Comparison Issues in Large Graphs: State of the Art and Future Directions

Hamida Seba* Sofiane Lagraa** Elsen Ronando*
*Université de Lyon, CNRS, Université Lyon 1
LIRIS, UMR5205, F-69622 Lyon, France.
hamida.seba@univ-lyon1.fr
**Université Grenoble Alpes
CNRS, TIMA, LIG, F-38031 Grenoble, France.

Abstract

Graph comparison is fundamentally important for many applications such as the analysis of social networks and biological data and has been a significant research area in the pattern recognition and pattern analysis domains. Nowadays, the graphs are large, they may have billions of nodes and edges. Comparison issues in such huge graphs are a challenging research problem.

In this paper, we survey the research advances of comparison problems in large graphs. We review graph comparison and pattern matching approaches that focus on large graphs. We categorize the existing approaches into three classes: partition-based approaches, search space based approaches and summary based approaches. All the existing algorithms in these approaches are described in detail and analyzed according to multiple metrics such as time complexity, type of graphs or comparison concept. Finally, we identify directions for future research.

1 Introduction

Comparing objects is one of the most frequently encountered tasks in computing: information retrieval, pattern recognition, biology, computer vision, etc. A comparison problem occurs whenever an object or a piece of it needs to be mapped to another object or part of it. Graphs are an attractive representation and modeling tool since they allow simple, intuitive and flexible representations of complex and interacting objects. Consequently, object comparison leads generally to a problem of graph comparison. Although significant progress has been made in graph comparison and related areas such as graph/subgraph isomorphism, pattern matching, etc., the recent explosion of the size of data generated and manipulated daily by applications and human activities has given rise to the big graph data challenge. In fact, real-world graphs are large and even huge,
i.e., thousands, millions and even billions nodes and edges. Social networks, web graphs and protein interaction graphs are some examples. For these graphs, existing solutions for graph analysis, mining, visualization, etc., do not scale at all. These algorithms must be revisited or even re-invented.

Traditional graph comparison approaches are generally classified into two categories: exact approaches and inexact approaches. Exact approaches, such as graph isomorphism, sub-graph isomorphism and the maximum common subgraph, aim to find out if an exact mapping between the vertices and the edges of the compared graphs or subgraphs is possible [10,18,48,73].

Inexact graph comparison aims generally to compute a distance between the compared graphs. This distance measures how much these graphs are similar and helps to deal with the errors and the noise that is inevitably introduced during the process needed to model objects by graphs. Inexact graph comparison is also useful for search/rank based applications where a distance between the compared objects is needed. In some applications, graph similarity measures are intended to compute relatively suboptimal distances [18] that are compensated by a large reduction of the computational complexity of the comparison process. Several graph similarity measures have been proposed in the literature and several approaches have been used including genetic algorithms [41,69], neural networks [51], the theory of probability [17,54], clustering techniques [12,66], spectral methods [65,74], decision trees [49,50], etc. We refer the reader to [9,10,18,27,76] for more exhaustive surveys. In order to cope with large graphs, new techniques, concepts and approaches have been proposed recently for performing graph comparison. Thus, in this paper we focus mainly on the solutions designed for large graphs.

The aim of this paper is to provide a survey of recent and current development of graph comparison and pattern matching approaches on large graphs. We describe and analyze in detail the existing approaches and we categorize them into different classes. We also highlight the advantages, disadvantages and the differences between the approaches and identify direction for future research.

The rest of the paper is organized as follows: Section 2 presents the problem definition and preliminaries. Section 3 presents the different approaches that we have categorized, analyzed and described in detail in order to compare them and to show their advantages and disadvantages. A summary of these approaches is presented and some important problems of graph comparison and pattern matching deserving further research are proposed in Section 4. Section 5 concludes the paper.

2 Problem Definition and Basics

In this section we present some basic definitions related to graphs and their comparison problems. We rely mainly on the terminology used in [3,23]. So, all the definitions below are adapted from [3,23].

**Definition 1** A graph $G$ is a 4-tuple $G = (V, E, f_V, f_E)$, where $V$ is a set of
nodes (also called vertices), $E \subseteq V \times V$ is a set of edges connecting the nodes, $f_V : V \rightarrow \Sigma_V$ and $f_E : E \rightarrow \Sigma_E$ are functions labeling the nodes and the edges respectively where $\Sigma_V$ and $\Sigma_E$ are the the sets of labels that can appear on the nodes and edges, respectively.

When omitting $f_E$ in the definition of $G$, we mean that $\Sigma_E$ is an empty set and the graph is not edge labeled. So, when there is no ambiguity, the notation $G = (V, E)$ defines vertex labeled graph. We will also use the terms vertex and node interchangeably in all this document.

The edges of a graph may have a direction associated with them. In this case, the graph is directed.

Generally, the number of vertices of a graph is called the order of the graph and the number of its edges is called the size of the graph.

A graph that is contained in another graph is called a subgraph and is defined as follows:

**Definition 2** A graph $G_1 = (V_1, E_1, f_{V_1}, f_{E_1})$ is a subgraph of a graph $G_2 = (V_2, E_2, f_{V_2}, f_{E_2})$, denoted $G_1 \subseteq G_2$, if $V_1 \subseteq V_2$, $E_1 \subseteq E_2 \cap (V_1 \times V_1)$, $f_{V_1}(x) = f_{V_2}(x) \forall x \in V_1$, and $f_{E_1}((x, y)) = f_{E_2}((x, y)) \forall (x, y) \in E_1$.

The distance between two nodes $u$ and $v$ in a graph $G$, denoted by $\text{dist}(u, v)$, is the length of the shortest undirected path from $u$ to $v$ in $G$. The diameter of a connected graph $G$, denoted by $d_G$, is the longest shortest distance of all pairs of nodes in $G$, i.e., $d_G = \max(\text{dist}(u, v))$ for all nodes $u, v$ in $G$. The eccentricity of a vertex in a graph is its maximum distance from any other vertex in the graph. The vertices of the graph with the minimum eccentricity are the centers of the graph, and the value of their eccentricity is the radius of the graph. The maximum value of eccentricity equals to the diameter of the graph.

Several applications that use graphs as a modeling tool such as pattern recognition, information retrieval, mining, etc., need to compare graphs. Graph comparison, also called graph matching, has been subject of several studies and surveys such as [18], [28], [1] and [76]. Graph comparison approaches are generally classified into two categories: exact approaches and inexact or fault-tolerant approaches. Exact approaches refer to the methods used to find out if two graphs are the same [18, 45, 73]. This means that we look for graph isomorphism.

Fault-tolerant graph comparison aims generally to compute a distance between the compared graphs. This distance measures how much these graphs are similar and is motivated mainly by three situations:

- the process of modeling objects by graphs may be subject to noise and distortions. This means that a modeling process executed twice on the same object may return two slightly different graphs. The different stages of image encoding is perhaps the most illustrative example of such noise that graph comparison must deal with [10].
• search/rank based applications such as database query processing or web search based applications need to compute a distance between the compared objects in order to rank the top-k results [16,72].

• In some applications, graph similarity measures are intended to compute relatively suboptimal distances [18] that are compensated by a large reduction of the computational complexity of the comparison process.

In both approaches and depending on the application, we need either to compare two whole graphs or a query graph with a large graph. According to this, graph comparison methods can be classified into two categories: graph similarity measures and graph pattern matching methods.

2.1 Graph similarity/dissimilarity measures

The aim of similarity/dissimilarity measures is to quantify the degree of resemblance between two graphs. The strongest similarity degree is graph "equality", called graph isomorphism and defined as follows:

**Definition 3** A graph $G_1 = (V_1, E_1, f_{V_1}, f_{E_1})$ and a graph $G_2 = (V_2, E_2, f_{V_2}, f_{E_2})$ are said to be isomorphic, denoted $G_1 \cong G_2$, if there exists a bijective function $h : V_1 \rightarrow V_2$ such that the following conditions are met:

1. $\forall x \in V_1 : f_{V_1}(x) = f_{V_2}(h(x))$
2. $\forall (x, y) \in E_1 : (h(x), h(y)) \in E_2$ and $f_{E_1}((x, y)) = f_{E_2}((h(x), h(y)))$
3. $\forall (h(x), h(y)) \in E_2 : (x, y) \in E_1$ and $f_{E_1}((h(x), h(y))) = f_{E_2}((x, y))$

Several relaxed approaches, i.e., "fault-tolerant graph comparison", are also proposed. They are useful for search/rank based applications where a distance between the compared objects is needed. In some applications, graph similarity measures are intended to compute relatively suboptimal distances [18] that are compensated by a large reduction of the computational complexity of the comparison process.

Several graph similarity measures have been proposed in the literature and several approaches have been using including genetic algorithms [41,69], neural networks [51], the theory of probability [17,54], clustering techniques [12,66], spectral methods [65,74], decision trees [49,50], etc. We refer the reader to [9,10,18,27,76] for more exhaustive surveys. Some of the existing approaches try to extend to graphs some of the properties defined in metric spaces.

**Definition 4** A metric space is an ordered pair $(M,d)$ where $M$ is a set and $d$ is a metric on $M$, i.e., a function

- $d(x,y) \geq 0$ (non-negativity),
- $d(x,y) = 0$ iff $x = y$ (uniqueness),
- $d(x,y) = d(y,x)$ (symmetry) and
• $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

Perhaps, the most referenced metric is edit distance which defines the similarity of graphs by the minimum costing sequence of edit operations that convert one graph into the other [8, 67]. An edit operation is either an insertion, a suppression or a re-labeling of a vertex or an edge in the graph. A cost function associates a cost to each edit operation. Figure 1 shows an example of edit operations that are necessary to get the graph $G_2$ from $G_1$ with the suppression of two edges and a vertex and the relabeling of two vertices.

![Figure 1: Example of edit operations](image)

Graph edit distance is a flexible graph similarity measure which is applicable to various kinds of graphs [2, 8, 55, 64, 67]. It also defines a common theoretical framework that allows comparing different approaches of graph comparison. In fact, Bunke showed in [6] that under a particular cost function, graph edit distance computation is equivalent to the maximum common subgraph problem. In [7], the same author shows that the graph isomorphism and subgraph isomorphism problems can be reduced to graph edit distance. However, computing graph edit distance suffers from two main drawbacks:

1. A high computational complexity. The problem of computing graph edit distance is NP-hard in general [81]. The most known method for computing the exact value of graph edit distance is based on $A^*$ [35], which is a best first search algorithm where the search space is organized as a tree. The root of the tree is the starting point of the algorithm. The internal vertices correspond to partial solutions and leaves represent complete solutions.

2. The difficulty related to defining cost functions [58].

The first drawback motivated several approximating solutions to compute graph edit distance. A comprehensive survey on graph edit distance and the approaches proposed to compute it can be found in [30]. To overcome the second drawback and avoid the definition of edit costs, similarity measures that do not use edit operations are also proposed. In [11], the authors propose a graph distance measure that is based on the maximal common subgraph of two graphs and prove that it is a metric, i.e., the measure satisfies the four properties of a usual metric namely: non-negativity, uniqueness, symmetry and triangle inequality. However, computing the maximal common subgraph of two graphs has
a high computational complexity [11]. For this reason, Raymond et al. [61] propose a modified version of the measure defined in [11] where an initial screening process determines whether it is possible for the measure of similarity between the two graphs to exceed a minimum threshold for which it is acceptable to compute the maximum common subgraph. This screening process is based on computing graph invariants. Graph invariants have been efficiently used to solve the graph comparison problem in general and the graph isomorphism problem in particular. They are used for example in Nauty [48] which is one of the most efficient algorithm for graph and subgraph isomorphism testing. A vertex invariant, for example, is a number \( i(v) \) assigned to a vertex \( v \) such that if there is an isomorphism that maps \( v \) to \( v' \) then \( i(v) = i(v') \). Examples of invariants are the degree of a vertex, the number of cliques of size \( k \) that contain the vertex, the number of vertices at a given distance from the vertex, etc. Graph invariants are also the basis of graph probing [44] where a distance between two graphs is defined as the norm of their probes. Each graph probe is a vector of graph invariants.

In [77], the distance metric based on the maximum common subgraph defined in [11] is extended by a proposal to define the problem size with the union of the two compared graphs rather than the larger of the two graphs used in [11].

In [80], the authors show that we can evaluate graph distance with a high degree of precision by considering complex graph sub-structures in the distance. In fact, in some applications such as analysis of protein interaction graphs, some sub-structures of these graphs represent certain functional modules of cells or organisms. Hence, comparing these graphs in terms of substructure information is biologically meaningful [80]. The authors defined a new metric based on the concept of Structure Abundance Vector. Each element of a Structure Abundance Vector of a graph \( G \) contains the size of an occurrence of a predefined sub-structure in \( G \). The Structure Abundance Vector is a generalization of the concept of graph invariants.

More recently, kernel based similarity measures are also proposed [5, 10, 32, 36, 56, 57]. The main idea is also to define similarity of graphs based on the similarity of substructures of these graphs.

### 2.2 Subgraph/Pattern matching

Given two graphs \( Q \) and \( G \), the graph pattern matching problem is to find all subgraphs of \( G \) that match \( Q \). In other words, find all the embeddings of \( Q \) in \( G \). Generally, \( Q \) is called the query graph or simply pattern and \( G \) is large compared to \( Q \). The exact version of graph pattern matching is called Subgraph isomorphism and is defined as follows:

**Definition 5** A graph \( Q = (V_Q, E_Q, f_{V_Q}, f_{E_Q}) \) is subgraph isomorphic to a graph \( G = (V_G, E_G, f_{V_G}, f_{E_G}) \) if there exists a subgraph \( G' \) of \( G \) such that \( Q \) and \( G' \) are isomorphic.
Subgraph isomorphism is an NP-complete problem \[31\]. The most known methods to enumerate the subgraphs of \( G \) that are isomorphic to a query \( Q \) are based on exploring search spaces. With these approaches, the number of possible matchings to be checked increases combinatorially with the number of nodes in the graphs. Even with the help of pruning methods that reduces the size of the search space \[19, 73\], these methods for subgraph isomorphism checking remain impractical for large graphs such as social networks. Furthermore, these graphs are directed, i.e., \((u, v)\) and \((v, u)\) denote different edges, and edge labeled. Moreover, the considered graph patterns are not simple graphs. A pattern in this kind of applications is a "regular expression"-like graph where a node is labeled by a search conditions which specifies a set of possible values for the node and the edge. In \[15\], an edge in query graph, is a directed edge and does not correspond to a direct edge between two nodes but to some reachability condition that means that the endpoint of the edge is reachable from the source node of the edge. This idea was extended in \[85\] by the introduction of a bound \( \delta \) such that if there is an edge between two nodes in the query, these nodes are mapped into the data graph to two nodes reachable within \( \delta \) edges, i.e., the shortest path between the two nodes is at most \( \delta \). More recently, \[\] introduces "regular expression"-like graph patterns that combine the concept of bounded edges of \[85\] with the power of regular expressions for defining the possible value taken by the labels of the nodes.

Consequently, relaxed approaches that achieve a better time complexity and that are more adapted to these pattern-based applications are proposed. In this context, Graph simulation \[37, 52\] receives an increasing interest specially for social network analysis. Graph simulation is defined as follows:

**Definition 6** A pattern \( Q = (V_Q, E_Q, f_{V_Q}, f_{E_Q}) \) matches a directed graph \( G = (V_G, E_G, f_{V_G}, f_{E_G}) \) via simulation, denoted by \( Q \preceq G \), if there exists a binary relation \( S \subseteq V_Q \times V_G \) such that:

1. for each \( u \in V_Q \), there exists \( v \in V_G \) such that \((u, v) \in S\);
2. for each \((u, v) \in S\), we have
   a. \( f_{V_Q}(u) = f_{V_G}(v) \);
   b. for each edge \((u, u') \in E_Q\) there is an edge \((v, v') \in E_G\) such that \((u', v') \in S\).

The graph that corresponds to simulation \( S \) is called the match graph and is defined as follows:

**Definition 7** Let \( Q = (V_Q, E_Q, f_{V_Q}, f_{E_Q}) \) be a query graph that matches a data graph \( G = (V_G, E_G, f_{V_G}, f_{E_G}) \) via simulation \( S \subseteq V_Q \times V_G \). The match graph that corresponds to \( S \) is a subgraph \( G_S \) of \( G \) such that \( G_S = (V_S, E_S) \), in which (1) a node \( v \in V_S \) iff it is in \( S \), and (2) an edge \((v, v') \in E_S\) iff there exists an edge \((u, u') \in E_Q\) with \((u, v) \in S\) and \((u', v') \in S\).
Contrarily to isomorphism, when two graphs $G_1$ and $G_2$ match by simulation, a node of one graph may be mapped to several nodes in the second graph. In Figure 2, the query graph is isomorphic to subgraph $G_1$ but it matches by simulation subgraphs $G_1$ and $G_2$. We note also that $G_2$ is not connected.

![Figure 2: Subgraph isomorphism Vs graph simulation.](image-url)

Note that a quadratic time algorithm for graph simulation is proposed in [37].

3 Approaches

In this section, we review graph comparison and pattern matching methods that focus on large graphs. Existing approaches can be categorized into three classes: partition based approaches, search space based approaches and summary based approaches. Figure 3 summarizes the approaches that will be reviewed in the rest of this section.

3.1 Partition-based Approaches

The basic idea of these approaches is to decompose graphs into sets of subgraphs and to compute the similarity between the initial graphs in function of a comparison between the obtained subgraphs. Partition-based approaches have two advantages:

1. They have a polynomial time complexity and thus may be suitable for large graph comparison.

2. They may highlight the existence of particular or meaningful structures within the compared graphs. These structures may enhance the accuracy of the comparison.

The first partition-based approach dates back to the 80s with the work of Eshera and Fu [21, 22]. The authors compute the edit distance between two attributed and directed graphs $G_1$ and $G_2$ in polynomial time ($O(n^2 \times m^2)(n + m)$) in the worst case, where $n$ is the order of the graph and $m$ is its size).
In this approach, the edit distance between \( G_1 \) and \( G_2 \) is mapped to the edit distance between their Basic Sub-Graphs called Basic Attributed Relational Graphs (BARGs) defined as follows:

**Definition 8** ([21], [22]) A Basic attributed relational graph (BARG or Basic graph) is a graph on the form of one level tree, i.e., it consists of a root node, the branches emanating from it, and the nodes on which these branches terminate.

In other words, a BARG is a star structure composed of a root vertex, its outcoming edges and the leaves associated to these edges. The mapping between two sets of BARGs is achieved via the exploration of a state space organized as a directed acyclic labeled lattice. Each state of the lattice is labeled with the set of matched BARGs and denotes the reconstruction of a subgraph from the query graph and a subgraph from the target graph as well as the matching of their respective BARGs. An edge between two states is labeled by the cost of the transition between two states. The final distance between the two graphs corresponds to the shortest costed path in the lattice. It is determined by dynamic programming.

In [68], the authors consider pair of vertices and their connecting edges, called Relational Descriptions (RD). They define a distance between two graphs based on the number of isomorphic RDs and prove that it is a metric. Given two graphs \( G_1 \) and \( G_2 \), the distance is defined by the number of RDs of \( G_1 \) that are not mapped to subgraphs of \( G_2 \) and the number of RDs of \( G_2 \) that are not mapped to subgraphs of \( G_1 \).
In [62, 63] the authors propose a modification of the approach of Eshera and Fu [21, 22] that considers undirected graphs and avoids the state exploration part of the distance computation. In this solution, an optimal match between the sets of star structures, called local structures, is obtained using the Hungarian algorithm [42, 53]. Given a source graph $G_1$ and a target graph $G_2$, the nodes of $G_1$ are mapped to the nodes of $G_2$ using the Hungarian algorithm by defining a cost matrix that records for each vertex from $G_1$ the edit operations that are needed to transform it to each vertex of $G_2$.

A similar approach is presented in [81]. In this case, the graphs are also undirected. They are decomposed into multisets of stars as in [21, 22]. In this approach, a star structure is defined around each vertex as in [62, 63] as follows:

**Definition 9** [81] A star structure $s$ is an attributed, a single-level, rooted tree which can be represented by a 3-tuple $s = (r, \mathcal{L}, \ell)$, where $r$ is the root vertex, $\mathcal{L}$ is the set of leaves and $\ell$ is a labeling function. Edges exist between $r$ and any vertex in $\mathcal{L}$ and no edge exists among vertices in $\mathcal{L}$.

Figure 5 shows an example of a graph and its star decomposition.

The edit operation between two stars is defined as follows:

**Definition 10** [81] Given two star structures $s_1$ and $s_2$, the edit distance between $s_1$ and $s_2$ is:

$$\lambda(s_1, s_2) = T(r_1, r_2) + d(\mathcal{L}_1, \mathcal{L}_2)$$

where

$$T(r_1, r_2) = \begin{cases} 
0 & \text{if } \ell(r_1) = \ell(r_2), \\
1 & \text{otherwise}.
\end{cases}$$

$$d(\mathcal{L}_1, \mathcal{L}_2) = ||\mathcal{L}_1| - |\mathcal{L}_2|| + \mathcal{M}(\mathcal{L}_1, \mathcal{L}_2)$$

$$\mathcal{M}(\mathcal{L}_1, \mathcal{L}_2) = \max\{|\Psi_{\mathcal{L}_1}|, |\Psi_{\mathcal{L}_2}|\} - |\Psi_{\mathcal{L}_1} \cap \Psi_{\mathcal{L}_2}|$$

$\Psi_{\mathcal{L}}$ is the multiset of vertex labels in $\mathcal{L}$.

The authors define the distance between two multisets of star structures. Subsequently, they define the mapping distance between two graphs based on the edit distance between their star representations using the Hungarian algorithm [42, 53].
Figure 5: A graph and its star decomposition according to [81]. The arrows indicate the root of the stars.

In [79], the authors proposed an index based on the star subdivision provided in [81]. This index is made up of two parts: an index for all distinct star structures from the given database, and an inverted list below each star structure. The star structures are sorted in alphabetical order. Each entry in the inverted lists contains the graph identity and the frequency of the corresponding star structure. All lists are sorted in increasing order of the graph size [79]. However, enumerating all the different stars in a large graph database may produce a huge index which is not a practical solution.

In [60], the authors also propose a polynomial time graph matching distance based on subgraph matching using the Hungarian algorithm [42, 53]. The subgraphs are also stars but consider edge labels which is not the case with [81] and [62, 63]. Each star structure is embedded within a vector of probes. Each probe gives the number of times that a given label appears in the star. An example is described in Figure 6.

![Graph with star decomposition and probe vectors](image)

Set of vertex and edge labels:
- \{a, b, c, d, e, f, m, n, p, r\}

Probe vector of the star s2:
- [1, 0, 1, 1, 1, 0, 1, 2, 0]

Figure 6: A graph and its decomposition into probe vectors.

Note that the decomposition into stars in the approaches of [62, 81] and [60] induce more overlappings than the decomposition into BARGs of [21, 22] as the number of BARGs is smaller than the number of stars in a graph.

Another resembling distance is also defined in [38] where a different representation of the star structure is used. In this similarity measure, the star structure is called node signature and is represented by a vector containing the
label of the root vertex, its degree, and the set of labels of its incident edges. So, in this representation, the labels of the leaves of the star are not considered in the subgraph as illustrated in Figure 7. A distance between two node signatures is also defined and the distance between two graphs is then defined as an assignment problem in the matrix containing the distances between nodes signatures of the two compared graphs.

\[ \text{Signature of node } a \quad \begin{array}{c} a \ 2 \ m \ n \end{array} \quad \text{Signature of node } b \quad \begin{array}{c} b \ 1 \ m \end{array} \quad \text{Signature of node } c \quad \begin{array}{c} c \ 3 \ n \ p \ p \end{array} \quad \text{Signature of node } d \quad \begin{array}{c} d \ 2 \ p \ r \end{array} \quad \text{Signature of node } f \quad \begin{array}{c} f \ 2 \ p \ r \end{array} \]

**Figure 7:** A graph and its decomposition into node signatures.

In [78], the authors propose to decompose the compared graphs into \( k \)-Adjacent Tree (\( k \text{AT} \)) patterns (like Q-Gram decomposition of strings [71]), then use the number of their common \( k \text{AT} \) patterns for edit distance estimation. The adjacent tree of a vertex \( v \) (\( AT(v) \)) in a graph \( G \) is a breadth-first search tree rooted at vertex \( v \), the children of each node of \( AT(v) \) are sorted by their labels in the graph. The \( k \)-adjacent tree of a vertex \( v \) (\( k \text{AT}(v) \)) in a graph \( G \) is the top \( k \)-level subtree of \( AT(v) \) [78]. This means that the star structure of [21, 22] and the related methods is a 1\text{AT}.

\[ \text{Signature of node } a \quad \text{Signature of node } b \quad \text{Signature of node } c \quad \text{Signature of node } d \quad \text{Signature of node } f \]

**Figure 8:** A graph and its 2\text{AT}s decomposition.

The set of all \( k \text{AT}s \) of a graph \( G \) is denoted \( k \text{AT}s(G) \). An example is illustrated in Figure 8. The number of common \( k \text{AT}s \), i.e., \( |k \text{AT}s(G_1) \cap k \text{AT}s(G_2)| \), of two graphs is called the matching number of the two graphs and is used to estimate their edit distance using the following inequality:

\[ |k \text{AT}s(G_1) \cap k \text{AT}s(G_2)| \geq |V(G_1)| - GED(G_1, G_2) - 2(\Delta(G_1) - 1)k - 1. \]

where \( GED(G_1, G_2) \) is the edit distance between \( G_1 \) and \( G_2 \) and \( \Delta(G_1) \) and \( \Delta(G_2) \) are the maximum degrees of \( G_1 \) and \( G_2 \) respectively with \( \Delta(G_2) > 1 \) and \( \Delta(G_2) > 1 \). This estimation has proven to be sufficiently tight but only for sparse graphs [78].

To avoid the above cited drawback of tree-based \( q \)-grams, [83] proposes to use a decomposition into path-based \( q \)-grams. A path-based \( q \)-gram in a graph \( G \) is
a path of length \( q \) with no repeated vertex. The edit distance can be estimated with path based \( q \)-grams with the following inequality: If \( GED(G_1, G_2) \leq \tau \) then \( G_1 \) and \( G_2 \) share at least \( \max(|Q_{G_1}|-\tau \cdot D_{path}(G_1), |Q_{G_2}|-\tau \cdot D_{path}(G_2)) \) path based \( q \)-grams, where \( |Q_G| \) is the size of the multiset of path based \( q \)-grams in \( G \) and \( D_{path}(G) \) is the number of path based \( q \)-grams of \( Q_G \) affected by an edit operation that occurs on \( G \). \( D_{path}(G) \) can be computed by:

\[
D_{path}(G) = \max_{u \in V_G} |Q_u^G|
\]

where \( Q_u^G \) denotes the multiset of path \( q \)-grams that contain vertex \( u \). To find the pairs of graphs that are within an edit distance of \( \tau \), the authors propose to use either an inverted index that maps each path \( q \)-gram to a list of identifiers of graphs that contain this path \( q \)-gram or a prefix filter such as those used in string similarity measures [13].

In [84], the authors point out that path-based \( q \)-grams still induce many overlapping structures. If there are some high-degree vertices, the estimated edit distance of the path-based \( q \)-grams is not tight. They propose to use a new \( q \)-gram based structure, called branch structure, so that a single edit operation can affect two structures at most allowing a tighter lower bound for edit distance than existing \( q \)-grams structures. A branch structure \( b \) is a vertex \( v \) and the multiset of edge labels incident to \( v \). A branch is represented by \( b(v) = (l_v, ES) \), where \( l_v = L_V(v) \) is the label of vertex \( v \), and \( ES = \{L_E(e) \mid \text{edge } e \text{ is adjacent to } v \} \) is the multiset of edge labels adjacent to \( v \). An example is given in Figure 9. A branch structure is equivalent to the node signature introduced in [38]. Figure 9 shows an example of a graph and its branch structures.

![Figure 9: A graph and its branch structures.](image)

The authors define the edit distance between two branches as in [81] and derives the distance between two multisets of branches \( B(G_1) \) and \( B(G_2) \) as the minimum weighted match in the bipartite graph which vertices represent the branches of \( B(G_1) \) and \( B(G_2) \) and edges represent transformations between any two branches (from \( B(G_1) \) and \( B(G_2) \) respectively) weighted with their pairwise branch edit distance. For solving the assignment problem, the authors use the Hungarian algorithm [42]. The authors also prove that the obtained branch based distance is tighter that the star based distance of [81].

To simplify the processing of a query graph in a large graph database, in [84] the authors propose to use an \( R \)-tree based index where each leaf is the set of branches of a graph of the database. An internal node of the tree is the union of the branches of its children. The query graph is processed by traversing
the index starting from the root. For an intermediate node, the branch based distance is computed between the query graph and the set of branches of the internal node. If this distance is greater than a given threshold, the subtree rooted at this internal node can be safely pruned. However, computing the distance with all the branch set of an internal node is not compatible with large databases and may induce an important overhead.

In [82], the authors propose to use variable-size non-overlapping partitions. The proposed partitioning is based on the half-edge concept defined as an edge with only one end node and denoted by \((u, .)\). Based on this concept, the authors introduce the notion of half-edge graph, i.e., a graph that contains half-edges, and half-edge subgraph isomorphism defined as follows:

**Definition 11** [82] A graph \(Q = (V_Q, E_Q, f_{V_Q})\) is half-edge subgraph isomorphic to a graph \(G = (V_G, E_G, f_{V_G})\), denoted as \(Q \sqsubseteq G\), if there exists an injection \(h: V_Q \rightarrow V_G\) such that (1) \(\forall u \in V_Q, h(u) \in V_G\) and \(f_{V_Q}(u) = f_{V_G}(f(u))\); (2) \(\forall (u, v) \in E_Q, (f(u), f(v)) \in E_G\) and \(f_{V_Q}((u, v)) = f_{V_G}((f(u), f(v)))\); and (3) \(\forall (u, .) \in E_Q, (f(u), w) \in E_G\) and \(f_{V_Q}((u, .)) = f_{V_G}((f(u), w)), w \in V_G \setminus h(V_Q)\)

Based on this, [82] develops a partition-based similarity search framework that contains two phases: an indexing phase that can be performed offline and a query processing phase performed for each query. The indexing phase takes as input a graph database \(D\) and an edit distance threshold \(\tau\) and constructs an inverted index as follows:

- For each data graph \(G \in D\), it first divides \(G\) into \(\tau + 1\) partitions. Figure 10 gives an example of a graph partition into 2 half edge-subgraphs.

```
Figure 10: A half-edge subgraph decomposition.
```

- Then, for each partition, it inserts \(G\)’s identifier into the corresponding postings list of the partition.

The processing of a query \(q\), starts by probing the inverted index for candidate generation. For each partition \(p\) in the inverted list, it tests whether \(p\) is contained by the query. If so, the graphs in the postings list of \(p\) are filtered based on their size and their labels. If the filtering produces a result within \(\tau\), the graph is produced as a candidate for the query. Finally, candidates are further examined with a classic graph edit distance algorithm. The main problem of this approach is related to the partitioning algorithm. In fact, such partitioning
is not unique for a given graph. Furthermore, the index is not practical for large graphs.

In [70], the authors propose a graph decomposition into STwigs. An STwig is a two level tree structure, $q = (r, L)$, where $r$ is the label of the root node and $L$ is the set of labels of its child nodes. Contrarily to the star structure used in [81] and [60], STwigs do not overlap (regarding edges), they are edge disjoint stars as illustrated in Figure 11.

![Figure 11: A graph and two of its possible decomposition into STwigs.](image)

Clearly, such decomposition is not unique and different decompositions of the same query incur different query processing cost. So, [70] proposes a query decomposition that minimizes the number of obtained STwigs. The authors proved that the minimum STwig cover problem is polynomial equivalent to the minimum vertex cover problem. Consequently, they construct an STwig cover from a vertex cover in polynomial steps using an existing 2-approximate algorithm [20] for the vertex cover problem. Given a query graph $q$, [70] first decomposes $q$ into a set of STwigs, then it uses exploration to find matches to each STwig. Exploration at this step avoids indexing on STwigs which is not feasible for billion node graphs. Finally, the approach joins the results to find the final solution. The authors also modified the 2-approximate algorithm for the STwig cover to incur an STwig order that optimizes the number of joins. In fact, it seems that given a set of STwigs produced by the decomposition step, an optimized order is the one that ensures that the root node of each STwig is a leaf node of at least one of the already processed STwigs.

In [39], the authors propose a tree $q$-gram like decomposition embedded in a vector representation. In this approach, each partition rooted at node $u$ encompasses the $h$-hop neighbors of $u$, i.e., the set of nodes $v$ whose distance from $u$ is less than or equal to $h$. The partition is encoded within a multidimensional vector, called neighborhood vector and denoted $R(u)$ for node $u$ with $R(u) = \{ (l, A(u,l)) \}$, where $l$ is a label presents in the neighborhood of $u$ and $A(u,l)$ represents the strength of $l$ in the neighborhood of node $u$ and is obtained
by:

\[ A(u, l) = \sum_{i=1}^{h} \alpha^i \sum_{d(u,v)=i} I(l \in L(v)) \]  

(1)

In this formula, \( L(v) \) is the label set of node \( v \), \( I(l \in L(v)) \) is an indicator function which takes the value 1 when \( l \) is in the label set of \( v \) and 0 otherwise. \( d(u, v) \) is the distance between \( u \) and \( v \). \( \alpha \) is a constant called the propagation factor that takes value between 0 and 1. Figure 12 gives an example of a graph and the neighborhood vectors associated to each of its vertices with \( h = 2 \) and \( \alpha = 0.5 \). The similarity between two neighborhood vectors \( R(u) \) and \( R(v) \) of two nodes \( u \) and \( v \) respectively is computed by the following cost function:

\[ C(u, v) = \sum_{l \in R(u)} M(A_Q(v, l), A_G(u, l)) \]  

(2)

\[ M(x, y) = \begin{cases} 
  x - y & \text{if } x > y, \\
  0 & \text{otherwise.} 
\end{cases} \]  

(3)

Figure 12: A graph and its neighborhood vectors (\( h = 2 \) and \( \alpha = 0.5 \)). Vertices are numbered to distinguish them.

Using neighborhood vectors, the authors propose an algorithm that finds all the embeddings of a query graph \( Q \) in a target graph \( G \) as follows:

1. compute the neighborhood vectors \( R_G(u) \) and \( R_Q(v) \) for all nodes \( u \in V(G) \), \( v \in V(Q) \),

2. for each node pair \( u \in V(G), v \in V(Q) \) s.t. \( L(v) \subseteq L(u) \), calculate the node matching cost, \( cost(u, v) \) as the difference of their neighborhood vectors, \( cost(u, v) = \sum_{l \in R(u)} M(A_Q(v, l), A_G(u, l)) \). Obtaining for each \( v \in V(Q) \) a list \( List(v) \) of possible matching nodes such that \( List(v) = \{ u \in V(G) | cost(u, v) \leq \varepsilon \} \) where \( \varepsilon \) is a similarity threshold. To speed up the computation of \( List(v) \) for all \( v \in V_Q \), two kinds of indexes can be constructed offline for \( G \):

- a label-based index with a hash table corresponding to each label of \( G \). This index is efficient if the labels are node selective.
- structure-based index which is built on the neighborhood vectors.
3. use dynamic programming to find the embeddings of $Q$ in $V$ from the final list of matched nodes for each node $v \in V_Q$.

In [40], the authors extend the approach proposed in [39] with an inference algorithm that iteratively boosts the score of more promising candidate nodes, considering both label and structural similarity. This approach, called NeMa is based on the neighborhood vector introduced in [39] with the slight difference that the neighborhood vector in NeMa gives more importance to the distance than to the labels. The authors motivate this by two remarks from real applications: (a) if two nodes are close in a query graph, the corresponding nodes in the result graph must also be close. However, (b) there may be some differences in labels of the matched nodes due to noises and heterogeneity in data.

The neighborhood of a node $u$ in a graph $G$ is given by

$$R_G(u) = \{ u', P_G(u, u') \}$$

where $u'$ is a node within $h$-hops of $u$, and $P_G(u, u')$ denotes the proximity of $u'$ from $u$ in $G$.

$$P_G(u, u') = \begin{cases} \alpha d(u, u') & \text{if } d(u, u') \leq h, \\ 0 & \text{otherwise.} \end{cases}$$

(4)

Where $d(u, u')$ is the distance between $u$ and $u'$. The propagation factor $\alpha$ is a parameter between 0 and 1; and $h > 0$ is the hop number delimiting the neighborhood. Given a matching function $\phi$, the matching cost of the neighborhood vectors of two nodes $v$ and $u = \phi(v)$ is given by:

$$N_\phi(u, v) = \frac{\sum_{v' \in N(v)} M(P_Q(v, v'), P_G(u, \phi(v')))}{\sum_{v' \in N(v)} P_Q(v, v')}$$

(5)

$M$ is defined by Equation 3.

The global cost $C(\phi)$ of the matching function $\phi$ between the query graph $Q$ and the target graph $G$ is given by:

$$C(\phi) = \sum_{v \in V_Q} F_\phi(v, \phi(v))$$

(6)

where, $F_\phi(v, \phi(v))$ is the individual node matching cost between $v$ and $u$ defined as a linear combination of the label difference function and the neighborhood matching cost function via a parameter $0 < \lambda < 1$, whose optimal value is set empirically.

$$F_\phi(v, \phi(v)) = \lambda \Delta_L(L_Q(v), L_G(\phi(v))) + (1 - \lambda)N_\phi(v, \phi(v))$$

(7)

The label difference function $\Delta_L$ between two node labels is defined by the Jaccard similarity.

To find a matching function $\phi$ that minimizes $C(\phi)$, [40] uses a heuristic based on the max-sum inference problem in graphical models [59].
3.2 State Space Exploring Approaches

In these classes of methods, we find mainly graph pattern matching approaches where a number of candidates vertices, subgraphs or regions are explored in a large data graph to find the different embeddings of some query graph or the subgraphs that match a given graph pattern.

In [34], the authors propose a solution, called TURBOISO, to robustly compute subgraph isomorphism with two mechanisms: a tree rewriting of the query graph and candidate region exploration. A candidate region for a query graph \( Q \) is a subgraph of the data graph \( G \) which may contain embeddings of the query graph. So, performing subgraph isomorphism search on all candidate regions will ensure that all embeddings can be obtained. However, minimizing the number of candidate regions and the size of each region is obviously important for faster matching. In order to minimize the size of each candidate region, the authors propose to:

1. rewrite the query \( Q \) into an equivalent NEC (Neighborhood Equivalence Class) tree \( Q' \). In \( Q' \) each set of vertices that have the same label and the same set of adjacent query vertices are merged into one NEC vertex. So, a NEC vertex is a compressed form of a set of vertices. Consequently, using \( Q' \) instead of \( Q \), will accelerate the candidate region exploration process, since the number of vertices is smaller.

2. construct candidate regions for the query \( Q \) in the data graph \( G \) by constructing for each region a BFS search tree \( T_G \) from the root node \( u'_s \) of the NEC tree \( Q' \) so that each leaf is on the shortest path from \( u'_s \). Then, for the start vertex \( v_s \) of each target candidate region, identify candidate data vertices for each query vertex by simply performing depth-first search using \( T_G \) and starting from \( v_s \).

Minimizing the number of regions comes through a careful choice of the root of the NEC tree. For this, TURBOISO ranks every query vertex \( u \) by \( \text{Rank}(u) = \frac{\text{freq}(G,L(u))}{\text{deg}(u)} \), where \( \text{freq}(G,l) \) is the number of data vertices in \( G \) that have label \( l \), and \( \text{deg}(u) \) means the degree of \( u \). This ranking function favours lower frequencies and higher degrees which will minimize the number of regions.

When exploring candidate regions, TURBOISO also minimizes the number of enumerated partial solutions by ordering the NEC vertices by increasing sizes. Thus, paths involving fewer vertices are explored first, the space is pruned rapidly if no isomorphism is possible.

[23] introduces bounded simulation, an extension of graph simulation intended to deal with graph queries expressed with graph patterns. In this case, all graphs are directed and a pattern graph is defined as follows:

**Definition 12** [23] A pattern graph is defined as \( P = (V_P, E_P, f_{V_P}, f_{E_P}) \), where

1. \( V_P \) and \( E_P \) are the set of nodes and the set of directed edges, respectively, as defined for data graphs;
2. \( f_{V_P} \) is a function defined on \( V_P \) such that for each node \( u \), \( f_{V_P}(u) \) is the predicate of \( u \), defined as a conjunction of atomic formulas of the form \( A \ op \ a \); here \( A \) denotes an attribute, \( a \) is a constant, and \( op \) is a comparison operator \(<, \leq, >, \geq, =, \neq\);

3. \( f_{E_P} \) is a function defined on \( E_P \) such that for each edge \( (u, u') \in E_P \), \( f_{E_P}((u, u')) \) is either a positive integer \( k \) or a symbol \(*\).

Intuitively, the predicate \( f_{V}(u) \) of a node \( u \) specifies a search condition and may induce several possible label values. The integer \( f_{E_P}((u, u')) \) means that the edge \((u, u')\) can be matched to a path of length at most \( f_{E_P}((u, u')) \). A simple graph query corresponds to a graph pattern where \( f_{V}(u) \) is simply the label of \( u \) and \( f_{E_P}((u, u')) = 1 \). In bounded simulation, the term "bounded" relates to the bound piggybacked by each edge in the pattern. This bound is the maximum length of a path in the data graph that matches the edge of the pattern. Bounded simulation is defined as follows:

**Definition 13** [23] A data graph \( G = (V, E, f_A) \) matches the pattern query \( Q = (V_Q, E_Q, f_{V_Q}, f_{E_Q}) \) via bounded simulation, denoted by \( Q \sqsubseteq G \), if there exists a binary relation \( S \subseteq V_Q \times V \) such that:

- for each \( u \in V_Q \), there exists \( v \in V \) such that \((u, v) \in S\);
- for each \((u, v) \in S\), (a) the attributes \( f_A(v) \) of \( v \) satisfies the predicate \( f_{V_Q}(u) \) of \( u \); and (b) for each edge \((u, u') \in E_{V_Q}\), there exists a non-empty path \( \rho = v/\ldots/v' \) in \( G \) such that \((u', v') \in S\), and \( \text{len}(\rho) \leq k \) if \( f_{V_Q}(u, u') \) is a constant \( k \).

In this paper, the authors also introduce the concept of **maximum match graph** to represent the union of all matches of a query in a data graph. This means that bounded simulation will search for a unique result graph that encompasses all the subgraphs that match the query pattern as illustrated in Figure 13.

![Figure 13: Maximum match graph for bounded simulation.](image)
Then, they propose an algorithm for *incremental matching* that avoids the cost related to re-computing the result graph when the graph data is modified. This ensures the scalability of the approach to large graphs.

In [45], the authors focused on reducing the number of matches returned by graph simulation and bounded simulation, the extension of graph simulation proposed in [23]. This is achieved by enforcing two conditions:

1. Duality which corrects the behavior of graph simulation concerning topology preservation of the query. In fact, as shown in Figure 2, graph simulation may return a disconnected subgraph for a connected graph query which augments the number of matches. To avoid this, [45] proposes dual simulation defined as follows:

**Definition 14** [45] A data graph \( G = (V, E, f_A) \) matches the pattern \( Q = (V_Q, E_Q, f_v, f_e) \) via dual simulation, denoted by \( Q \prec_D G \), if \( Q \prec G \) with a binary match relation \( S_D \subseteq V_Q \times V \), and for each pair \((u, v) \in S_D\) and each edge \((u_2, u) \in E_Q\), there exists an edge \((v_2, v)\) \(\in E\) with \((u_2, v_2) \in S_D\).

Thus, dual simulation requires that two related nodes have the same edges and by the way avoids to simulate a connected graph with a disconnected one. Accordingly, in the example of Figure 2 only subgraph \( G_1 \) is returned as the result graph match.

2. Locality which reduces the diameter of the returned subgraph of bounded simulation. In fact, bounded simulation returns a maximum match that encompasses all the matches of the query. This maximum match is unique but may be a too large graph. Locality is enforced by requiring matches to be within a ball of radius equal to the diameter of the query. A ball is defined as follows:

**Definition 15** For a node \( v \) in a graph \( G \) and a non-negative integer \( r \), the ball with center \( v \) and radius \( r \) is a subgraph of \( G \), denoted by \( G[v, r] \), such that (1) for all nodes \( v' \in G[v, r] \), the shortest distance \( \text{dist}(v, v') \prec r \), and (2) it has exactly the edges that appear in \( G \) over the same node set.

**Definition 16** [45] A data graph \( G = (V, E, f_A) \) matches the query pattern \( Q = (V_Q, E_Q, f_v, f_e) \) via strong simulation, denoted by \( Q \prec_S G \), if there exist a vertex \( v \in V \) and a connected subgraph \( G_s \) of \( G \) such that:

- \( Q \prec_D G_s \) with the maximum match relation \( S \);
- \( G_s \) is exactly the match graph of \( Q \) with \( S \), and
- \( G_s \) is contained in the ball \( G_D[v, d_Q] \) of center \( v \) and radius \( d_Q \) the diameter of \( Q \).
Figure 14: Bounded simulation Vs Dual and Strong simulation.

Figure 14 illustrates an example adapted from [45]. In this example, with simulation and bounded simulation the query graph matches all the data graph. However, dual simulation returns $G_3$ while strong simulation returns $G_2$. Note that the diameter of the query graph is 2. Subgraph isomorphism returns $G_1$.

The authors show that strong simulation has the same complexity than simulation and bounded simulation while preserving graph topology. They propose a cubic-time algorithm that returns the set of subgraphs of a data graph that matches by strong simulation a graph query. The algorithm inspects the balls of radius equal to the query diameter and centred at each node of the data graph.

In [25], the authors propose strict simulation to further improve graph simulation and adapt its computation within a vertex-centric Bulk Synchronous Parallel (BSP) programming model [75] used by several graph processing frameworks such as Pregel [47]. They introduce an extra step in the algorithm of strong simulation proposed in [23]. Strict simulation reduces the size, i.e., the number of nodes, of the ball inspected by strong simulation. For this, the idea is to first compute the match for dual simulation before inspecting the balls. So, the balls are computed on the result of dual simulation and are consequently much smaller than those computed by strong simulation. Formally, strict simulation is defined as follows:

**Definition 17** [25] A data graph $G = (V, E, f_A)$ matches the query pattern $Q = (V_Q, E_Q, f_v, f_e)$ via strict simulation, denoted by $Q \prec \sum_{\text{sim}} G$, if there exists a vertex $v \in V$ such that:

- $v \in V_D$ where $G_D(V_D, E_D, l_D)$ is the result match graph with respect to $Q \prec D_{\text{sim}} G$;

- $Q \prec D_{\text{sim}} G_D[v, d_Q]$ where $G_D[v, d_Q]$ is a ball extracted from $G_D$; and

- $v$ is a member of the maximum match graph.

In the example of Figure 14, strong simulation will first compute the match graph for dual simulation, i.e., subgraph $G_3$, and then begin inspecting the balls. [25] also proposes distributed algorithms to compute simulation, bounded simulation, strong simulation and strict simulation.
Similarly to strict simulation, [26] introduces tight simulation that improves strict simulation and approaches subgraph isomorphism. Tight simulation focuses on reducing the number of the balls inspected by strict simulation. To do so, the authors propose to select a single vertex \( u \) of the pattern \( Q \) and to use it as a candidate match to the center of a potential ball in the data graph. \( u \) is chosen to be the vertex of minimum eccentricity, i.e., it is a center of \( Q \), which has the highest ratio of degree to label frequency (in \( Q \)). This allows to reduce the radius of the balls and also their number. So, tight simulation is defined as follows:

**Definition 18** [26] A data graph \( G = (V, E, f_A) \) matches the query pattern \( Q = (V_Q, E_Q, f_v, f_e) \) via tight simulation, denoted by \( Q \sim_{\text{sim}}^T G \), if there are vertices \( u \in Q \) and \( u' \in G \) such that

1. \( u \) is a center of \( Q \) with highest defined selectivity;
2. \((u, u') \in R_D \) where \( R_D \) is dual relation set between \( Q \) and \( G \);
3. \( Q \sim_{\text{sim}}^D \tilde{G}_D[u', r_Q] \) where \( \tilde{G}_D[u', r_q] \) is a ball extracted from \( G_D(V_D, E_D, l_D) \) which is the result match graph with respect to \( Q \sim_{\text{sim}}^D G \), and \( r_Q \) is the radius of \( Q \), and
4. \( u' \) is a member of the resulting maximum match graph.

In the example of Figure 14, the node having label \( b \) is a center of the query graph and will be used to extract the balls in the result of dual simulation, i.e., the match graph \( G_3 \). The authors show that tight simulation has better results than strong simulation and strict simulation.

### 3.3 Summary-based approach

Graph summarizing/compression offers interesting perspectives for large graph storage and processing. A graph summarizing method that retains an “acceptable amount” of the graph properties may be used as a preprocessing step to several graph algorithms. The idea here is not to reduce the size of a huge graph just to minimize its storage requirement and to decompress the graph to process it. Rather, the aim is to obtain a compressed representation of the graph that can be used, instead of the original graph, by the processing algorithms, i.e., analysis, mining, comparison, querying, etc. In this vein, [14] proposes an algorithm that finds all frequent subgraphs in a database of large graphs where the database graphs are summarized. Summarizing is achieved by grouping the nodes that have the same label into supernodes as follows:

**Definition 19** (Summarized Graph) [14]. Given a labeled graph \( G \) such that its vertices \( V(G) \) are partitioned into groups, i.e., \( V(G) = V_1(G), V_2(G), \ldots, V_k(G) \), such that: (1) \( V_i(G) \cap V_j(G) = \phi, 1 \leq i \neq j \leq k \) (2) all vertices in \( V_i(G), 1 \leq i \leq k \), have the same labels.

We can summarize \( G \) into a compressed version \( \text{comp}(G) \) where:
(1) $\text{comp}(G)$ has exactly $k$ nodes $v_1, v_2, \ldots, v_k$ that correspond to each of the groups of $V(G)$ (i.e., $V_i(G) \mapsto v_i$). The label of $v_i$ is set to be the same as those vertices in $V_i(G)$, and

(2) an edge $(v_i, v_j)$ with label $l$ exists in $\text{comp}(G)$ if and only if there is an edge $(u, u')$ with label $l$ between some vertex $u \in V_i(G)$ and some other vertex $u' \in V_j(G)$.

The obtained summarized graphs may then be mined for frequent patterns using any existing algorithm. To ensure that all patterns are found, the authors do not systematically summarize all the graphs of the database, rather they proceed with several iterations each of which consists of two steps:

- Step 1: For each $G_i$ in a graph database $D$, randomly partition its vertex set $V(G_i)$.
- Step 2: Execute a pattern mining algorithm of the resulting summarized database.
- Step 3: Compute the support of each resulting pattern in the original database, i.e., the number of graphs that contain the pattern. Discard the pattern if its support is lower than a predefined threshold. The number of iteration is controlled by the probability of missing a frequent pattern.

In [24], the authors observe that users typically adopt a class $Q$ of queries when querying a data graphs $G$. They propose a graph compression preserving queries of $Q$. This means that each query in $Q$ returns the same result when applied to $G$ and when applied to the compression of $G$. They define the compression functions for two kind of graph queries: reachability queries and pattern queries. Roughly speaking, for reachability queries which aims to define if a node is reachable from another, the compression function groups the nodes that have the same ancestors and the same descendants. For pattern queries, the compression function is equivalent to the one given by Definition 19.

In [43], the authors propose a new solution for the comparison of large graphs. Their approach relies on a compact encoding of graphs called prime graphs. Prime graphs are smaller and simpler than the original ones but they retain the structure and properties of the encoded graphs. An example of a graph and its prime is given in Figure 15. In [43], the authors propose to approximate the similarity between two graphs by comparing the corresponding prime graphs. Their proposed approach involves the following steps:

- Building the prime graph of the compared graphs. Prime graphs are obtained by modular decomposition of the original graphs. Modular decomposition is one of the most known graph decompositions [33]. It was introduced by Gallai [29] to solve optimization problems. Modular decomposition generates a representation of a graph that highlights groups of vertices that have the same neighbors outside the group. These subsets of vertices are called modules. The prime graph correspond to the graph obtained by compressing all the modules recursively.
• Partitioning the compared prime graphs into stars of modules as in [81].

• Computing the distance between two prime graphs based on the distance of each pair of the stars of modules. Given a query prime graph $PG_1$ and a target prime graph $PG_2$, the nodes of $PG_1$ are mapped to the nodes of $PG_2$ using the Hungarian algorithm by defining a cost matrix that records for each star of modules from $PG_1$ the edit operations that are needed to transform it to each star of modules of $PG_2$.

• Solving the assignment problem by using the Hungarian algorithm [32] to obtain the minimum distance.

(a) A protein graph of 1818 nodes and 1833 edges from the RI database [4].
(b) The corresponding prime graph having 271 nodes and 321 edges.

Figure 15: Example of a graph and its prime graph.

4 Discussion

Tables 1, 2, 3 summarize all the presented approaches within the three categories: partition-based approaches, search-space exploring approaches and summary-based approaches, respectively.

The tables summarize these approaches according to the following facets:

• Graphs: the type of graphs on which the graph comparisons are performed: directed/undirected graph, labeled/unlabeled edges.

• Decomposition unit: the type of graph partitioning given by the name of the subgraph structure.

• Comparison concept: the type of similarity used for graph comparison.

• Application: describes the application area of the approach.
• Program: the type of program, it can be sequential or parallel.

• Size of the query: describes the range of the size of the graph query used for matching.

• Size of data graph: describes the range of the size of the data graph used for matching. The size here is given in terms of the number of nodes and edges in the graph. It can be thousand (k), million (M) or billion.

• Time complexity of the approach when computed.

Throughout this survey, we can see that various solutions are considered and there is not a generic algorithm for graph comparison or graph pattern matching that takes into consideration any type of graph (labeled/unlabeled and directed/undirected). Partition based approaches become increasingly used for graph comparison and pattern matching approaches. In fact, these approaches have a good time complexity and are easy to project toward parallel algorithms. The problem of matching in partition-based approaches is simplified by decomposing the graphs to be matched into smaller subgraphs. However, the best graph decomposition technique that should be adopted for computing distance remains an open problem for large graphs even if we note that the majority of partitioning approaches relay on a star decomposition. Besides, the approaches that use the Hungarian algorithm [42, 53] on a large cost matrix such as [81] suffer memory problems. Heuristics or other methods that compute the minimum cost while avoiding the construction of the cost matrix are appreciated. Also, a parallel version of the Hungarian algorithm that relies on a partitioning of the matrix storage and computation will scale these approaches to larger graphs.

Furthermore, using partition based approaches in subgraph search is generally associated with joins or indexing methods. Both of them are time consuming and complex tasks especially for large graphs. So, research must focus on methods to avoid them or develop them to deal with large graphs.

We can also note that several graph matching techniques have not been investigated in large scale graphs. Among these solutions we can cite clustering based methods and polynomial heuristics to the greatest common subgraph. Invariant-based graph comparison [43, 80] may also give good results.

To cope with large graphs, one among the solutions is graph compression without loss of information and performing the matching on the compressed graph. However, it does not exist enough summary-based approaches. Reducing and compressing a graph for graph matching is a very interesting approach. There are two benefits: obtaining more storage space in the hard disk and performing the matching in a compressed and reduced graph without decompression [43]. In addition, graph compression techniques that retain all the information of the original graphs and that can be used for matching remain a challenge.

In the majority of approaches, the space complexity of graph matching has not been investigated. The different approaches do not deal much about space
Table 1: Summary of partition-based approaches

| Approach | Graphs | Decomposition unit | Comparison concept | Application | Size of the query | Size of data graph | Program | Time Complexity |
|----------|--------|-------------------|------------------|-------------|------------------|-------------------|---------|----------------|
| [21, 22] directed labeled edges | BARG | Edit | distance | Image processing | No experiment | No experiment | Sequential | $O(|V_G|^2|V_Q|(|V_G| + |V_Q|))$ |
| [68] directed unlabeled edges | Relational Description | Number of common RDs | Image processing | No experiment | No experiment | Sequential | Not computed |
| [60] undirected labeled edges | Star | Edit | distance | Image processing | 4-12 nodes | 4-12 nodes | Sequential | $O(|V_Q|^2)$ |
| [61] undirected unlabeled edges | Star | Edit | distance | Chemistry Networks | 5-65 nodes | 1 - 80 nodes | Sequential | $O(|V_Q|^2)$ |
| [82][83] undirected labeled edges | Star | Edit | distance | Image processing | - | 8-126 nodes | Sequential | Not computed |
| [69] undirected labeled edges | Signature | Edit | distance | Retrieving Image | - | 9-117 nodes | Sequential | Not computed |
| [82] undirected unlabeled edges | Half-edge subgraph | Edit | distance | Chemistry Networks | - | 40 - 100k nodes | Sequential |
| [83] undirected unlabeled edges | path-based q-gram | Edit | distance | Chemistry Networks | - | 40 - 126 nodes | Sequential | $O(\tau(|V_G| + |V_Q| \log |V_Q|))$ |
| [76] undirected unlabeled edges | kAT | Edit | distance | Chemistry Networks | - | 40 - 100k nodes | Sequential | Not computed |
| [70] undirected unlabeled edges | STwig | Subgraph Matching | Web Networks | 3-10 nodes | 80 - 4096K nodes | Parallel | $O(|q|^\gamma)$ |
| [89] undirected unlabeled edges | Neighborhood vector | Edit | distance | Web Networks | 8-12 nodes | 172k-100000k nodes | Sequential | $O(|V_Q| \cdot d^\gamma)$ |
| [10] undirected unlabeled edges | Neighborhood vector | Edit | distance | Web Networks | 3-7 nodes | 2M - 12M nodes | Sequential | $O(|V_Q| \cdot |V| + I \cdot |V_Q| \cdot m_Q \cdot d_Q)$ |
| [84] undirected labeled edges | Branch structure | Edit | distance | Biology | 40k - 100k nodes | 40k - 100k nodes | Sequential | Not computed |

$\tau$: graph edit distance threshold.
$h$: hops. $d$: the average degree of each node.
$d_Q$: the maximum number of $h$-hop neighbors of each query node.
$m_Q$: maximum number of candidates per query node.
complexity and memory consumption of algorithms which are important performance metrics either in theory or practice coping with large graphs.

The problem of matching dynamic graphs has not received enough interest in the literature. Currently with social networks and the web, graphs change continuously: new nodes and edges are added or deleted from the graph through time. The problem is then to take into consideration the evolution of dynamic graphs in graph comparison or pattern matching approaches. Apart from the work of [23] we found little literature on this question.

5 Conclusion

The dominance of graphs as a representation tool in real world applications demand new graph matching techniques, concepts, and languages to match large graph datasets efficiently. We have presented a review of recent works on graph comparison and graph pattern matching approaches on large graphs, highlighting the different notions, techniques and concepts used for matching and their impact coping with large graphs. We classified the approaches into three categories: partition based approaches, search space exploring approaches and summary-based approaches. Each of them has its advantages and application areas. Many recent graph comparison and graph pattern matching approaches converge towards partitioning of the compared graphs. The problem is simplified by decomposing the graphs to be matched into smaller subgraphs. However, these approaches are not always possible and there are few algorithms suitable for all kinds of graphs and applications. Globally and as discussed in the previous section several problems and area of investigations deserve future research despite the substantial results of current and past investigations. According to the International Technology Roadmap for Semiconductors (ITRS), as many as 6000 processors are expected on a single system-on-chip by the end of year 2026. Moreover, the memory size will follow the same trends. Thus, parallel graph matching algorithm is needed for the next generation in order to run quickly the matching processes and exploit efficiently the hardware resources such as the number of processors and memory size. Moreover, due to the huge size of graphs, compressing graphs for matching without decompression remains a challenging issue. Combining parallelism with compressing or partitioning is also very interesting. Furthermore, dynamic graphs and graphs in streaming applications are not sufficiently addressed in the actual research effort.

References

[1] C. C. Aggarwal and H. Wang. Managing and mining Graph data. Springer, 2010.

[2] R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging and its application to diatom identification. Graph Based Representations in Pattern Recognition - GBR, pages 95–106, 2003.
[3] M. Basu and T. K. H. BBA. *Data Complexity in Pattern Recognition*. Springer, 2006.

[4] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, 14(Suppl 7)(S13), 2013.

[5] K. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *5th Int. Conference on Data Mining*, pages 74–81, 2005.

[6] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18:689–694, 1997.

[7] H. Bunke. Error correcting graph matching: On the influence of the underlying cost function. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(9):917–922, 1999.

[8] H. Bunke and G. Allerman. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters-PRL*, 1(4):245–253, 1983.

[9] H. Bunke and B. T. Messmer. Recent Advances in Graph Matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 11:169–203, 1997.

[10] H. Bunke and K. Riesen. Recent advances in graph-based pattern recognition with applications in document analysis. *Pattern Recognition*, 44:1057–1067, 2011.

[11] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998.

[12] M. Carcassoni and E. R. Hancock. Weighted graph-matching using modal clusters. pages 142–151, 2001.

[13] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE ’06, pages 5–, Washington, DC, USA, 2006. IEEE Computer Society.

[14] C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han. Mining graph patterns efficiently via randomized summaries. *Proc. VLDB Endow.*, 2(1):742–753, Aug. 2009.

[15] J. Cheng, J. Yu, B. Ding, P. Yu, and H. Wang. Fast graph pattern matching. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 913–922, April 2008.

[16] J. Cheng, X. Zeng, and J. Yu. Top-k graph pattern matching over large graphs. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1033–1044, April 2013.
[17] W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI*, 17(8):749–764, 1995.

[18] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18:265–298, 2004.

[19] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1367–1372, 2004.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[21] M. Eshera and K. Fu. A similarity measure between attributed relational graphs for image analysis. In *7th International Conference on Pattern Recognition*, pages 75–77, 1984.

[22] M. Eshera and K.-S. Fu. A graph distance measure for image analysis. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-14(3):398–408, May 1984.

[23] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *Proc. VLDB Endow.*, 3(1-2):264–275, Sept. 2010.

[24] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’12, pages 157–168, New York, NY, USA, 2012. ACM.

[25] A. Fard, M. U. Nisar, L. Ramaswamy, J. A. Miller, and M. Saltz. A distributed vertex-centric approach for pattern matching in massive graphs. In *Proceedings of the 2013 IEEE International Conference on Big Data*, 6-9 October 2013, Santa Clara, CA, USA, pages 403–411, 2013.

[26] A. Fard, M. U. Nisar, L. Ramaswamy, J. A. Miller, and M. Saltz. Distributed and scalable graph pattern matching: Models and algorithms. *International Journal of Big Data*, 1(1):1–14, 2014.

[27] B. Gallagher. *Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching*. PhD thesis, 1939.

[28] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[29] T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18:25–66, 1967.
[30] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis Applications*, (13):113–129, 2010.

[31] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[32] T. Gartner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In Springer, editor, *Annual Conf. Computational Learning Theory*, pages 129–143, 2003.

[33] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[34] W.-S. Han, J. Lee, and J.-H. Lee. Turboiso: Towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’13, pages 337–348, New York, NY, USA, 2013. ACM.

[35] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[36] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.

[37] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing Simulations on Finite and Infinite Graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[38] S. Jouili and S. Tabbone. Attributed graph matching using local descriptions. In *ACIVS 2009, LNCS 5807*, pages 89–99, 2009.

[39] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011*, Athens, Greece, June 12-16, pages 901–912, 2011.

[40] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. Nema: Fast graph search with label similarity. *PVLDB*, 6(3):181–192, 2013.

[41] K. G. Khoo and P. N. Suganthan. Multiple relational graphs mapping using genetic algorithms. pages 727–737, 2001.

[42] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[43] S. Lagraa, H. Seba, A. M’Baya, R. Khennoufa, and H. Khedouci. A Distance Measure for Large Graphs based on Prime Graphs. *Pattern Recognition*, 2013.
[44] D. P. Lopresti and G. T. Wilfong. Comparing Semi-Structured Documents via Graph Probing. In Workshop on Multimedia Information Systems, pages 41–50, 2001.

[45] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing topology in graph pattern matching. Proc. VLDB Endow., 5(4):310–321, Dec. 2011.

[46] S. Ma, Y. Cao, J. Huai, and T. Wo. Distributed graph pattern matching. In Proceedings of the 21st International Conference on World Wide Web, WWW ’12, pages 949–958, New York, NY, USA, 2012. ACM.

[47] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing - "abstract". In Proceedings of the 28th ACM Symposium on Principles of Distributed Computing, PODC ’09, pages 6–6, New York, NY, USA, 2009. ACM.

[48] B. McKay. Practical graph isomorphism. Congress Numerantium, 87:30–45, 1981.

[49] B. Messmer. Efficient Graph Matching Algorithms for Preprocessed Model Graphs. PhD thesis, University of Bern, Switzerland, 1995.

[50] B. T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI, 32(12):1979–1998, 1999.

[51] A. Micheli. Neural network for graphs : A contextual constructive approach. IEEE Transactions on Neural Networks, 20(3):498–511, 2009.

[52] R. Milner. Communication and Concurrency. Prentice Hall, 1989.

[53] J. Munkres. Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics, 5:32–38, 1957.

[54] R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian graph edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI, 22(6):628–635, 2000.

[55] M. Neuhaus and H. Bunke. An Error-Tolerant Approximate Matching Algorithm for Attributed Planar Graphs and Its Application to Fingerprint Classification. In International Workshop on Structural and Syntactic Pattern Recognition, pages 180–189, 2004.

[56] M. Neuhaus and H. Bunke. A Random Walk Kernel Derived from Graph Edit Distance. In International Workshop on Structural and Syntactic Pattern Recognition, pages 191–199, 2006.
[57] M. Neuhaus and H. Bunke. A convolution edit kernel for error-tolerant graph matching. In *IEEE international conference on pattern recognition, Hong Kong*, pages 220–223, 2006.

[58] M. Neuhaus and H. Bunke. Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177:239–247, 2007.

[59] J. Pearl. Reverend bayes on inference engines: a distributed hierarchical approach. In *in Proceedings of the National Conference on Artificial Intelligence*, pages 133–136, 1982.

[60] R. Raveaux, J.-C. Burie, and J.-M. Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31:394–406, 2010.

[61] J. W. Raymond, E. J. Gardiner, and P. Willett. RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs. *The Computer Journal*, 45:631–644, 2002.

[62] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27:950–959, 2009.

[63] K. Riesen, M. Neuhaus, and H. Bunke. Bipartite graph matching for computing the edit distance of graph. pages 1–12, 2007.

[64] A. Robles-kelly and E. R. Hancock. Graph Edit Distance from Spectral Serration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:365–378, 2005.

[65] A. Robles-kelly and E. R. Hancock. A Riemannian approach to graph embedding. *Pattern Recognition -PR*, 40(3):1042–1056, 2007.

[66] A. Sanfeliu, R. Alquézar, and F. Serratosa. Clustering of attributed graphs and unsupervised synthesis of function-described graphs. volume 2, pages 6022–6025, 2000.

[67] A. Sanfeliu and K. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 13(3):353–363, 1983.

[68] L. G. Shapiro and R. M. Haralick. A metric for comparing relational descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 7(1):90–94, 1985.

[69] P. N. Suganthan. Structural pattern recognition using genetic algorithms. *Pattern Recognition - PR*, 35(9):1883–1893, 2002.

[70] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.
[71] E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In P. Spirakis, editor, Algorithms-ESA’95, volume 979 of Lecture Notes in Computer Science, pages 327–340. Springer Berlin Heidelberg, 1995.

[72] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’07, pages 737–746, New York, NY, USA, 2007. ACM.

[73] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. J. ACM, 23(1):31–42, Jan. 1976.

[74] S. Umeyama. An eigen decomposition approach to weighted graph matching problems. IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI, 10(5):695–703, 1988.

[75] L. G. Valiant. A bridging model for parallel computation. Commun. ACM, 33(8):103–111, Aug. 1990.

[76] M. Vento. A One Hour Trip in the World of Graphs, Looking at the Papers of the Last Ten Years. In W. G. Kropatsch, N. M. Artner, Y. Haxhimusa, and X. Jiang, editors, Graph-Based Representations in Pattern Recognition, volume 7877 of Lecture Notes in Computer Science, pages 1–10. Springer Berlin Heidelberg, 2013.

[77] W. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. Pattern Recognition Letters, 22(6-7):701 – 704, 2001.

[78] G. Wang, B. Wang, X. Yang, and G. Yu. Efficiently indexing large sparse graphs for similarity search. Knowledge and Data Engineering, IEEE Transactions on, 24(3):440–451, March 2012.

[79] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin. An efficient graph indexing method. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, ICDE ’12, pages 210–221, Washington, DC, USA, 2012. IEEE Computer Society.

[80] Y. Xiao, H. Dong, W. Wu, M. Xiong, W. Wang, and B. Shi. Structure-based graph distance measures of high degree of precision. Pattern Recognition, 41:3547–3561, 2008.

[81] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. Proceedings of The Vldb Endowment - PVLDB, 2(1):25–36, 2009.

[82] X. Zhao, C. Xiao, X. Lin, Q. Liu, and W. Zhang. A partition-based approach to structure similarity search. Proc. VLDB Endow. PVLDB, 7(3):169–180, 2013.
[83] X. Zhao, C. Xiao, X. Lin, and W. Wang. Efficient Graph Similarity Joins with Edit Distance Constraints. In IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, pages 834–845, 2012.

[84] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Graph similarity search with edit distance constraint in large graph databases. In 22nd ACM International Conference on Information and Knowledge Management, CIKM’13, San Francisco, CA, USA, October 27-November 1, 2013, pages 1595–1600, 2013.

[85] L. Zou, L. Chen, and M. T. Özsu. Distance-join: Pattern match query in a large graph database. Proc. VLDB Endow., 2(1):886–897, Aug. 2009.
Table 2: Summary of search-space exploring approaches

| Approach          | Graphs   | Comparison Concept | Application | Size of the query | Size of data graph | Program | Time Complexity |
|-------------------|----------|--------------------|-------------|-------------------|--------------------|---------|-----------------|
| [34]              | undirected | unlabeled isomorphism | Biology     | 2-15 nodes        | 0.5M-4M nodes      | Sequential | $O(|V G|)$     |
|                    | directed  | bounded edges       | Web         | 4-10 nodes        | 1k-20k nodes       | Sequential | $O(|V G| + |E G| + |V Q|)$ |
|                    | directed  | unlabeled simulation | Web         | 3-15 nodes        | millions of nodes  | Sequential | Not computed   |
|                    | directed  | unlabeled edge      | strict social networks | 10-20 nodes | millions of nodes | Parallel | Not computed   |
|                    | directed  | unlabeled simulation | tight social networks | 5-100 nodes | millions of nodes | Parallel | $O(|V Q|)$   |
Table 3: Existing summary-based approaches

| Approach | Graphs | Comparison Concept | Application | Size of the query | Size of data graph | Program | Time Complexity |
|----------|--------|--------------------|-------------|-------------------|-------------------|---------|-----------------|
| [14]     | undirected labeled edges | Subgraph mining data | Program Not Necessary | 100-20k nodes | 220k edges | Sequential | Not computed |
| [24]     | directed labeled edges | Compression preserving query | Social Networks 3-8 edges 6k-2.4M nodes 21k-5M edges | Sequential | $O(|V(G)|^2 + |E(G)|)$ |
| [43]     | undirected unlabeled edges | Prime graph Biological graphs 8-34000 nodes 9-33k nodes 9-332k edges | Sequential | $O(k^2 + |V(G)| + |E(G)|)$ |

$k$ is the number of vertices in the largest prime graph.