Mathematical Programming Models Based on Hub Covers in Graph Query Processing

Belma Yelbay, Ş. İlker Birbil and Kerem Bülbül

Sabancı University, Industrial Engineering, Orhanlı-Tuzla, 34956 Istanbul, Turkey.

byelbay@sabanciuniv.edu, sibirbil@sabanciuniv.edu, bulbul@sabanciuniv.edu

ABSTRACT: The use of graph databases for social networks, images, web links, pathways and so on, has been increasing at a fast pace and promotes the need for efficient graph query processing on such databases. In this study, we discuss graph query processing – referred to as graph matching – and an inherent optimization problem known as the minimum hub cover problem. This optimization problem is introduced to the literature as a new graph representation model to expedite graph queries. With this representation, a graph database can be denoted by a small subset of graph vertices. Searching over only this subset of vertices decreases the response time of a query and increases the efficiency of graph query processing. We also discuss that finding the minimum hub cover alone does not guarantee the efficiency of graph matching computations. The order in which we match the query vertices also affects the cost of querying. For this purpose, we propose a shortest path formulation which gives a hub cover and a search sequence yielding a least cost query. The minimum hub cover problem is $\mathcal{NP}$-hard and there exist just a few studies related to the formulation of the problem and the associated solution approaches. In this study, we present new alternate models and partially fill that gap. Similar to the other problems in the $\mathcal{NP}$-hard class, solving large-scale MHC instances may prove very difficult. Therefore, we introduce relaxations and some rounding heuristics to find optimal or near optimal solutions fairly quickly. We provide a new binary integer programming formulation as well as a quadratic integer programming formulation. Our relaxation for the quadratic model leads to a semidefinite programming formulation. A solution to the minimum hub cover problem can be obtained by solving the relaxations of the proposed models and then rounding their solutions. We introduce two rounding algorithms to be applied to the optimal solutions of the linear programming and semidefinite programming relaxations. In addition, we also implement two other well-known rounding algorithms for the set covering formulation of the problem. Our computational study demonstrates that the results of the rounding algorithms obtained with the relaxations of the proposed mathematical models are better than those obtained with the standard set covering formulation.

Keywords: graph query processing, minimum hub cover, linear programming, semidefinite programming, rounding heuristics

1. Introduction.

Graph databases store relational data in various applications such as social networks, web, protein interactions, image processing and so on. In a graph database, vertices and edges represent entities and relationships, respectively. For readers not familiar with the subject, graph query processing, known as graph matching is to a one-to-one mapping between the vertices of two graphs under a set of label constraints. In other words, querying a graph database refers to searching for a structural similarity between two graphs. For instance, Figure 1 demonstrates a query and a database graph of two molecular compounds. Note that the database graph on the right has a subgraph, which is structurally identical to the query graph on the left. Therefore, carrying out a query with this subgraph returns a positive response.

Managing very large graph databases has two main obstacles: (i) large memory requirement to store a graph (ii) high query response times. In the recent years, these obstacles and the increasing popularity of graph databases have attracted many researchers to graph query processing. Existing state-of-the-art techniques such as Ullmann (Ullmann, 1976), VFLib (Cordella et al., 2001), VF2 (Cordella et al., 2004), GraphQL (He and Singh, 2008), QuickSI (Shang et al., 2008), TALE (Tian and Patel, 2008), GADDI (Zhang et al., 2009), SPath (Zhao and Han, 2010) and SAPPER (Zhang and Jin, 2010) are all developed to

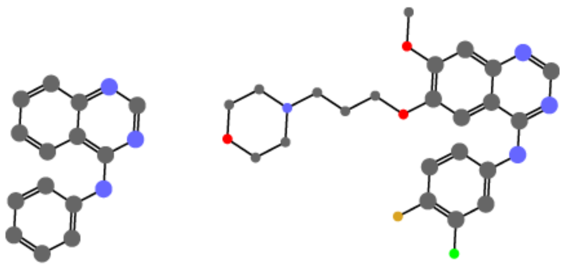


Figure 1: A query and a database graph of a molecular compound (Kawabata, 2014).

increase the efficiency of graph query processing. Among those studies, Jamil (2011) has introduced a new graph representation model to expedite graph queries. With this representation, a graph database can be represented by a small subset of graph vertices, which has a low memory footprint. Moreover, searching over only that subset of vertices decreases the response time of a query and increases the efficiency of graph query processing as illustrated by Rivero and Jamil (2014, 2016). Jamil (2011); Rivero and Jamil (2014, 2016) propose a graph matching algorithm, which uses the solution of an optimization problem to find out the subset of vertices to represent a query graph and to prune the search space. Rivero and Jamil (2016) show that proposed algorithm using the solution of that optimization problem outperforms the contemporary algorithms, i.e., VF2, GraphQL, QuickSI, GADDI and SPath for larger graphs with different structures. Yelbay et al. (2013) formalize the problem of finding a representative subgraph of the query graph including the minimum number of vertices as an optimization problem referred to as the minimum hub cover (MHC) problem. The objective of the MHC problem is to cover all edges of a graph by selecting the minimum number of vertices. Selecting a vertex covers both its incident edges and the edges between its adjacent neighbors. For instance in Figure 2, vertex g covers edges $(f,g),(e,g),(c,g)$ and (c,e). The formal definition of the problem follows.

Definition 1.1 *Let $G=(V,E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. A subset of the vertices, $HC \subseteq V$ is a* hub cover *of G if for every edge $(i,j) \in E$, either $i \in HC$ or $j \in HC$ or there exists a vertex k such that $(i,k) \in E$ and $(j,k) \in E$ with $k \in HC$. The* MHC problem *is about finding a hub cover that has the minimum number of vertices.*

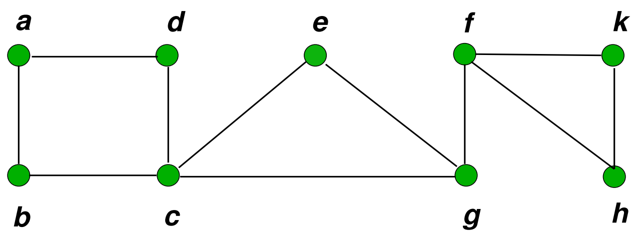


Figure 2: A sample graph for the minimum hub cover problem.

The efficiency of the graph matching algorithm proposed by Yelbay et al. (2013); Rivero and Jamil (2014, 2016) depends on the performance of the solution method of the MHC problem (in terms of solution time and quality). The MHC problem is known to be $\mathcal{NP}$-hard in the strong sense (Yelbay et al., 2013). It remains $\mathcal{NP}$-hard even when restricted to planar graphs (Yelbay et al., 2016). Therefore, one may resort to algorithms that yield good, preferably near-optimal solutions, fairly quickly. In the first category, we have approximation algorithms with proven performance bounds. To this date, the only approximation result for the MHC problem has been given by Yelbay et al. (2016). This result applies exclusively to planar graphs. In the second category, we can consider fast heuristic approaches. Yelbay et al. (2013) implement two greedy algorithms and one mathematical programming-based heuristic. Their results indicate that the mathematical programming-based heuristic outperforms the greedy algorithms in terms of the optimality gap. With this motivation, in this paper we focus on introducing alternate mathematical programming models and their relaxations to solve the MHC problem efficiently. For large-scale graphs, in which the MHC problem cannot be solved to optimality, we introduce new mathematical programming relaxations and rounding heuristics. Our motivation is two-fold: First, with the advances in the computational machinery, very large instances of mathematical programming relaxations can be solved efficiently. Second, strong formulations coupled with rounding-based relaxation heuristics provide good feasible solutions in a reasonable time.

The performance of the proposed graph matching algorithm also depends on the order of the vertices matched (Rivero and Jamil, 2014, 2016). Processing two successive vertices corresponds to joining those two vetices, and the cost of the join operation can be computed by multiplying the number of the candidate vetices in the database graph that can be matched to those query vertices. Since each search order may entail very different computation times when performing the graph matching task, Rivero and Jamil (2014, 2016) compute all possible orderings of MHCs and their associated query costs and select the search order with the least query cost. The focus of this study is on effective solution methods for the MHC problem with the intent of decreasing the total cost of querying. In a nutshell, we make the following research contributions: (i) We introduce a new binary integer programming model along with a quadratic integer programming model for the MHC problem. The relaxation of the former formulation is a linear programming (LP) problem, while the relaxation of the latter gives rise to a semidefinite programming (SDP) problem. (ii) We present several rounding heuristics to accompany the proposed relaxations. (iii) We conduct an extensive computational study to illustrate the empirical performance of the rounding heuristics. We also observe that the quality of the rounding algorithms may vary when applied to different relaxations. (iv) When the exact formulation has alternate optimal solutions, we propose an algorithm to generate all optimal MHCs. The algorithm relies on iteratively solving an integer programming model and removing the current optimal solution from the feasible region by adding a cut. After finding all optimal solutions, as proposed by Rivero and Jamil (2014), all possible orderings for each alternate solution may be generated and their querying costs may be calculated (Rivero and Jamil, 2016). (v) Currently, there is no study to generate the sequence of vertices with the least query cost. Rivero and Jamil (2016) state that finding the hub sequence with the least query cost is an interesting future research study. With this motivation, we introduce an integrated shortest path formulation to find both the minimum hub cover and the associated hub sequence.

2. Graph Query Processing.

Studies related to query processing can be categorized into two groups: exact and inexact graph matching algorithms. Exact graph matching algorithms are proposed to solve either the graph isomorphism problem (Cordella et al., 2001; Santo et al., 2003) or the more general subgraph isomorphism problem (Cordella et al., 2001; Lee et al., 2012; Lipets et al., 2009; Shang et al., 2008; Ullmann, 1976; Weber et al., 2012; Yelbay et al., 2013; Zhu et al., 2010). In this study, we specifically focus on subgraph isomorphism. Jamil (2011) and Yelbay et al. (2013) propose a *graphlet* representation, which keep topological data to help detect the most similar vertices between query and database graphs. Yelbay et al. (2013) devise a technique to perform graph matching that takes a query graph q, a data graph g, and an MHC M of q as input, and it outputs all of the possible matchings of q in g in four steps: (i) First, we compute an MHC for q. (ii) Next, we define the search space. That is, for all vertices of MHC we find out the structurally similar vertices in g. (iii) Then, we determine the order in which the query vertices are processed for matching. (iv) Finally, we construct a search tree to compute a graph matching. In the succeeding sections, we go through all these steps in detail.

2.1 MHC Computation. The first step is to find the vertices in the optimal or near-optimal solution of the MHC problem. We first state the set covering formulation of the problem introduced in (Yelbay et al., 2013) for the sake of a self-contained presentation. If an edge corresponds to an item, and a set is defined for each vertex whose elements are the edges covered by that vertex, then the connection between the set covering and the MHC problems can be easily established. This mathematical programming formulation is given as follows:

$$\begin{aligned} &\text{minimize} && \sum_{j\in V} x_j, && && (1)\\ &\text{subject to} && x_i + x_j + \sum_{k\in\mathcal{K}^{(i,j)}} x_k \geq 1, && (i,j)\in E, && (2)\\ & && x_j \in \{0,1\}, && j \in V. && (3) \end{aligned}$$

Here, x_j is a binary variable, which is equal to one when vertex j is selected. For $(i,j) \in E$, $\mathcal{K}^{(i,j)}$ denotes all those vertices $k \in V$ such that $(i,k) \in E$ and $(j,k) \in E$. The objective function (1) minimizes the number of the selected vertices. Constraints (2) ensure that every edge is covered by at least one vertex in the hub cover. Finally, constraints (3) enforce the binary restrictions on the variables.

The well-known minimum vertex cover problem is a special case of the MHC problem when the cardinality of the set $\mathcal{K}^{(i,j)}$ is zero; that is, $|\mathcal{K}^{(i,j)}| = 0$. The minimum vertex cover problem has a complementary problem formulation known as the maximum independent set problem. This relationship inspired us to introduce a new optimization problem, which we call the maximum triangular set (MTS) problem. The formal definition of MTS follows.

Definition 2.1 *For a given graph $G = (V,E)$, $TS \subseteq V$ is a triangular set if and only if for every edge (i,j) at most $|\mathcal{K}^{(i,j)}|+1$ of the vertices in $\bar{\mathcal{K}}^{(i,j)} := \mathcal{K}^{(i,j)} \cup \{i,j\}$ are also in TS. The MTS problem is about finding a triangular set which has the maximum number of vertices.*

The careful reader may notice that MTS is equivalent to MHC in the sense that the solution of one problem will yield a solution for the other one. This point is formalized in the next lemma.

Lemma 2.1 *In any graph $G = (V,E)$, HC is a hub cover in G if and only if $V \setminus HC$ is a triangular set and TS is a triangular set in G if and only if $V \setminus TS$ is a hub cover.*

Proof. Suppose TS is a triangular set in G. Then for any edge (i,j), at most $|\mathcal{K}^{(i,j)}|+1$ of the vertices in $\bar{\mathcal{K}}^{(i,j)}$ are in TS. Since the number of vertices that can cover edge (i,j) is equal to $|\mathcal{K}^{(i,j)}|+2$, at least one of the vertices in S must be in $V \setminus TS$. Thus, $V \setminus TS$ must be a hub cover. Conversely, suppose $V \setminus TS$ is a hub cover. Then, at least one of the vertices in $\bar{\mathcal{K}}^{(i,j)}$ must be in $V \setminus TS$ so that $V \setminus TS$ is a hub cover. That is for each edge, the cardinality of the subset of $\bar{\mathcal{K}}^{(i,j)}$ included in TS is less than $|\mathcal{K}^{(i,j)}| + 2$ and hence, TS is a triangular set. □

The mathematical programming formulation of MTS is given by

$$\text{maximize} \quad \sum_{j \in V} x_j, \tag{4}$$

$$\text{subject to} \quad x_i + x_j + \sum_{k \in \mathcal{K}^{(i,j)}} x_k \leq |\mathcal{K}^{(i,j)}| + 1, \qquad (i,j) \in E, \tag{5}$$

$$x_j \in \{0,1\}, \qquad j \in V, \tag{6}$$

where x_j is a binary variable that is equal to 1 when vertex j is selected. The objective function (4) maximizes the number of selected vertices. Constraints (5) ensure that the solution is a TS per Definition 2.1. The binary restrictions on the variables are enforced by the final set of constraints (6).

Our last reformulation is a quadratic integer program, where each term is a product of two binary variables. This formulation shall form the basis of the semidefinite programming relaxation that we will introduce in Section 3.1:

$$\text{minimize} \quad \sum_{j \in V} (1 + y_0 y_j)/2, \tag{7}$$

$$\text{subject to} \quad (y_0 - y_i)(y_0 - y_j) + (2y_0 - y_i - y_j) \sum_{k \in \mathcal{K}^{(i,j)}} (y_0 - y_k) \leq 8|\mathcal{K}^{(i,j)}|, \qquad (i,j) \in E, \tag{8}$$

$$y_j \in \{+1, -1\}, \qquad j \in V \cup \{0\}. \tag{9}$$

The optimal solution of the MHC problem is given by those vertices $j \in V$ such that $y_j = y_0$. The set of constraints (8) is obtained after simplifying the following constraint for each $(i,j) \in E$.

$$(y_0 - y_i)(y_0 - y_j) + \sum_{k \in \mathcal{K}^{(i,j)}} (y_0 - y_i)(y_0 - y_k) + \sum_{k \in \mathcal{K}^{(i,j)}} (y_0 - y_j)(y_0 - y_k) \leq 8|\mathcal{K}^{(i,j)}|. \tag{10}$$

The following example illustrates relation (10) on a clique of three vertices.

Example 2.1 *Suppose that we consider the clique consisting of the vertices i, j, and k. Since in a clique, every pair of vertices are connected by an edge, constraint (10) for edge (i,j) takes the form*

$$(y_0 - y_i)(y_0 - y_j) + (y_0 - y_i)(y_0 - y_k) + (y_0 - y_j)(y_0 - y_k) \leq 8. \tag{11}$$

Given $y_i \in \{-1, +1\}$, this constraint ensures that the solution $y_0 \neq y_i = y_j = y_k$ is infeasible. Thus, at least one of the three vertices is selected.

2.2 Search Space Computation. Graph matching algorithms, in general, construct a search tree, which stores all possible matchings to respond to a query. Since matching is performed between two similar vertices during graph matching computations, we need the search space computation to identify the vertices that are similarly structured between a query and a database graph. Jamil (2011) introduces a new graph representation to prune the search space and to decrease the possible number of matchings. Similarity among the vertices are determined based on their graphlet representations as defined as $r_v = \langle v, N_v, B_v \rangle$. In this representation, $v \in V$ is a vertex in graph $G(V,E)$; $N_v \subseteq V$, and $B_v \subseteq E$ are the set of immediate neighbors of vertex v and the edges between the vertices in N_v, respectively. If query processing is performed on graphs with labeled vertices, then we add one more dimension L_v to graphlet $r_v = \langle v, L_v, N_v, B_v \rangle$ in order to incorporate the vertex labels. Whether g is labeled or unlabeled, we take advantage of the representation of the graphs by means of hubs in the following way: let r_u and r_v be two hubs in q and g, respectively; vertex v belongs to the search space of u if and only if the number of neighbors/triangles of r_u is less or equal than that of r_v. Thanks to this property, we are able to prune the search space even when we are dealing with unlabeled graphs. Additionally, for labeled graphs, v belongs to the search space of u if and only if their labels match.

Each time we match a query vertex v in the search tree, we gradually construct a subgraph of a query graph by adding vertices v, N_v, the edges between v and N_v as well as the set of edges in B_v. Once we map all query vertices, we finish the construction of the the whole query graph and find a graph match. According to Definition 1.1, matching the set of vertices in a hub cover guarantees the coverage of a query graph and implies that a query graph can be represented by only the vertices in the hub cover rather than all query vertices. This results in a reduction in the number of query graphlets to be processed and the memory requirement to store the query graph. Therefore, it is very critical to solve the MHC problem efficiently to be able to get the maximum benefit in terms of both memory usage and query processing efficiency.

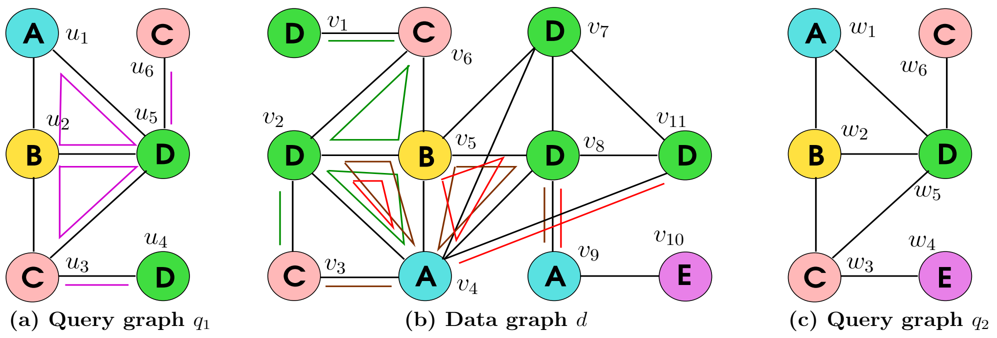


Figure 3: Query graphs q_1, q_2 and data graph d.

Example 2.2 *Let us determine the search space for the query graph in Figure 3(a) with respect to data graph in Figure 3(b). Notice that q_1 has multiple MHCs as (u_3, u_5) and (u_4, u_5).*

***Graphlet representation of query graph** q_1**, given MHC** (u_4, u_5)*

$r_{u_4} = \langle u_4, \{u_3\}, \{\emptyset\} \rangle$

$r_{u_5} = \langle u_5, \{u_1, u_2, u_3, u_6\}, \{(u_1, u_2), (u_2, u_3)\} \rangle$

***Graphlet representation of the data graph** d*

$r_{v_1} = \langle v_1, \{v_6\}, \{\emptyset\} \rangle$

$r_{v_2} = \langle v_2, \{v_3, v_4, v_5, v_6\}, \{(v_3, v_4), (v_4, v_5), (v_5, v_6)\} \rangle$

$r_{v_3} = \langle v_3, \{v_2, v_4\}, \{(v_2, v_4)\} \rangle$

$r_{v_4} = \langle v_4, \{v_2, v_3, v_5, v_7, v_8, v_{11}\}, \{(v_2, v_3), (v_2, v_5), (v_5, v_7), (v_5, v_8), (v_7, v_{11}), (v_8, v_{11})\} \rangle$

$r_{v_5} = \langle v_5, \{v_2, v_4, v_6, v_7, v_8\}, \{(v_1, v_2), (v_2, v_4), (v_4, v_7), (v_4, v_8), (v_7, v_8)\} \rangle$

$r_{v_6} = \langle v_6, \{v_1, v_2, v_5\}, \{(v_2, v_5)\} \rangle$

$r_{v_7} = \langle v_7, \{v_4, v_5, v_8, v_{11}\}, \{(v_4, v_5), (v_4, v_{11}), (v_5, v_8), (v_8, v_{11}\}$
$r_{v_8} = \langle v_8, \{v_4, v_5, v_7, v_9, v_{11}\}, \{(v_4, v_5), (v_4, v_7), (v_4, v_{11}), (v_5, v_7), (v_7, v_{11}\}$
$r_{v_9} = \langle v_9, \{v_8, v_{10}\}, \{\emptyset\}\rangle$
$r_{v_{10}} = \langle v_{10}, \{v_9\}, \{\emptyset\}\rangle$
$r_{v_{11}} = \langle v_{11}, \{v_4, v_7, v_8\}, \{(v_4, v_7), (v_4, v_8), (v_7, v_8)\}\rangle$

Taking boundaries and neighbors into account, the search space of our example is given as below. Notice that v_6 cannot be a candidate for u_5 because both the number of neighbors and boundary edges of u_5 are greater than that of v_6. Once we find a mapping for the graphlets r_{u_4} and r_{u_5}, the vertices u_4, u_5 and all their neighbors are matched so we do not have to keep the graphlet representations of the vertices other than u_4 and u_5.
$u_4 : \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$
$u_5 : \{v_2, v_4, v_5, v_7, v_8\}$

2.3 Query Plan Computation. The graph matching algorithm discussed in this study iteratively matches the vertices of the query and the database graph. It is well-known that different matching orderings may entail very different computation times when performing the graph matching task (He and Singh, 2008). To select an ordering, Yelbay et al. (2013); Rivero and Jamil (2016) first compute the ordering of all vertices of the query graph in a way similar to that in He and Singh (2008), in which a search order is evaluated by analyzing the costs of the joins of the query vertices. This search order is also called the query plan, which is represented as a binary tree in which the leaves are query vertices and the internal vertices denote the join operations. The cost of joining the first two query vertices u and v in the query plan is computed by multiplying the cardinalities of the set of candidates for u and v in the search space. The cost of joining one more vertex z is computed by multiplying the cost of the previous join operation with the size of the candidate vertices for z in the search space, and then applying a reduction factor γ^{α}, where α represents the number of edges between vertex z and its predecessors in the query plan and $\gamma \in (0, 1)$. The total cost of a query plan is the sum of the costs of all join operations. Rivero and Jamil (2016) approximate the reduction factor by a constant number 0.5 and select the least cost query plan p among all possible orderings of the MHCs. Rivero and Jamil (2016) also take into account all optimal MHCs, if the problem has alternate optimal solutions. The authors compute the least cost query plan for each optimal solution and select the least query plan among all. A way of obtaining all optimal MHCs was not stated in (Rivero and Jamil, 2016). Algorithm 1 takes care of this issue by solving the following integer programming (IP) formulation iteratively:

$$\text{minimize} \sum_{j \in V} x_j, \tag{12}$$

$$\text{subject to } x_i + x_j + \sum_{k \in \mathcal{K}^{(i,j)}} x_k \geq 1, \qquad (i, j) \in E, \tag{13}$$

$$\sum_{j \in HC^i} x_j \leq |HC^i| - 1, \qquad i \in \{1, \ldots, t-1\}, \tag{14}$$

$$x_j \in \{0, 1\}, \qquad j \in V. \tag{15}$$

The IP formulation (12)-(15) is solved to compute the t^{th} optimal solution by adding a set of constraints

(14) to the IP formulation (1)-(3). Here, HC^i includes all variables that are set to one in the ith optimal solution. Constraints (14) ensure that the optimal solution obtained at iteration t is different from those obtained at the previous iterations. Algorithm 1 iterates as long as the cardinality of the optimal solution is equal to that of the very first optimal solution.

Algorithm 1 Computing all optimal MHCs.

```
$t \leftarrow 1$
Solve the IP model (12)-(15)
$HC^t \leftarrow \{j \in V | x_j = 1\}$
while $|HC^t| = |HC^1|$ do
    Solve (12)-(15)
    $t \leftarrow t + 1$
    $HC^t \leftarrow \{j \in V | x_j = 1\}$
end while
return $\{HC^1, \ldots, HC^{t-1}\}$
```

Since the MHC problem is $\mathcal{NP}$-hard, solving the model (12)-(15) iteratively as in Algorithm 1 is impractical especially for large-scale query graphs. In this study, we present a novel approach to find the MHC that yields the least cost query cost. We formulate the problem of finding the least cost MHC query plan as a shortest path problem (SPP). In other words, with this formulation, we can find both the best set of vertices and the search order that yield the least cost query. The SPP has some side constraints to ensure that the vertices included in the shortest path is also a hub cover for the MHC problem. Before stating the formulation, we introduce some notation. Suppose $G = (V, E)$ denotes the query graph, where V and E represent the set of vertices and edges in G, respectively. For notational simplicity, in the rest of the paper we assume that query graph G is undirected so the edges (i, j) and (j, i) are equivalent. For each query graph G, we have an associated directed graph $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V}$ and $\bar{E}$ denote the set of vertices and edges in $\bar{G}$, respectively. $\bar{G}$ is a labeled graph; that is, each vertex in $\bar{G}$ has an associated label defined by a map $f : \bar{V} \mapsto V$ such that for every vertex $i \in \bar{V}$, there is a unique vertex $f(i) \in V$.

For each query graph G, we construct a directed graph $\bar{G}$ such that each path in $\bar{G}$ represents a query plan. The total length of a path is equal to the cost of the join operations, if the query search is performed according to the query plan specified by the path. The edge weights are computed based on the Cochran formula given in (Rivero et al., 2013). $\bar{G}$ is a layered graph such that each edge connects the vertices in successive layers. The set of vertices located in layer k is denoted as $\bar{V}_k$ and vertex $i \in \bar{V}_k$ if and only if $f(i) \in V$ is the kth vertex in the query plan of the path passing over vertex i.

Suppose we add a source vertex s and a sink vertex t to $\bar{V}$ such that $f(s) = f(t) = \emptyset$. For $k = 1$, we create n vertices for $\bar{V}_1$ such that $\bar{V}_1 = \{1, 2 \ldots, n\}$ and $f(i) = i$, for all $i \in \bar{V}_1$. We also create edges between s and all other vertices $i \in \bar{V}_1$. For $k > 1$, we create a new vertex j and an edge (i, j) for all $f(j) \in V$, if the following conditions are met:

(i) $i \in \bar{V}_k$ and $j \in \bar{V}_{k+1}$,

(ii) $f(r) \neq f(j)$ for all r on the path from s to j in $\bar{G}$

(iii) There exists at least one vertex $r \in \bar{V}_l$, $l < k$ on the path from s to j in $\bar{G}$ such that $(f(r), f(j)) \in$

E or there exists a vertex i such that $(f(r), i) \in E$ and $(f(j), i) \in E$.

The first condition ensures that the adjacent vertices in graph $\bar{G}$ will be searched successively. The second condition guarantees the uniqueness of the query vertices for all possible query plans. The final condition makes sure that two successive matching operations are connected. The arc weight of edge $w(i,j)$ is computed as given below:

(i) $w(s,j) = 0$ and $w(i,t) = 0$ for all i,j

(ii) $w(i,j) = |C_i| \times |C_j|$ for all $i \in \bar{V}_1$, where $|C_i|$ is the cardinality of the search space of vertex $f(i)$.

(iii) $w(i,j) = w(k,i) \times |C_j| \times 0.5^{\alpha}$, where α is the number of neighbors of $f(j)$ that are matched before the vertex j based on the query plan represented by the path from s to j and $k \in \bar{G}$ where $(k,i) \in \bar{E}$.

Figure 5 is a directed graph to find the least cost query plan for the query and the database graphs shown in Figure 4. In Figure 5, the vertices 1, 2, and 3 lying in the first layer represent the first query vertices that will be searched. The vertices 4 to 9 in $\bar{G}$ map to the vertices 2, 3, 1, 3, 1 and 2 in G. Similarly, the vertices 10 to 15 in $\bar{G}$ map to the vertices 3, 2, 3, 1, 2, and 1 respectively. Arc weight between vertices 1 and 4 is computed as the multiplication of the number of candidates for query vertices 1 and 2. Since only vertices B and D are candidates for vertex 2 and all the database vertices are candidates for 1, the arc weight is simply two times five. The arc weight between vertex 4 and 10 is the multiplication of $w(1,4)$ (10), the number of candidates for vertex 3 (5) and (0.5) since vertex 2 is the only neighbor of vertex 3 that is already matched. The weights of all other edges are computed accordingly.

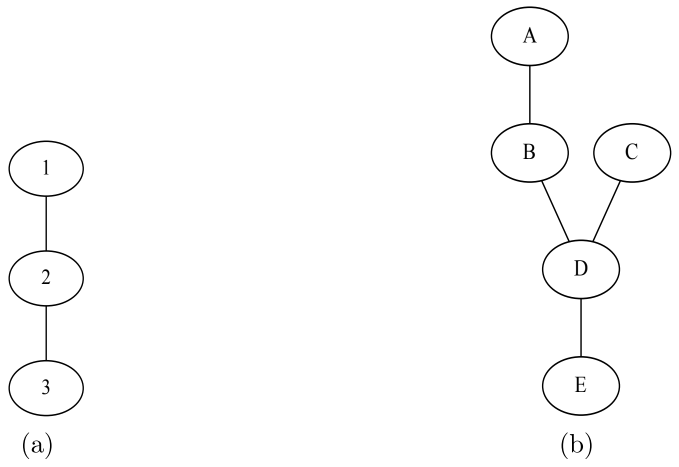


Figure 4: A query (a) and a database (b) graph, which will be queried.

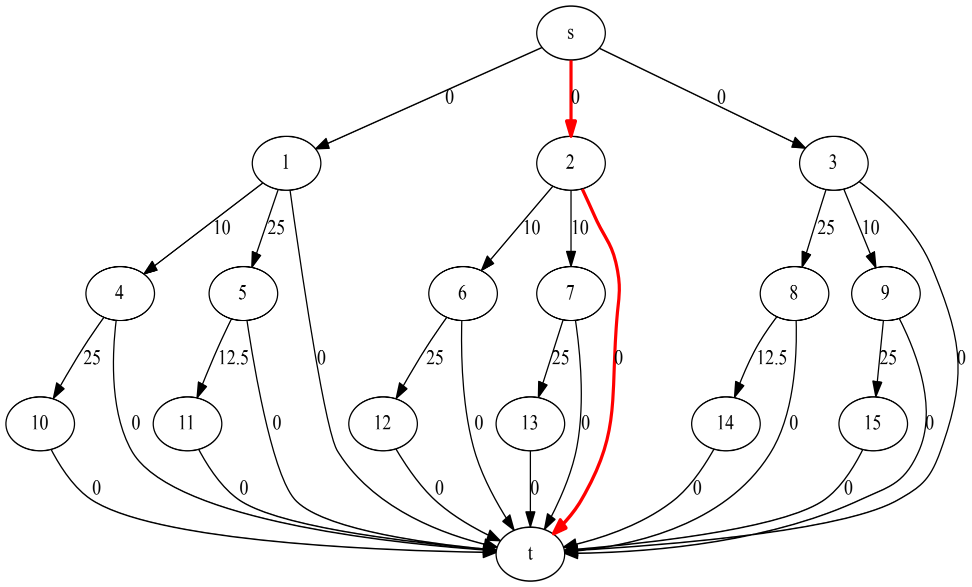


Figure 5: Transformed directed graph $\bar{G}$ for query graph given in Figure 4

We are now ready to give the shortest path formulation:

$$\text{minimize} \quad \sum_{(i,j)\in\bar{E}} x_{(i,j)} w_{(i,j)}, \tag{16}$$

$$\text{subject to} \quad \sum_{j\in\bar{V}} (x_{(i,j)} - x_{(j,i)}) = 0, \qquad i \in \bar{V} \setminus \{s,t\} \tag{17}$$

$$\sum_{j\in\bar{V}} x_{(s,j)} = 1, \qquad j \in \bar{V} \tag{18}$$

$$\sum_{j\in\bar{V}} x_{(j,t)} = 1, \qquad j \in \bar{V} \tag{19}$$

$$\sum_{l,f^{-1}(i)\in\bar{V}} (x_{(f^{-1}(i),l)} + x_{(l,f^{-1}(i))}) + \sum_{l,f^{-1}(j)\in\bar{V}} (x_{(f^{-1}(j),l)} + x_{(l,f^{-1}(j))}) + \sum_{k\in\mathcal{K}(f^{-1}(i),f^{-1}(j))} \sum_{l,f^{-1}(k)\in\bar{V}} (x_{(f^{-1}(k),l)} + x_{(l,f^{-1}(k))}) \geq 1, \qquad (i,j)\in E, \tag{20}$$

$$x_{(i,j)} \in \{0,1\}, \qquad (i,j)\in\bar{E}. \tag{21}$$

Here, $x_{(i,j)}$ is a binary variable that is equal to one when edge (i,j) is on the shortest path. The objective function (16) minimizes the total path length. Constraints (17) ensure the flow balance at each vertex in $\bar{G}$. Constraint (18)-(19) ensures the one unit flow between s and t, and finally the set of constraints (20) guarantees that the vertices on the shortest path form a hub cover. Notice that each path and each vertex in G represent a query plan and a position of a query vertex in the query plan, respectively. Therefore, we use the inverse function $f^{-1}(i)$ to find out the query vertex denoted by vertex i in $\bar{G}$. The final set of constraints (21) ensure the integrality of the binary variables. In Figure 5, the paths $s \to 1 \to t$, $s \to 2 \to t$, and $s \to 3 \to t$ are all shortest paths but the only path that satisfies the constraints (20) is $s \to 2 \to t$. Thus, it is the optimal hub cover yielding the least cost query.

2.4 Graph Matching Computation. In the final step, we use the previously computed search space and query plan to perform the graph matching. Our technique takes the initial query hub according to the plan, and it uses the search space to find those database vertices that may match with it. Then, the graph matching task focuses on the structural unification of a query and a data hub, i.e., we have to match all the neighbors and triangles of the query vertex with some of the neighbors and triangles of the database vertex. When the whole query vertex is matched, we perform a recursive call to process the next query vertex in the MHC plan. When the whole MHC is processed, we report the complete matching and continue the backtracking process until all matchings are found. Here, we give an example to demonstrate the graph matching iterations. For the details, we refer to (Yelbay et al., 2013; Rivero and Jamil, 2016).

EXAMPLE 2.3 *Suppose we select u_5, u_4 as a hub cover in q_1 and $u_5 \bowtie u_4$ as a query plan. The hub vertex u_5 has five candidates as stated in Example 5.1. Suppose $u_5 = v_2$, then the graphlet representation of the query vertices of q_1 changes as follows:*
$\langle u_4, \{u_3\}, \{\emptyset\}\rangle$
$\langle v_2, \{u_1, u_2, u_3, u_6\}, \{(u_1, u_2), (u_2, u_3)\}$

Next, we have to find a one-to-one mapping between the neighbors of u_5 and v_2 by considering the connections among the neighbors, i.e., the boundary edges. Suppose $u_1 = v_4$, $u_2 = v_5$, $u_3 = v_6$, and $u_6 = v_3$. After those mappings, the graphlet representation is given as follows:
$\langle u_4, \{v_6\}, \{\emptyset\}\rangle$
$\langle v_2, \{v_4, v_5, v_6, v_3\}, \{(v_4, v_5), (v_5, v_6)\}$

Once we map all vertices in u_5, we select the second hub vertex u_4. Remember that all the vertices of d are the candidates of u_4. However, the previous mappings reduce the search space of u_4. The candidates are the hub vertices having a neighbor v_6. We select v_1 as a candidate so $u_4 = v_1$. Since we find a one-to-one mapping for all query vertices we are done. We can continue like that to generate all possible subgraph isomorphs in Figure 3 (b), which are represented with different colors.

3. Handling the MHC Computation.

We discussed that in the literature, it has been demonstrated that MHC-based graph matching increases the efficiency of query processing in terms of both memory usage and response time. On the other hand, solving MHC optimally itself is a major endeavor that affects the query processing efficiency. With this motivation, we focus on different solution approaches for MHC in this section and conduct an extensive set of computational experiments to illustrate the empirical performance of these approaches.

3.1 Relaxations and Rounding Heuristics.

Clearly, the mathematical programming models that we have introduced so far are very difficult to solve to optimality especially for large scale instances. Nonetheless, the relaxations of these problems can be solved efficiently. These relaxations have two uses: (i) Their optimal objective function values yield bounds. These bounds can be employed to increase the efficiency of exact methods. (ii) The optimal solutions of the relaxations can be used to obtain feasible solutions for the original problem. In certain cases, these solutions can even play a role in proving approximation bounds. These relaxations as well as the corresponding rounding heuristics are introduced next.

Linear Programming Relaxation. The LP relaxation of a binary integer program is obtained simply by replacing the binary constraints on the variable x_j with the inequality $0 \leq x_j \leq 1$. We relax the integrality constraints in models (1)-(3) and (4)-(6) and obtain the LP relaxations for the MHC and MTS problems, respectively. These two relaxations are referred to as LP1 and LP2 for MHC and MTS, respectively.

We first introduce a new rounding algorithm for the MTS problem. As previously mentioned, our first model (1)-(3) is a special case of the set covering problem. Therefore, we also customize two other rounding algorithms that were originally proposed to solve the set covering problem.

Primal Rounding Algorithm for MTS (*PRMTS*): The algorithm uses the optimal solution of LP2. The pseudo-code of the algorithm is given in Algorithm 2. In line 3, we solve LP2 and obtain the optimal solution $\mathbf{x}^*$. We select the kth variable, which is the largest component in $\mathbf{x}^*$ in line 5. In lines 8-13, we check if increasing x_k violates the feasibility of any constraint including x_k. If it is not the case, then kth vertex is set to 1 and the right hand sides of the constraints including that vertex are decreased by 1. The algorithm continues to select the next vertex with the largest value as long as none of the constraints is violated. The algorithm returns $\mathbf{x}$, which is a feasible solution obtained by the algorithm.

Algorithm 2 Primal Rounding Algorithm for the MTS Problem

```
1: $x_j = 0,\ j \in V$
2: $y_{ij} \leftarrow |\mathcal{K}^{(i,j)}| + 1$                ▷ Right hand side of (5)
3: $\mathbf{x}^* \leftarrow$ Solve LP relaxation of (4)-(6)
4: for $i = 1$ to $|V|$ do
5:     pick the $k$th variable, which is the $i$th largest component in $\mathbf{x}^*$
6:     find the set of constraints $C \subseteq E$ including the $k$th variable
7:     $flag = 1$
8:     for all $(i,j) \in C$ do
9:         if $y_{ij} = 0$ then
10:            $flag = 0$
11:        end if
12:    end for
13:    if $flag = 1$ then
14:        $x_k \leftarrow 1$
15:        for all $(i,j) \in C$ do
16:            $y_{ij} \leftarrow y_{ij} - 1$
17:        end for
18:    end if
19: end for
20: return $\mathbf{x}$
```

Primal Rounding Algorithm for MHC (*PRMHC*): Algorithm 3 is adapted from a set covering algorithm proposed by Hochbaum (1982). The algorithm uses the optimal solution of LP1 denoted by $\mathbf{x}^*$. Any component of $\mathbf{x}^*$ with a value greater than or equal to $1/f$ is set to 1. In the hub cover formulation, f is the maximum number of vertices that can cover an edge. This approach is guaranteed to yield a feasible solution for MHC. Suppose $PRMHC$ does not yield a feasible solution, then there exists at least one constraint for edge (i,j) such that $x^*_j < 1/f$ for all $j \in \bar{\mathcal{K}}^{(i,j)}$. If this is the case, then

$x_i^* + x_j^* + \sum_{k \in \mathcal{K}^{(i,j)}} x_k^* < 1$ because $|\bar{\mathcal{K}}^{(i,j)}| \leq f$. This contradicts our assumption that $\mathbf{x}^*$ is the optimal solution of LP1.

Algorithm 3 Primal Rounding Algorithm for the MHC Problem

```
1: $x_j = 0,\ j \in V$
2: $\mathbf{x}^* \leftarrow$ Solve LP relaxation of (1)-(3)
3: for all $j \in V$ do
4:     if $x_j^* \geq 1/f$ then
5:         $x_j \leftarrow 1$
6:     end if
7: end for
8: return x
```

Dual Rounding for MHC (*DRMHC*): The algorithm proposed by Hochbaum (1982) for the set covering problem is applied to obtain an integral solution for MHC. It uses the optimal solution of the dual problem given by

$$\begin{aligned}
&\text{maximize} && \sum_{(i,j)\in E} y_{(i,j)}, && && (22)\\
&\text{subject to} && \sum_{(i,j)\in E} y_{(i,j)} + \sum_{(j,i)\in E} y_{(j,i)} + \sum_{j\in\mathcal{K}^{(i,k)}} y_{(i,k)} \leq 1, && j \in V, && (23)\\
& && y_{(i,j)} \geq 0, && (i,j) \in E, && (24)
\end{aligned}$$

where $y_{(i,j)}$ is a dual variable corresponding to the coverage constraint (2) for edge (i,j). The steps of the algorithm are given in Algorithm 4. The optimal solution of (22)-(24) is denoted by y^*. Based on the LP duality theory, the main idea of the algorithm is to set the primal variable to one whenever the corresponding dual constraint is tight.

Algorithm 4 Dual Rounding Algorithm for the MHC Problem

```
1: $x_j = 0,\ j \in V$
2: Solve LP relaxation of (22)-(24)
3: for all $j \in V$ do
4:     if $\sum_{(i,j)\in E} y^*_{(i,j)} + \sum_{(j,i)\in E} y^*_{(j,i)} + \sum_{j\in\mathcal{K}^{(i,k)}} y^*_{(i,k)} = 1$ then
5:         $x_j \leftarrow 1$
6:     end if
7: end for
8: return x
```

Semidefinite Programming Relaxation. Semidefinite programming (SDP) is about optimizing a linear function of a symmetric matrix over the cone of positive semidefinite matrices. LP is a special case of SDP. Semidefinite relaxations have been developed for many $\mathcal{NP}$-hard optimization problems. The important point is that very good approximation bounds can be obtained after solving the SDP relaxations of hard combinatorial problems (Goemans and Williamson, 1995; Halperin, 2002; Karakostas, 2005). Before introducing the SDP relaxation, we first remove the integrality constraint from (7)-(9) and obtain

$$\begin{aligned} &\text{minimize} && \sum_{j\in V}(1+y_0y_j)/2, && && (25)\\ &\text{subject to} && (y_0-y_i)(y_0-y_j)+(2y_0-y_i-y_j)\sum_{k\in\mathcal{K}^{(i,j)}}(y_0-y_k)\le 8|\mathcal{K}^{(i,j)}|, && (i,j)\in E, && (26)\\ & && y_j^2=1, && j\in V\cup\{0\}. && (27)\end{aligned}$$

We next introduce the matrix variable $\mathbf{Y}=\mathbf{y}\mathbf{y}^T$, where $\mathbf{y}$ is the vector consisting of the components y_0 and $y_i, i\in V$. We also define $\mathbf{A}\bullet\mathbf{B}$:=trace($\mathbf{A}^T\mathbf{B}$). Using now this notation, we can give the following equivalent formulation:

$$\begin{aligned} &\text{minimize} && \mathbf{C}\bullet\mathbf{Y} && && (28)\\ &\text{subject to} && \mathbf{A}^{(i,j)}\bullet\mathbf{Y}\le 8|\mathcal{K}^{(i,j)}|, && (i,j)\in E, && (29)\\ & && \text{diag}(\mathbf{Y})=\mathbf{e}, && && (30)\\ & && \mathbf{Y}\succeq 0, && && (31)\\ & && \text{rank}(\mathbf{Y})=1, && && (32)\end{aligned}$$

where $\mathbf{C}$ and $\mathbf{A}^{(i,j)}$ are symmetric matrices, $\mathbf{e}$ is the vector of ones and $\mathbf{Y}\succeq 0$ means that the matrix $\mathbf{Y}$ is positive semidefinite. Before specifying $\mathbf{C}$ and $\mathbf{A}^{(i,j)}$, let us relax the constraint (32) and drop it from the model.

The symmetric matrices in the SDP relaxation are defined as follows: Let C_{mn} denote the components of the matrix $\mathbf{C}$. Then,

$$C_{mn}=\begin{cases}1/4, & \text{if } m=0 \text{ and } n\in V;\\ 1/4, & \text{if } m\in V \text{ and } n=0;\\ 0, & \text{otherwise.}\end{cases}$$

When it comes to the matrix $\mathbf{A}^{(i,j)}$, we observe that

$$(y_0-y_i)(y_0-y_j)=\mathbf{M}\bullet\mathbf{y}\mathbf{y}^T,$$

where $\mathbf{M}$ is a symmetric matrix and its nonzero components are given by

$$M_{00}=1,\quad M_{0i}=M_{i0}=M_{0j}=M_{j0}=-1/2,\quad M_{ij}=M_{ji}=1/2.$$

For a given $(i,j)\in E$ note that the constraint (10) is constructed by summing up matrices like $\mathbf{M}$ above. Consequently, the matrix $\mathbf{A}^{(i,j)}$ is also symmetric.

Formally, we write $\mathbf{Y}=\mathbf{V}^T\mathbf{V}$, where the columns of $\mathbf{V}$ are given by $\mathbf{v}_m, m\in V\cup\{0\}$. Then, we obtain,

$$\begin{aligned} &\text{minimize} && \sum_{m,n}C_{mn}\mathbf{v}_m^T\mathbf{v}_n && && (33)\\ &\text{subject to} && \sum_{m,n}\mathbf{A}_{mn}^{(i,j)}\mathbf{v}_m^T\mathbf{v}_n\le 8|\mathcal{K}^{(i,j)}|, && (i,j)\in E, && (34)\\ & && \mathbf{v}_m^T\mathbf{v}_n=1, && m\in V\cup\{0\}, && (35)\\ & && \mathbf{v}_m\in\mathbb{R}^{|V|+1}, && m\in V\cup\{0\}. && (36)\end{aligned}$$

SDP Rounding Algorithm for the MHC Problem (*RSDP*): We implemented a rounding algorithm inspired from a method proposed for the minimum vertex cover problem (Halperin, 2002). This rounding method uses the optimal solution $\mathbf{v}^*$ of the SDP relaxation and returns the set $S = \{j \in V | \mathbf{v}_0^{*T} \mathbf{v}_j^* \geq 0\}$ as an approximate solution. This solution is not necessarily a feasible hub cover but the number of uncovered edges is much less with respect to the number of covered edges. On the other hand, we observe that the method results in many redundant vertices in the hub cover. Alternatively, we propose Algorithm 5, which obtains $S = \{j \in V | \mathbf{v}_0^{*T} \mathbf{v}_j^* > 0\}$ and then repairs any resulting infeasibility by iteratively selecting a vertex $i \in V \setminus S$ which covers the highest number of uncovered edges until all edges are covered.

Algorithm 5 Semidefinite Programming Algorithm for the MHC Problem

```
1:  $x_j = 0,\ j \in V$
2:  $\mathbf{v}^* \leftarrow$ Solve SDP relaxation of (33)-(36)
3:  for all $j \in V$ do
4:      if $\mathbf{v}_0^{*T} \mathbf{v}_j^* > 0$ then
5:          $x_j \leftarrow 1$
6:      end if
7:  end for
8:  Find the set of uncovered edges $U \subseteq E$
9:  while $|U| > 0$ do
10:     Find the vertex $j$ that covers the maximum number of edges in $U$
11:     $x_j \leftarrow 1$
12:     for all $(i,k)$ covered by vertex $j$ do
13:         $U = U \setminus (i,k)$
14:     end for
15: end while
16: return $x$
```

Postprocessing. Rounding algorithms may return solutions in which some edges are covered several times. Therefore, we apply a postprocessing algorithm to the output of a rounding algorithm in order to decrease the number of redundant vertices in the final hub cover and improve the solution quality. Algorithm 6 summarizes the iterations of the postprocessing algorithm. After obtaining the solution by any of the rounding algorithms in line 1, we compute the number of times that each edge is covered by the selected vertices. In line 4, for each vertex in the solution, we check if the vertex is redundant. If it is redundant, then we remove that vertex from the solution and update the number of times each edge is covered by the remaining vertices.

3.2 Numerical Experiments. In this section, we conduct a set of experiments to test the performance of the LP and SDP relaxations as well as the rounding algorithms using the optimal solutions of those relaxations. We first define our problem classes and experimental setup and then discuss our results. Our data set includes a total of 210 graphs (30 graphs from each class) with known optimal solutions. The first five classes are from a well-known graph database by Santo et al. (2003) and the others are synthetically generated graphs used in various application areas. The LP and SDP relaxations

Algorithm 6 Postprocessing Algorithm

```
1: Get the solution $x$ from any one of the rounding algorithms
2: $C_{(i,j)} = x_i + x_j + \sum_{k \in \mathcal{K}^{(i,j)}} x_k \quad \forall (i,j) \in E$
3: $V' = \{j \in V \mid x_j = 1\}$
4: for all $j \in V'$ do
5:     Find the set of edges $E'$ covered by vertex $j$
6:     $flag = 1$
7:     for all $(i,k) \in E'$ do
8:         if $C_{(i,k)} = 0$ then
9:             $flag = 0$
10:        end if
11:    end for
12:    if $flag = 1$ then
13:        $x_j \leftarrow 0$
14:        for all $(i,k) \in E'$ do
15:            $C_{(i,k)} \leftarrow C_{(i,k)} - 1$
16:        end for
17:    end if
18: end for
```

are obtained by MATLAB 2010b. To solve the SDP relaxation, we used the SDPA-M solver which is a MATLAB interface for the semidefinite programming algorithm (SDPA) solver developed by Kojima et al. (2005). The solver is developed to solve small and medium size semidefinite programming models. Therefore, our problem set includes only small to medium size instances. The number of vertices and edges range from 20 to 1000.

- ◇ **(a) Random graphs:** Randomly generated graphs with varying densities.
- ◇ **(b) Bounded valence graphs:** The vertices in these graphs possess the same number of neighbors. Bounded valence graphs are generally employed in the modeling of molecular structures.
- ◇ **(c) Irregular bounded valence graphs:** These graphs are obtained by introducing irregularities to the graphs in (b) by deleting and adding some edges.
- ◇ **(d) Regular Meshes:** The 2D, 3D, and 4D meshes where each vertex has connections with 4, 6 and 8 neighbors. 3D objects can be represented as 3D mesh graphs in object recognition.
- ◇ **(e) Irregular Meshes:** Meshes obtained by introducing irregularities into the graphs in (d) by adding extra edges.
- ◇ **(f) Scale-free graphs:** The degree distribution of the vertices in these graphs follows a power law. These graphs are obtained through the scale-free graph generator of the C++ Boost Graph Library. The World Wide Web, social networks, and flight networks are a few examples of scale-free graphs.
- ◇ **(g) Planar graphs:** These graphs have planar embeddings so that no edges cross. Molecular structures and graph databases in biometric identification satisfy the planarity condition.

In this section, we carried out a computational experiment to test the performances of the relaxation models and rounding methods on various types of graph databases. To benchmark the lower bounds obtained by the LP and SDP relaxations over all instances, we solve the LP and SDP relaxations of MHC, respectively, and compute the percentage gap with respect to the optimal solution of the IP formulation (1) -(3). Figure 6 demonstrates the empirical cumulative distribution of these percentage gaps obtained over all instances. It indicates that the LP relaxation yields a tighter lower bound relative to the SDP relaxation. In almost 90% of the instances, the gaps between the optimal and the LP solutions are less than 10%. On the other hand, the SDP relaxation achieves that gap in 75% of the instances.

Figures 7(a) and 7(b) compare the upper bounds obtained by the rounding methods applied to the optimal solutions of the LP and SDP relaxations before and after postprocessing, respectively. The results without postprocessing indicate that the SDP rounding algorithm is superior to the primal and dual rounding algorithms by providing tighter upper bounds. In 70% of the instances, the SDP rounding algorithm provides upper bounds with optimality gaps less than 30%. The corresponding fraction of the instances with similar solution quality decreases to 25% and 15% for the primal and dual rounding algorithms, respectively. On the other hand, surprisingly the rounding algorithm developed for MTS outperforms all other rounding algorithms. The rounding algorithm using the optimal LP solution of MTS provides the optimal solution in almost 45% of the instances. With postprocessing, the percentage of the instances that can be solved to optimality increases to 55%. The results indicate that the postprocessing algorithm eliminates the redundant vertices and improves the solution quality considerably for other algorithms as well. While $PRMHC$ and $DRMHC$ derive the most benefit from postprocessing, the relative ordering of the algorithms stays intact before and after postprocessing. The cumulative fraction of the instances for which $PRMHC$ returns an optimal solution gets a boost from 15% to 40% by postprocessing. The corresponding change for $DRMHC$ and SDP are from 8% to 32% and 20% to 46%, respectively. After the postprocessing algorithm, in 70% of the instances, the SDP rounding algorithm provides upper bounds with optimality gaps less than 5%. Without postprocessing the same optimality gap is achieved in about 25% of the instances.

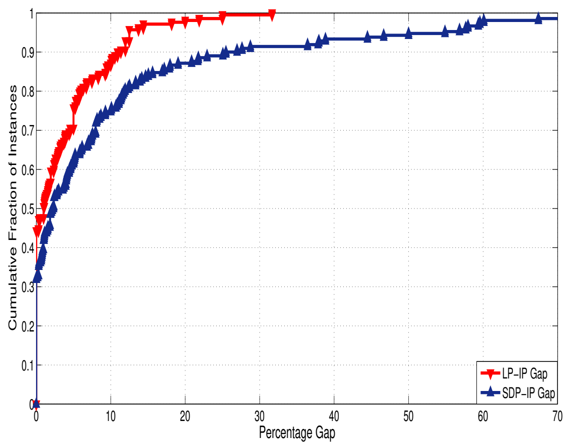
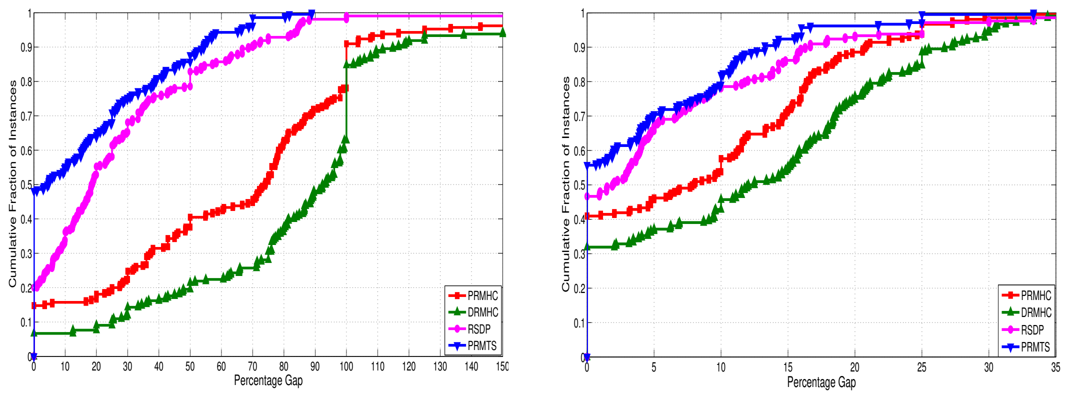


Figure 6: The empirical cumulative distributions of the optimality gaps of the LP and SDP relaxations.



(a) Before postprocessing (b) After postprocessing

Figure 7: The empirical cumulative distributions of the optimality gaps of the rounding algorithms before and after postprocessing.

4. Conclusion.

In this study, we briefly introduced graph query processing and elaborated on the role of an optimization problem – known as the MHC problem – for the efficiency of the graph matching computations. Given an optimal or a near-optimal hub cover, we outlined the technique to compute the subgraph isomorphism of a query graph and discussed that the cost of a query plan changes with respect to the set of vertices coming from different solutions of MHC or the order of the query vertices in a solution. Generating an optimal hub cover, which yields the least cost query plan is critical to further increase the performance. To this end, we proposed a shortest path formulation which computes an optimal hub cover with the best query plan. Unfortunately, solving this integrated formulation is practically much harder than solving MHC alone. As a future study, we plan to attack the integrated problem of identifying a hub cover with the minimum query cost.

In the literature, only a few studies are present on the MHC problem (Yelbay et al., 2013, 2016). In this study, we advance the state-of-the-art on MHC by presenting several other new mathematical programming formulations along with their relaxations. We conducted a numerical study to compare the bounds obtained from the SDP and LP relaxations of MHC and observed that the LP relaxation gives a tighter bound. We also introduced two rounding algorithms $RSDP$ and $PRMTS$, and compared those with two well-known rounding algorithms proposed for the set covering problem in the literature. The results indicate that the algorithms proposed in this study are superior to the benchmark algorithms in terms of solution quality. We also observed that the performances of the LP rounding algorithms vary when applied to different mathematical programming models. This observation underlines the significance of introducing alternate models for the same optimization problem.

Our semidefinite programming relaxation may be used to develop an approximation bound for MHC. However, such a result remains elusive at this time. Even for special problem classes, where the number of candidate vertices to cover an edge is less than or equal to three, a formal analysis to obtain an approximation bound seems beyond reach. Nonetheless, based upon our empirical results, we conjecture that our SDP relaxation may achieve an approximation bound less than two for MHC.

In order to solve the SDP relaxation, we used the SDPA-M solver, which was developed to solve small to medium size instances limited to 2,000 constraints and 2,000 variables. Recently, Fujisawa et al. (2014)

report that the parallel version of SDPA – referred to as SDPARA – can solve instances with up to a million constraints. As a future study, we plan to employ the parallel implementation of the semidefinite programming solver and test the performance of the SDP relaxation on large-scale MHC instances.

References

Cordella, L., Foggia, P., Sansone, C., and Vento, M. (2001). An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, Cuen, Italy.

Cordella, L., Foggia, P., Sansone, C., and Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans Pattern Anal Mach Intell*, 26(10):1367–1372.

Fujisawa, K., Endo, T., Yasui, Y., Sato, H., Matsuzawa, N., Matsuoka, S., and Waki, H. (2014). Peta-scale general solver for semidefinite programming problems with over two million constraints. In *The 28th IEEE International Parallel and Distributed Processing Symposium*, India.

Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145.

Halperin, E. (2002). Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31:1608–1623.

He, H. and Singh, A. K. (2008). Graphs-at-a-time: query language and access methods for graph databases. In *SIGMOD Conference*, pages 405–418, Vancouver, Canada.

Hochbaum, D. (1982). Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556.

Jamil, H. M. (2011). Computing subgraph isomorphic queries using structural unification and minimum graph structures. In *SAC*, pages 1053–1058, Taichung, Taiwan.

Karakostas, G. (2005). A better approximation ratio for the vertex cover problem. In *32nd International Colloquium on Automata, Languages and Programming(ICALP)*, pages 1043–1050, Lisboa, Portugal.

Kawabata, T. (2014). What is MCS? (link - Last accessed 10 November 2015).

Kojima, M., Fujisawa, K., Nakata, K., and Yamashita, M. (2005). SDPA (semidefinite programming algorithm) user's manual. Technical report, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Japan.

Lee, J., Han, W.-S., Kasperovics, R., and Lee, J.-H. (2012). An in-depth comparison of subgraph isomorphism algorithms in graph databases. *PVLDB*, 6(2):133–144.

Lipets, V., Vanetik, N., and Gudes, E. (2009). Subsea: an efficient heuristic algorithm for subgraph isomorphism. *Data Min. Knowl. Discov.*, 19:320–350.

Rivero, C. and Jamil, H. M. (2014). On isomorphic matching of large disk resident graphs using an xquery engine. The 5th International Workshop on Graph Data Management: Techniques and Applications, Chicago, USA.

Rivero, C. R., Hernandez, I., Ruiz, D., and Corchuelo, R. (2013). Benchmarking data exchange among semantic-web ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 25:1997–2009.

Rivero, C. R. and Jamil, H. M. (2016). Efficient and scalable labeled subgraph matching using sgmatch. *Knowledge and Information Systems*, pages 1–27.

Santo, M., Foggia, P., Sansone, C., and Vento, M. (2003). A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24:1067–1079.

Shang, H., Zhang, Y., Lin, X., and Yu, J. (2008). Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. In *Journal Proceedings of the VLDB Endowment*, volume 1, pages 364–375, Auckland, New Zealand.

Tian, Y. and Patel, J. M. (2008). Tale: A tool for approximate large graph matching. In *International Conference on Data Engineering*, pages 963–972, Cancun, Mexico.

Ullmann, J. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM*, 23:31–42.

Weber, M., Liwicki, M., and Dengel, A. (2012). Faster subgraph isomorphism detection by well-founded total order indexing. *Pattern Recognition Letters*, 33:2011–2019.

Yelbay, B., Birbil, Ş. İ., Bülbül, K., and Jamil, H. (2016). Approximating the minimum hub cover problem on planar graphs. *Optimization Letters*, 10:33–45.

Yelbay, B., Birbil, Ş. İ., Bülbül, K., and Jamil, H. M. (2013). Trade-offs computing minimum hub cover toward optimized graph query processing. (arXiv).

Zhang, S. and Jin, W. (2010). Sapper: Subgraph indexing and approximate matching in large graphs. In *Journal Proceedings of the VLDB Endowment*, volume 3, pages 1185–1194, Singapore, Singapore.

Zhang, S., Li, S., and Yang, J. (2009). GADDI: distance index based subgraph matching in biological networks. In *EDBT*, pages 192–203.

Zhao, P. and Han, J. (2010). On graph query optimization in large networks. *PVLDB*, 3(1):340–351.

Zhu, K., Zhang, Y., Lin, X., Zhu, G., and Wang, W. (2010). A novel and efficient framework for finding subgraph isomorphism mappings in large graphs. In *15th International Conference on Database Systems for Advanced Applications*, pages 140–154, Tsukuba, Japan.