AJIT-e: Online Academic Journal of Information Technology
2018 Yaz/Summer – Cilt/Vol: 9 - Sayı/Num: 33
DOI: 10.5824/1309-1581.2018.3.002.x
Received: 25.01.2018 **Editorial Process Begin: 30.07.2018** **Published: 05.10.2018**

Trade-offs Computing Minimum Hub Cover toward Optimized Labeled Graph Query Processing

Belma YELBAY, *Department of Industrial Engineering, Sabancı University, Turkey*, byelbay@sabanciuniv.edu, *ORCID: 0000-0002-6609-2775*
S. İlker BİRBİL, *Econometric Institute, School of Economics, Erasmus University Rotterdam, The Netherlands,* birbil@ese.eur.nl, *ORCID: 0000-0001-7472-7032,*
Kerem BÜLBÜL, *Department of Industrial Engineering, Sabancı University, Turkey*, bulbul@sabanciuniv.edu, ORCID: 0000-0001-8955-0911,
Hasan M. JAMİL, Department of Computer Science, University of Idaho, USA, jamil@uidaho.edu, ORCID: 0000-0002-3124-3780

ABSTRACT. As techniques for graph query processing mature, the need for optimization is increasingly becoming an imperative. Indices are one of the key ingredients toward efficient query processing strategies via cost-based optimization. Due to the apparent absence of a common representation model, it is difficult to make a focused effort toward developing access structures, metrics to evaluate query costs, and choose alternatives. In this context, recent interests in covering-based graph matching appears to be a promising direction of research. In this paper, our goal is to formally introduce a new graph representation model, called *Minimum Hub Cover*, and demonstrate that this representation offers interesting strategic advantages, facilitates construction of candidate graphs from graph fragments, and helps leverage indices in novel ways for query optimization. However, like other covering problems, minimum hub cover is NP-hard, and thus is a natural candidate for optimization. We claim that computing the minimum hub cover leads to substantial cost reduction for graph query processing. We present a computational characterization of minimum hub cover based on integer programming to substantiate our claim and investigate its computational cost on various graph types.

1 Introduction

Queries over graph databases can be classified broadly into whole graph at-a-time, and node at-a-time processing, and framed as a subgraph isomorph computation problem (e.g., [29, 51]) under a set of label mapping constraints, generally known as graph matching. Techniques such as GraphQL [21], QuickSI [39] and earlier research such as VFLib [12] and Ullmann [44] fall in the former category while TALE [41], and SAP- PER [56] are representative of the latter. The advantage of the node at-a-time graph processing approach is its inherent ability to prune search space based on target node matching conditions. Node indices are the most common pruning aid used in most of these processing methods although indices on paths [17], frequent structures [54],

DOI: 10.5824/1309-1581.2018.3.002.x

node distances [29, 54, 52], etc. have also been used. The key difference is in the ways indices are exploited for the construction of the target database graphs from their parts (i.e., the edges).
The effectiveness of the node at-a-time methods, however, largely depends on the query type such as subgraph isomorphism, approximate matching, path queries and so on, as well as on the index structure used. In other words, universal indexing methods are not always suitable for all queries, and therefore specialized indices are often constructed to process a query (e.g., GraphGrep [17], TALE, and SAPPER) and never maintained. Thus, it is not apparent if the index structure is switched, how an algorithm will perform leaving an open question if generic index structures can be leveraged in a way similar to relational query processing with popular indices such as B+ trees, extendible hashing and inverted files.
In order to decouple the index selection from the query expressions, and to subsequently use indices as a strategic instrument to compute alternative query plans, we focus on a representation method for graphs that is independent of the underlying access structures. Our goal is to propose the "hub" as the unit of graph representation that tells us all we need to know about a node or vertex of a graph. Intuitively, each node in a graph as hub "covers" all the edges involving its neighbors and itself. For example, the hub u_5 in Figure 1(a) covers the edges (u_1, u_5), (u_1, u_2), (u_2, u_5), (u_2, u_3), (u_3, u_5) and (u_5, u_6) as a unit structure (shown as the purple edges).

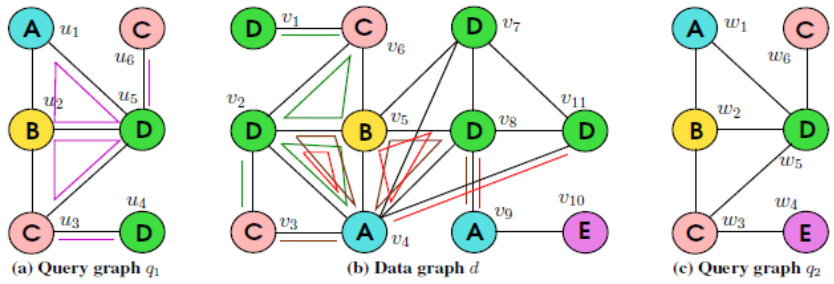


Fig. 1. Example: Query graphs q_1 and q_2, and data graph d.

The concept of hub we have in mind can be thought of as a convenient extension of Ullmann's adjacency matrix [44] and feature structure indexing [27, 9] in that we localize the adjacency matrix at the node level and consider only a single feature, edges among the neighbors. Consequently, only structures that are part of a hub are stars (neighbors with no shared edge with other neighbors) and triangles (neighbors sharing edge with other neighbors). In Figure 1(a), vertex u_5 has two triangles $\Delta u_1u_2u_5$ and $\Delta u_2u_3u_5$, and a star (u_5, u_6) (in this case just an edge). Whereas vertices v_9 and v_{11} in Figure 1(b) have a star (with two edges (v_8, v_9) and (v_9, v_{10}) with v_9 as their center), and three triangles ($\Delta v_4v_7v_{11}$, $\Delta v_8v_7v_{11}$, and $\Delta v_4v_8v_{11}$) respectively.

Our goal is to use these atomic structural cues to match shapes for the purpose of graph matching. For example, to match the query graph q_1 in Figure 1(a) with the data graph d in Figure 1(b), we look for individual node structures that are identically connected and depending on the matching requirement, have identical labels. The next step is to piece together these individual

matches to see if the composed structure is the target graph. The cost of this search usually is dominated by the cost of piecing together the components and testing if the process is yielding the target graph.
In this example, we can contemplate several different types of graph matching that can be conceived as the variants of subgraph isomorphism though in the literature, only the structural isomorphs and match isomorphs defined below are prevalent. We therefore will consider only these two types of matching in the remainder of this paper.

- *structural subgraph isomorph*, where only the node IDs (not the labels) are mapped from query graphs to data graphs using and injective function.
- *label subgraph isomorph*, on the other hand, requires an injective mapping of both node IDs and node labels from query graph to data graph.
- *full subgraph isomorph* extends label subgraph isomorphic matching to include edge labels in the mapping.
- *match subgraph isomorph* uses an equality function on the definition of full sub- graph isomorph to achieve exact matching of node and edge labels while maps node IDs using an injective function[1].

Among the above four modes of matching, structural subgraph isomorphism is the least restrictive or selective, and so most computationally expensive. While the match sub- graph isomorphic matching (in the literature it is known as labeled graph matching) is the most restrictive/selective, full and label subgraph isomorphism are increasingly less so. The idea here is that by combining different selection and mapping constraints, called *matching mode*, we can capture most popular graph matching concepts and go beyond current definitions. Traditional deep equality $\stackrel{d}{=}$ operator [1] in object-oriented databases can be used to test if two graphs (or a subgraph) are equal (or contained in the other graph) by requiring that node IDs and labels be identical. Finally, by requiring that the two graphs have equal number of nodes, we can also achieve graph isomorphism for each case above.

1.1. Main Motivation

The vertex labeled[2] graphs in figure 1 show two query graphs q_1 and q_2, and a data graph d respectively. In these graphs, each node has a unique node ID such as u_i, v_j and w_k, and a label such as A, B and C (shown in uppercase with unique color codes). If all the labels are empty, the graph is considered unlabeled. If we matched query graph q_1 with the data graph d for structural subgraph isomorphism, we will compute the green, brown and red matches, among others, as we are only required to map node IDs. However, if we were to compute match subgraph isomorphs, only solution we can compute then is the green subgraph. If label subgraph isomorphs were being sought instead, the solution will include the brown subgraph where we

[1] *Injective mapping of node IDs ensures structural match while equality of label mapping ensures that the graphs are identical even though the node IDs are different*

AJIT-e: Online Academic Journal of Information Technology
2018 Yaz/Summer – Cilt/Vol: 9 - Sayı/Num: 33
DOI: 10.5824/1309-1581.2018.3.002.x

map pink (C) to pink, green (D) to blue (A), mustard (B) to mustard, and blue to green, along with the obvious green subgraph. Let us illustrate this naive matching process using Figures 1 and 2.

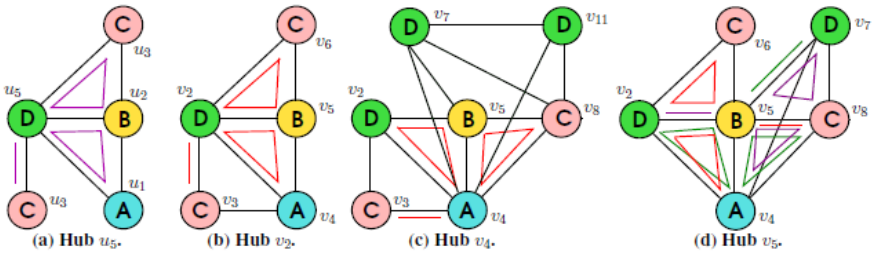


Fig. 2. Candidate generation for query graph hub u_2 on data graph d.

Since there are six nodes in the graph q_1, a total of six hubs are possible. Similarly, the graph d can be represented by a total of eleven hubs. Let us first match q_1 with d in match subgraph isomorphism mode. Under this constraint, for hub u_5 (shown clock- wise rotated by 180°C in purple in Figure 2(a)), we can only find one hub, hub v_2 shown in Figure 2(b), in d which is a super-graph of hub u_5 and can produce a structure identical to u_5 (shown as red edges) on proper mapping of the node IDs.

These structures are called candidate graphlets. However, if we choose to match in structural or label sub-graph isomorphism mode, we can find more hubs as capable of generating candidate structures. For example, in label subgraph isomorphism mode, hubs v_4 and v_5 can also generate candidates (possible candidate structures are shown in red, purple and green) as shown in Figures 2(c) and 2(d).

To complete the matching, we can continue to match all the other hubs in q_1 in a similar way, and by applying the substitutions generated for each set of previous matches to the candidate hubs to eventually compute the green match as shown in Figure 1(b). These graphlets can be joined or pieced together in both bottom-up fashion using a process similar to natural join, or in top-down manner using a depth-first search[2]. In both cases, the graphlets that do not stitch to form the target graph will eventually be eliminated.

Clearly, the dominant cost in this naive algorithm is in candidate generation. Therefore, it would be prudent to seek opportunities to curtail the candidates that do not have a realistic chance of contributing to the result or will produce redundant candidates. For example, we can be smarter and choose to match u_4 or u_5 only without compromising the outcome. We can further speed up the process by noticing that u_4 is a green D node and there are four such nodes in d, all of which will generate a total of twelve candidates even though only four will survive the node mapping, and only one will join with the first candidate to complete the computation. On the other hand, if

[2] The complete top-down matching process is shown in algorithm 2. In this algorithm μ is the structure mapping function that generates the mapping, θ is the substitution list from previous steps, $[\theta]$ is the application of the substitution, and • is the composition function of two substitutions.

we decided to use u_3, we will generate six candidates (v_3 is not one of them) of which only one will survive the mapping and eventually the response. Furthermore, if we started initially with u_4, and then try to map u_5, the number of candidates generated will be even higher although the response computed will still be the same. Interestingly though, if the query is q_2 (note the similarity of the two queries except that w_4 is now purple), it is better to start matching with w_4, because there is only one candidate and it will fail to produce the response in the next step, as expected.

These observations lead us to devise the following query processing strategy. First, we reduce the number of hubs or graphlets in the query graph that we must match based on a new notion of edge covering in graph theory, called the *minimum hub cover* (MHC). A minimum hub cover essentially means a subset of the nodes in a graph accounts for all the edges in a graph. Secondly, the concept of MHC helps exploit available meta-data on nodes to order the nodes in priority order based on their selectivity to prune search space, that we call a *query plan*. In ordering the nodes, we explore the nodes that will most likely produce the least number of candidates first[3]. Given the fact that a query graph may have multiple MHCs, it also offers us the opportunity to choose the best query plan for a database instance. Finally, we are now able to use access structures such as hash index and set index to find only the nodes that are relevant for expanding nodes at a given point in a query plan. In fact, the matching Algorithm 2 uses two such indices I_H and I_S. The query plan can be implemented as a top-down or bottom-up procedure based on the expected number of candidates and a choice can be made based on the expected cost. It is also possible to reorder the query plan in a top-down procedure to prune search space dynamically in a way like best-first search.

1.2. Significance

In overwhelming majority of graph matching research, the unit of storage and matching is basically the edge between two vertices. The major cost of such approach is the reconstruction of target graph from the edge set for the determination of topological concordance. As opposed to edge-based matching [21], to reduce the cost of reconstruction, several research used indexing to collate similar fragments of graphs, or *graphlets*, to speed up retrieval of the target topology [46, 59]. Depending on the query class and types of data graphs, structure indexing has been shown to prune search space better and thus improve matching cost. One of the methods used to successfully search for graphlets was based on set or edge coverings [32, 23] to reduce the number nodes needed to be explored. But, such covering based matching did not exploit the power of indexing of structural features of graphs.

In one of our recent research, we have explored the idea of exploiting our notion of hub covering as discussed above and sub-goal ordering to prune search space [23]. This qualitative investigation suggested that developing scalable algorithms and optimization techniques for efficient graph query processing based on subgraph isomorphism matching is entirely feasible. We have further

[3] The ordering algorithm for query graphlets is shown in algorithm 1.

DOI: 10.5824/1309-1581.2018.3.002.x

demonstrated that traditional graph matching algorithms in systems such as Neo4j performs poorly for queries over disk resident graphs [36]. The impetus for the adoption of graphlet based tree matching in our re- search on large phylogenetic databases reported in [26] was the observation that judicial choice of graphlets in the sub-goal processing order pays dividend over traditional techniques. In [25], we have shown that sub-goal ordering with access support can be combined to drastically improve query processing time over large disk resident phylogenetic databases. Finally, we show that for a very large class of general graph queries, our technique delivers superior performance on many benchmark and real-life data sets [37] over contemporary approaches.

In all the above research, the common technique used is the minimum hub cover to reduce the number of graphlets to be processed for matching. As we discuss in this paper, the problem of minimum hub cover computation is NP-Hard. We have already reported a similar finding in [49] for planar graph. Such observations led to the adoption of heuristic hub cover computation in all our previous research [36, 26, 25, 37] because cover computation is query specific and the cost of processing includes the cost of cover computation. The reason an optimum cover was not sought in our previous research above was that we wished to avoid a blind attempt at computing optimum solutions and exacerbate the cost of actual query processing. Because the cost varies significantly depending on the query graph type as we show in this paper. In this paper, our goal thus is to develop a formal treatment of minimum hub cover, and quantitatively characterize its computation costs over various graph types to suggest a theoretical basis for us to judiciously choose an algorithm for cover computation depending on the graph at hand. We present our approach to graph query processing based on minimum hub cover using a generic matching technique in sections 3.2 and 3.4 that are not specific for any database graph type. The examples of specific algorithms heuristic cover computation can be found in [26, 25, 37].

1.3. Organization of the Paper

Covering based graph matching is proving to be an interesting and emerging research direction although we are aware of only Sigma [33] which used set covering directly for matching very small graphs, while [22, 10] indirectly used covering for graph matching tangentially. Our goal in this paper, however, is to formally introduce the idea of graph representation using graphlets and graph query processing using the minimum hub cover of query graph graphlets[5]. Our focus is to convince the skeptics that these two concepts help achieve the separation in graph representation and storage, indexing, query plan generation, and query optimization conveniently.

Once this model is accepted in principle, two main computational problem emerge both of which are computationally hard – computing MHC and graph matching using subgraph isomorphism as the primary vehicle. In this paper, we only address the first issue, that is the computational aspects of MHC. But for the sake of completeness, we also briefly present an outline of the cost-based optimization strategy for the ordering of graphlets in the MHC as a candidate query plan, and a query processing algorithm that uses indices for the execution of a query plan. By doing so, we demonstrate that cost-based query optimization is feasible if we can compute the MHC of a query graph. Finally, we believe that even if algorithms such as SUMMA [55], NOVA [58], TALE,

and SAPPER do not use a notion similar to hubs as we do, they do use the notion of neighborhood and will benefit from the development presented in this paper if they consider a similar covering, i.e., edge or vertex covers, of the queries. The results in this paper then becomes directly relevant to those research as well because we show how the cost of covering computation may vary depending on the type and parameters of the graphs being considered.

The remainder of the paper is organized as follows. We discuss background of the research related to MHC in Section 2. In this section, we also discuss related research in covering computation based on which we formulate our characterization of MHC. The formal treatment of MHC and its application in query plan generation is discussed in Section 3. Similar to other covering problems such as set cover, and minimum vertex cover, MHC turns out to be an NP-complete problem as well. Therefore, it can be framed as an optimization problem and made a candidate for heuristic solutions. In Section 3.5, we discuss an integer programming formulation of the MHC problem as a prelude to our main results on its computability. We have implemented the algorithm using the IBM ILOG optimization engine CPLEX. The experimental results in Section 5 based on the design in Section 4 suggest that solving MHC to optimality is not a concern for many graph types. A summary of interesting and possible future research issues that are still outstanding is discussed in Section 6. We finally conclude in Section 7.

2 Background and Related Research

In our earlier research on IsoSearch [23], we have shown that the notion of *structural unification* helps to extract all possible matches of two hubs under a mapping function or a substitution list. While this atomic matching process generates a potentially large candidate pool, we were able to avoid the large cost related to testing for conformity of the candidate target structure with that of the query graph that most other algorithms incur. In our case, conformity is an eventuality and automatic if a match exists. We have also shown that IsoSearch performs significantly better than traditional algorithms such as Ullmann and VFLib, have a significantly low memory footprint, and is able to handle arbitrary sized query and data graphs (because we handle only pairs of graphlets at a time). The concepts of hubs and minimum hub covers also help model various definitions of exact graph matching along the lines of [43, 53, 54], as well as approximate graph matching in the spirit of TALE.[5]

In our recent research on a declarative graph query language called *NyQL* [24], we have informally introduced the idea of the MHC and discussed how it can be exploited to represent graphs as nested relations and develop graph query operators in a way similar to the notion of the deep equality operator [1] in object-oriented databases.

Tangentially to this research, in our recent top-*k* graph matching algorithm TraM [2], we have explored the idea of hub matching as a unit of comparison and computed structural distance of attributed hubs without the need for explicit use of indices. In this approach, we have developed a quantification for a hub's structural feature as a random walk score [6]. Since random walk scores encompass the global topological properties of a node as a hub, from the standpoint of graph matching, it can be used to compare topological orientation and relative importance of graph nodes. These scores thus effectively capture the topological likeness and structural cues

DOI: 10.5824/1309-1581.2018.3.002.x

shared among the hubs and were effectively exploited for approximate graph matching in TraM. A similar method was used in [16] to compute computation of similarities of nodes of a graph for collaborative recommendation. The random walk-based approach, however, does not offer much opportunity for cost-based query optimization based on the evolving states of the database extension in ways similar to [57, 42] because it is largely similar in nature to many algorithmic counterparts such as SUMMA, NOVA, TALE, and SAPPER.

As we shall elaborate in the subsequent part of this section, the MHC problem is closely related to two well-known combinatorial optimization problems, the *set covering problem* (SCP) and the *minimum vertex cover* (MVC) problem, which are, in turn, share a similar mathematical programming model. We next discuss the relationship be- tween the MHC problem and the MVC problem, and then tie this discussion to a general set covering formulation that we also adopt in this work.

Let us start with the integer programming (IP) model for the MVC problem:

$$\text{minimize} \quad \sum_{j \in V} x_j, \tag{1}$$

$$\text{subject to} \quad x_i + x_j \geq 1 \qquad (i,j) \in E, \tag{2}$$

$$x_j \in \{0,1\}, \qquad j \in V, \tag{3}$$

where x_j is a binary variable that is equal to 1, if vertex j is in the cover and 0, otherwise. The objective function (1) evaluates the total number of vertices in the cover. Constraints (2) ensure that every edge is covered by at least one vertex, and constraints (3) enforce binary restrictions on the variables. To give a concrete example, suppose that we are trying to find the optimal MVC in the graph shown in figure 1(a). The corresponding IP model is then given by

$$\begin{aligned}
\text{minimize} \quad & x_1+x_2+x_3+x_4+x_5+x_6, \\
\text{subject to} \quad & x_1+x_2 \geq 1, \quad [(u_1,u_2)] \\
& x_1 + x_5 \geq 1, \quad [(u_1,u_5)] \\
& x_2+x_3 \geq 1, \quad [(u_2,u_3)] \\
& x_2 + x_5 \geq 1, \quad [(u_2,u_5)] \\
& x_3+x_4 \geq 1, \quad [(u_3,u_4)] \\
& x_3 + x_5 \geq 1, \quad [(u_3,u_5)] \\
& x_5+x_6 \geq 1, \quad [(u_5,u_6)] \\
& x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6 \in \{0,1\}.
\end{aligned}$$

The model minimizes the total number of selected vertices while satisfying the coverage constraints written for each edge. For the sake of clarity, the edge corresponding to a constraint is designated at the end of each line within the brackets. The first constraint, for example, implies that the edge (u_1, u_2) can be covered by vertices u_1 and u_2. Since an edge can be covered only by

the vertices incident to it, each constraint in the IP formulation of the MVC problem involves exactly two variables. In this example, $\{u_2, u_5\}$ is the unique optimal solution.
When it comes to the mathematical programming model of the MHC problem, we need to pay attention to the fact that a vertex (as a hub) covers not only the edges incident to itself but also those edges between its immediate neighbors. Using this fact, we obtain the following IP formulation of the MHC problem for the graph in Figure 1(a):

$$\begin{aligned}
\text{minimize } & x_1+x_2+x_3+x_4+x_5+x_6, \\
\text{subject to } & x_1+x_2 \quad +x_5 \quad \geq 1, \\
& x_2+x_3 \quad +x_5 \quad \geq 1, \\
& x_3+x_4 \quad \geq 1, \\
& x_5+x_6 \geq 1, \\
& x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6 \in \{0,1\}.
\end{aligned}$$

Notice that unlike the IP formulation of the MVC problem, the number of constraints re- duces since multiple edges can be covered by the same set of vertices. For instance, the second constraint shows that vertices u_2, u_3 and u_5 cover the edges (u_2, u_3), (u_2, u_5), and (u_3, u_5). Because of this hub property, the number of variables appearing in a constraint is greater than or equal to two. In fact, this number can easily go up to the number of vertices, because the vertices incident to an edge may be connected to all other vertices forming an abundant number of triangles. Clearly, the cardinality of the MHC can be far less than that of the cardinality of the MVC due to the additional non-incident edges covered by those vertices in a triangle. Therefore, for triangle-free graphs, the optimal solutions for the MHC problem and the MVC problem naturally coincide.

2.1 Minimum Hub Cover: As a Special Case of Set Covering

The above formulations of the MVC and MHC problems show that both problems are just the special cases of the SCP. Given a fixed number of items and a family of sets collectively including (covering) all these items, the objective of the SCP is to select the least number of sets (minimum cardinality collection) such that each item is in at least one of these selected sets. If an edge corresponds to an item and a set is formed with the edges that can be covered by each vertex, then the connection between the SCP and the MHC problem as well as the MVC problem can easily be established. To formalize this discussion, below we give the generic IP formulation for the MHC problem:

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

$$\text{subject to } x_i + x_j + \sum_{\substack{(i,k)\in E \\ (j,k)\in E}} x_k \geq 1, \qquad (i,j) \in E, \tag{5}$$

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

DOI: 10.5824/1309-1581.2018.3.002.x

Again, the binary variable x_j is equal to 1, if vertex j is in the hub cover, and the objective is to minimize the number of vertices used in the cover. Constraints (5) ensure that every edge is covered by at least one hub node in the cover. Finally, the constraints
(6) enforce the binary restrictions on the variables. Although the number of constraints seems equal to the number of edges, we remind that multiple edges can be covered by the same set of vertices (see MHC example above).
The introduction of the hub cover concept to the literature is quite recent [24]. Thus, to the best of our knowledge, the solution methods for the MHC problem have not been examined in the literature until recently in [50, 48]. However, closely related problems, the MVC problem and the SCP, have been extensively studied before. Take the MVC problem; approximation algorithms [18, 19], heuristic solutions [4], evolutionary algorithms [28, 14] can be listed among those numerous solution methods. The SCP is no different. From approximation algorithms [7, 18] that have good empirical performances to randomized greedy algorithms [15, 20], and from local search heuristics [30, 45] to different meta-heuristics [8, 35, 3] have been proposed for solving the SCP.

3 Minimum Hub Cover: The Formal Model

As mentioned earlier, in graph q_1, node or hub u_5 covers the edges (u_1, u_5), (u_1, u_2), (u_2, u_5), (u_2, u_3), (u_3, u_5) and (u_5, u_6). Similarly, the hub u_3 covers edges (u_3, u_2), (u_5, u_2), (u_3, u_5) and (u_3, u_4). Since the set of hubs $\{u_5, u_3\}$ covers all the edges of q_1, this set is called a *hub cover* of q_1, denoted $cover(q_1)$, and so are the sets $\{u_5, u_4\}$, $\{u_2, u_6, u_4\}$, and $\{u_1, u_5, u_4\}$. However, for the purposes of graph matching, it is sufficient if we matched the set $\{u_5, u_3\}$ with another graph as any super set this will not add any more new node or edge matching. We thus call this set the *minimum hub cover* of q_1, denoted $MHC(q_1)$ and formalized in the following definition.

Definition 1 (Minimum Hub Cover). For a given graph $G =< V, E >$, $M \subseteq V$ is a minimum hub cover of G, denoted $MHC(G)$, if M is the smallest set, and for every edge $< s, d > \in E$, either $d \in M$ or $s \in M$, or there exists edges $< s, c > \in E$ and $< d, c > \in E$, and $c \in M$. The set of all $MHC(G)$ is denoted as $\Gamma(G)$.

It should be apparent that given a graph $G = \langle V, E \rangle$, the number of hub nodes in M lies within the range $1 \leq |M| \leq \frac{|V|}{2}$. For example, both $\{u_5, u_3\}$ and $\{u_5, u_4\}$ are in $\Gamma(q_1)$, and $|\{u_5, u_3\}| = |\{u_5, u_4\}|$. It should also be apparent that hubs in a given MHC ensures edge connectivity because all edges are covered. But it is likely that a given sequence of hubs may not retain edge connectivity, and thus ordering of the elements in $MHC(G)$ matters and as such this ordering relationship is a key to processing queries efficiently. Intuitively, the query optimization approach we present essentially is the problem of computing the set $\Gamma(Q)$ when computable, then ordering the elements in each set $c_i \in \Gamma(G)$ using a cost function χ (discussed in detail in section 3.2) such that $\chi(c_i)$ is minimized, and then choose MHC c_j such that $\chi(c_j)$ is the minimum among all MHCs in $\Gamma(Q)$ under χ. The least cost ordering relationship implied by $\chi(c_j)$ is then taken as the query plan to be executed using a depth-first matching algorithm.

3.1. Hubs as Graph Representation

Traditionally, graphs are represented as a pair $\langle V, E\rangle$ where V is a set of vertices and E is a set of edges over V. Such a representation does not carry any structural information of which vertices are a part of, and they are not visible until structures are constructed from the set of edges. To ease computational hurdles and aid analysis, some models have used vertices and their neighbors as a unit of representation [41], i.e., $r_v = \langle v, N\rangle$ where v is vertex in V, and N is a set of neighbors such that $(v, n) \in E \Rightarrow n \in N$. A graph is then modeled as a set of such units. While this representation captures some structural cues, it still is pretty basic.

The decision on the degree of structural information that can be captured and exploited is very hard. In one extreme, we have the traditional representation $\langle V, E\rangle$ with no structural information other than the edges, and the GraphQL model where pretty much the entire graph structure is represented as a single XML document. While GraphQL offers most selectivity due to its representation, taking advantage of this in cost-based query optimization is extremely difficult because meta-data now has to be at the largest structural level. The models that are in between are the star structures [41] and dynamically mined frequent feature structures [47, 59] which we believe can generally be difficult to index and maintained with the evolution of the database.

We believe representing a graph as a set of hubs is a prudent compromise because it assures a deterministic model, and yet offers a realistic chance of efficient storage and processing of graph queries. It is deterministic because each hub can be represented as a triple of the form $r_v = \langle v, N_v, B_v\rangle$, called a *graphlet*, where $v \in V$ is a node in graph $G = \langle V, E\rangle$, and $N_v \subseteq V$, and $B_v \subseteq E$ such that for each $v \in V$ all $n \in N_v$ are its immediate neighbors, and every edge $b \in B_v$ are edges involving neighbors in N_v[6]. For unlabeled and undirected graphs, this representation model is sufficient. But for labeled and directed graphs, this simple model can be extended without any structural overhaul[7]. We can now simply define a graph as a set of graphlets, i.e., $G = \bigcup_{\forall v, v \in V} r_v$. In fact, in NyQL, we have shown that such a representation can be conveniently captured using traditional storage structures such as nested relations or XML documents.

[6] We follow this convention of representing a graphlet, equivalently called a node or hub, throughout the paper unless stated otherwise.

[7] For example, a hub of a node labeled undirected graph can be represented simply as $r_v = \langle v, L_v, N_v, B_v\rangle$ where L_v additionally represents the node label. The hubs in a fully labeled graphs can be modeled as yet another extension as $r_v = \langle v, L_v, N_{v_l}, B_{v_l}\rangle$ where N_{v_l} are a set of pairs (n, n_l) and B_{v_l} are triples (n_1, n_2, e_l) such that n_l and e_l are edge labels for edges between the hub and the neighbors, and among the neighbors respectively. The directionality of the edges can be captured by partitioning the sets N_v and B_v to imply directions. For example, the expression $\langle v, (N_{v_t}, N_{v_f}), (B_{v_t}, B_{v_f})\rangle$ means that (i) there are edges from v to every node in N_{v_t}, and from N_{v_f} to v, and (ii) for each edge (n_1, n_2) in B_{v_t}, the sink node is n_2, and for edges in B_{v_f}, it is reversed.

[8] Recall that G is now a set of graphlets, or basically a set of nested tuples.

AJIT-e: Online Academic Journal of Information Technology
2018 Yaz/Summer – Cilt/Vol: 9 - Sayı/Num: 33
DOI: 10.5824/1309-1581.2018.3.002.x

3.2 Cost Function χ and Graphlet Ordering Relation $\prec$

Given two graphlets r_v and r_u, and a graph D, we define an ordering relation $r_v \preceq r_u$, called the *selectivity ordering*, between them to mean r_v precedes r_u (or r_v costs less than r_u in processing time) in two principal ways.

- First, when the selectivity of r_v, denoted $sel(r_v, D)$, is higher than $sel(r_u, D)$. Technically, $sel(x, G)$ means the number of graphlet instances r_y in graph G that satisfy the condition $|N_y| \geq |N_x|$ and $|B_y| \geq |B_x|$, and selectivity of r_v is higher than that of r_u in a graph G if $sel(r_v, D) \leq sel(r_u, D)$. We call it *node selectivity*, denoted $\preceq_v$.
- Second, when the selectivity of N_v and B_v, called the *structural selectivity* and denoted $\preceq_s$, is higher than the selectivity of N_u and B_u, denoted respectively as $sel(N_v, B_v, D)$ and $sel(N_u, B_u, D)$, i.e., $sel(N_v, B_v, D) \leq sel(N_u, B_u, D)$, where $sel(N_x, B_x, D)$ is the number of graphlet instances r_y in graph G that satisfy the condition $|N_y| \geq |N_x|$ and $|B_y| \geq |B_x|$.
- Finally, the *neighbor selectivity* (denoted $\prec_n$) $sel(N_v, D) \leq sel(N_u, D)$ holds where $sel(N_x, D)$ represents the number of graphlet instances r_y in graph G that satisfy the condition $|N_y| \geq |N_x|$.

The query plan as the total ordering $\prec$ then can be induced from the precedence relationship $\preceq$ of graphlets in G using the procedure **Graphlet Ordering** in algorithm 1. Algorithm 1 is intuitively explained in figure 3 where we show that the dot nodes are identified first followed by the star nodes and ordered in terms of their selectivity. In figure 3(a), the three components of a graphlet are shown in brown, green and yellow, where the neighbors N_v are partitioned in to two sets $mapped(N_v) \subseteq N_v[\theta]$ and unmapped subsets (shown in uppercase). Since an instantiated dot node may only fetch a single hub at the most, this ordering is likely to result in a least cost plan.

3.3 Indices for Efficient Access

Let us also assume that we have a hash index I_H that given a node ID (i.e., $v \in V$), returns the graphlet corresponding to v, i.e., $I_H(v) = r_v$. Furthermore, we also have another index I_S similar to [40] such that it returns a set of graphlets R given a set of nodes V_q, and an integer n_c, i.e., $I_S(V_q, n_c) \subseteq G^8$, such that for each $r_v \in R, V_q \subseteq N$, and $|N| \geq n_c$. In other words, I_S returns all graphlets that contain the set of nodes in V_q and have a minimum of $|V_q| \leq n_c$ neighbors.

Algorithm 1: Graphlet Ordering

Input: A graph G, a cost estimator χ, a matching mode and data dictionary.
Output: A total ordering $\prec$ on G.

```
1  Set i = 1.
2  Choose the most selective graphlet as r_{v_i} using node selectivity.
3  while not all graphlet in ≺ marked dead do
4      foreach unmarked graphlet r_v in ≺ order do
5          while ∃r_x, r_x ∈ G not marked dead and x ∈ N_v (i.e., x is an immediate
             neighbor of v) do
6              for ∀r_y, r_y ∈ N_v do
7                  if r_x ≺_v r_y or r_y ≺_v r_x ∧ r_x ≺_s r_y holds then
8                      Increment i.
9                      Let r_{v_{i+1}} = r_x.
10                     Mark r_{v_{i+1}} as a dot node, i.e., r•_{v_{i+1}}.
11     foreach unmarked graphlet r_v in ≺ order do
12         while ∃r_x, r_x ∈ G not marked dead and N_v ∩ N_x ≠ ∅ (i.e., x is a 2-hop
             neighbor of v and share the nodes N_v ∩ N_x as neighbors) do
13             for ∀r_y, N_y ∩ N_v ≠ ∅ do
14                 if r_x ≺_n r_y or r_y ≺_n r_x ∧ r_x ≺_s r_y holds then
15                     Increment i.
16                     Let r_{v_{i+1}} = r_x.
17                     Mark r_{v_{i+1}} as a star node, i.e., r*_{v_{i+1}}.
18         Mark r_v dead.
19 return Order ≺.
```

3.4 Outline of a Query Processor

The graph query processor for exact matching is simple. It is composed of a depth-first search algorithm **Find Solutions** that maps a sequence of graphlets in a query graph presented in $\prec$ order to a set of graphlets of a data graph using a composable term mapping function μ. Algorithm 2 is assumed to have access to a data graph D, one of four graph matching modes, and the indices I_H and I_S.

3.5 Computational Characterization of Minimum Hub Cover

We formally present the computational characterization of MHC in terms of its complexity class, and using the decision version of the MHC problem (MHC-D) show that the MHC problem is indeed difficult and that it is NP-complete.

Theorem 1. *MHC-D is NP-complete.*

Proof. To show that MHC-D is NP-Complete, we answer the following question: given a graph $G(V, E)$ and an integer $k \leq |V|$, does there exist a subset $C \subseteq V$ with $|V| \leq k$ such that for every edge $(s, d) \in E$, $s \in C$ or $d \in C$ or there exists a vertex $c \in C$ such

AJIT-e: Online Academic Journal of Information Technology
2018 Yaz/Summer – Cilt/Vol: 9 - Sayı/Num: 33
DOI: 10.5824/1309-1581.2018.3.002.x

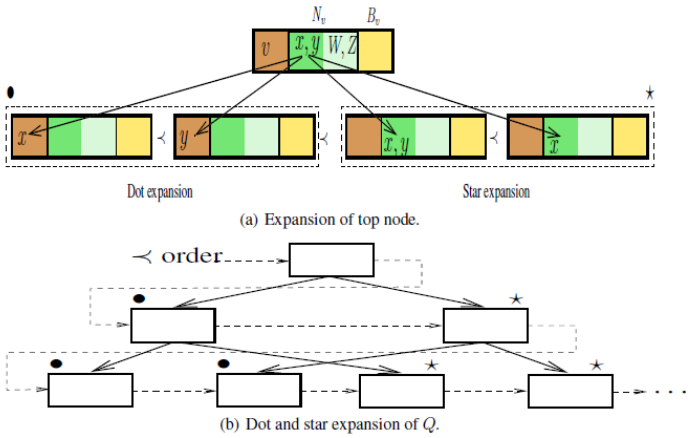


Fig. 3. Ordering Q based on expected cost using selectivity.

that both $(s, c) \in E$ and $(d, c) \in E$? First, we argue that MHC-D is in NP because for a given yes-instance and $C' \subseteq V$ with $|C'| \leq k$, we can verify in polynomial time that every edge in E is covered by C'. We now complete the proof by a reduction from the MVC problem on triangle free graphs [34]. In triangle-free graphs, for all $(s, d) \in E$ there does not exist a vertex c such that $(s, c) \in E$ and $(d, c) \in E$. Thus, either s, or d must be in C' to cover the edge (s, d). Consequently, the MHC and the MVC of a graph are equivalent in this class of graphs and MVC is NP-complete for this class of graphs [34].

4 Experimental Design

The NP-completeness of MHC speaks to the hardness of the solvability in general. The optimization strategy and query plan generation presented in section 3, however, calls for the computation of all MHCs of a given query Q, i.e., $\Gamma(Q)$, which is indeed a hard problem. In this section, our goal is to design a set of experiments using different graph types, size and graph density parameters to study how the solvability and the quality of MHC solutions depend on these parameters. The goal is to experimentally identify problem classes for which available solutions are practical and acceptable, and the classes for which new heuristic solutions are warranted. We therefore employ different algorithms to show the trade-offs between optimality and computation time over different graph types. Although our experiments and analysis involve producing only one optimal solution for a graph, the insights gained can be leveraged to develop new algorithms to compute $\Gamma(Q)$ or to choose other strategies when computing $\Gamma(Q)$ is infeasible.

Algorithm 2: Find Solutions

Input: A stack T with query graph Q in $\prec$ order (initially r_{v_1} as the top element), and a substitution list θ (initially empty).
Output: Set sol of all matched subgraphs of D.

```
if stack T empty then
    Include Q[θ] in sol.
    return.
Let r_v = pop(T).
Apply substitution θ to r_v, i.e., r_v ← r_v[θ].
if r_v is not a dot node and a star node (it is the top node in ≺ order) then
    Retrieve all r_x from D such that |N_x| ≥ |N_v and |B_x| ≥ |B_v.
else
    if r_v is a dot node then
        Retrieve r_x ← I_H(v) from D.
    else
        if r_v is a star node then
            Let M = mapped(N_v).
            Retrieve all r_x ∈ I_S(M, |N_v|)
foreach r_x do
    if exists one substructure of r_x similar to r_v then
        foreach substructure r_y of r_x[θ] similar to r_v do
            Find Solutions (T, μ(r_y, r_v) • θ).
    else
        return.
return sol
```

The optimal linear programming (LP) and IP solutions are obtained by `ILOG IBM CPLEX 12.4` on a personal computer with an Intel Core 2 Dual processor and 3.25 GB of RAM. In all problem instances, the upper limit on the computation is set at 3,600 seconds. The batch processing of the instances is carried out through simple C++ scripts. Our data set includes a total of 830 instances. We have 5 different instances for each combination of a graph type, size, and density parameter to be able to draw conclusions.

4.1 Selected Graph Types and Problem Classes

We have chosen to use the benchmark database graph instances in [38] and our own synthetically generated data set for our numerical study. This is a very large database of different graph types and sizes designed specifically to test the sophistication of (sub)graph isomorphism algorithms. Since we are using subgraph isomorphism as a basic vehicle for graph matching, the instances selected are thus representative of the class of queries we are likely to handle when we solve the MHC problem. The descriptions of the graph instances we have chosen from this collection are listed below.

Randomly connected graphs These graphs have no special structure and the number of vertices range from 20 to 1000 ($|V|$ = 20, 60, 100, 200, 600, 1000). The parameter η denotes the probability

DOI: 10.5824/1309-1581.2018.3.002.x

of having an edge between any pair of vertices. Thus, this parameter, in a sense, specifies the sparsity of a graph. In the database, three different values of η (0.01, 0.05, and 0.10) are considered. Our data set includes a set of graphs of different sizes for each value of η.

Bounded valence graphs The vertices of the graphs in this class have the same degree (*fixed valence*). The sizes of the instances are like those of the problem class (a). We use three different values of valence – 3, 6 and 9 to obtain graphs of different size and valence.

Irregular bounded valence graphs These graphs are generated by introducing irregularities in the problem class (b). Irregularity comes from randomly deleting edges and adding them elsewhere in the graph. With this modification, the average degree is again bounded but some of the vertices may have higher degrees. The sizes of the instances are like those of problem classes (a) and (b).

Regular meshes with 2D, 3D, and 4D In graphs with 2D, 3D, and 4D meshes, each vertex has connections with 4, 6, and 8 neighbors, and the numbers of vertices range from 16 to 1024, 27 to 1000, and 16 to 1296, respectively. Similar to the problem classes (a), (b), and (c), we have a set of graphs for each combination of size and dimension.

Irregular meshes As in class (c), irregular meshes are generated by introducing small irregularities to the regular meshes. Irregularity comes from the addition of a certain number of edges to the graph. The number of edges added to the graph is ρ x |V|, where $\rho \in \{0.2, 0.4, 0.6\}$.The number of vertices is the same as in problem class (d).

Scale-free graphs This problem class includes the graphs that follow a power-law distribution of the form

$$P(k) \sim ck^{-\alpha},$$

where $P(k)$ is the probability that a randomly selected vertex has exactly k edges, c is the normalization constant, and $2 \leq \alpha \leq 3$ is a fixed parameter. We employed the scale-free graph generator of C++ Boost Graph Library. The generator (Power Law Out Degree algorithm) takes three inputs. These are the number or vertices, α and β. Increasing the value of β increases the average degree of vertices. On the other hand, increasing the value of α decreases the probability of observing vertices with high degrees. The sizes of the instances range from 20 to 1000; $|V| \in \{20, 60, 100, 200, 600, 1000\}$ to be precise. We considered two values for $\alpha \in \{1.5, 2.5\}$ and three values of $\beta \in \{100 \text{ x } |V|, 200 \text{ x } |V|, 500 \text{ x } |V|\}$. Graphs in social networks, protein-protein interaction networks, and computer networks are examples of this class.

4.2 Solution Methods Used

We choose three solution methods to compute MHC – (i) an exact method to solve the problem to optimality, (ii) adapt two approximation algorithms from the vertex cover literature capable of computing feasible solutions fast, and (iii) a mathematical programming-based heuristic originally proposed for solving the SCP.

Exact algorithm The IP formulation (4) - (6) is solved by an off-the-shelf solver to optimality. Since the MHC problem is shown to be NP-Hard, this approach may have practical value only for small-to-medium-scale graphs. However, it sets a definitive benchmark for comparing the performances of various heuristics.

Approximation and greedy algorithms We implemented two different approximation algorithms. First algorithm selects the vertex with the highest degree at each iteration. The aim is to cover as many edges as possible. Next, all covered edges as well as the vertices in the cover are removed from the graph. The algorithm ends when there is no uncovered edge in the graph. The algorithm is called the $H(\Delta)$-*approximation algorithm* (GR1) for the MVC problem. Here, Δ is the maximum degree in the graph, and $H(\Delta)$ is evaluated by

$$H(\Delta) = 1 + 1/2 + \ldots + 1/\Delta.$$

The second algorithm (GR2), the *2-approximation algorithm*, is an adaptation of [5] originally proposed for computing a near-optimal solution for the MVC problem. Un- like the previous algorithm, it selects an edge arbitrarily, then both vertices incident to that edge are added to the cover.

Mathematical programming-based heuristics Yelbay et al. [50] propose a heuristic (MBH) that uses the dual information obtained from the LP relaxation of the IP model of SCP. They show the efficacy of the heuristic on a large set of SCP instances. In their work, the dual information is used to identify the most promising columns and then form a restricted problem with those columns. Then, an integer feasible solution is found by one of the two approaches. In the first approach (MBH), the exact IP optimal solution is obtained by solving *the restricted problem*. In the second approach, a METARAPS [30] local search heuristic (LSLP) is applied over those promising columns. We use both approaches.

5 Analysis of Experimental Results

We focus on analyzing and understanding the MHC solution methods in section 4.2 on the instances in section 4.1 in three different axes: (i) optimal solvability of MHC, (ii) quality of the solutions, and (iii) computational cost of optimal solution. These analyses are aimed at understanding which problem classes are inherently more difficult relative to others so that depending on the application and query, a suitable algorithm can be selected to compute MHC. We also discuss the factors that increase the complexity of the problems.

5.1 Optimal Solvability of Minimum Hub Covers

Figure 4 (continued in Figure 5) shows how the optimal solution time of `CPLEX`, an exact method, varies depending on the problem size, class, and structure. The x-axis and the y-axis represent the number of vertices and the average computation time, respectively. The right-most data point on a line shows the size of the largest instance that can be solved to optimality in a group. In general, its performance is good for small to medium scale graphs. However, in our study, 39 out of 90, 73 out of 285 and 39 out of 180 instances in problem classes (a), (e) and (f), respectively, could not be solved optimally using `CPLEX` within the time limit. This observation opens the door for heuristics to find acceptable but possibly suboptimal solutions.

Random graphs From Figure 4(a) we conclude that for randomly connected graphs with more than 200 nodes, optimal solution is not achievable within the bounded time. It also suggests that the

DOI: 10.5824/1309-1581.2018.3.002.x

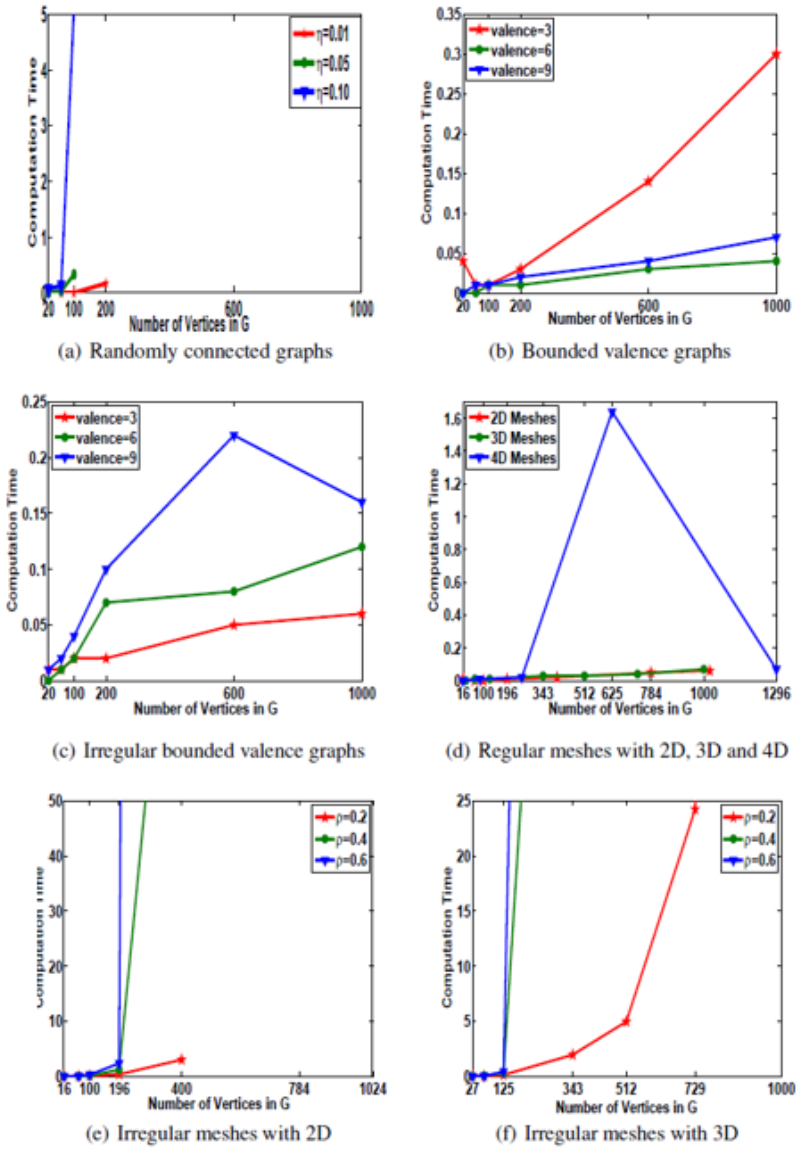
density of graphs is a factor that affects the solvability. The solver does increasingly better as the density η goes down (up to 0.01) for the same number of vertices. Its sensitivity with respect to the size and density is apparent in the plots for η equal to 0.05 and 0.10, i.e., a 16-fold increase in solution time.

Bounded valance graphs Compared to random graphs, Figure 4(b) shows an improved performance on bounded valence graphs solving all instances under 0.3 seconds. The reason for the performance difference may be due to the considerably higher number of edges in a randomly connected graph (which forces the number constraints in the IP model to go higher) than that of a bounded valence graph. However, although we expect higher solution time for graphs with larger valence, Figure 4(b) shows substantially higher time for valence 3 than valences 6 and 9 suggesting other factors may also be playing a role.

Irregular bounded valance graphs Although the degree distribution is neither constant nor fully randomly distributed, CPLEX performs similarly to bounded valence graphs. As Figure 4(c) shows, all solutions are computed in less than 0.25 seconds, and that the computation time increases with the increase in valence.

Regular mesh graphs Figure 4(d) shows that the size of meshes (2D, 3D or 4D) usually does not have any influence on the performance barring the abrupt behavior of the 4D mesh graph. In general, the solution time appears to linearly increase with the increase in graph size, though the increase in time is extremely small.

Irregular mesh graphs Unlike the irregular bounded valence graphs, mesh graphs are more susceptible to irregularity and the computation time substantially increases with the degree of irregularity. Figures 4(e), 4(f) and 5(a) show that the sizes of the problems that can be solved to optimality decrease and the computation times increase with increasing degree of irregularity. This result is quite reasonable and expected because increasing irregularity increases the number of edges, and thus the computation time as well. This is also because randomly adding edges to a mesh graph makes it structurally more similar to random graphs, which, as discussed earlier, is inherently hard to solve.



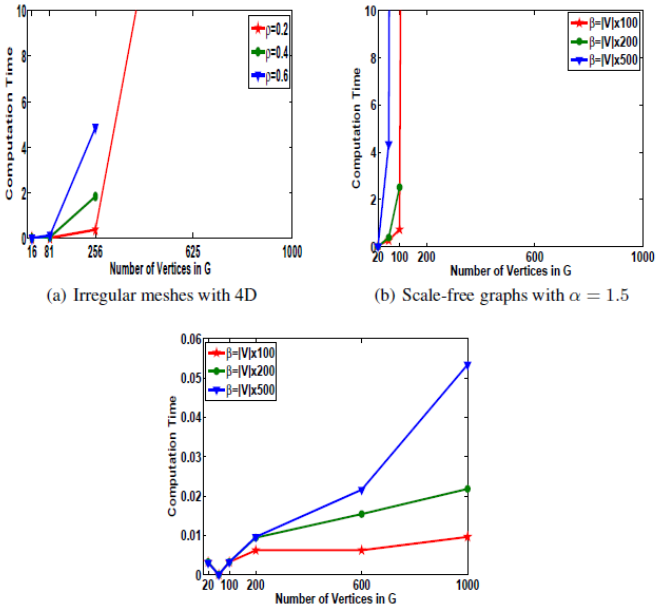
(a) Randomly connected graphs (b) Bounded valence graphs

(c) Irregular bounded valence graphs (d) Regular meshes with 2D, 3D and 4D

(e) Irregular meshes with 2D (f) Irregular meshes with 3D

Fig. 4. Average computation time of CPLEX on problem classes as a function of the number of vertices in G and the parameters of the problem classes

DOI: 10.5824/1309-1581.2018.3.002.x



(a) Irregular meshes with 4D (b) Scale-free graphs with $\alpha = 1.5$

Fig. 5. Figure 4 continued: Average computation time of `CPLEX` on problem classes as a function of the number of vertices in G and the parameters of the problem classes

Scale-free graphs We consider the effect of the two parameters α and β on the solvability of the problems. On one hand, increasing α makes the degree distribution sharper, i.e., we observe smaller number of vertices with high degrees. On the other hand, in- creasing the value of β increases the degrees of non-hub nodes. Figures 5(b) and 5(c) represent the optimal solution times of scale-free instances. The difficulty of the problem is closely related to parameters α and β. The figures show that computation times decrease significantly with increasing values of α. When α = 1.5, the instances with more than 100 vertices cannot be solved to optimality within the time limit. When α = 2.5, however, all instances can be solved optimally in less than 0.06 seconds. These figures also show that the computation time increases with increasing values of β. This means that increasing degrees of non-hub nodes makes the problem more difficult.

5.2 Performance Profile of Solution Methods

To study the quality of solutions generated by other solution methods with respect to the optimal solutions computed using CPLEX, we refer to figures 6(a) through 6(e). These plots are called performance profiles of algorithms that depict the fraction of problems for which the algorithm is within a factor of the best solution [13]. Thus, they compare the performance of an algorithm s on an instance p with the best performance observed by any other algorithm on the same instance. The x-axis represents the *performance ratio* given by

$$r_{p,s} = \frac{\alpha_{p,s}}{\min\{\alpha_{p,s} : s \in S\}},$$

DOI: 10.5824/1309-1581.2018.3.002.x

where $\alpha_{p,s}$ is the number of hub nodes in the hub cover when the instance p is solved by algorithm s and S is the set of all benchmark algorithms. The y-axis shows the percentage of the instances that gives a solution that is less than or equal to τ times the best solution. Recall that CPLEX cannot solve all instances in problem classes (a), (e) and (f) to optimality. However, the solver can find feasible solutions for some of those unsolved instances (11 out of 39, 44 out of 73, and 24 out of 39 in (a), (e), and (f) respectively). Figure 6 includes all instances except those for which CPLEX cannot find either feasible or optimal solutions within the time limit.

We first analyze how much we sacrifice from the optimality by employing mathematical programming-based heuristics MBH and LSLP. Recall that MBH and LSLP solve the same restricted problem. While MBH tries to solve the problem to optimality, the latter visits alternate solutions in the feasible region. Figure 6(a) shows that 12% of the instances in class (a) where both MBH and LSLP find better feasible solutions than that of CPLEX. Note that, this can happen if and only if CPLEX returns a feasible solution rather than an optimal solution within the time limit. For other problem classes, the performances of the CPLEX and MBH are quite similar. Moreover, these figures show that LSLP is outperformed by MBH and CPLEX on almost 40% and 30% of in- stances in problem classes (b) and (c) respectively. For the remaining problem classes, the performance of LSLP is also comparable to MBH and CPLEX.

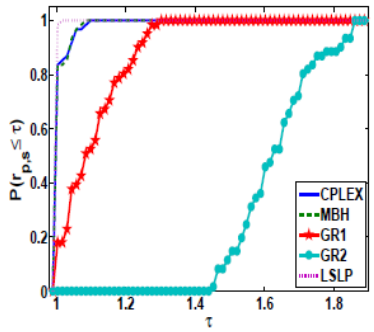
The greedy algorithms return feasible but sub-optimal solutions quickly. Except the scale-free networks, the performances of the greedy algorithms do not change with respect to problem classes. GR2 is known as 2-approximation algorithm for MVC. Figures 6(c) and 6(f) show that there are some instances for which performance ratios of GR2 are higher than 2. Obviously, the approximation ratio of GR1 for MHC problem is higher than 2. Intuitively, the performance of GR1 is supposed to be better when the degree distribution of the vertices is not uniform. Since the average degrees of the vertices are identical or are quite similar for the instances in problem classes (a)-(e), the performance of GR1 does not vary for these problem classes. However, figure 6(f) shows that GR1 finds the optimal or the best solution in 30% of the scale-free instances. This means that the performance of the GR1 is better for the graphs that follow the power- law distribution.

5.3 Cost Profile of Solution Methods

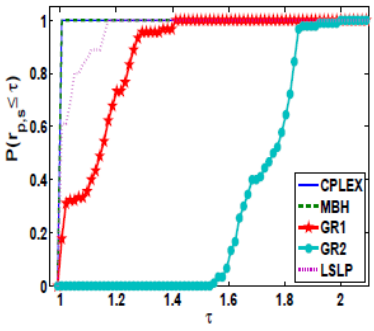
The previous two analyses focused on the optimal solvability and the quality of the solu- tions. In this section we turn our attention to the cost of computing a feasible or optimal MHC solutions in terms of time. Figures 7(a) through 7(f) summarize the distribution of the average computation times of the algorithms over the problem classes. In these plots, the instances for which feasible solutions were not found by any of the algorithm within a time limit are excluded. Each bar in the figure represents the percentage of the instances that are solved within the time interval stated in the legend, e.g., the blue bar for 0.0 to 0.05 seconds. Since LSLP is a local search algorithm, we show both the total computation time and the first time when the best solution is found.

The results clearly show that the solution times of greedy algorithms (GR1, GR2) are much shorter than that of the other algorithms. MBH can solve the restricted problem to optimality

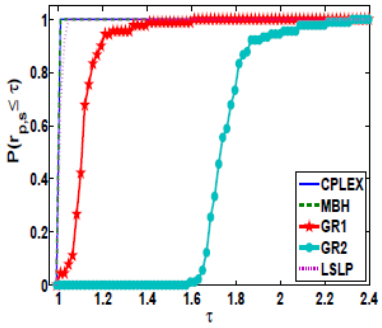
in a reasonable amount of time for a great majority of the instances. We have already discussed earlier that the performance of the MBH is good in terms of its solution quality. However, the main drawback for MBH is its inability to solve the restricted problem to optimality.
In such cases, LSLP may serve as an alternative to MBH as it is comparable to MBH in terms of both solution quality and time, and because LSLP is a local search algorithm, it is also guaranteed to produce a feasible solution. However, the performance of LSLP is dependent upon prudent selection of algorithmic parameters, e.g., the total number of iterations, the number of improvement iterations (see [50] for details). There is a trade-off between solution time and the solution quality. Decreasing the total number of iterations may result in a decrease in the total solution time. However, it may increase the optimality gap.



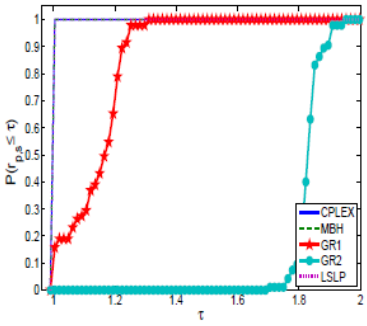
(a) Randomly connected graphs



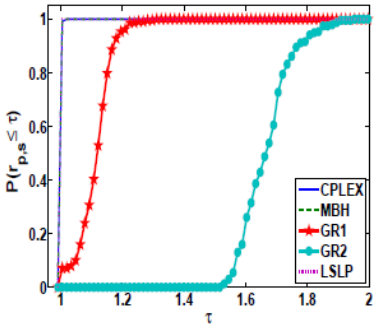
(b) Bounded valence graphs



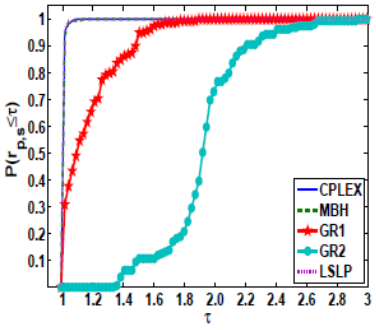
(c) Irregular bounded valence graphs



(d) Regular meshes with 2D, 3D and 4D



(e) Irregular meshes



(f) Scale-free graphs

DOI: 10.5824/1309-1581.2018.3.002.x

Fig. 6. Performance profiles for the algorithms on the problem classes in terms of solution quality

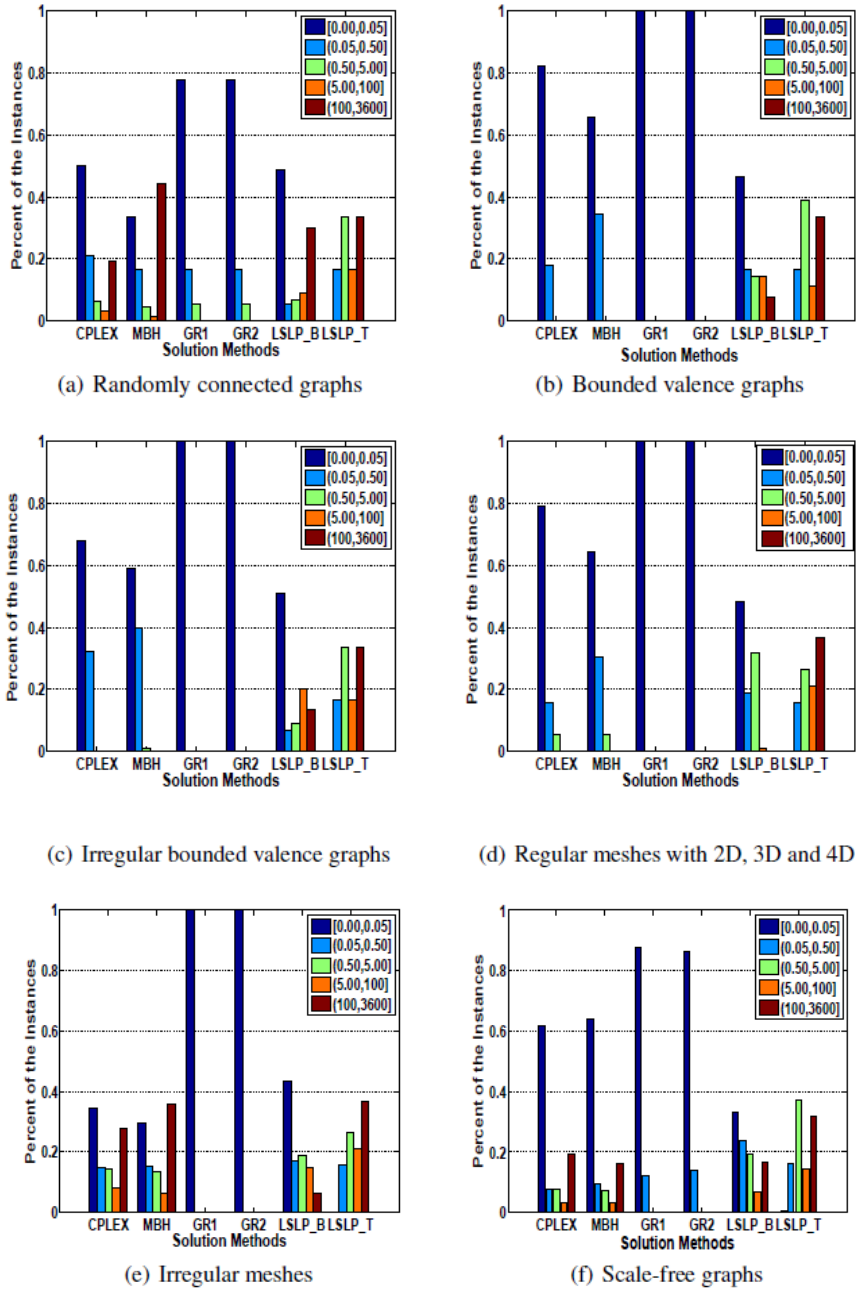


Fig. 7. Computation time distributions of the solution methods on the problem classes

6 Future Research Opportunities

Constraint solvers such as CPLEX usually do not offer all optimal solutions. Such solutions also do not exploit database meta-data in computing the most desired solution. For example, for the graph in figure 1(c), there is no guarantee that the solver will pro- duce the desirable

solution w_5, w_4 when the data graph is known to be d. Therefore, we are required to compute all possible MHCs of q_2, i.e., $\Gamma(q_2)$, so that we can identify this least cost query plan. Unfortunately, it is not guaranteed that solvers can even always compute a solution, let alone the whole family of solutions $\Gamma(Q)$ of Q.

The discussion in section 5 also suggests that although the cost of computing MHC is significantly low for small graphs, it remains high for many graph types when the query graph is large. Therefore, even though some of the existing solvers may be useful for applications involving small scale-free graphs such as protein-protein interaction networks, they may not be a great candidate for big data applications in social networks and world wide web. Though it may be challenging, we believe the low MHC solution time for many graph types offers hope that designing algorithms for $\Gamma(Q)$ is feasible for most practical applications but remains as an interesting problem.

It is also important to recognize that while developing the least cost MHC using meta-data may be feasible for a single data graph, devising such algorithm for a large set of data graphs may not be feasible. It is thus worth investigating if a general but a single optimal query plan (without computing $\Gamma(Q)$), for which we have a solution, can be dynamically adjusted for best performance over a set of graphs. Finally, it remains an open question if a suboptimal MHC produced by a greedy algorithm can be improved enough to defeat or match the overall processing performance using an optimal MHC solution, i.e., total cost of MHC, plan generation, plan selection and execution.

7 Conclusions

In this paper we have formally introduced the idea of graphlets as a basic unit for graph representation in a way like RDF triple store, and the concept of minimum hub cover of graphlets as a basic ingredient toward graph query optimization. We have demonstrated on intuitive grounds that such an approach can leverage generic access structures such as hash [11] and set indices [40, 31] for query optimization. Though computationally hard, we have also demonstrated that query processing and optimization using MHC and subgraph isomorphism is computationally feasible and intellectually intriguing. We have shown that for many application domains of cur- rent interest such as social networks, and protein-protein interaction networks, existing constraint solvers can deliver optimal solutions for MHC, and therefore can be used to develop optimization strategies. It is our thesis that covering based graph processing we have presented opens new research directions and holds enormous promise. The logical next step is to develop a query processor by integrating the algorithms in sections 3.2 and 3.4, with new algorithms outlined in section 6. These are some of the tasks we plan to continue as our future research.

8 Acknowledgements

Belma Yelbay's research was partially supported by TUBITAK 2214 Ph.D. Research Scholarship Program, and Hasan Jamil's research was supported in part by NSF grants IIS 0612203 and DRL 1515550.

DOI: 10.5824/1309-1581.2018.3.002.x

Refefences

1. S. Abiteboul and J. V. den Bussche. Deep equality revisited. In *DOOD*, pages 213–228, 1995.

2. S. Amin, J. Russell L. Finley, and H. M. Jamil. Top-k similar graph matching using TraM in biological networks. *ACM/IEEE TCBB*, 2012. http://doi.ieeecomputersociety.org/10.1109/TCBB.2012.90.

3. Z. N. Azimi, P. Toth, and L. Galli. An electromagnetism metaheuristic for the unicost set covering problem. *European Journal of Operational Research*, 205:290–300, 2010.

4. S. Balaji, V. Swaminathan, and K. Kannan. Optimization of unweighted minimum vertex cover. *World Academy of Science, Engineering and Technology*, 67:508–513, 2010.

5. R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

6. P. Boldi, M. Santini, and S. Vigna. A deeper investigation of page rank as a function of the damping factor. In *Web Information Retrieval and Linear Algebra Algorithms*, 2007.

7. A. Caprara, P. Toth, and M. Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98:353–371, 2000.

8.M. Caserta. *Metaheuristics: progress in complex systems optimization*, chapter 3, pages 43–

63. Springer, Berlin, 2007.

9.C. Chen, X. Yan, P. S. Yu, J. Han, D.-Q. Zhang, and X. Gu. Towards graph containment search and indexing. In *VLDB*, pages 926–937, 2007.

10.J. Chen and I. A. Kanj. On approximating minimum vertex cover for graphs with perfect matching. *Theor. Comput. Sci.*, 337(1-3):305–318, 2005.

11.S. M. Chung. Indexed extendible hashing. *Inf. Process. Lett.*, 44(1):1–6, 1992.

12.L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Performance evaluation of the vf graph matching algorithm. In *Image Analysis and Processing*, pages 1172–1177, 1999.

13.E. Dolan and J. More. Benchmarking optimization software with performance profiles.

Mathematical Programming, 91:201–213, 2002.

14.I. Evans. Evolutionary algorithms for vertex cover. In *7th Annual Conference Evolutionary Programming*, pages 377–386, New York, 1998.

15.T. A. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

16.F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *TKDE*, 19(3):355–369, 2007.

17.R. Giugno and D. Shasha. GraphGrep: A fast and universal method for querying graphs. In *ICPR*, volume 2, pages 112–115, 2002.

18.F. Gomes, C. Meneses, P. Pardalos, and G. Viana. Experimental analysis of approximation algorithms for the veretx cover and set covering problems. *Computers and Operations Research*, 33:3520–3534, 2006.

19.E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31:1608–1623, 2001.

20.M. Haouari and J. S. Chaouachi. A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem. *Journal of the Operational Research Society*, 53:792–799, 2002.

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

22.T. Imamura, K. Iwama, and T. Tsukiji. Approximated vertex cover for graphs with perfect matchings. *IEICE Transactions*, 89-D (8):2405–2410, 2006.

23.H. M. Jamil. Computing subgraph isomorphic queries using structural unification and minimum graph structures. In *ACM International Symposium on Applied Computing*, pages 1053–1058, Taichung, Taiwan, March 2011.

24.H. M. Jamil. Design of declarative graph query languages: On the choice between value, pattern and object-based representations for graphs. In *ICDE Workshop on Graph Data Management*, April 2012.

25.H. M. Jamil. Pruning forests to find the trees. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management, SSDBM 2016, Budapest, Hungary, July 18-20, 2016*, pages 18:1–18:12, 2016.

26.H. M. Jamil. A visual interface for querying heterogeneous phylogenetic databases. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–14, January 2016. DOI 10.1109/TCBB.2016.2520943.

27.H. Kardes and M. H. Günes. Structural graph indexing for mining complex networks. In *ICDCS Workshops*, pages 99–104, 2010.

28.S. Khuri and T. Back. An evolutionary heuristic for the minimum vertex cover problem. In *Workshop 18th Annual German Conference Artificial Intelligence*, pages 86–90, Saarbrucken, Germany, 1994.

29.Y. Kou, Y. Li, and X. Meng. Dsi: A method for indexing large graphs using distance set. In *WAIM*, pages 297–308, 2010.

30.G. Lan, G. W. DePuy, and G. E. Whitehouse. An effective and simple heuristic for the set covering problem. *European Journal of Operational Research*, 176:1387–1403, 2007.

31.N. Mamoulis. Efficient processing of joins on set-valued attributes. In *SIGMOD Conference*, pages 157–168, 2003.

32.M. Mongiov `ı, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. Sigma: a set-cover-based inexact graph matching algorithm. *J. Bioinformatics and Computational Biology*, 8(2):199–218, 2010.

33.M. Mongiov `ı, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. Sigma: a set-cover-based inexact graph matching algorithm. *J. Bioin. and Comp. Bio.*, 8(2):199–218, 2010.

34.S. Poljak. A note on stable sets and coloring of graphs. *Commun. Math. Univ. Carolinae*, 15:307–309, 1974.

35.Z. G. Ren, Z. R. Feng, L. J. Ke, and Z. J. Zhang. New ideas for applying ant colony optimization to the set covering problem. *Computers & Industrial Engineering*, 58:774–784, 2010.

36.C. R. Rivero and H. M. Jamil. On isomorphic matching of large disk-resident graphs using an xquery engine. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE, Chicago, IL, USA, March 31 - April 4*, pages 20–27, 2014.

37.C. R. Rivero and H. M. Jamil. Efficient and scalable labeled subgraph matching using SG- Match. *Knowledge and Information Systems*, 2016. DOI 10.1007/s10115-016-0968-2.

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

39.H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *PVLDB*, 1:364–375, 2008.

40.M. Terrovitis, P. Bouros, P. Vassiliadis, T. K. Sellis, and N. Mamoulis. Efficient answering of set containment queries for skewed item distributions. In *EDBT*, pages 225–236, 2011.

41.Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. *ICDE*, pages 963–972, 2008.

42.S. Trißl. Cost-based optimization of graph queries. In *Workshop on Innovative Database Research, SIGMOD*, 2007.

43.S. Trißl and U. Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD Conference*, pages 845–856, 2007.

44.J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

45.M. Yagiura, M. Kishida, and T. Ibaraki. A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, 172:472–499, 2006.

46.X. Yan, P. S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. *ACM Trans. Database Syst.*, 30(4):960–993, 2005.

47.X. Yan, P. S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. *TODS*, 30(4):960–993, 2005.

48.B. Yelbay. *Minimum hub cover problem: Solution methods and applications*. Dissertation, Sabanci University, Turkey, 2014.

49.B. Yelbay, S. I. Birbil, K. Bülbül, and H. M. Jamil. Approximating the minimum hub cover problem on planar graphs. *Optimization Letters*, 10(1):33–45, 2016.

50.B. Yelbay, S¸. İ.Birbil, and K. Bülbül. The set covering problem revisited: An empirical study of the value of dual information. *Journal of Industrial and Management Optimization*, 11(2):575–594, 2015. DOI 10.3934/jimo.2015.11.575.

51.D. Yuan and P. Mitra. A lattice-based graph index for subgraph search. In *WebDB*, 2011.

52.L. A. Zager and G. C. Verghese. Graph similarity scoring and matching. *Appl. Math. Lett.*, 21(1):86–94, 2008.

53.S. Zhang, J. Li, H. Gao, and Z. Zou. A novel approach for efficient supergraph query processing on graph databases. In *EDBT*, pages 204–215, 2009.

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

55.S. Zhang, S. Li, and J. Yang. Summa: subgraph matching in massive graphs. In *CIKM*, pages 1285–1288, 2010.

56.S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1):1185–1194, 2010.

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

58.K. Zhu, Y. Zhang, X. Lin, G. Zhu, and W. Wang. Nova: A novel and efficient framework for finding subgraph isomorphism mappings in large graphs. In *DASFAA (1)*, pages 140–154, 2010.

59.L. Zou, L. Chen, H. Zhang, Y. Lu, and Q. Lou. Summarization graph indexing: Beyond frequent structure-based approach. In *DASFAA*, pages 141–155, 2008.

DOI: 10.5824/1309-1581.2018.3.002.x