Models for test cost minimization in database migration*

ANONYMOUS AUTHOR(S)*

Database migration is a ubiquitous need faced by enterprises that generate and use vast amounts of data. This is due to database software updates, or from changes to hardware, project standards, and other business factors [25]. Migrating a large collection of databases is a way more challenging task than migrating a single database due to the presence of additional constraints. These constraints include capacities of shifts, and sizes of databases. In this paper, we present a comprehensive framework that can be used to model database migration problems of different enterprises with customized constraints, by appropriately instantiating the parameters of the framework. These parameters are the size of each database, the size of each shift, and the cost of testing each application. Each of these parameters can be either constant or arbitrary. Additionally, the cost of testing an application can be proportional to the number of databases that application uses. Additionally, we examine a variant of the problem in which all the parameters are constant, there are only two shifts, and each application calls at most two databases. We establish the computational complexities of a number of instantiations of this framework. We present fixed-parameter intractability results for various relevant parameters of the database migration problem. We also provide approximability and inapproximability results as well as lower bounds for the running time of any exact algorithm for the database migration problem. We show that the database migration problem is equivalent to a variation of the classical Hypergraph Partitioning problem. Our theoretical results also imply new theoretical results for the Hypergraph Partitioning problem that are interesting in their own right. Finally, we adapt heuristic algorithms devised for the Hypergraph Partitioning problem to the database migration problem, and give experimental results for the adapted heuristic algorithms.

CCS Concepts: • **Theory of computation → Problems, reductions and completeness**; • **Computing methodologies** → *Planning and scheduling*.

Additional Key Words and Phrases: Database Migration, Capacity constraint, Inapproximabilty, Fixed-parameter tractability

ACM Reference Format:
Anonymous Author(s). 2022. Models for test cost minimization in database migration. *ACM Trans. Datab. Syst.* 1, 1 (May 2022), 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Many factors drive the Big Data age. The mass deployment of sensors as part of the Internet-of-Things (IoT) paradigm generates an enormous amount of information [20]. The adoption of video cameras, surveillance devices, and smart appliances has given rise to data-spewing smart cities [34]. In the health-care field, billions of radiological images are produced annually that require processing [13]. In addition to these domains, there is an ongoing wave of user-generated content. Indeed, these disruptive technologies have begun to transform the way we store, process, and analyze data. The resulting deluge of information has been mitigated in large part thanks to cloud computing infrastructures that store and process the data [12].

Cloud computing has also surged in popularity among consumers in recent years. Indeed, cloud computing leverages the availability of data centers with three main consumer services: Software-as-a-Service (SaaS) [32], Platform-as-a-Service (PaaS) [6], and Infrastructure-as-a-Service (IaaS) [26]. In SaaS, the end-user is provided with software applications running on the cloud. PaaS provides the platform for the user to execute his own applications. IaaS provides storage and hardware resources through a virtual machine. These services are provided via a multitude of interconnected data servers

*Title note

2022. 0362-5915/2022/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

housed in one or more data centers. As such, their reliability, availability, and latency can be affected by the movement of data between said servers.

The database migration problem entails the movement of data between different databases. Such migration is often necessary due to database software updates, or from changes to hardware, project standards, and other business factors [25]. As per established rules of software reliability, when a database is migrated, every application that is dependent upon it *must* be tested (i.e., run through regression suites [11]). It is known that testing an application is an expensive aspect of maintaining the application [30].

While the ubiquity of computational and storage capabilities of cloud computing are undeniable, there remain open challenges with regards to resource allocation and data management [22]. In fact, the migration of data in data centers remains a significant problem, due to the massive throughput of data and the limited bandwidth of communication channels [21]. This problem is further exacerbated by the overhead incurred from retesting applications after data migration and by Quality-of-Service (QoS) requirements, which demand minimal interruptions to end-user applications [33]. As such, minimizing the cost associated with bandwidth-constrained database migration is of great interest.

If there are no constraints imposed on the database migration, then minimizing the application testing cost would be a trivial task (i.e., all of the databases would be migrated at once). This means each application would be tested only once. However, in practical situations, there are a number of constraints which need to be met. In a typical company, all the databases cannot be migrated at once since the databases will be inaccessible during the migration process. The company will need a scheduling plan for the database migration operation. In the database migration operation, the databases need to be clustered into several subsets such that the constraints are met and the databases in the same subset are migrated together. We say that the set of databases clustered into the same subset forms a *shift* of the migration schedule. When a set of databases is migrated, every application that is dependent upon one of those databases must be tested. Consequently, the principal goal in the migration process is to minimize the application testing cost [31].

Patil et al. [23] introduced the first systematic study on the database migration problem, and proved that the problem is **NP-hard** for some special cases. They provided an integer programming formulation, which can only be used for small instances, as well as a greedy heuristic that can be used for the large instances of the problem. In this paper, we define a general framework that subsumes the model in [23]. Our framework accommodates the modeling of database migration needs of various enterprises with customized constraints. We present hardness results for all of the models in our framework as well as fixed-parameter intractability results for various relevant parameters. We also present approximability and inapproximability results as well as lower bounds for the running time of any exact algorithm for the models in our framework. We discuss the relation between our framework and the Hypergraph Partitioning (HGP) problem, and prove new theoretical results for the HGP problem that are interesting in their own right. We also adopt heuristic algorithms devised for the HGP problem for the models in our framework and test the performance of these adapted heuristic algorithms on randomly generated instances. The best performing algorithm finds solutions that are within 0.68% and 16.68% of the optimum in the average-case and worst-case, respectively. Additionally, 63.45% of the instances are solved optimally. We also present a randomized approximation algorithm for a simple but interesting special case of our framework.

The principal contributions of this paper are as follows:

(1) The CCDM problem is **NP-hard** in all models (see Section 3).
(2) The CCDM problem is **NP-hard** when database sizes are arbitrary, even if there is only one application (see Section 3).

(3) The CCDM problem is **NP-hard** when when shift sizes are arbitrary and application costs are either constant or arbitrary, even if each application uses at most two databases (see Section 4).
(4) The CCDM2 problem cannot be solved in time $2^{o(n)}$, unless the Exponential Time Hypothesis (ETH) fails (see Section 5).
(5) A randomized $(\frac{2 \cdot k-1}{k} + \epsilon)$-approximation algorithm for a restricted version of the CCDM problem where k is the number of shifts (see Section 6).
(6) **APX-hardness** of the CCDM problem with constant database and shift sizes, when each application calls at most two databases implies **APX-hardness** of the Weighted Minimum-Bisection problem (see Section 6).
(7) There is a relationship between certain variants of the CCDM problem and certain variants of the HGP problem (see Section 7).
(8) An empirical study of heuristic algorithms for the CCDM problem (see Section 8).

The rest of the paper is organized as follows: In Section 2, we formally define our framework which we refer to as the Capacity-Constrained Database Migration (CCDM) problem and introduce the notation used in the paper. In Section 3, we study the computational complexities of all of the models of the CCDM problem and prove that the problem is **NP-hard** for all of the models. The fixed-parameter tractability of the CCDM problem with respect to four relevant parameters is discussed in Section 4. The $2^{o(n)}$ lower bound for the running time of any exact algorithm for the CCDM problem is presented in Section 5. In Section 6, we present a randomized approximation algorithm for a special case of the CCDM problem and show that **APX-hardness** of some models of the CCDM problem implies **APX-hardness** for the weighted Minimum Bisection problem. In Section 7, we discuss the relation between the CCDM problem and the HGP problem, and present the implications of our theoretical results for the CCDM problem to the HGP problem. A detailed implementation profile of various models of the CCDM problem on representative data is described in Section 8. Finally, we conclude in Section 9, by summarizing our contributions and pointing out avenues for future research.

2 THE NOTATION AND PROBLEM FORMULATION

In this section, we provide a formal definition of the framework for the CCDM problem. We also introduce the terminology used in this paper. Assume we have a collection of m applications $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ and n databases $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, with each application calling one or more databases. The call relationship is stored in the $m \times n$ matrix $\mathbf{D} = \|d_{ij}\|$, where

$$d_{ij} = \begin{cases} 1, & \text{if application } A_i \text{ calls database } B_j \\ 0, & \text{otherwise.} \end{cases}$$

The matrix $\mathbf{D} = \|d_{ij}\|$, which represents a bipartite graph as shown in Figure 1, is part of the input. Associated with the set of applications $\mathcal{A}$ is a cost-vector $\mathbf{c} = [c_1, c_2, \ldots, c_m]^T$, where c_i represents the cost of testing application A_i once. For each application A_i, we let x_i be an integer variable that denotes the number of times A_i will have to be tested in the migration schedule. The size-vector $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$ represents the size of databases, with w_i representing the size of B_i.

In the CCDM problem, the set of databases $\mathcal{B}$ is to be clustered into disjoint subsets which we call shifts. The databases in each shift are migrated at the same time. When a shift of databases migrates, each application that calls at least one database in that shift needs to be tested immediately. For example, if the set of databases called by an application A_i is scheduled into 5 distinct shifts, then application A_i needs to be tested 5 times throughout the migration process, i.e., $x_i = 5$. The total size of the databases that may migrate in shift i (i.e., the size of shift i) is bounded by l_i. The shift

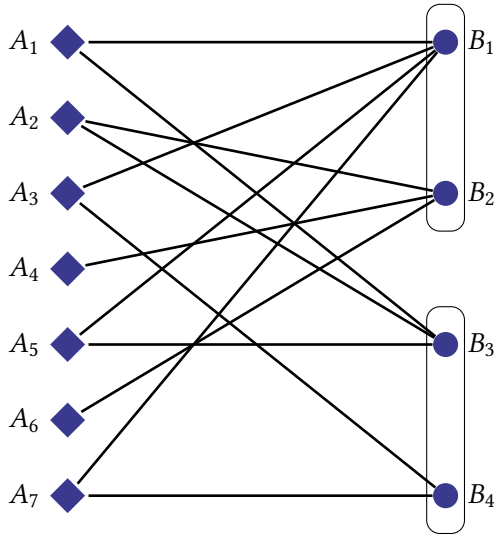

Fig. 1. The bipartite graph shows the call relationship between the applications and the databases. The nodes in the left partition represent the applications, while the nodes in the right partition represent the databases in the system. An edge (A_i, B_j) exists in the graph if application A_i calls database B_j. This means A_i must be tested right after B_j migrates. We note that each database is associated with a nonempty set of applications, and each application is associated with a nonempty set of databases. This figure shows a partition of the databases into 2 shifts.

size-vector $\mathbf{l} = [l_1, l_2, \ldots, l_k]^T$ is also part of the input. Thus, the input to the CCDM problem must contain the 4-tuple $\langle \mathbf{c}, \mathbf{w}, \mathbf{D}, \mathbf{l} \rangle$.

For instance, consider the following 4-tuple:

$$\left\langle \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 3 \end{pmatrix}, \begin{pmatrix} 5 \\ 7 \\ 10 \\ 12 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}^T, \begin{pmatrix} 10 \\ 15 \\ 12 \\ 7 \\ 28 \\ 34 \end{pmatrix} \right\rangle$$

In this example, we have seven applications and four databases, since the matrix $\mathbf{D}$ has seven rows and four columns. We will name the applications as $A_1, A_2, \ldots, A_7$, and the databases as B_1, B_2, B_3, B_4. Matrix $\mathbf{D}$ indicates which applications call which databases. We observe that database B_1 is called by applications A_1, A_3, A_5, and A_7. The database B_2 is called by applications A_2, A_4, and A_6. The database B_3 is called by applications A_1, A_2, and A_5, and database B_4 is called by applications A_3, and A_7. The testing costs of applications $A_1, A_2, \ldots, A_7$ are given as 1, 1, 1, 2, 2, 3, and 3, respectively, in $\mathbf{c}$. The sizes of the databases B_1, B_2, B_3, and B_4 are given as 5, 7, 10, and 12, respectively, in $\mathbf{w}$. The database migration operation in this example is to be completed in at most 6 shifts since $|\mathbf{l}| = 6$ in the input. The cumulative size of the databases that migrate in each shift is constrained by 10, 15, 12, 7, 28, and 34, respectively, in $\mathbf{l}$.

There are several parameters associated with the CCDM problem:

(i) Application Testing cost (α) - The testing cost of an application depends primarily on two factors: The time it takes to test the application, and the skills required to test a given application. We consider the following three cost models associated with testing an application in order to take these factors into account:
   (a) Constant (*const*) - In this model, the cost of testing each application is the same and is equal to some known fixed constant C. Note that this model considers the scenario in which each application requires the same level of skills and takes roughly the same time to test.
   (b) Proportional (*prop*) - In this model, the cost of testing an application is proportional to the total size of the databases it calls. Note that this model considers the situation where each application in the system requires the same level of skills, but the testing time may vary from application to application.
   (c) Arbitrary (*arb*) - In this model, there is no relation among the costs of testing different applications. This model captures the problem of companies that use application software from several different companies such that each application requires personnel with different skill sets.

(ii) Size of Databases (β) - The size of a database is a factor in the amount of time required for its migration. This is because the database migration operation must read the database at the original location, write it at the new location, and then delete the original database. Consider a bank that maintains data for credit card accounts, savings accounts, and checking accounts in different databases. Typically, a bank will have more customers with a checking account than a savings account. Similarly, the number of credit card customers will be significantly more than the number of savings account customers since a typical customer has one savings account but several credit cards. This means we need two size models associated with the databases:
   (a) Constant (*const*) - In this model, the size of each database is the same and is equal to some fixed constant W. This allows us to model the database migration operation for companies whose databases have more or less the same size.
   (b) Arbitrary (*arb*) - In this model, the sizes of the databases are arbitrary. This lets us model the database migration operation for companies whose databases may significantly vary in size.

(iii) Shift size (γ) - During the database migration operation, some parts of the database will be inaccessible. For some companies, there is no ideal time to make a database unavailable. For instance, Facebook and Youtube have users all over the world which means the database access rate is roughly uniform. In this case, regardless of when a database becomes inaccessible, there will be a subset of users who cannot access the database until the migration is complete. For companies (e.g., banks) that operate during regular business hours, it is favorable for the databases to be unavailable when the companies are closed rather than when they are open. In order to model the needs of several different companies, we use two size models associated with the shifts:
   (a) Constant (*const*) - In this model, the total size of the databases migrating in each shift is bounded by a fixed constant L (i.e., $\mathbf{l} = \langle L, L, \ldots, L \rangle$).
   (b) Arbitrary (*arb*) - In this model, the bound on the total size of the databases migrating in each shift is arbitrary.

Thus, a model of the capacity-constrained database migration problem has three parameters, and it is specified as a triple $\langle \alpha \mid \beta \mid \gamma \rangle$. For instance, $\langle arb \mid const \mid const \rangle$ refers to the capacity-constrained database migration problem in which the application testing costs are arbitrary, all databases have the same size, and the shift sizes are uniform. For notational convenience, we use $*$ as an entry of the triple when we present a statement that is true for all of the models for that entry. For instance, the

notation $\langle arb \mid * \mid * \rangle$ refers to all 4 models of the CCDM problem in which the application testing costs are arbitrary. The following is a formal definition of the CCDM problem:

CCDM*: Given a* 4*-tuple* $\langle \mathbf{c}, \mathbf{w}, \mathbf{D}, \mathbf{l} \rangle$*, cluster the databases into shifts so that the total application testing cost is minimized, while respecting the shift size constraints.*

Given that we have 3 different models for application testing costs, 2 different models for databases sizes, and 2 different models for shift sizes, the CCDM problem formulation gives us a framework with a total of 12 different models, each of which is suitable for the database migration needs of different companies. We use CCDM$\langle \alpha \mid \beta \mid \gamma \rangle$ to refer to the set of CCDM problems that use models captured by $\langle \alpha \mid \beta \mid \gamma \rangle$. For example, CCDM$\langle const \mid * \mid const \rangle$ is the set of CCDM problems that use the two models captured by $\langle const \mid * \mid const \rangle$.

We refer to the problem in CCDM$\langle const \mid const \mid const \rangle$, with the additional constraints that there are only two shifts and each application calls at most two databases, as the CCDM2 problem. Though the setting of the CCDM2 problem is rather restricted, most of the hardness results for the CCDM problem also hold for the CCDM2 problem. Thus, we will use the CCDM2 problem in our hardness proofs for the CCDM problem.

2.1 Integer Linear Program for the CCDM problem

We now present an integer linear programming formulation for the CCDM problem. In this formulation, the decision variable b_{jk} is 1 if the database B_j is assigned to shift k and 0 otherwise. The decision variable a_{ik} is 1 if the application A_i needs to be tested after the shift j is migrated and is 0 otherwise. The decision variable x_i is the total number of times the application A_i needs to be tested.

$$\text{minimize} \sum_{i=1}^{m} c_i x_i$$

subject to

$$\sum_{k=1}^{n} b_{jk} = 1, \qquad \forall j = 1,\ldots,n \tag{1}$$

$$\sum_{j=1}^{n} w_j \cdot b_{jk} \leq l_k, \qquad \forall k = 1,\ldots,n \tag{2}$$

$$\sum_{j=1}^{n} b_{jk} \cdot d_{ik} \leq M_i \cdot a_{ik}, \qquad \forall i = 1,\ldots,m, \forall k = 1,\ldots,n, \tag{3}$$

$$\sum_{k=1}^{n} a_{ik} = x_i, \qquad \forall i = 1,\ldots,m \tag{4}$$

$$b_{jk} \in \{0,1\}, \qquad \forall j,k = 1,\ldots,n \tag{5}$$

$$a_{ik} \in \{0,1\}, \qquad \forall i = 1,\ldots,m, \forall k = 1,\ldots,n \tag{6}$$

$$x_i \in \{1,\ldots,n\}, \qquad \forall i = 1,\ldots,m \tag{7}$$

The first set of constraints ensures that each database is assigned to a shift. The second set of constraints enforces the shift sizes. The parameter M_i is equal to the number of databases used by application A_i, i.e., $M_i = \sum_{j=1}^{n} d_{ik}$. The third set of constraints sets a_{ik} to 1 if any of the databases used by the application A_i is assigned to the shift k. The last set of constraints counts how many times the application A_i needs to be tested after all shifts are migrated.

3 COMPUTATIONAL COMPLEXITY OF THE CCDM PROBLEM

The formulation given in [23] corresponds to the problem in CCDM$\langle arb \mid arb \mid const \rangle$. [23] proved that the CCDM problem is **NP-hard** for that model. In this section, we strengthen their hardness result via Theorem 3.1, which states that the CCDM problem is **NP-hard** for all of the models in our framework, even under the additional restrictions that there are only 2 shifts, and each application calls at most 2 databases.

THEOREM 3.1. *Each of the problems in CCDM$\langle * \mid * \mid * \rangle$ is* **NP-hard***, even when there are only two shifts, and each application calls at most two databases.*

PROOF. Recall that CCDM$\langle * \mid * \mid * \rangle$ captures all of the models in our framework. Notice that the CCDM2 problem trivially reduces to each of the problems in CCDM$\langle * \mid * \mid * \rangle$, under the additional restrictions given in the theorem statement. Therefore, all we need is to show that the CCDM2 problem is **NP-hard**. We will do this via a polynomial time reduction from the classical Minimum-Bisection problem, a formal definition of which is given below.

Definition 3.2 (Minimum-Bisection). Given an undirected graph $G = (V, E)$, partition V into two subsets V_1 and V_2 of equal size such that the number of edges with one endpoint in V_1 and one endpoint in V_2 is minimized. It is assumed that $|V|$ is even.

For a given instance $G = (V, E)$ of the Minimum-Bisection problem, we construct the corresponding CCDM2 instance as follows:

- For every vertex i of the graph of the Minimum-Bisection instance, the CCDM2 instance has a corresponding database B_i with unit size, i.e., $w_i = 1$,
- For every edge $e = (i, j)$ of the graph of the Minimum-Bisection instance, the CCDM2 instance has a corresponding application A_e with unit application testing cost ($c_e = 1$) that only calls databases B_i and B_j,
- The CCDM2 instance has only two shifts and the size of each shift is $\frac{|V|}{2}$.

Notice that the constructed CCDM2 instance has $|V|$ databases and $|E|$ applications such that each application calls exactly two databases. In any feasible solution to the CCDM2 instance, exactly $l = \frac{|V|}{2}$ databases are assigned to the first shift and the remaining l databases are assigned to the second shift. If both of the databases called by an application A_i are assigned to the same shift, then application A_i needs to be tested only once ($x_i = 1$) in the database migration process, and it needs to be tested twice ($x_i = 2$) otherwise. Let d denote the number of applications that call one database from each shift, and thus need to be tested twice. Then, the total application testing cost of the CCDM2 instance is $(|E| + d)$, since d of the $|E|$ applications are tested twice and all other applications are tested only once. Since $|E|$ is fixed, the optimal solution to the CCDM2 instance is the one that minimizes d.

Given a solution to the constructed CCDM2 instance, consider the following solution to the given Minimum-Bisection instance:

- If database B_i of the constructed CCDM2 instance is assigned to the first shift, then assign vertex i of G to V_1,
- If database B_i of the constructed CCDM2 instance is assigned to the second shift, then assign vertex i of G to V_2.

Notice that an edge $e = (i, j)$ of the given Minimum-Bisection instance has one endpoint in V_1 and one endpoint in V_2 if and only if the databases B_i and B_j of the CCDM2 instance are assigned to different shifts, and thus the application A_e is to be tested twice. Therefore, the number of edges with one endpoint in V_1 and one endpoint in V_2 of the given Minimum-Bisection instance is equal to the

number of applications that needs to be tested twice in the constructed CCDM2 instance, which is denoted by d. □

4 FIXED-PARAMETER INTRACTABILITY OF THE CCDM PROBLEM

In this section, we study the fixed-parameter tractability of the CCDM problem for 4 different input parameters. These are the number of applications, the number of shifts, the maximum number of databases that are called by an application, and the maximum number of applications that calls a database. Notice that Theorem 3.1 establishes fixed-parameter intractability for all 12 models of the CCDM problem, where the parameter is the number of shifts, or the maximum number of databases that are called by an application. In the rest of the section, we prove intractability results for the remaining parameters.

Theorem 4.1 establishes fixed-parameter intractability for all the problems in CCDM$\langle * \mid arb \mid * \rangle$, when parameterized by the number of applications.

THEOREM 4.1. *The 6 problems in CCDM$\langle * \mid arb \mid * \rangle$ are* **NP-hard***, even when the number of applications $|\mathcal{A}|$ is 1.*

PROOF. Note that the set of instances of the problem in CCDM$\langle const \mid arb \mid const \rangle$ is a subset of the instances of any problem in CCDM$\langle * \mid arb \mid * \rangle$ when $|\mathcal{A}| = 1$. Thus, we only need to show that the problem in CCDM$\langle const \mid arb \mid const \rangle$ is **NP-hard**, even when $|\mathcal{A}| = 1$. We will do that via a polynomial reduction from the classical Partition problem, a formal definition of which is given below.

Definition 4.2 (Partition). Given a multiset S of positive integers, is S can be partitioned into two subsets S_1 and S_2 such that the sum of the numbers in S_1 is equal to the sum of the numbers in S_2?

Given a Partition instance $S = \{s_1, s_2, \ldots, s_n\}$, we construct a corresponding CCDM in CCDM$\langle const \mid arb \mid co$ with one application as follows:

- For every integer s_i in the multiset S of the Partition instance, the CCDM instance has a corresponding database B_i with size s_i, i.e., $w_i = s_i$,
- The CCDM instance has one application A_1 with unit testing cost, that calls all of the n databases,
- The CCDM instance has sufficiently many shifts and the size of each shift is $\frac{\sum_{i=1}^{n} s_i}{2}$.

Since A_1 calls all the databases, the total application testing cost is equal to the number of shifts with at least one database assigned. Thus, the CCDM instance has a solution with total application testing cost of 2 if the databases can be clustered into two shifts. The theorem holds since the databases can be clustered into two shifts, if and only if the answer to the Partition instance is "yes". □

Theorem 4.3 establishes fixed-parameter intractability for all 4 of the problems in CCDM$\langle const \mid * \mid arb \rangle \cup$ CCDM$\langle arb \mid * \mid arb \rangle$, when parameterized by the maximum number of applications that call a database.

THEOREM 4.3. *All of the problems in CCDM$\langle const \mid * \mid arb \rangle \cup$ CCDM$\langle arb \mid * \mid arb \rangle$ are* ***NP-hard****, even if each database is called by at most two applications.*

PROOF. Note that the set of instances of the problem in CCDM$\langle const \mid const \mid arb \rangle$ is a subset of the instances of any problem in CCDM$\langle const \mid * \mid arb \rangle \cup$ CCDM$\langle arb \mid * \mid arb \rangle$. Thus, all we need is to show that the problem in CCDM$\langle const \mid const \mid arb \rangle$ is **NP-hard**, even when each database is called by at most two applications. This is done via a polynomial reduction from Clique problem, a formal definition of which is given below.

Definition 4.4 (Clique). Given a graph $G = (V, E)$ and an integer k, is there a fully connected subgraph $G' \subseteq G$ consisting of k vertices?

Given a Clique instance $\langle G = (V, E), k \rangle$, we construct an instance of the problem in CCDM$\langle const \mid const \mid arb \rangle$ as follows:

- For every vertex v_i of the Clique instance, the CCDM instance has a corresponding application A_i with unit application testing cost, i.e., $c_i = 1$.
- For every edge e of the Clique instance, the CCDM instance has a corresponding database B_e with unit size, i.e., $w_e = 1$.
- For every edge $e = (v_i, v_j)$ of the Clique instance, the applications A_i and A_j of the CCDM instance calls database B_e. Notice that each database is called by exactly 2 applications.
- The CCDM instance has $|E| - \frac{k \cdot (k-1)}{2} + 1$ shifts. The size of the first shift is $\frac{k \cdot (k-1)}{2}$, and the size of each of the remaining $|E| - \frac{k \cdot (k-1)}{2}$ shifts is 1.

Figure 2 explains this reduction. In Figure 2, we have a Clique instance on the left and the corresponding CCDM instance constructed by the above reduction on the right.

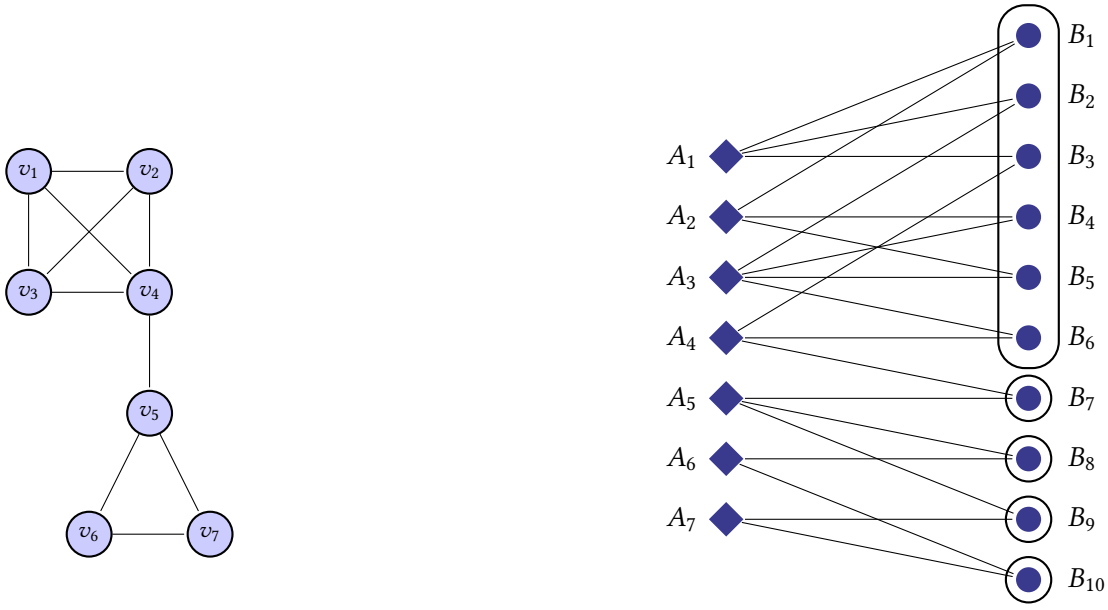

Fig. 2. There is a Clique instance with $k = 4$ on the left. The corresponding CCDM instance is presented on the right. The CCDM instance has 7 applications (each of which corresponds to a node on the left) and 10 databases (each of which corresponds to an edge on the left). All applications have unit testing cost, and all databases have unit size. The call relationship is captured with the bipartite graph. In the CCDM instance, we have a total of 5 shifts. The size of the first shift is $\frac{k(k-1)}{2} = 6$, and the size of each of the remaining 4 shifts is 1. In the optimal solution to the CCDM instance, the databases $B_1, \ldots, B_6$ are assigned to the first shift, and the remaining databases are assigned to the remaining shifts separately. In the optimal solution, the applications A_1, A_2, A_3, and A_4 are tested once since all the databases they call are assigned to the first shift. Each of the remaining applications is tested twice. And hence, the total application testing cost of the optimal solution is 10.

We next show that there is a k-clique in G, if and only if the corresponding CCDM instance has a solution with a total application testing cost of $2 \cdot |E| - k^2 + 2 \cdot k$. Let c^* denote the cost of the

minimum-cost solution for the constructed CCDM instance. Since each database is called by exactly 2 applications, we have $c^* > 2 \cdot (|E| - \frac{k \cdot (k-1)}{2}) = 2 \cdot |E| - k^2 + k$. This follows from the fact that both of the applications calling a database assigned to one of the $|E| - \frac{k \cdot (k-1)}{2}$ shifts with size 1 must be tested immediately after the migration of that shift, leading to a test cost of $2(|E| - \frac{k \cdot (k-1)}{2}) = 2 \cdot |E| - k^2 + k$. For the first shift, whose size is $\frac{k \cdot (k-1)}{2}$, we have a total application test cost of k, if and only if the vertices in G corresponding to the databases in this shift induce a clique of size k. This follows trivially from the fact that the number of edges in a k-clique is $\frac{k \cdot (k-1)}{2}$. Thus, $c^* \geq 2 \cdot |E| - k^2 + 2 \cdot k$ and $c^* = 2 \cdot |E| - k^2 + 2 \cdot k$ if and only if G has a clique of size k. Since all values in the reduction are polynomially bounded, it follows that the problem in CCDM$\langle const \mid const \mid arb \rangle$ is strongly **NP-hard**, even when each database is called by at most two applications. □

5 LOWER BOUND ON THE RUNNING TIME OF EXACT ALGORITHMS

In this section, we present a lower bound on the running time of any exact algorithm for the CCDM problem. In particular, we show that, unless the Exponential Time Hypothesis (ETH) fails, there is no $2^{o(n)}$ time algorithm for the CCDM problem. In fact, we show that, unless the ETH fails, there is no $2^{o(n)}$ time algorithm for the CCDM2 problem. Note that this lower bound rules out a $O\left(2^{(n^{1-\epsilon})}\right)$ algorithm for any $\epsilon > 0$.

Consider a 3CNF formula Φ with m' clauses and n' variables. From Φ we construct a CCDM2 instance I as follows:

(1) Create two shifts S_1 and S_2 of size n'.
(2) For each variable x_i, create two databases B_i^+ and B_i^-, both of which has size 1. Additionally, create applications $A_{i,1}$ through $A_{i,m'-1}$ that calls the databases B_i^+ and B_i^-. The testing cost of each of these applications is 1.
(3) For each pair of variables x_i and x_j, $i < j$, and each clause ϕ_k:
   (a) If ϕ_k uses the literals x_i and x_j, then create the following applications with a testing cost of 1:
      (i) The application $A_{i,j,k}^{+-}$ calling the databases B_i^+ and B_j^-.
      (ii) The application $A_{i,j,k}^{-+}$ calling the databases B_i^- and B_j^+.
      (iii) The application $A_{i,j,k}^{--}$ calling the databases B_i^- and B_j^-.
   (b) If ϕ_k uses the literals x_i and $\neg x_j$, then create the following applications with a testing cost of 1:
      (i) The application $A_{i,j,k}^{++}$ calling the databases B_i^+ and B_j^+.
      (ii) The application $A_{i,j,k}^{-+}$ calling the databases B_i^- and B_j^+.
      (iii) The application $A_{i,j,k}^{--}$ calling the databases B_i^- and B_j^-.
   (c) If ϕ_k uses the literals $\neg x_i$ and x_j, then create the following applications with a testing cost of 1:
      (i) The application $A_{i,j,k}^{++}$ calling the databases B_i^+ and B_j^+.
      (ii) The application $A_{i,j,k}^{+-}$ calling the databases B_i^+ and B_j^-.
      (iii) The application $A_{i,j,k}^{--}$ calling the databases B_i^- and B_j^-.
   (d) If ϕ_k uses the literals $\neg x_i$ and $\neg x_j$, then create the following applications with a testing cost of 1:
      (i) The application $A_{i,j,k}^{++}$ calling the databases B_i^+ and B_j^+.
      (ii) The application $A_{i,j,k}^{+-}$ calling the databases B_i^+ and B_j^-.
      (iii) The application $A_{i,j,k}^{-+}$ calling the databases B_i^- and B_j^+.

(e) If ϕ_k does not use both of the variables, then create the following applications with a testing cost of 1:
  (i) The application $A^{++}_{i,j,k}$ calling the databases B^+_i and B^+_j.
  (ii) The application $A^{+-}_{i,j,k}$ calling the databases B^+_i and B^-_j.
  (iii) The application $A^{-+}_{i,j,k}$ calling the databases B^-_i and B^+_j.
  (iv) The application $A^{--}_{i,j,k}$ calling the databases B^-_i and B^-_j.

From Φ, we also create the CCDM2 instance I' with the same shifts and databases as I, but with the following applications:

(1) For each variable x_i, create the application $A_{i,m'}$ with a testing cost of 1 that calls the databases B^+_i and B^-_i.
(2) For each clause $\phi_k = (l_i \vee l_j \vee l_h)$, $i < j < h$, create:
  (a) The application $A^{s_i s_j}_{i,j,k}$ calling databases $B^{s_i}_i$ and $B^{s_j}_j$ where s_i and s_j are the signs of literals l_i and l_j ($s_i = +$ if l_i is a positive literal and $s_i = -$ if l_i is a negative literal).
  (b) The application $A^{s_i s_h}_{i,h,k}$ calling databases $B^{s_i}_i$ and $B^{s_h}_h$.
  (c) The application $A^{s_j s_h}_{j,h,k}$ calling databases $B^{s_j}_j$ and $B^{s_h}_h$.

Note that I' has $(n' + 3 \cdot m')$ applications. Additionally, note that no application in I' is in I and that, when considering both I and I', there are a total of m' applications using each pair of databases. Thus, there are a total of $(m' \cdot \binom{2 \cdot n'}{2} - 3 \cdot m' - n')$ applications in I.

We now show that Φ is NAE-satisfiable, if and only if the databases in I can be migrated with a total testing cost of at most $(m' \cdot \binom{2 \cdot n'}{2} + m' \cdot n'^2 - 5 \cdot m' - 2 \cdot n')$.

LEMMA 5.1. *Let Φ be a 3CNF formula and let I and I' be the CCDM2 instances created from Φ by the above construction. Φ is NAE-satisfiable, if and only if the databases in I can be migrated with a testing cost of at most $(m' \cdot \binom{2 \cdot n'}{2} + m' \cdot n'^2 - 5 \cdot m' - 2 \cdot n')$.*

PROOF. Note that the databases in I can be migrated with a total application testing cost of at most $(m' \cdot \binom{2 \cdot n'}{2} + m' \cdot n'^2 - 5 \cdot m' - 2 \cdot n')$, if and only if at most $(m' \cdot n'^2 - 2 \cdot m' - n')$ applications in I need to be tested twice. In any database migration schedule, a total of $m' \cdot n'^2$ applications need to be tested twice if we consider both the applications in I and the applications in I'. Thus, migration of the databases in I has a total testing cost of at most $(m' \cdot \binom{2 \cdot n'}{2} + m' \cdot n'^2 - 5 \cdot m' - 2 \cdot n')$, if and only if at least $(2 \cdot m' + n')$ applications in I' need to be tested twice.

Let M be a migration of the databases such that at least $(2 \cdot m' + n')$ applications in I' need to be tested twice. Let $\mathbf{x}$ be an assignment to the variables in Φ such that x_i is assigned **true** if database B^+_i is assigned to shift S_1 and **false** otherwise.

Consider the clause $\phi_k = (l_i \vee l_j \vee l_h)$. Let s_i, s_j, and s_h be the signs of literals l_i, l_j, and l_h. If the databases $B^{s_i}_i$, $B^{s_j}_j$, and $B^{s_h}_h$ are all assigned to the same shift, then none of the applications $A^{s_i s_j}_{i,j,k}$, $A^{s_i s_h}_{i,h,k}$, $A^{s_j s_h}_{j,h,k}$ need to be tested twice. If the databases are assigned to separate shifts, then it must be the case that two databases are assigned to one shift and one database is assigned to the other shift. Thus, two of these applications will need to be tested twice and one application will need to be tested once. Consequently, no migration will test more than $2 \cdot m'$ of the applications in I' associated with clauses of Φ twice.

Thus, M needs to test all n' applications in I' associated with the variables of Φ twice. This means that for each i, the databases B^+_i and B^-_i are assigned to separate shifts. Thus, if x_i is assigned **true**, then $B^+_i \in S_1$ and $B^-_i \in S_2$. Conversely, if x_i is assigned **false**, then $B^+_i \in S_2$ and $B^-_i \in S_1$. From the arguments made previously, at least two of the applications associated with the clause ϕ_k need to be tested twice. Thus, at least one of the literals in ϕ_k is assigned **true** and at least one of the literals in ϕ_k is assigned **false**. Consequently, $\mathbf{x}$ NAE-satisfies Φ.

Now assume that $\mathbf{x}$ is an assignment that NAE-satisfies Φ. From $\mathbf{x}$ we construct a migration M of the databases in I as follows:

For each variable x_i:

(1) If x_i is assigned **true** by $\mathbf{x}$, then assign B_i^+ to shift S_1 and B_i^- to shift S_2.
(2) If x_i is assigned **false** by $\mathbf{x}$, then assign B_i^+ to shift S_2 and B_i^- to shift S_1.

Note that M tests all n' applications in I' associated with the variables of Φ twice.

Consider the clause $\phi_k = (l_i \vee l_j \vee l_h)$. Let s_i, s_j, and s_h be the signs of literals l_i, l_j, and l_h. Since $\mathbf{x}$ NAE-satisfies Φ, at least one literal in ϕ_k is **true** and at least one literal is **false**. If all three databases $B_i^{s_i}$, $B_j^{s_j}$, and $B_h^{s_h}$ were assigned to the same shift, then by construction of M, $\mathbf{x}$ would either set all three literals in ϕ_k to **true** or it would set all three literals in ϕ_k to **false**. Since $\mathbf{x}$ NAE-satisfies Φ, this cannot happen. Thus, the databases $B_i^{s_i}$, $B_j^{s_j}$, and $B_h^{s_h}$ need to be assigned to separate shifts. Consequently, two of the applications $A_{i,j,k}^{s_i s_j}$, $A_{i,h,k}^{s_i s_h}$, $A_{j,h,k}^{s_j s_h}$ need to be tested twice. This means that M tests $2 \cdot m'$ of the applications in I' associated with clauses of Φ twice.

A total of $(n' + 2 \cdot m')$ applications in I' need to be tested twice. From the arguments uses previously, this means that migrating the databases in I has total cost at most $(m' \cdot \binom{2 \cdot n'}{2} + m' \cdot n'^2 - 5 \cdot m' - 2 \cdot n')$ as desired. □

From Lemma 5.1, the CCDM2 problem cannot be solved in time $2^{o(n)}$ unless the ETH fails.

THEOREM 5.2. *The CCDM2 problem cannot be solved in time $2^{o(n)}$, unless the ETH fails.*

PROOF. Let Φ' be a 3-CNF formula with m^* clauses and n^* variables. From Φ' we can easily construct a 3-CNF formula Φ with $n' \in O(m^*)$ variables and $m' \in O(m^*)$ clauses such that Φ is NAE-satisfiable if and only if Φ' is satisfiable [27].

From Φ, we can construct a CCDM2 instance I with $n = 2 \cdot n' \in O(m^*)$ databases. From Lemma 5.1, Φ is NAE-satisfiable, if and only if the databases in I can be migrated with cost at most $(m' \cdot \binom{2 \cdot n'}{2} + m' \cdot n'^2 - 5 \cdot m' - 2 \cdot n')$. Thus, if an algorithm can solve the CCDM2 problem in $2^{o(n)}$ time, then 3-SAT could be solved in $2^{o(m)}$ time.

From the Sparsification Lemma, this violates the ETH [16]. □

Corollary 5.3 below generalizes Theorem 5.2 for all of the models of the CCDM problem.

COROLLARY 5.3. *There is no $2^{o(n)}$ algorithm for any of the problems in CCDM$\langle * \mid * \mid * \rangle$, unless the ETH fails. This holds even when there are only two shifts, and each application calls at most two databases.*

PROOF. Recall that CCDM$\langle * \mid * \mid * \rangle$ represents all of the models in our framework. Notice that the CCDM2 problem trivially reduces to all the problems in CCDM$\langle * \mid * \mid * \rangle$, even with the additional restrictions required by this corollary. The corollary follows directly from Theorem 5.2. □

6 APPROXIMATION COMPLEXITY

This section is devoted to approximability and inapproximability results for the CCDM problem. We first present Algorithm 1, which is a randomized algorithm for the problems in CCDM$\langle * \mid const \mid const \rangle$ when each application calls at most two databases. Throughout this section, let k be the number of shifts.

Algorithm 1 Approximation algorithm for CCDM$\langle * \mid const \mid const \rangle$ when each application calls at most two databases.

Function MIN-TEST-COST($\langle \mathbf{c}, \mathbf{w}, \mathbf{D}, \mathbf{l} \rangle$, ϵ)
if $(n < 1 + \frac{k-1}{k \cdot \epsilon})$ **then**
  Find the optimal solution by brute force.
else
  Let $i = 1$.
  while (there are unassigned databases) **do**
    Select $\frac{n}{k}$ of the unassigned databases uniformly at random.
    Assign the selected databases to the shift i.
    $i = i + 1$.
  end while
end if

Theorem 6.1 establishes the approximation factor of Algorithm 1 by bounding the expected number of times each application is tested in the solution constructed by Algorithm 1.

THEOREM 6.1. *For any given $\epsilon > 0$, Algorithm 1 returns a solution whose expected total application testing cost is at most $(\frac{2 \cdot k-1}{k} + \epsilon)$ times that of the optimum, for the CCDM problem under the models $\langle * \mid const \mid const \rangle$, when each application calls at most two databases.*

PROOF. Since Algorithm 1 finds the optimal solution in polynomial time by brute force if $n < 1 + \frac{k-1}{k \cdot \epsilon}$, in the rest of the proof, we will assume the contrary, i.e., $\epsilon \geq \frac{k-1}{k \cdot (n-1)}$.

Let X_i be a random variable that denotes the number of times application A_i is tested in the solution generated by Algorithm 1. Notice that $X_i \in \{1, 2\}$ for all i, since each application calls at most two databases. The total application testing cost of the solution returned by Algorithm 1 is $\sum_{i=1}^{m} c_i \cdot X_i$. Note that $\sum_{i=1}^{m} c_i$ is a lower bound for the cost of the optimal solution, since each application has to be tested at least once. To complete the proof, all we need is to show that $\mathbb{E}(\sum_{i=1}^{m} c_i \cdot X_i) \leq \left(\frac{2 \cdot k-1}{k} + \epsilon\right) \sum_{i=1}^{m} c_i$. Due to linearity of expectations, it suffices to show that $\mathbb{E}(X_i) \leq \left(\frac{2 \cdot k-1}{k} + \epsilon\right)$ for any i. If A_i calls exactly one database then this inequality is trivially satisfied since $\mathbb{E}(X_i) = 1$. So, in the rest of the proof, without loss of generality, we only consider applications that call exactly two databases.

Let A_i be an application that calls databases B_j and B_l. Let E_i denote the event that databases B_j and B_l are assigned to the same shift. Accordingly, $\overline{E_i}$ is the event that the databases B_j and B_l are assigned to different shifts. $\mathbb{E}(X_i)$ can be bounded as in the following.

$$
\begin{aligned}
\mathbb{E}(X_i) &= 1 \cdot \mathbf{Pr}(E_i) + 2 \cdot \mathbf{Pr}(\overline{E_i}) \\
&= 1 \cdot \left(\frac{\frac{n}{k}-1}{n-1}\right) + 2 \cdot \left(1 - \left(\frac{\frac{n}{k}-1}{n-1}\right)\right) \\
&= 2 - \left(\frac{\frac{n}{k}-1}{n-1}\right) = 2 - \frac{n-k}{k\cdot(n-1)} \\
&= 2 - \left(\frac{n-1}{k\cdot(n-1)} + \frac{1-k}{k\cdot(n-1)}\right) \\
&= 2 - \frac{n-1}{k\cdot(n-1)} + \frac{k-1}{k\cdot(n-1)} \\
&= 2 - \frac{1}{k} + \frac{k-1}{k\cdot(n-1)} \\
&= \frac{2\cdot k-1}{k} + \frac{k-1}{k\cdot(n-1)} \\
&\leq \frac{2\cdot k-1}{k} + \epsilon, \text{ as desired.}
\end{aligned}
$$

□

The Minimum-Bisection problem and its variants are among the most intriguing problems in the area of approximation algorithms [17]. This is because very little is known regarding their approximability. For instance, though the best approximation algorithm for the Minimum-Bisection problem achieves a guarantee of $O(\log n)$ [24], there is no result that rules out the possibility of a PTAS. In the Weighted Minimum-Bisection problem, we are given an edge-weighted undirected graph $G = (V, E)$, and the goal is to partition V into V_1 and V_2 such that $||V_1| - |V_2|| \leq 1$, and the sum of the weights of the edges with endpoints in different sets is as small as possible. Theorem 6.2 below establishes that **APX-hardness** of a particular variant of the CCDM problem implies the same for the Weighted Minimum-Bisection problem.

THEOREM 6.2. *If any problem in CCDM⟨∗ | const | const⟩ is* **APX-hard***when there are two shifts and each application calls at most two databases, then the Weighted Minimum-Bisection problem is* **APX-hard**.

PROOF. Let P be a problem in CCDM⟨∗ | *const* | *const*⟩ when there are two shifts and each application calls at most two databases. Assume that P is **APX-hard**. All we need is to prove that the Weighted Minimum-Bisection problem is **APX-hard** under this assumption. We will do that via a PTAS reduction to the weighted Minimum Bisection problem.

Given an instance I of P, we construct a corresponding Weighted Minimum-Bisection instance F as in the following:

(1) For each database B_i of I, create a vertex v_i in F, i.e., the vertex set of F is $V = \{v_1, \cdots, v_n\}$.
(2) For each application A_i of I that calls exactly two databases, say B_j and B_k, create the (v_j, v_k) edge in F with weight c_i.
(3) If there are multiple edges between a pair of vertices in F, replace them with a single edge whose weight is the sum of the weights of the replaced edges.

Given I, one can construct F in polynomial-time as described above and the size of F is no more than the size of I. Thus this construction forms the function f required for a PTAS reduction.

Let o and o' denote the value of the optimal solutions for the instances F and I, respectively. Let C_T represent the total cost of testing the applications in I once, i.e., $C_T = \sum_{A_i \in I} c_i$. Notice that

$o' = o + C_T$. Without loss of generality, assume that $o > 0$. This is because it is trivial to decide whether the optimal solution to a Weighted Minimum-Bisection instance is 0 or not.

For any $\epsilon > 0$, let a be the $(1+\epsilon)$-approximate solution for instance F returned by the PTAS for the Weighted Minimum-Bisection problem. We construct the solution a' for I using a as follows. For each vertex $v_i \in F$, if $v_i \in V_1$ in a then assign the corresponding database B_i to the first shift in a'. Otherwise, assign B_i to the second shift in a'. This forms the function g required for a PTAS reduction.

Let z and z' denote the objective function values of a and a', respectively. Let $\alpha(\epsilon) = \epsilon$ be the last function required for a PTAS reduction. All we need to show is $\frac{z'}{o'} \leq 1 + \alpha(\epsilon) = (1+\epsilon)$. To do this we use the facts that $z' = z + C_T$ and $o' = o + C_T$. Thus,

$$\frac{z'}{o'} = \frac{z + C_T}{o + C_T} < \frac{z}{o} \leq 1 + \epsilon$$

holds since $C_T > 0$ and $z \geq o$.

This means that the reduction from P to the Weighted Minimum-Bisection problem is a PTAS reduction. Thus, if P is **APX-hard**, then so is the Weighted Minimum-Bisection problem. □

7 THEORETICAL RESULTS FOR THE HYPERGRAPH PARTITIONING PROBLEM

The theoretical results we obtained for the CCDM problem imply new theoretical results for the classical Hypergraph Partitioning (HGP) problem and its variants. This section is devoted to derive the new theoretical results for the HGP problem and its variants. First, we define the classical balanced k-way HGP problem **P1** and its two variants **P2** and **P3**. While **P2** is supported by two state-of-the-art HGP tools PaToH[7] and KaHyPar [15], **P3** is introduced in this paper.

An undirected hypergraph H is a binary pair (V, N), where V is a set of vertices with weights $w : V \rightarrow \mathcal{R}_{\geq 0}$, and N is a set of nets (hyperedges) with weights $c : N \rightarrow \mathcal{R}_{\geq 0}$.

A k-way partition $\Pi = \{V_1, V_2, \ldots, V_k\}$ of V into k blocks is called *ϵ-balanced* if each block satisfies the balance constraint $\sum_{v \in V_i} w(v) \leq (1+\epsilon) W_{avg}$ where $W_{avg} = \frac{\sum_{v \in V} w(v)}{|V|}$, and ϵ is called the maximum imbalance ratio.

Classical Balanced k-way Hypergraph Partitioning Problem (P1): Given an undirected hypergraph $H = (V, N, w, c)$, an integer k, and a maximum imbalance ratio ϵ, Classical Balanced k-way Hypergraph Partitioning Problem is to find a ϵ-balanced k-way partition Π that minimizes an objective function defined over the nets.

Variable Block Size k-way Hypergraph Partitioning Problem (P2): Given an undirected hypergraph $H = (V, N, w, c)$, a block size upper-bound vector $\mathcal{L} = \langle L_1, L_2, \ldots, L_k \rangle$, Variable Block Size k-way Hypergraph Partitioning Problem is to find a k-way partition Π that minimizes an objective function defined over the nets such that the total weight of the vertices in each block is at most the corresponding block size upper bound.

Variable Block Size at most k-way Hypergraph Partitioning Problem (P3): Given an undirected hypergraph $H = (V, N, w, c)$, a block size upper-bound vector $\mathcal{L} = \langle L_1, L_2, \ldots, L_k \rangle$, Variable Block Size k-way Hypergraph Partitioning Problem is to find a k'-way partition Π for some $k' \leq k$ that minimizes an objective function defined over the nets such that the total weight of the vertices in each block is at most the corresponding block size upper bound.

There are three common objective functions used in the hypergraph partitioning context. These are the *connectivity*, *cut*, and *fan-out* objective functions. For a given solution Π, let $\lambda(e)$ denote the number of blocks that contains a vertex incident on hyperedge e. The connectivity (*con*) objective function is defined as $con(\Pi) = \sum_{e \in N} c(e)(\lambda(e) - 1)$. The *fan-out* objective function is defined as $fan-out(\Pi) = \sum_{e \in N} c(e)\lambda(e)$. The *cut* objective function is defined as $cut(\Pi) = \sum_{e \in N_C} c(e)$, where

N_C denotes the set of hyperedges whose incident vertices are placed into at least two distinct blocks. Optimizing the *fan-out* and *con* objective functions are equivalent since their values differ by exactly $\sum_{e \in N} c(e)$. Also, when $k = 2$, the *con* and *cut* objective functions are equivalent, since for each hyperedge e, we have $\lambda(e) \in \{1, 2\}$. In the rest of the section, when we say "some objective function", we mean any one of the *con*, *cut*, and *fan-out* objective functions.

Theorem 7.2, Proposition 7.3, and Proposition 7.4 relate the variants of the HGP problem and the Weighted Minimum-Bisection problem from the perspectives of approximation complexity and computational complexity of exact algorithms. The formal definition of the Weighted Minimum-Bisection problem is given below.

Definition 7.1 (Weighted Minimum-Bisection). Given an undirected graph $G = (V, E)$ where E is a set of edges with weights $w : E \rightarrow \mathcal{R}_{\geq 0}$, partition V into two subsets V_1 and V_2 such that $||V_1| - |V_2|| \leq 1$ and the total weight of the edges with one endpoint in V_1 and one endpoint in V_2 is minimized.

THEOREM 7.2. *If there is an α-approximation algorithm for **P2** for some objective function, then there is an α-approximation algorithm for **P3** for the same objective function. Furthermore, for any function f, if there is an $O(f(n))$-time exact algorithm for **P2** for some objective function, then there is a $O^*(f(n))$-time exact algorithm for **P3** for the same objective function.*

PROOF. First, let us first fix an objective function for each HGP problems. Assume that there is an α-approximation algorithm A for **P2**. We need to show that there is an α-approximation algorithm B for **P3**. Given a **P3** instance $(H, \mathcal{L} = \langle L_1, L_2, \ldots, L_k \rangle)$, algorithm B first constructs at most k **P2** instances and solves each **P2** instance using algorithm A. Algorithm B then constructs the **P2** instances as followings:

- Sort the block size upper-bound vector $\mathcal{L}$ in decreasing order and relabel blocks in this order.
- Let i be the smallest integer such that the total size of the first i blocks is greater than or equal to the sum of the weights of all the vertices.
- For each j such that $i \leq j \leq k$, create the **P2** instance $(H, \mathcal{L}_j = \langle L_1, L_2, \ldots, L_j \rangle)$.

Algorithm B then solves each of the $(k - i + 1)$ **P2** instances by using algorithm A. It then returns the best solutions found for these $(k - i + 1)$ instances.

Let o be an optimal solution for the given **P3** instance. Without loss of generality, assume that o is a k'-way partition of vertices for some $k' \leq k$. The proof follows since the objective function value of o is equal to the objective function value of the optimal solution for the **P2** instance $(H, \mathcal{L}_{k'})$.

Let us now assume that there is an $O(f(n))$-time exact algorithm A' for **P2**. We need to show that there is an $O^*(f(n))$-time exact algorithm B' for **P3**. B' is the same as B except that it uses A' as a subroutine instead of A. Thus, the running time of B' is $O(k \cdot f(n)) \subseteq O^*(f(n))$. □

PROPOSITION 7.3. *If there is an α-approximation algorithm for **P2** for some objective function, then there is an α-approximation algorithm for **P1** for the same objective function. Furthermore, for any function f, if there is an $O(f(n))$-time exact algorithm for **P2** for some objective function, then there is a $O(f(n))$-time exact algorithm for **P1** for the same objective function.*

PROOF. The proof immediately follows since **P1** is the special case of the **P2** where the size of each block is $\dfrac{\sum_{v \in V} w(v) \cdot (1 + \epsilon)}{k}$. □

PROPOSITION 7.4. *If there is an α-approximation algorithm for **P1** (or **P3**) for some objective function, then there is an α-approximation algorithm for the Weighted Minimum-Bisection problem as well. Furthermore, for any function f, if there is an $O(f(n))$-time exact algorithm for **P1** (or **P3**)*

for an objective function, then there is a $O(f(n))$-time exact algorithm for the Weighted Minimum-Bisection problem.

PROOF. The proof immediately follows since the Weighted Minimum-Bisection problem is the special case of both **P1** and **P3**. □

Corollary 7.6 below establishes that there is no $2^{o(n)}$ algorithm for the Weighted Minimum-Bisection problem, unless the ETH fails. This statement is true even if the weights of the edges are given in unary notation. This result immediately follows from Theorem 5.2 and Theorem 7.5 below.

THEOREM 7.5. *If there is a $2^{o(n)}$ algorithm for the Weighted Minimum-Bisection problem, then there is a $2^{o(n)}$ algorithm for the CCDM2 problem.*

PROOF. Assume that there is a $2^{o(n)}$ exact algorithm A for the Weighted Minimum-Bisection problem. All we need is to show that there is a $2^{o(n)}$ exact algorithm B for the CCDM2 problem. Given a CCDM2 instance I, algorithm B first constructs a Weighted Minimum-Bisection instance F, then solves F by using A, and then constructs a solution for I from the solution for F. Given I, construction of F is as follows:

- For each database B_j of I, create a vertex v_j in F.
- For each application A_i that calls B_j and B_k of I, create the edge $e = (v_j, v_k)$ with weight 1.
- If there are multiple edges between a pair of vertices in F, then replace all of these edges with a single edge whose weight is equal to the number of constituent edges.

Notice that the size of F is at most the size of I, and the above construction takes polynomial time. B obtains an optimal solution for F by using A. In the optimal solution for F, some vertices are assigned to V_1 and some vertices are assigned to V_2. In the solution to I, B assigns the databases corresponding to the vertices in V_1 to the first shift, and the databases corresponding to the vertices in V_2 to the second shift. The theorem follows since this mapping takes polynomial time, and the optimal solution for F corresponds to the optimal solution for I. □

COROLLARY 7.6. *There is no $2^{o(n)}$ algorithm for the Weighted Minimum-Bisection problem, unless the ETH fails. This statement is true even if the weights of the edges are given in unary notation.*

PROOF. The first statement immediately follows from the conjunction of the statements of Theorem 5.2 and Theorem 7.5. The result also holds even if the weights of the edges are given in unary notation since in the reduction given in the proof of Theorem 7.5, the size of the Weighted Minimum-Bisection instance F is equal to the size of the CCDM2 instance I if the weights of the edges in F are given in unary notation. □

Theorem 7.7 below establishes $\mathbf{AC^0}$-equivalence between the CCDM problem and **P3** under the *fan-out* (and thus, *con*) objective function. It is instrumental in obtaining the fixed-parameter intractability results for the HGP variants given by Corollary 7.8 and Corollary 7.9.

THEOREM 7.7. *The CCDM problem is $\mathbf{AC^0}$-equivalent to **P3** under the fan-out (and thus, con) objective function.*

PROOF. To prove the theorem, we first give an $\mathbf{AC^0}$-reduction from the CCDM problem to **P3** under the *fan-out* objective function. Then, we present an $\mathbf{AC^0}$-reduction from **P3** under the *fan-out* objective function to the CCDM problem.

Given a CCDM instance I, we construct a **P3** instance under the *fan-out* objective function in parallel by a CREW-PRAM machine as follows:

- For each database B_j of I, use a single processor to create the vertex v_j with weight w_j. This requires a total of n processors.
- For each application A_i of I, use as many processors as the number of databases A_i calls to create a hyperedge e_i. This requires a total of at most $n \cdot m$ processors.
- For each shift i, use a single processor to create a block i with size l_i. This requires a total of k processors.

Each of these steps takes constant time. Thus, the **P3** instance can be constructed in constant time using a CREW-PRAM machine with at most $(n \cdot m + n + k)$ processors. Note that a CREW-PRAM is a parallel random access machine whose processors can simultaneously read data from the same location but cannot simultaneously write data to the same location. Therefore, this is a $\mathbf{AC^0}$-reduction.

Given a **P3** instance under the *fan-out* objective function, we construct a CCDM instance I in parallel by a CREW-PRAM machine as follows:

- For each vertex v_j of F, use a single processor to create the database B_j with weight w_j. This requires a total of n processors.
- For each hyperedge e_i of F, use as many processors as the number of vertices e_i is incident on to create an application A_i. This requires a total of at most $n \cdot m$ processors.
- For each block i, use a single processor to create a shift i with size L_i. This requires a total of k processors.

Each of these steps takes constant time. Thus, the CCDM instance can be constructed in constant time using a CREW-PRAM machine with at most $(n \cdot m + n + k)$ processors. Therefore, this is a $\mathbf{AC^0}$-reduction. □

COROLLARY 7.8. *Both **P2** and **P3** under any objective function are fixed-parameter intractable, when parameterized by the number of hyperedges. This statement is true even if $k = 2$.*

PROOF. Recall that for $k = 2$, the *cut* and the *con* objective functions are equivalent. Thus, for $k = 2$, the CCDM problem is $\mathbf{AC^0}$-equivalent to **P3** under any objective function due to Theorem 7.7. Recall from Theorem 4.1 that the CCDM problem is **NP-hard** even when the number of applications $|\mathcal{A}|$ is 1. In the $\mathbf{AC^0}$-reduction used in the proof of Theorem 7.7, the number of applications of the CCDM instance coincides with the number of hyperedges of the **P3** instance. Thus, **P3** under any objective function is fixed-parameter intractable, when parameterized by the number of hyperedges. From Theorem 7.2, this result also holds for **P2**. □

COROLLARY 7.9. *Both **P2** and **P3** under the con and fan-out objective functions are fixed-parameter intractable, when parameterized by the maximum degree of any vertex. This statement is true even if both the vertices and the hyperedges are unweighted.*

PROOF. The CCDM problem is $\mathbf{AC^0}$-equivalent to **P3** under the *con* and *fan-out* objective functions due to Theorem 7.7. Recall from Theorem 4.3 that the CCDM problem is **NP-hard** even if each database is called by at most two applications. In the $\mathbf{AC^0}$-reduction used in the proof of Theorem 7.7, the number of applications calling a database of the CCDM instance coincides with degree of the corresponding vertex of the **P3** instance. Thus, **P3** under the *con* and *fan-out* objective functions is fixed-parameter intractable, when parameterized by the maximum degree of any vertex. Note that the CCDM instance constructed in the proof of Theorem 4.3 is an instance of the problem in CCDM$\langle const \mid const \mid arb \rangle$. Additionally, in the proof of Theorem 7.7 application testing costs coincide with hyperedge weights, and database sizes coincide with vertex weights. Thus, this result holds even if both the vertices and the hyperedges are unweighted. From Thereom 7.2, this result holds for **P2** as well. □

8 EMPIRICAL STUDY

In this section, we first review the heuristic algorithms developed for **P2**. Then, we describe how we adapt them for **P3**. Finally, we evaluate their performances on randomly generated $1,080$ instances for each of the 12 CCDM models.

8.1 The Hypergraph Partitioning Problem and Heuristics

As we showed in the previous section, the CCDM problem is $\mathbf{AC^0}$-equivalent to **P3** under the *con* and *fan-out* objective functions. As shown in the proof of Theorem 7.2, a solution to an instance of **P3** can be found using the solvers for **P2**. So, we devote this subsection to a review of heuristics developed for **P2**.

Since the Classical Balanced k-way Hypergraph Partitioning Problem is **NP-hard** [9], a large amount of effort is put into heuristic algorithms [3, 4, 7, 15, 18, 19, 28] to tackle the problem in practice. Most of these heuristic algorithms use *multilevel* hypergraph partitioning scheme, which consists of the following three stages: *coarsening*, *initial partitioning*, *uncoarsening*. In the coarsening stage, vertices or nets of the hypergraph are contracted to obtain a series of smaller hypergraphs. In the initial partitioning stage, an initial partition is obtained by partitioning the smallest hypergraph either by computing k-way partition directly or using recursive bisectioning until k blocks are found. In the uncoarsening stage, the initial partition uncoarsened back to obtain solutions to the larger hypergraphs until a solution to the original hypergraph is obtained. A local search procedure is used at each level of the uncoarsening stage for a more global view. The most commonly used local search procedure is the FM [8] heuristic. There are various tools for the **P1** problem, but only PaToH and KaHyPar support the **P2** problem. So, below we describe the components of these tools and how we adapt them to solve the CCDM problem.

PaToH uses a multilevel hypergraph partitioning scheme and recursive bisectioning to find an initial partition and a variation of the FM heuristic in the uncoarsening stages. It provides three different settings: speed, default, and quality. We use it on both default and quality settings with their default parameters. We refer to them as PaToH-D and PaToH-Q, respectively.

KaHyPar is an n-level hypergraph partitioning framework developed from a series of papers [3, 4, 10, 14, 15, 28]. The latest version of the direct k-way partitioner in the KaHyPar framework, which is referred to as kKaHyPar, uses an n-level hypergraph partitioning scheme. As the local search algorithm, it uses an FM-based heuristic along with the Minimum Flow Refinement heuristic (MFR) [10]. There is also a meta-heuristic in the KaHyPar framework, which is referred to as kKaHyPar-E, that employs a sophisticated genetic algorithm that uses recombination and mutation operators specifically tailored for the HGP problem to explore the solution space efficiently (for more details, see [4]). In more recent work, new problem specific recombination and mutation operators are combined with kKaHyPar-E to create a more effective algorithm. We denote this algorithm with kKaHyPar-EBQ [1].

While kKaHyPar, kKaHyPar-E and kKaHyPar-EBQ support variable block sizes, PaToH-D and PaToH-Q can take *target* block sizes. Target block sizes do not correspond to the maximum block size; rather, they are the desired sizes of the blocks. All heuristics use all k blocks necessarily since they find k-way partitions. However, in the CCDM problem, not all shifts need to be used. Furthermore, our initial experiments showed that PaToH-D and PaToH-Q might compromise solution quality to respect the target block sizes. Therefore, we made the following adaptations to the heuristic algorithms.

Adaptations: We first sort the blocks in the decreasing order of their sizes. Let $L_i = \sum_{j=1}^{i} l_i$, where l_i denotes size of the block i. Additionally, let $W_t = \sum_{v \in V} w(v)$. Let x be the smallest integer such that $L_x \geq W_t$. In our experiments, we run kKaHyPar-E and kKaHyPar-EBQ multiple times with

different numbers of blocks ranging between x and k. In order to prevent PaToH-D and PaToH-Q from compromising solution quality to respect the target block sizes, we run them multiple times with different numbers of blocks. For each number of blocks i such that $L_i \geq W_t$, we first run PaToH-D and PaToH-Q with the first i blocks having their original block sizes as their target block sizes. We then decrease the size of the last block by one and run it again until either the new total block size is less than the total vertex size, or the last block size is less than 0 to obtain more diverse partitions. To make a fair comparison between PaToH-D, PaToH-Q, and kKaHyPar, we run kKaHyPar the same number of times for each number of blocks but without changing the last block size.

8.2 Instance Generation and Experimental Setup

The parameters that are the same for all 12 of the models are described below.

- Number of databases (n): We select 3 values for $n : 10, 15, 20$.
- Number of applications (m): We select 3 values for $m : 2 \cdot n, 3 \cdot n, 5 \cdot n$.
- Number of shifts (k): We select 4 values for $k : 2, 3, 4, 5$.
- Number of databases called by applications (p): For each application, p is chosen uniformly at random from $\{2, 3, 4, 5\}$ and then p databases are selected uniformly at random.

For each of these parameters we generate 10 instances, which adds up to $360 (= 3 \cdot 3 \cdot 4 \cdot 10)$ instances. We ensure that each database is called by at least one application. Database sizes and application testing costs in these instances are chosen according to their respective models. For models with arbitrary database sizes or application testing costs, they are chosen uniformly at random from $\{1, \ldots, 10\}$, and they are chosen as 1 for the models with constant database sizes or constant application testing costs. For models with proportional testing costs, the cost of testing an application A_i is the total size of the databases that A_i calls.

3 shift sizes with different levels of imbalance are chosen for all of the generated instances to extend them into a total of 1080 instances for each model, and 12960 instances in total.

For the models with uniform shift sizes and constant database sizes, we choose shift sizes as $\lceil W_{avg} \rceil + \epsilon$, $\forall \epsilon = 0, 1, 2$. For the models with uniform shift sizes and arbitrary database sizes, we choose shift sizes as $\lceil (1 + \epsilon) \cdot W_{avg} \rceil$, $\forall \epsilon = 0.1, 0.2, 0.3$.

For the models with non-uniform shift sizes and constant database sizes, for shift k, $l_k = \lceil W_{avg} \rceil + \epsilon_k$, where ϵ_k is chosen uniform at random from $\{0, 1, 2\}$, $\{0, \ldots, 4\}$, $\{0, \ldots, 6\}$. For the models with non-uniform shift sizes and arbitrary database sizes, for shift i, $l_i = \lceil (1 + \epsilon_i) \cdot W_{avg} \rceil$, where ϵ_i is chosen uniform at random from $[0, 0.2]$, $[0, 0.4]$, $[0, 0.6]$. Optimal solutions for these instances are found by solving the Integer Linear Program (ILP) given in Appendix 2.1.

All experiments are performed on the same computer that has a 64-bit AMD Ryzen 7 2700X CPU and 32GB DDR4 3200 MHz RAM running Ubuntu 18.04.02. The ILP given in appendix 2.1 is implemented in Java version 8 and solved using CPLEX version 12.6.2. The ILP uses all 8 cores with 16 threads, while the heuristic algorithms use a single core.

8.3 Performance of the Heuristic Algorithms

Recall that we generated a total of $12,960$ instances of the CCDM problem. On each of these instances, we ran CPLEX on the ILP given in Appendix 2.1 to generate an optimal solution. For each heuristic, we used the relative gap between the solution generated by the heuristic and the optimal solution generated by CPLEX to measure the quality of that heuristic.

Several of the $12,960$ total instances were discarded for a variety of reasons. 7 instances were discarded because CPLEX declared them to be infeasible. An additional 13 instances were discarded because CPLEX was unable to solve them due to lack of memory. Another 28 instances were discarded because databases could be migrated in a single shift. Finally, 608 instances were discarded

because at least one of the heuristics was unable to find a feasible solution. Table 1 details the distribution of these discarded instances.

Model	PaToH-D			PaToH-Q			kKaHyPar			kKaHyPar-E			kKaHyPar-EBQ		
	inf	avg	max	inf	avg	max	inf	avg	max	inf	avg	max	inf	avg	max
$\langle const \mid const \mid const \rangle$	0	1.34	10.29	0	1.09	8.33	0	2.15	15.58	0	0.52	8.16	1	0.36	4.35
$\langle prop \mid const \mid const \rangle$	0	1.53	8.76	0	1.20	8.76	0	2.31	17.92	0	0.65	6.88	0	0.49	6.88
$\langle arb \mid const \mid const \rangle$	0	1.52	11.50	0	1.20	11.50	0	2.15	17.37	0	0.59	8.91	1	0.47	6.47
$\langle const \mid arb \mid const \rangle$	47	2.55	18.18	61	2.20	14.05	9	2.37	17.31	10	0.72	10.53	15	0.54	10.53
$\langle prop \mid arb \mid const \rangle$	48	2.52	15.03	82	2.35	16.42	16	2.84	13.78	13	0.84	7.57	26	0.56	7.48
$\langle arb \mid arb \mid const \rangle$	50	2.93	16.77	69	2.44	21.31	16	2.46	16.72	14	0.71	8.68	24	0.50	9.29
$\langle const \mid const \mid arb \rangle$	0	1.59	12.68	0	1.26	12.68	0	3.18	39.13	0	0.86	8.82	7	0.54	10.00
$\langle prop \mid const \mid arb \rangle$	0	1.36	13.28	0	1.09	11.23	0	5.23	47.59	0	0.81	10.07	9	0.42	9.01
$\langle arb \mid const \mid arb \rangle$	0	1.75	15.84	0	1.38	22.98	0	3.26	41.13	0	0.88	8.91	11	0.53	6.52
$\langle const \mid arb \mid arb \rangle$	31	3.17	17.91	44	2.73	18.64	22	3.33	22.00	19	1.60	13.85	40	1.29	9.86
$\langle prop \mid arb \mid arb \rangle$	28	2.93	18.62	59	2.88	18.68	36	3.69	22.75	27	1.67	16.68	48	1.25	16.68
$\langle arb \mid arb \mid arb \rangle$	41	3.67	20.28	58	2.99	20.28	27	3.51	20.55	31	1.69	12.44	66	1.39	12.44
$\langle * \mid * \mid * \rangle$	245	2.20	20.28	373	1.87	22.98	126	3.03	47.59	114	0.95	16.68	248	0.68	16.68

Table 1. Table shows the number of infeasible instances, average and maximum relative gaps from the optimal solution for each algorithm

To evaluate the performance of heuristic algorithms, for each instance, we divide the solutions found by heuristic algorithms according to the optimal solution for that instance. We use geometric mean to compute the average performances of heuristic algorithms. Table 1 shows the average and worst case performances of each algorithm for each CCDM model on the remaining 12, 304 instances.

Model	PaToH-D			PaToH-Q			kKaHyPar			kKaHyPar-E			kKaHyPar-EBQ		
	opt	avg	max	opt	avg	max	opt	avg	max	opt	avg	max	opt	avg	max
$\langle const \mid * \mid * \rangle$	37.52	2.13	18.18	46.06	1.79	18.64	34.89	2.75	39.13	60.00	0.92	13.85	67.77	0.67	10.53
$\langle prop \mid * \mid * \rangle$	33.19	2.06	18.62	43.72	1.84	18.68	27.99	3.51	47.59	51.99	0.98	16.68	61.82	0.67	16.68
$\langle arb \mid * \mid * \rangle$	29.73	2.42	20.28	40.81	1.96	22.98	30.10	2.83	41.13	52.21	0.95	12.44	60.70	0.71	12.44
$\langle * \mid const \mid * \rangle$	41.66	1.52	15.84	51.87	1.20	22.98	35.01	3.03	47.59	59.37	0.72	10.07	69.42	0.47	10.00
$\langle * \mid arb \mid * \rangle$	24.59	2.96	20.28	34.46	2.59	21.31	26.66	3.03	22.75	49.75	1.20	16.68	56.95	0.92	16.68
$\langle * \mid * \mid const \rangle$	32.12	2.04	18.18	41.93	1.72	21.31	32.55	2.37	17.92	59.68	0.67	10.53	68.32	0.48	10.53
$\langle * \mid * \mid arb \rangle$	34.89	2.37	20.28	45.17	2.02	22.98	29.45	3.70	47.59	49.76	1.23	16.68	58.50	0.88	16.68
$\langle * \mid * \mid * \rangle$	33.49	2.20	20.28	43.54	1.87	22.98	31.01	3.03	47.59	54.76	0.95	16.68	63.45	0.68	16.68

Table 2. Table shows the percentage of instances solved optimally, average and maximum relative gaps from the optimal solution for each algorithm for CCDM models grouped by each parameter

Table 2 shows the percentage of instances solved optimally, average and worst case performances of each algorithm for CCDM models grouped by each parameter. As seen in Table 2, the heuristic algorithms perform better on CCDM instances in which database size is constant than they do on instances in which database size is arbitrary. The performances of PaToH-D and PaToH-Q are affected more compared to the KaHyPar variants for the database size parameter. Additionally, the heuristics perform better on instances with constant shift size than they do on instances with arbitrary shift size. This time, the performances of KaHyPar variants are affected more than PaToH variants for the shift size parameter. The application testing cost parameter has the least effect on the performance of the heuristics. Even though, heuristics perform worse, when the application testing costs are proportional or arbitrary compared to the constant, its effect is lower than database size and shift

size parameter. These results show that as the problem gets harder, the performance of the heuristics degrades as expected.

On average, PaToH-Q returns solutions closer to the optimum solution than PaToH-D. However, PaToH-Q failed to find a feasible solution for more instances than PaToH-D did. The kKaHyPar-EBQ returns solutions closer to the optimum than any other heuristic while finding the second highest number of infeasible solutions.

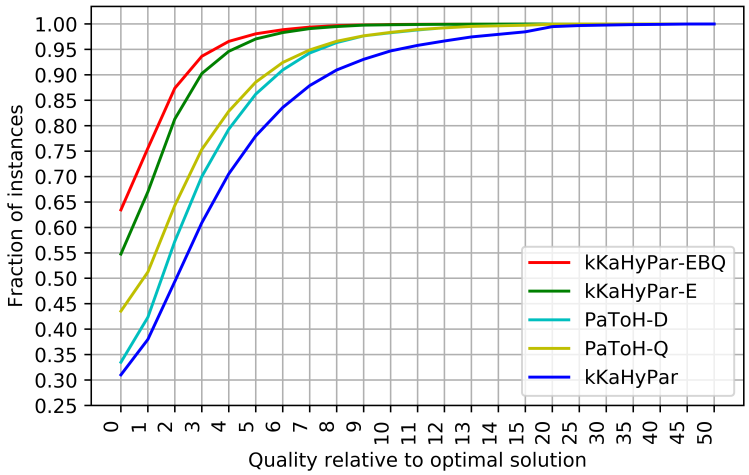

Fig. 3. Relative gap of all algorithms with respect to the fraction of the instances

Figure 3 shows that KaHyPar-EBQ outperforms all other algorithms and solves 63.45% of the instances optimally. PaToH-D, PaToH-Q, kKaHyPar and kKaHyPar-E solve 33.49%, 43.54%, 31.01%, 54.76% of the instances optimally, respectively. PaToH-Q, PaToH-D, kKaHyPar, kKaHyPar-E, and kKaHyPar-EBQ find solutions that are 5.78%, 5.36%, 7.69%, 2.97% and 2.34% within the optimal in 90% of the instances, respectively.

	Algorithm				
Algorithm	PaToH-D	PaToH-Q	kKaHyPar	KaHyPar-E	KaHyPar-EBQ
PaToH-D	-	12.17	38.66	9.92	6.45
PaToH-Q	30.48	-	42.17	11.58	6.80
KaHyPar	29.35	18.57	-	0.03	0.24
KaHyPar-E	51.48	39.47	51.38	-	1.66
KaHyPar-EBQ	55.49	43.90	54.56	18.77	-

Table 3. Each row of the table corresponds to the percentage of instances that algorithm find better solutions compared to the other algorithm

Table 3 shows that kKaHyPar-E finds solutions that are closer to optimal than the solutions found by PaToH-D, PaToH-Q, kKaHyPar and kKaHyPar-EBQ in 6,334, 2,285, 6,322 and 204 instances, respectively. However, PaToH-Q finds solutions that are closer to optimal than the solutions found by

Model	Algorithm							
	PaToH-D		PaToH-Q		kKaHyPar		ILP	
	avg.	max	avg.	max	avg.	max	avg.	max
$\langle const \mid const \mid const \rangle$	0.95	5	10.67	56	97.36	580	25030.97	2667573
$\langle prop \mid const \mid const \rangle$	0.99	6	11	55	99.06	587	31403.13	9262589
$\langle arb \mid const \mid const \rangle$	0.97	6	10.80	56	97.19	584	18825.80	2961430
$\langle const \mid arb \mid const \rangle$	3.09	16	32.91	165	265.27	1264	11785.15	1792407
$\langle prop \mid arb \mid const \rangle$	3.47	19	39.17	209	267.96	1405	24738.52	3819110
$\langle arb \mid arb \mid const \rangle$	3.22	16	34.70	178	263.57	1283	13939.25	1995377
$\langle const \mid const \mid arb \rangle$	1.15	9	12.18	104	115.93	1176	41284.85	11262781
$\langle prop \mid const \mid arb \rangle$	1.97	17	21.12	147	197.86	1535	17416.48	1522711
$\langle arb \mid const \mid arb \rangle$	1.15	10	12.50	109	116.58	1180	21787.17	2620246
$\langle const \mid arb \mid arb \rangle$	3.19	20	33.37	214	270.45	1814	62326.42	10482791
$\langle prop \mid arb \mid arb \rangle$	4.02	33	44.28	429	313.45	2981	58715.74	4965643
$\langle arb \mid arb \mid arb \rangle$	3.32	22	35.85	225	273.78	1864	66833.88	16052122
$\langle * \mid * \mid * \rangle$	2.24	33	24.33	429	194.59	2981	32445.66	16052122

Table 4. Table shows the average and maximum running times in milliseconds.

PaToH-D, kKaHyPar, kKaHyPar-EBQ in $3,750$, $5,188$ $1,425,837$ instances, respectively. Therefore, to find better solutions using these heuristics, it might be better to combine kKaHyPar-EBQ and PaToH-Q instead of kKaHyPar-EBQ and kKaHyPar-E.

We run PaToH-D, PaToH-Q, and kKaHyPar more times for the instances with arbitrary database sizes compared to the instances with constant database sizes. This has a noticeable effect on the average running times. Table 4 shows that for the instances with arbitrary database sizes, heuristic algorithms take 2.5 to 3 times more time when compared to the instances with uniform database sizes. Since we run PaToH-D, PaToH-Q, and kKaHyPar the same number of times, we can compare their running times directly. PaToH-D is approximately 11 times faster than PaToH-Q and 87 times faster than kKaHyPar on average. The average running time of CPLEX on the ILP is 33.2 seconds, whereas the maximum running time is slightly less than 4.5 hours. It should not be forgotten that CPLEX uses 16 threads, whereas each heuristic algorithm uses a single thread. So, direct comparison is not possible.

9 CONCLUSION AND FUTURE RESEARCH DIRECTIONS

This paper introduced the CCDM problem, a general framework that is suitable for modeling the database migration needs of a variety of enterprises with customized constraints. We showed that the CCDM problem is **NP-hard** for all the models, even under the very restricted scenario, where there are only 2 shifts and each application is calling at most 2 databases. We also studied the parameterized complexity of the CCDM problem for four relevant parameters and presented fixed-parameter intractability results for all of them. We have also presented a randomized $(\frac{2 \cdot k-1}{k} + \epsilon)$-approximation algorithm for an interesting but a quite restricted special case of the CCDM problem and showed that **APX-hardness** of the problems in CCDM$\langle * \mid const \mid const \rangle$, with the restrictions that there are two shifts and each application calls at most two databases, implies the same for the Weighted Minimum-Bisection problem. We showed that our theoretical results imply new theoretical results for variants of the classical Hypergraph Partitioning problem. On the experimental front, we tested the performance of the problem specific adaptations of the well-known heuristics PaToH-D,

PaToH-Q, and kKaHyPar as well as the state-of-the-art meta-heuristics kKaHyPar-E and kKaHyPar-EBQ developed for the HGP problem on randomly generated instances of the CCDM problem. Our results indicate that the adaptations of kKaHyPar-EBQ outperform the other algorithms. kKaHyPar-EBQ found solutions that are 0.68% within optimal on average, and it solved 63.45% of the instances optimally.

Every model of the CCDM problem is an interesting combinatorial optimization problem by itself. It would be interesting to know for which models of the CCDM problem there are low factor approximation algorithms. From our perspective, the following avenues of research are interesting:

(1) Derandomizing the randomized approximation algorithm: The approximation algorithm in this paper randomly assigns databases to each shift. This lets us derive a bound on the expected cost to test each application based on the probability that the two databases used by that application are assigned to the same shift. Note that if certain complexity theoretic assumptions hold true, then any randomized algorithm can be derandomized with only a polynomial slowdown [5]. However, for each method of derandomizing this algorithm, we have found a set of applications for which the algorithm does not guarantee the desired approximation bound.
(2) Designing approximation algorithms and/or obtaining inapproximability results for all the models of the CCDM problem: The randomized algorithm in this paper assumes that each database and shift has constant size, and that each application uses at most two databases. We plan to develop approximation algorithms for models of the CCDM problem without these restrictions. Additionally, we plan to develop inapproximability results that are not dependent on the Weighted Minimum-Bisection problem.
(3) Establishing fixed-parameter tractability or intractability for all models of the CCDM problem - In this paper, we showed that the 6 problems in CCDM$\langle * \mid arb \mid * \rangle$ are fixed parameter intractable when parameterized by the number of applications. We plan to extend these results to the remaining CCDM problems (those in CCDM$\langle * \mid const \mid * \rangle$). Additionally, we plan to investigate the parameterized complexity of the CCDM problem using alternative parameters, such as the maximum number of applications that calls a database.

REFERENCES

[1] Utku Umur Acikalin and Bugra Caskurlu. 2022. Multilevel Memetic Hypergraph Partitioning with Greedy Recombination. https://doi.org/10.48550/ARXIV.2204.03730
[2] Utku Umur Acikalin, Bugra Caskurlu, Piotr J. Wojciechowski, and K. Subramani. 2021. New Results on Test-Cost Minimization in Database Migration. In *Algorithmic Aspects of Cloud Computing - 6th International Symposium, ALGOCLOUD 2021, Lisbon, Portugal, September 6-7, 2021, Revised Selected Papers (Lecture Notes in Computer Science)*, Gianlorenzo D'Angelo and Othon Michail (Eds.), Vol. 13084. Springer, 38–55. https://doi.org/10.1007/978-3-030-93043-1_3
[3] Yaroslav Akhremtsev, Tobias Heuer, Peter Sanders, and Sebastian Schlag. 2017. Engineering a direct k-way Hypergraph Partitioning Algorithm. In *19th Workshop on Algorithm Engineering and Experiments, (ALENEX 2017)*. 28–42.
[4] Robin Andre, Sebastian Schlag, and Christian Schulz. 2018. Memetic multilevel hypergraph partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 347–354.
[5] Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: a modern approach*. Cambridge University Press.
[6] Daniel Beimborn, Thomas Miletzki, and Stefan Wenzel. 2011. Platform as a service (PaaS). *Business & Information Systems Engineering* 3, 6 (2011), 381–384.
[7] Umit V Catalyurek and Cevdet Aykanat. 1999. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on parallel and distributed systems* 10, 7 (1999), 673–693.
[8] Charles M Fiduccia and Robert M Mattheyses. 1982. A linear-time heuristic for improving network partitions. In *19th design automation conference*. IEEE, 175–181.
[9] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[10] Lars Gottesbüren, Michael Hamann, Sebastian Schlag, and Dorothea Wagner. 2020. Advanced Flow-Based Multilevel Hypergraph Partitioning. In *18th International Symposium on Experimental Algorithms (SEA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[11] Mary Jean Harrold, James A Jones, Tongyu Li, Donglin Liang, Alessandro Orso, Maikel Pennings, Saurabh Sinha, S Alexander Spoon, and Ashish Gujarathi. 2001. Regression test selection for Java software. *ACM Sigplan Notices* 36, 11 (2001), 312–326.

[12] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. 2015. The rise of "big data" on cloud computing: Review and open research issues. *Information systems* 47 (2015), 98–115.

[13] Jer Hayes. 2015. Multimedia big data: Content analysis and retrieval. *Big-Data Analytics and Cloud Computing* (2015), 37–51.

[14] Tobias Heuer, Peter Sanders, and Sebastian Schlag. 2019. Network flow-based refinement for multilevel hypergraph partitioning. *Journal of Experimental Algorithmics (JEA)* 24 (2019), 1–36.

[15] Tobias Heuer and Sebastian Schlag. 2017. Improving coarsening schemes for hypergraph partitioning by exploiting community structure. In *16th International Symposium on Experimental Algorithms (SEA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[16] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. 2001. Which Problems Have Strongly Exponential Complexity? *J. Comput. System Sci.* 63, 4 (2001), 512 – 530.

[17] Marek Karpinski. 2002. Approximability of the minimum bisection problem: An algorithmic challenge. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 59–67.

[18] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1999. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7, 1 (1999), 69–79.

[19] George Karypis and Vipin Kumar. 2000. Multilevel k-way hypergraph partitioning. *VLSI design* 11, 3 (2000), 285–300.

[20] Alexandros Labrinidis and Hosagrahar V Jagadish. 2012. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment* 5, 12 (2012), 2032–2033.

[21] Ping Lu, Liang Zhang, Xiahe Liu, Jingjing Yao, and Zuqing Zhu. 2015. Highly efficient data migration and backup for big data applications in elastic optical inter-data-center networks. *IEEE Network* 29, 5 (2015), 36–42.

[22] Dimas C Nascimento, Carlos Eduardo Pires, and Demetrio Mestre. 2015. Data quality monitoring of cloud databases based on data quality SLAs. In *Big-data analytics and cloud computing*. Springer, 3–20.

[23] Sangameshwar Patil, Sasanka Roy, John Augustine, Amanda Redlich, Sachin Lodha, Harrick M Vin, Anand Deshpande, Mangesh Gharote, and Ankit Mehrotra. 2010. Minimizing Testing Overheads in Database Migration Lifecycle.. In *COMAD*. 191.

[24] Harald Räcke. 2008. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 255–264.

[25] YV Ravikumar, KM Krishnakumar, and Nassyam Basha. 2017. Oracle Database Migration. In *Oracle Database Upgrade and Migration Methods*. Springer, 213–277.

[26] Mario Santana. 2016. Infrastructure as a Service (IaaS). In *Cloud Computing Security: Foundations and Challenges*. CRC Press, 59.

[27] T.J. Schaefer. 1978. The complexity of Satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, Alfred Aho (Ed.). ACM Press, New York City, NY, 216–226.

[28] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. 2016. k-way Hypergraph Partitioning via *n*-Level Recursive Bisection. In *18th Workshop on Algorithm Engineering and Experiments, (ALENEX 2016)*. 53–67.

[29] K Subramani, Bugra Caskurlu, and Alvaro Velasquez. 2018. Minimization of testing costs in capacity-constrained database migration. In *International Symposium on Algorithmic Aspects of Cloud Computing*. Springer, 1–12.

[30] Silvia Regina Vergilio, José Carlos Maldonado, Mario Jino, and Inali Wisniewski Soares. 2006. Constraint based structural testing criteria. *Journal of Systems and Software* 79, 6 (2006), 756–771.

[31] W Eric Wong, Joseph R Horgan, Aditya P Mathur, and Alberto Pasquini. 1999. Test set size minimization and fault detection effectiveness: A case study in a space application. *Journal of Systems and Software* 48, 2 (1999), 79–89.

[32] Linlin Wu, Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. 2014. SLA-Based Resource Provisioning for Hosted Software-as-a-Service Applications in Cloud Computing Environments. *IEEE Transactions on Services Computing* 3, 7 (2014), 465–485.

[33] Xiaonian Wu, Mengqing Deng, Runlian Zhang, Bing Zeng, and Shengyuan Zhou. 2013. A task scheduling algorithm based on QoS-driven in cloud computing. *Procedia Computer Science* 17 (2013), 1162–1169.

[34] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. 2014. Internet of things for smart cities. *IEEE Internet of Things journal* 1, 1 (2014), 22–32.