation for the query batching problem accordingly. Finally, in the last phase, our proposed approach uses two custom-built heuristic approaches, i.e., RCSA-I and RCSA-II, to quickly solve the obtained formulation in practice. We tested our proposed approach

on three well-known database benchmarks by conducing a comprehensive computational study. The results showed that a reduction of up to 61.8% is achievable for the total processing times in the database benchmarks when employing our proposed approach.

We hope that the simplicity, versatility, and performance of our proposed approach encourage practitioners/researchers to consider employing/developing effective query batching optimizers. There are several future research directions that can be considered for this study. One direction can be developing better exact or heuristic solution approaches for solving instances with larger number of queries. Alternatively, developing effective machine learning methods for predicting the outcome, i.e., an optimal solution, of the proposed mixed binary quadratic program can be an interesting research direction too. Another research direction can be developing some theories and/or methodologies for identifying an optimal amount of time that one should wait to accumulate queries before starting to solve the query batching problem.

References

[1] Arumugam, S., Dobra, A., Jermaine, C. M., Pansare, N., Perez, L., 2010. The datapath system: A data-centric analytic processing engine for large data warehouses. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. SIGMOD '10. pp. 519–530.

[2] Codd, E. F., Jun. 1970. A relational model of data for large shared data banks. Commun. ACM 13 (6), 377–387.
URL `http://doi.acm.org/10.1145/362384.362685`

[3] Coronel, C., Morris, S., 2016. Database systems: design, implementation, & management. Cengage Learning.

[4] Elmasri, R., Navathe, S. B., 2006. Fundamentals of Database Systems (5th Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[5] Elmasri, R., Navathe, S. B., 2015. Fundamentals of Database Systems, 7th Edition. Pearson.

[6] Eslami, M., Tu, Y., Charkghard, H., 2019. A system for batched query processing and optimization. Tech. Rep. CSE/18-088, URL: `www.csee.usf.edu/~tuy/pub/tech18-088.pdf`, CSE Dept., University of South Florida, Tampa, FL, USA.

[7] Garcia-Molina, H., Ullman, J. D., Widom, J., 2008. Database Systems: The Complete Book, 2nd Edition. Prentice Hall Press, Upper Saddle River, NJ, USA.

[8] Giannikis, G., Alonso, G., Kossmann, D., 2012. Shareddb: Killing one thousand queries with one stone. Proc. VLDB Endow. 5 (6), 526–537.

[9] Giannikis, G., Makreshanski, D., Alonso, G., Kossmann, D., 2013. Workload optimization using shareddb. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. SIGMOD '13. ACM, pp. 1045–1048.

[10] Harizopoulos, S., Shkapenyuk, V., Ailamaki, A., 2005. Qpipe: A simultaneously pipelined relational query engine. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. SIGMOD '05. ACM, pp. 383–394.

[11] Hoerl, A. E., Kennard, R. W., 1970. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 12 (1), 55–67.

[12] Hoffer, J. A., Ramesh, V., Topi, H., 2011. Modern database management. Upper Saddle River, NJ: Prentice Hall,.

[13] Kester, M. S., Athanassoulis, M., Idreos, S., 2017. Access path selection in main-memory optimized data systems: Should i scan or should i probe? In: Proceedings of the 2017 ACM International Conference on Management of Data. SIGMOD '17. ACM, pp. 715–730.

[14] Kuhn, D., Alapati, S. R., Padfield, B., 2016. Expert Oracle Indexing and Access Paths: Maximum Performance for Your Database. Apress.

[15] Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T., Nov. 2015. How good are query optimizers, really? Proc. VLDB Endow. 9 (3), 204–215.
URL http://dx.doi.org/10.14778/2850583.2850594

[16] Link, Apr 2015. Global database industry market size 2012-2017.
URL https://www.statista.com/statistics/724611/worldwide-database-market/

[17] Oh, G., Kim, S., Lee, S.-W., Moon, B., Aug. 2015. Sqlite optimization with phase change memory for mobile applications. Proc. VLDB Endow. 8 (12), 1454–1465.
URL http://dx.doi.org/10.14778/2824032.2824044

[18] Pansart, L., Catusse, N., Cambazard, H., 2018. Exact algorithms for the order picking problem. Computers & Operations Research 100, 117 – 127.

[19] Pöss, M., Floyd, C., 2000. New TPC benchmarks for decision support and web commerce. SIGMOD Record 29 (4), 64–71.
URL https://doi.org/10.1145/369275.369291

[20] Ramakrishnan, Raghu, G. J., 2000. Database management systems. McGraw Hill.

[21] Ratliff, H. D., Rosenthal, A. S., 1983. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Operations Research 31 (3), 507–521.

[22] Roussopoulos, N., 1982. View indexing in relational databases. ACM Trans. Database Syst. 7 (2), 258–290.

[23] Ruben, R. A., Jacobs, F. R., 1999. Batch construction heuristics and storage assignment strategies for walk/ride and pick systems. Management Science 45 (4), 575–596.

[24] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., Price, T. G., 1979. Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD international conference on Management of data. ACM, pp. 23–34.

[25] Sellis, T. K., Mar. 1988. Multiple-query optimization. ACM Trans. Database Syst. 13 (1), 23–52.
URL http://doi.acm.org/10.1145/42201.42203

[26] Seppi, K. D., Barnes, J. W., Morris, C. N., 1993. A bayesian approach to database query optimization. ORSA journal on Computing 5 (4), 410–419.

[27] Sherali, H. D., Smith, J. C., 2001. Improving discrete model representations via symmetry considerations. Management Science 47 (10), 1396–1407.

[28] TPC, 2001. Tpc benchmark by transaction processing performance council.
URL `http://www.tpc.org`

[29] Triantis, K. P., Egyhazy, C. J., 1988. An integer programming formulation embedded in an algorithm for query processing optimization in distributed relational database systems. Computers & Operations Research 15 (1), 51 – 60.

[30] Tsitsiklis, J. N., Xu, K., 2017. Flexible queuing architectures. Operations Research 65 (5), 1398–1413.

[31] Wäscher, G., 2004. Order Picking: A Survey of Planning Problems and Methods. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 323–347.
URL `https://doi.org/10.1007/978-3-540-24815-6_15`

[32] Yu, M., de Koster, R. B., 2009. The impact of order batching and picking area zoning on order picking system performance. European Journal of Operational Research 198 (2), 480 – 490.
URL `http://www.sciencedirect.com/science/article/pii/S0377221708007601`

Appendix: Proof of Theorem 1

Theorem 1. *The query batching problem when employing the proposed processing time prediction function* (1) *is NP-hard.*

Proof. In order to prove that the QBP is NP-hard, it is sufficient to show that its corresponding decision problem, `QBP`, is NP-complete. Note that `QBP` is obviously in NP and an instance of `QBP` can be shown by

$$[m, n, U, \beta_0, \beta_1, \beta_2 \log_2(s_1), \ldots, \beta_{m+1} \log_2(s_m), Q_1, \ldots, Q_m].$$

In the remaining, we show that `Partition-Problem` is polynomially reducible to `QBP`, where `Partition-Problem` is defined as follows: given a set $\{c_1, \ldots, c_{m'}\}$ of positive integers, does there exist a subset $A \subseteq \{1, \ldots, m'\}$ such that

$$\sum_{i \in A} c_i = \sum_{i \in A \setminus \{1,\ldots,m'\}} c_i.$$

Observe that an integer programming formulation for `Partition-Problem` can be stated as follows (the second constraint is redundant but we keep it for the sake of the proof),

$$\begin{aligned}
&-\frac{\sum_{i=1}^{m'} c_i}{2} + \sum_{i=1}^{m'} c_i x_i = 0 \\
&-\frac{\sum_{i=1}^{m'} c_i}{2} + \sum_{i=1}^{m'} c_i (1 - x_i) = 0 \\
&x_i \in \{0, 1\} \qquad\qquad \forall i \in \{1, 2, \ldots, m'\}
\end{aligned}$$

Now, given an arbitrary instance of `Partition-Problem`, denoted by $[m', c_1, \ldots, c_{m'}]$, we construct an instance of `QBP` by setting,

- $m = n = k = m'$
- $U = 0$
- $\beta_0 = -\frac{\sum_{t=1}^{m'} c_t}{2}$
- $\beta_1 = 0$
- $\beta_{t+1} log(s_t) = c_t$ for all $t \in \{1, \ldots, m'\}$
- $Q_t = \{t\}$ for all $t \in \{1, \ldots, m'\}$

Observe that the created instance of `QBP` is valid in a sense that it can be constructed in polynomial time and each of its queries involves at least one table. Now, we show that an instance of `Partition-Problem`, $[m', c_1, \ldots, c_{m'}]$, is a Yes-instance if an only if its corresponding `QBP` instance is a Yes-instance. For doing so, we use the formulation proposed for `QBP`, i.e., MBQP(k), and we show that when we plug the constructed instance into it, the formulation will be simplified/equivalent to the formulation of `Partition-Problem`.

In light of the above, observe that because $\beta_1 = 0$ and $n_j = \sum_{i=1}^{n} x_{ij}$, the formulation of `QBP` can be simplified to,

$$\sum_{j=1}^{k} \left[\beta_0 z_j + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y_{jt}\right]^2 \leq U$$

$$\sum_{j=1}^{k} x_{ij} = 1 \qquad \forall i \in \{1, 2, \ldots, n\}$$

$$x_{ij} \leq y_{jt} \qquad \forall t \in T,\ \forall j \in \{1, 2, \ldots, k\},\ \forall i \in Q_t$$

$$y_{jt} \leq z_j \qquad \forall t \in T,\ \forall j \in \{1, 2, \ldots, k\}$$

$$y_{jt} \leq \sum_{i \in Q_t} x_{ij} \qquad \forall j \in \{1, 2, \ldots, k\},\ \forall t \in T$$

$$z_j \leq \sum_{i=1}^{n} x_{ij} \qquad \forall j \in \{1, 2, \ldots, k\}$$

$$x_{ij}, y_{jt}, z_j \in \{0, 1\} \qquad \forall j \in \{1, 2, \ldots, k\},\ \forall i \in \{1, \ldots, n\},\ \forall t \in T.$$

Moreover, since $Q_t = \{t\}$ for all $t \in \{1, \ldots, m'\}$ the term $\sum_{i \in Q_t} x_{ij}$ is equivalent to x_{tj}. So, the formulation can be simplified to,

$$\sum_{j=1}^{k} \left[\beta_0 z_j + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y_{jt}\right]^2 \leq U$$

$$\sum_{j=1}^{k} x_{ij} = 1 \qquad \forall i \in \{1, 2, \ldots, n\}$$

$$x_{tj} \leq y_{jt} \qquad \forall t \in T,\ \forall j \in \{1,2,\ldots,k\}$$
$$y_{jt} \leq z_j \qquad \forall t \in T,\ \forall j \in \{1,2,\ldots,k\}$$
$$y_{jt} \leq x_{tj} \qquad \forall j \in \{1,2,\ldots,k\},\ \forall t \in T$$
$$z_j \leq \sum_{i=1}^{n} x_{ij} \qquad \forall j \in \{1,2,\ldots,k\}$$
$$x_{ij}, y_{jt}, z_j \in \{0,1\} \qquad \forall j \in \{1,2,\ldots,k\},\ \forall i \in \{1,\ldots,n\},\ \forall t \in T.$$

The formulation can be simplified further to,

$$\sum_{j=1}^{k} \big[\beta_0 z_j + \sum_{t \in T} \beta_{t+1} \log_2(s_t) y_{jt}\big]^2 \leq U$$
$$\sum_{j=1}^{k} x_{ij} = 1 \qquad \forall i \in \{1,2,\ldots,n\}$$
$$x_{tj} = y_{jt} \qquad \forall t \in T,\ \forall j \in \{1,2,\ldots,k\}$$
$$y_{jt} \leq z_j \qquad \forall t \in T,\ \forall j \in \{1,2,\ldots,k\}$$
$$z_j \leq \sum_{i=1}^{n} x_{ij} \qquad \forall j \in \{1,2,\ldots,k\}$$
$$x_{ij}, y_{jt}, z_j \in \{0,1\} \qquad \forall j \in \{1,2,\ldots,k\},\ \forall i \in \{1,\ldots,n\},\ \forall t \in T.$$

Consequently, the decision variable y_{jt} can be replaced by x_{tj} since $m = n = m'$. So, the formulation can be simplified further to,

$$\sum_{j=1}^{m'} \big[\beta_0 z_j + \sum_{i=1}^{m'} \beta_{i+1} \log_2(s_i) x_{ij}\big]^2 \leq U$$
$$\sum_{j=1}^{m'} x_{ij} = 1 \qquad \forall i \in \{1,2,\ldots,m'\}$$
$$x_{ij} \leq z_j \qquad \forall i \in \{1,\ldots,m'\},\ \forall j \in \{1,2,\ldots,m'\}$$
$$z_j \leq \sum_{i=1}^{m'} x_{ij} \qquad \forall j \in \{1,2,\ldots,m'\}$$
$$x_{ij}, z_j \in \{0,1\} \qquad \forall i \in \{1,\ldots,m'\},\ \forall j \in \{1,2,\ldots,m'\}.$$

Now, since $U = 0$, $\beta_0 = -\frac{\sum_{t=1}^{m'} c_t}{2}$, and $\beta_{t+1} log(s_t) = c_t$ for all $t \in \{1,\ldots,m'\}$, the formulation can be rewritten as,

$$\sum_{j=1}^{m'} \big[-\frac{\sum_{i=1}^{m'} c_i}{2} z_j + \sum_{i=1}^{m'} c_i x_{ij}\big]^2 \leq 0$$
$$\sum_{j=1}^{m'} x_{ij} = 1 \qquad \forall i \in \{1,2,\ldots,m'\}$$

$$x_{ij} \leq z_j \qquad \forall i \in \{1, \ldots, m'\},\ \forall j \in \{1, 2, \ldots, m'\}$$
$$z_j \leq \sum_{i=1}^{m'} x_{ij} \qquad \forall j \in \{1, 2, \ldots, m'\}$$
$$x_{ij}, z_j \in \{0, 1\} \qquad \forall i \in \{1, \ldots, m'\},\ \forall j \in \{1, 2, \ldots, m'\}.$$

Since the sum of squares is not positive, each square must be zero. So, the formulation can be simplified to,

$$-\frac{\sum_{i=1}^{m'} c_i}{2} z_j + \sum_{i=1}^{m'} c_i x_{ij} = 0 \qquad \forall j \in \{1, 2, \ldots, m'\} \tag{15}$$
$$\sum_{j=1}^{m'} x_{ij} = 1 \qquad \forall i \in \{1, 2, \ldots, m'\} \tag{16}$$
$$x_{ij} \leq z_j \qquad \forall i \in \{1, \ldots, m'\},\ \forall j \in \{1, 2, \ldots, m'\} \tag{17}$$
$$z_j \leq \sum_{i=1}^{m'} x_{ij} \qquad \forall j \in \{1, 2, \ldots, m'\} \tag{18}$$
$$x_{ij}, z_j \in \{0, 1\} \qquad \forall i \in \{1, \ldots, m'\},\ \forall j \in \{1, 2, \ldots, m'\}. \tag{19}$$

Observe that Constraints (17) and (18) guarantee that $z_j = 1$ if and only if $\sum_{i=1}^{m'} x_{ij} > 0$. This combined with Constraint (15) ensure that for any $j \in \{1, \ldots, m'\}$, $\sum_{i=1}^{m'} c_i x_{ij} = \frac{\sum_{i=1}^{m'} c_i}{2}$ if and only if $z_j = 1$ (or $\sum_{i=1}^{m'} x_{ij} > 0$). This itself combined with Constraint (16), which is defined for partitioning the set of queries, guarantee that (if the instance is feasible then) there will be exactly two batches $j, j' \in \{1, \ldots, m'\}$ with $j \neq j'$ such that $\sum_{i=1}^{m'} c_i x_{ij} = \frac{\sum_{i=1}^{m'} c_i}{2}$ and $\sum_{i=1}^{m'} c_i x_{ij'} = \frac{\sum_{i=1}^{m'} c_i}{2}$. Note that the remaining batches must be empty since by assumptions $c_1, \ldots, c_{m'} > 0$. So, the above formulation is equivalent to,

$$-\frac{\sum_{i=1}^{m'} c_i}{2} + \sum_{i=1}^{m'} c_i x_{ij} = 0 \qquad \forall j \in \{1, 2\}$$
$$\sum_{j=1}^{2} x_{ij} = 1 \qquad \forall i \in \{1, 2, \ldots, m'\}$$
$$x_{ij} \in \{0, 1\} \qquad \forall i \in \{1, 2, \ldots, m'\}, \forall j \in \{1, 2\}$$

This itself is equivalent to,

$$-\frac{\sum_{i=1}^{m'} c_i}{2} + \sum_{i=1}^{m'} c_i x_i = 0$$
$$-\frac{\sum_{i=1}^{m'} c_i}{2} + \sum_{i=1}^{m'} c_i (1 - x_i) = 0$$
$$x_i \in \{0, 1\} \qquad \forall i \in \{1, 2, \ldots, m'\}$$

This is precisely an integer programming formulation of `Partition-Problem`. Therefore, the result follows.