A Sub-graph Matching Method based on Calibration of Characteristics of Topological Footprint

David F. Nettleton
Pompeu Fabra University
Dept. Inf. Technology and Communications
Barcelona, Spain

Anton Dries
KU Leuven
Dept. Computer Science
Leuven, Belgium

ABSTRACT
Approximate sub-graph matching is important in many graph data mining fields. At present, current solutions can be difficult to implement, have an expensive pre-processing phase, or only work for given types of graph. In this paper a novel generic approach is presented which addresses these issues. An approximate sub-graph matcher (A-SGM) calculates the distance between the topological characteristics (footprint) of the sub-graphs to be matched, applying a weighting to the different sub-graph characteristics and those of neighbor nodes. The weights are calibrated for each dataset with a simulated annealing process using sample sets of graph nodes to reduce computational cost, and an exact isomorphism matcher as a fitness function which takes into account how well the match maintains the neighboring node degree distributions. Benchmarking is performed on several state of the art methods and real and synthetic graph datasets to evaluate the precision, recall and computational cost. The results show that the A-SGM is competitive with state of the art methods in terms of precision, recall and execution time.

General Terms
Machine Intelligence (Data and Web Mining), Applications of Computer Science in Modeling, Data and Information Systems.

Keywords
Graph Matching, topology, graph characteristics, weight calibration, simulated annealing, graph queries.

1. INTRODUCTION
The matching of sub-graphs is a key task for graph mining which however may have a high computational cost for large data domains. Different matching paradigms exist in the literature. On the one hand there is the exact one to one mapping of labeled nodes exemplified by isomorphism matchers. For these, different optimizations exist in order to reduce the computational cost, such as compressed matrix representations of the graph structure. On the other hand, approximate matchers relax the requirement for a total matching.

The need for an exact isomorphic matching is also application and domain dependent. Applications such as molecular discovery[1][2] and chemical analysis[3] may require exact isomorphic matching, whereas for data mining online social networks (OSNs) a good approximate match may be sufficient. Example applications for OSN sub-graph matching are user profiling, classification and clustering, network analysis, behavior analysis, among others.[4]

It must also be mentioned that an approximate matcher can be used as a first step to reduce the search space, after which an exact matcher can be applied. This approach is used by different authors and some of these works will be summarized in the state of the art.

The matcher presented in this paper uses topological graph characteristics identified by data mining and statistical analysis techniques to compose a descriptive footprint. A corresponding weighting scheme is applied for each characteristic to capture an approximate "model" of the GED (Graph Edit Distance) [5] isomorphism matcher which is used as fitness function. The runtime computational cost is reduced because the calculations required for the sub-graph characteristics (statistical metrics such as degree, clustering coefficient, number of edges, degrees of neighbors) are not NP-hard and are pre-calculated. Then the distance metric calculates the difference between the respective weighted characteristics of each sub-graph. This is used as a first step to reduce the search space then a label mapping is used to obtain the most exact match. The principal overhead is the training of the weights, for which simulated annealing is used as a pre-process and GED[5] as the fitness function. The computational cost of the training is reduced by using relatively small but representative samples of the complete dataset, and cross validating the results.

In this paper the approximate graph matcher presented is evaluated against two state of the art methods, in terms of computational cost, precision and recall. The sub-graph matching algorithm in the present work has been submitted as a European and International Patent application[6]. It has also been successfully used in a specific application for data privacy (k-anonymization) [7].

The structure of the paper is as follows: in Section 2 the state of the art and related work is discussed for the issues considered in this paper; in Section 3 definitions are given for the key concepts used in the remainder of the paper; in Section 4 the distance metric is described, together with some example calculations and details of the data processing scheme; in Section 5 the experimental setup, datasets and benchmarking methods are described; in Section 6 the empirical results for the training and test (apply) phases are presented; finally, in Section 7 the work is summarized.

2. RELATED WORK AND STATE OF THE ART
For convenience, the related work and state of the art will be divided into two general areas: (i) isomorphic matching approaches and (ii) recent work which includes different approximate approaches, indexing schemes and scalable solutions for big data.

2.1 Isomorphic Matching

A key property of interest in graphs is the isomorphic property, which refers to an exact match between two given graphs, in terms of structure, dimensionality, connectivity and mapping of corresponding nodes. Graph matching is a key activity in different pattern recognition applications, in which high and low level information may be represented. However, graph matching has a high computational cost, and the task of finding isomorphic sub-graphs is a problem which is NP-complete.

Two key references for isomorphic matchers are nauty[8] and VF2[9]. Nauty uses group theory concepts in order to efficiently construct the automorphism group of each of the input graphs, from which a canonical labeling is derived. This obtains a node ordering that is uniquely defined for each equivalence class of isomorphic graphs, such that two graphs can be isomorphically matched by a relatively simple verification of the equality of the adjacency matrices of their canonical forms. The equality verification has $O(N^2)$ time, however the preprocessing of the canonical labeling can require an exponential time in the worst case. VF2, on the other hand, is based on converting the graph into a tree structure and then performing a depth-first search, using a set of rules to efficiently prune the search tree. It uses data structures which have been adapted in order to further reduce the computational cost of matching. VF2 has subsequently become widely used in the graph mining community.

The Graph Edit Distance (GED) [5] calculates the number of changes required in order to convert one graph into another. In general, the number of changes necessary is directly related to the similarity of graphs. The operators used to change the graph can be, for example, add edge, delete edge, add node, delete node, and so on. GED represents an optimized version of the graph edit distance measure, in which Bunke evaluated the relation between the cost function and the optimal matching of two graphs.

2.2 Recent Work

In this section a selection of recent sub-graph matching methods and distance measures are reviewed, with an emphasis on those which are most often referenced and used for benchmarking within the graph mining community.

Tale[10] is a method which uses graph characteristics and stochastic techniques in order to perform approximate matching of sub-graphs. This method uses an indexing technique, called Neighborhood Index (NH-Index), which uses a node's neighborhood characteristics. Queries are launched against the graph, and a graph node matches the query node, only if the two nodes match and their neighborhoods also match. A neighborhood is defined as the induced sub-graph of a node and its neighbors (adjacent nodes). The similarity function is defined as follows: 'nbConnection' is the neighbor connectivity of a given node; ρ is the percentage of neighbors of a query node that can have no corresponding matches in the neighborhood of a graph node. Hence, 'nbmiss' = ρ × (Nq.degree) represents the number of neighbors of the query node which can be missing in the match to a graph node and 'nbcmiss' is the number of missing neighbor connections. Then the fraction of missing neighbors of the query node is defined as fnb = nbmiss / Nq.degree, and the fraction of missing neighbors connections is defined as fnbc = nbcmiss / Nq.nbconnection .

Wei[11] presents 'Tedi', a method for efficiently calculating shortest paths for graph queries. It employs an indexing and query processing scheme in which the graph is first decomposed into a tree such that each (tree) node contains multiple graph vertices. The shortest paths are stored in the tree nodes and these local paths together with the tree are used to form a lookup index to the graph. A graph search can then be performed as a bottom-up process on the tree.

Zou et al.[12] present a distance-based pattern matching method for queries over a large data graph G. In order to process the large search space, the authors adopt a filter-and-refine framework to answer pattern match queries over the graph. The method is benchmarked against Tedi[11] and Sapper[13]. First, a set of candidate matches is found by a graph embedding technique and then the set of matches is evaluated to find the exact matches. Zhao et al.[14] present 'SPath', which the authors describe as being a high performance graph indexing mechanism, that addresses the graph query problem on large networks. 'SPath' decomposes shortest paths around vertex neighborhoods and uses them as basic indexing units, which was found to be effective for graph search space pruning and scalable for index construction and deployment. 'SPath' processes and optimizes a graph query 'one path at a time', instead of the less efficient, 'vertex at a time' method. They benchmark their method against the GraphQL[15] algorithm.

Khan et al.[16] present 'Ness', a neighborhood based similarity search designed for graph datasets with a low incidence of automorphisms and a high incidence of noise (such as online social networks). Their method is based on an information propagation model which transforms a large network into a set of multidimensional vectors, which are then processed by indexing and similarity search algorithms. The neighborhood cost function aggregate the differences for all node pairs (v, u), where u = f(v).

Sun et al.[17] study the problem of sub-graph matching on big data graphs. They present an algorithm that supports efficient sub-graph matching for graphs deployed on a distributed memory store. The method uses efficient graph exploration and massive parallel computing for query processing. Given a query, a cluster graph is created to model the data distribution among different machines in the cluster with regard to the query.

Another method which takes into account noisy data is Sapper, presented by Zhang et al. in [13]. Sapper (Sub-graph Indexing and Approximate Matching in Large Graphs) considers the existence of noise (such as missing edges) in large database graphs, and how it affects approximate sub-graph indexing and thus finding the occurrences of a query sub-graph in a large database graph. Sapper uses hybrid neighborhood unit structures in the index with pre-generated random spanning trees and a specific graph enumeration order. The authors benchmark their method against Tale[10] and Gaddi[18] using real and synthetic graph datasets.

Gaddi[18] is a method which uses a distance measurement based on frequent substructure counts over a single large graph. The distance metric, called NDS (neighboring discriminating substructure distance) is used for graph indexing which the authors claim has a high pruning power in the search space, and scales linearly with the number of neighboring vertex pairs. Consider a pair of vertices v_1 and v_2 within distance L in the overall graph G, and an integer k defined such that $2 \times k > L$. Then an intersecting sub-graph $Int(G, v_1, v_2)$ is generated from v_1 and v_2 as follows: (i) the k-neighborhood set of v_1 and v_2 is generated and defined as $N_k(G, v_1)$ and $N_k(G, v_2)$; (ii) the intersection of $N_k(G, v_1)$ and N_k

(G, v_2) is obtained , that is N_k (G, v_1) ∩ N_k (G, v_2); (iii) the induced sub-graph of the intersection set Int(G, v_1, v_2) is obtained as S(N_k (G, v_1) ∩ N_k (G, v_2)). Once the set of intersecting sub-graphs has been obtained, frequent sub-graph mining techniques are used to find frequent substructures and to select the most discriminative ones.

To conclude this section, it is noted that a specific application of the sub-graph matcher A-SGM was presented by Nettleton et al. in [7], in which it was successfully used to match local neighborhoods of a graph for a data privacy application. In [7], a k-anonymization process is applied in which objects (local neighborhoods in this case) are generalized into groups of k members. This requires finding, for a given local neighborhood, the k-1 most similar local neighborhoods.

3. DEFINITIONS AND PRELIMINARY DISCUSSION

A graph is defined as a set of vertices V interconnected by a set of edges, thus: G = (V, E). In the current work each vertex has an identifier for data processing purposes which is considered as its label.

Isomorphism: two graphs G_1 = (V, E_1), G_2 = (V, E_2) are designated as being isomorphic if a permutation *p* exists such that $p(G_1) = G_2$. That is, with the same set of vertices, the edges of G_1 can be rearranged to fit G_2. In Fig. 1 an example of two graphs is shown, one of which is an isomorphism of the other, by the permutation {(A, V), (B, W), (C, X), (D, Z), (E, Y)}.

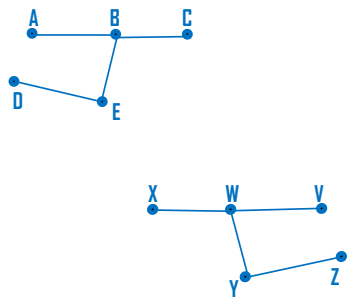

Fig. 1. Graph Isomorphisms: the upper and lower graphs are isomorphic if an adequate mapping can be defined between them.

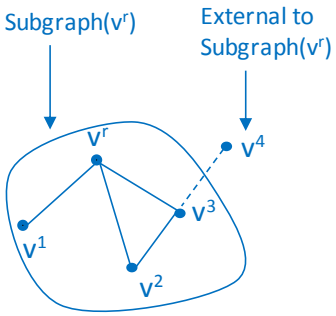

Fig. 2. Sub-graph depiction

Sub-graph: a sub-graph G^n = (V', E') is defined as a subset of G around a given reference vertex v^r at a distance of one. Hence $v^r \in V'$ and all other vertices $v' \in V'$ are immediate neighbors of v^r. An example of a sub-graph can be seen in Fig. 2.

Approximate topological matcher: an approximate topological matcher is defined as a function *A-SGM*(G_1, G_2, {T}, {W}), which returns a similarity value ϕ for the two sub-graphs G_1 and G_2, given a set of topological characteristics (or features) {T} and a set of weights {W} which act on the features. The set of features {T} is defined as {t_1, t_2, t_3, t_4, t_5} and the set of weights {W} is defined as {w_1, w_2, w_3, w_4, w_5}. It is observed that in the current implementation there are five features and their corresponding weights, although in a different implementation a distinct number of features and weights could be used.

Approximate Label Pair Edge Matcher: assume that a sub-graph G_1 has a set of labels L_1 = {l_1, l_2, l_3, … , l_n} and a set of edges between label E_1 = {{l_1, l_2}, {l_1, l_3}, …, {l_n, l_m}}. Likewise, a sub-graph G_2 has a set of labels L_2 and a set of edges between labels E_2. Then a label edge matcher will count the number of edges which are different, defined as E_Δ , using the following formula:

$$E\Delta = \frac{|\{E1\} \cup \{E2\}|}{|\{E1 \cap E2\} + \{E2 \cap E1\}|} \quad \textbf{(1)}$$

which gives a distance measure between the sub-graphs in terms of the label pair similarity.

Exact matcher: an exact matcher is defined as a function *GED*(G_1, G_2) which returns a distance value Γ indicating the number of modifications (edits) required to make the sub-graphs G_1 and G_2 isomorphic.

Samples: a set of *n* training samples {S} is defined as { s_1, .. s_m} which is randomly selected from the complete graph G, each sample s_i consisting of *m* vertices. Each sample vertex has to be at least at distance three from any other sample node in order to obtain disjunct neighborhoods.

Fitness: The fitness of the approximate matcher is defined in terms of the information retrieval concepts of precision and recall, for which the *'relevant'* sub-graph set is that which is found by the GED matcher. Then, the *'returned'* sub-graph set is that which is found by the A-SGM matcher.

Relevant sub-graphs: The reference (ground truth) function is defined as GED and the function to be tested as A-SGM. Now let the list L_E be the list of (relevant) sub-graphs returned by the *GED* function, in ascending order of the edit distance value with respect to a given sub-graph G_r with reference node v^r. Let the list L_U be the ranked list of (returned) sub-graphs returned by the *A-SGM* function, in ascending order of the similarity value with respect to the sub-graph G_r corresponding to a given reference node v^r. It is observed that each sub-graph in the list is identified by its corresponding reference node v^r in the graph.

Then, the fitness of the A-SGM function is measured in terms of the information retrieval metrics 'precision', 'recall' and F_1. These metrics are defined in terms of two sets of results: (i) the 'ideal' set of results returned by an 'optimum' method (GED, in this case) and (ii) another set of results returned by a method which is to be benchmarked (e.g. A-SGM). The former set is known as the "relevant results' and the latter set is known as the 'returned results'.

3.1 Defining the Relevant Set of Results

The definition of the 'relevant set' is often defined by the precision@(*n*) measure (precision at *n*), that is, the precision and recall given by considering the first *n* results retrieved by the reference matcher (in the present case, GED) as the relevant set. The 'returned results' set will then also be defined as the first *n* results, ordered by the similarity metric of the matcher used (e.g. A-SGM). However this is an artificial cutoff, and the returned results are dependent on the items being queries. For example, Tian and Patel[10] reported an optimum precision of approx. 85% for a recall of 80% for the 'Tale' matcher processing the ASTRAL protein structure dataset. For higher recall values the precision dropped rapidly. These relatively high values were obtained because of the nature of the dataset and given that the methods tend to return relevant results as their top results. Walters[19] presented a detailed study of precision and recall for 8 different dataset corpuses including Google Scholar, finding an optimum precision of 80% for a corresponding recall of 50%. However, for all corpuses, lower precisions (approx 50%) and recall (approx. 30%) were obtained with an optimum relevant document set size of between 10 and 25 (for precision) and between 20 and 40 (for recall).

Relevant Results: in the current context, in order to limit the set of "relevant results" returned by the GED function for a given sub-graph query, it is necessary to specify which results will be included in the set. Firstly, all results are included whose distance is zero, that is, GED designates them as exact matches (isomorphisms). It is observed that sub-graphs of distance zero cannot be ordered, thus if an artificial cutoff is defined for the relevant set, some of the isomorphisms returned by GED may be excluded. As a consequence, if a test method returns an isomorphism in the returned set which was not in the relevant set, it will be considered as not relevant. Hence, in order to guarantee that the relevant set always contains all the isomorphisms, its minimum size is set to the maximum number of isomorphisms τ (with distance zero with respect to the query) returned for any query. On the other hand, if there are less than τ isomorphisms returned, the results with the top non-zero distances are included, ordered by distance until a set size of τ is obtained. In practice, this procedure obtains an average relevant set size of approx. τ=25 results. It is observed that the test datasets (see Section 5) will have different characteristics with respect to the average number of isomorphisms returned per query. This is particularly so for the protein graph dataset which has a more regular topological structure than the online social network graph datasets. It is also true that queries consisting of small sub-graph structures, for example, two vertices connected by an edge, would return a much higher number of isomorphisms. However, in the present work, a sub-graph query is always formed by a reference node and its immediate neighbors. Consequently, given that the average degree tends to be relatively high, this will in the majority of cases result in quite large sub-graph structures as query targets which have relatively few isomorphisms in the complete graph.

To formalize what has been commented about relevant result sets, L_E will represent the set of "relevant sub-graphs" and L_U the set of "returned sub-graphs". Then, the information retrieval metrics of precision P and recall R are defined as:

$$P = \frac{|\{L_E\} \cap \{L_U\}|}{|\{L_U\}|} \quad \textbf{(2)}$$

and

$$R = \frac{|\{L_E\} \cap \{L_U\}|}{|\{L_E\}|} \quad \textbf{(3)}$$

Then the F_1 measure (information retrieval metric) is defined in terms of the precision and the recall sub-graph sets. The F_1 measure combines precision and recall as their harmonic mean:

$$F = 2 \times \frac{P \times R}{P + R} \quad \textbf{(4)}$$

In the case of F_1, recall and precision are evenly weighted. This measure has been used in order to obtain an equilibrium between these metrics for the current application. Other variants include F_2, which biases recall over precision, and $F_{0.5}$ which gives a greater weight to precision.

4. DESCRIPTION OF THE METHOD

In this Section, firstly the distance metric of the sub-graph matcher will be described, followed by some examples of how the matching is calculated. Next there is a description of the overall data processing scheme. In Fig. 3 a schematic representation of the process is shown.

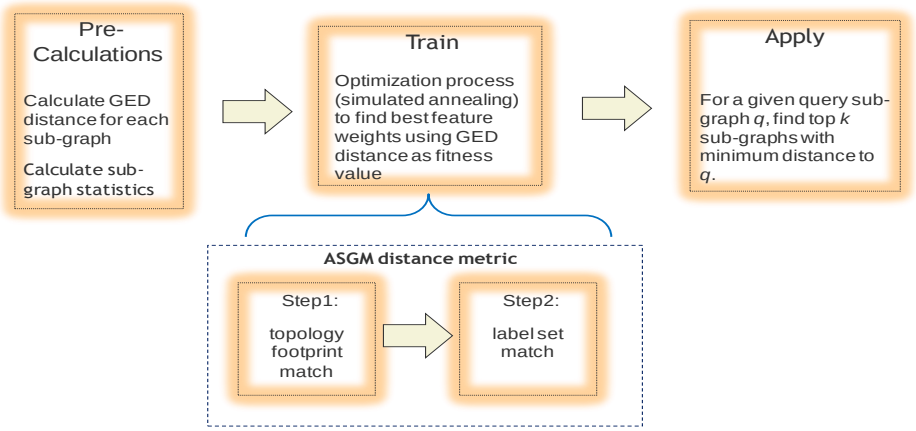

Fig. 3. Overall Scheme of Data Processing

4.1 Distance Metric

In order to calculate the similarity between two sub-graphs, a two step distance metric is used. The first step serves to reduce the search space and finds similar topologies based on graph statistical features. The second step acts only on the top sub-graphs identified by Step 1, and performs a label

matching. It is recalled that a sub-graph is defined as a reference vertex, its immediate neighbor vertices and the links between them (see Fig. 2).

Step 1: The topology similarity metric calculates a distance based on sub-graph characteristics which are pre-calculated.

The sub-graph characteristics (footprint) are:

- degree of the target node D_T
- number of edges in the sub-graph N_E
- clustering coefficient CC
- normalized average (internal) degree of adjacent nodes AD_{AN}
- normalized standard deviation of (internal) degree of adjacent nodes SD_{AN}

The five characteristics were selected by statistical evaluation so as to best reflect the internal structure of the sub-graph, while including characteristics about the neighbors (such as their degree including links outside the sub-graph). In order to perform the calculation, all values are normalized against the maximum and minimum corresponding values in the complete graph. The immediate neighborhood sub-graphs of two reference vertices v_1 and v_2 are designated as sub-graphs G_1 and G_2, respectively. Then, the distance metric will be as follows:

$$\textit{A-SGM1}(G_1, G_2) = \Delta(D_T(G_1), D_T(G_2))\times\alpha + \Delta(N_E(G_1), N_E(G_2))\times\beta + \Delta(CC(G_1), CC(G_2))\times\chi + \Delta(AD_{AN}(G_1), AD_{AN}(G_2))\times\delta + \Delta(SD_{AN}(G_1), SD_{AN}(G_2))\times\varepsilon \quad \textbf{(5)}$$

The weight vector $\{\alpha, \beta, \chi, \delta, \varepsilon\}$ is trained using a simulated annealing process. $\Delta(D_T(G_1), D_T(G_2))$ means the normalized difference between metric D_T (degree of target node) of graph G_1 and metric D_T of graph G_2.

Step 2: Using as input the most similar sub-graphs identified by Step 1, the label matcher calculates a distance based on a list of vertex labels {L} and a list of label pairs which are connected {LP}. Again consider two sub-graphs G_1 and G_2. The distance metric will be as follows:

$$\textit{A-SGM2}(G_1, G_2) = |\{LP_1\} \cap \{LP_2\}| \quad \textbf{(6)}$$

Equation (6) gives a numerical value for the number of label pairs which are different between the two sub-graphs G_1 and G_2. A value of zero will imply an exact match between G_1 and G_2 based on label pairs. It is this value which is compared with the value generated by GED, for each sub-graph pair in the sample training set, the difference being used as the fitness value for the simulated annealing process which calibrates the weights for Equation (5). It is observed that, for each sub-graph, the assignment of the vertex label sets {L} and the connected vertex label sets {LP} is performed within the clustering coefficient calculation, with no extra cost. Efficient adjacency list data structures hold the sets for each vertex.

At runtime, the intersection is calculated by direct access to the adjacency list data structures using the attribute-value as index. This inevitably requires a one to one checking of each label and label pair, rather than the simple numerical differences of Step 1 which are pre-calculated. However, Step 1 greatly reduces the search space by selecting only a small subset of the closest matching sub-graphs in terms of their topology, and hence Step 2, which contains the most expensive step of label matching, is performed only on this reduced subset.

4.2 Example Calculations of Distance Metric

With reference to Fig. 4 and Table 1, eight similar graphs are shown with a slightly different number of vertices, edges and labels. Each graph has a 'reference node'. For example, in the case of graph G_1 in Fig. 4 the reference node is labeled as R, which in turn has a number of 'neighbor nodes', which for graph G_1 are labeled as A to F. From the graph statistics of Table 1, it can be seen how the topological features vary in relation to the topologies, and how they are sensitive to small changes in the sub-graph. The set of returned graphs shown in Fig. 4 is ordered by the distance value given by Equation (5), which is shown in column 1 of Table 2. It is observed that graphs G_2 and G_3 have a mutual distance of zero with respect to G_1, as calculated by Equation (5) due to their identical topology. However, when the label matcher of Equation (6) is applied, it is found that the top match to G_1 is G_4 (col. 2 of Table 2).

Example of edge pair calculation: With reference to Fig. 4, considering graphs G_1 and G_6, first a matching is made on G_1's edges in G_6. It is found that A-B C-D, F-E and F-R are missing, which gives a provisional missing count of 4. Then the inverse comparison is performed, looking for G_6's edges in G_1. It is found that two edges are missing, B-C and D-E. Hence, the total edge pair difference for graphs G_1 and G_6 will be 6.

Table 1. Example values for graph topological characteristics used by the distance measure (Eq. 5) for the graphs shown in Fig. 4

	G_1	**G_2**	**G_3**	**G_4**	**G_5**	**G_6**	**G_7**	**G_8**
Nodes	7	7	7	7	7	6	8	8
Edges	9	9	9	8	9	7	10	10
Clustering Coefficient	0.886	0.886	0.886	0.827	0.707	0.84	0.802	0.695
Avg. Degree neighbors	2.0	2.0	2.0	1.67	2.0	1.80	1.86	1.86
Std. Dev. degrees neighbors	0	0	0	0.516	0.894	0.447	0.690	0.899

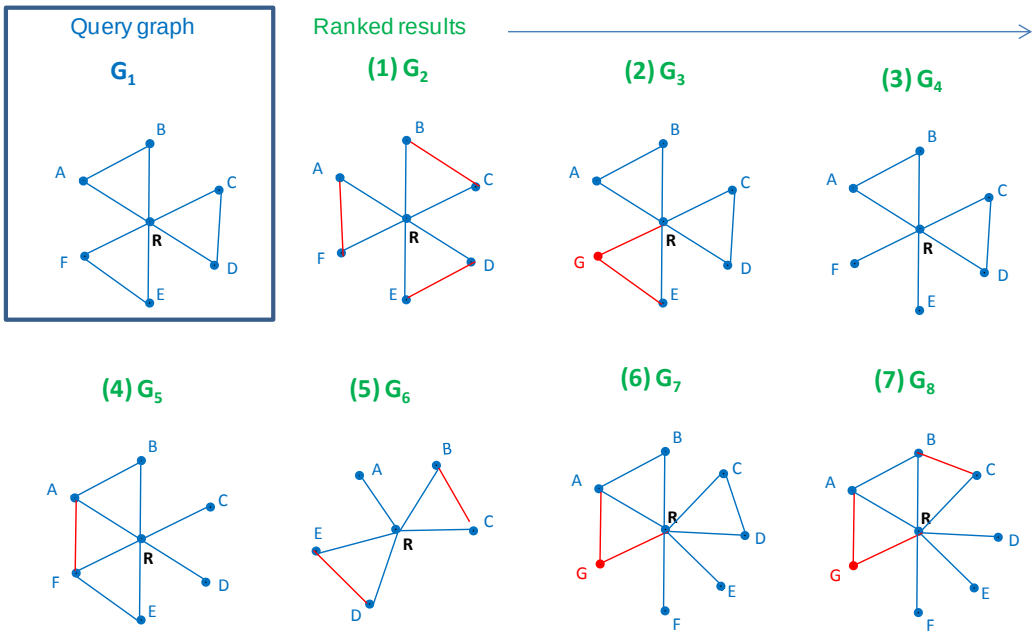

Fig. 4. Example of a sub-graph query (G_1) and returned results (G_2 to G_8) ordered by distance measure (Eq. 5).

Table 2. Distance measure results for the graphs of Fig. 4 and feature values of Table 1.

Graph	Eq. (5) distance* to G_1	Eq. (6) distance to G_1	GED distance to G_1
G_2	0.00	6	1.00
G_3	0.00	4	0.67
G_4	1.17	1	0.17
G_5	1.40	2	0.33
G_6	1.68	6	1.00
G_7	1.70	3	0.50
G_8	2.18	5	0.83

*Using initial weights of {3, 3, 2, 1, 1}

In order to calculate the distance (Eq. 5), first all the values in Table 1 are normalized, then an initial weighting vector {3, 3, 2, 1, 1} is applied, which produces the values in the first column of Table 2. The initial weighting vector can have an arbitrary assignment, or it can be assigned from some previous statistical analysis. The initial weight assignments will then be refined by the simulated annealing optimization process, which will be commented in the next section. Then the edge pair distance is calculated using Eq. 6 as described previously, and the values are normalized, which gives column 2 of Table 2. It can be seen that the first two graphs, G_2 and G_3, give a distance of zero ranked uniquely on topology, whereas the lowest label pair distances correspond to graphs G_4, G_5, G_7, G_3, G_8, G_2 and G_6, in that order. However, if the set size of results returned by Eq. 5 is progressively extended, the majority of the top graphs ranked by GED will be found among them. By training the weights of Eq. 5 (using a simulated annealing process) to find the same subset of similar graphs to the GED function, a closer fit is obtained of the results set of Eq. 5 to that of GED. It is recalled that Eq. 5 is used as a first step to reduce the search space and then Eq. 6 is used to produce the final ranking. Hence, it is finally the set overlap (Eq. 6, col. 2 of Table 2) which measures the fitness, and not the ordering (rank position) of the results. From columns 2 and 3 of Table 2 a good fit can be seen between the set overlap and GED.

4.3 Data Processing Scheme

The data processing scheme comprises three main steps: pre-calculations, train and apply.

Pre-calculations: In the pre-calculations step the sub-graph statistics and the GED distance between all sub-graphs are calculated. It is observed that GED is calculated for all sub-graphs for purposes of runtime benchmarking (see Sections 5 and 6). Otherwise, it is only necessary to calculate the GED for the sample vertex set S (input to the train step). For each vertex v_i, the reference vertices (v_j, v_{j+1}, …) of the k sub-graphs which are closest to v_i's sub-graph are assigned as a vector. The pseudo-code of this process can be seen in the Annex Section as 'Procedure 1'.

Train: The '*A-SGM*' matcher distance metric is defined in terms of a set of descriptive statistical features each of which has a corresponding weight to ponder its contribution to the overall distance result. The weights are optimized for each dataset by a process which uses the simulated annealing technique. The fitness function used for the optimization technique matches the result obtained by the '*A-SGM*' method against the 'ideal' result obtained by the GED matcher, as described in the previous section. Training is performed using subsets of randomly assigned sample nodes. The sub-graph around each reference (sample) node is then used as a graph search query and the top k results are found. The use of subsets of sample nodes greatly reduces the computational cost. The best weight assignments are found by executing *n* training runs (n = 5). The pseudo-code of this process can be seen in the Annex Section as 'Procedure 2' and 'Procedure 3'.

Apply: Once the feature weights have been optimized in the 'Train' step, the A-SGM distance measure is calculated for different sub-graph queries against the whole graph to find the top k results. The GED method (the gold standard) is also run, and two state of the art benchmark comparison methods (see Section 5.2). The pseudo-code of this process can be seen in the Annex Section as 'Procedure 4'.

5. EXPERIMENTAL SETUP

The experimental setup will now be described in terms of the datasets, the benchmark methods and the hardware/software used.

5.1 Datasets

The datasets chosen have been frequently used by other authors of sub-graph matching methods in the state of the art literature. The real datasets are: DBLP[20], downloaded from https://snap.stanford.edu/data/com-DBLP.html; CondMat[21], downloaded from https://snap.stanford.edu/data/ca-CondMat.html; and Protein[22], downloaded from http://vlado.fmf.uni-lj.si/pub/networks/data/. In Table 3 the basic statistics are shown for these graph datasets. The synthetic datasets, generated using a Java implementation of RMAT[23], had the following configurations {vertices, edges}: {10k, 100k}, {100k, 1M}, {1M, 3M}.

Table 3. Summary of graph statistics for the test datasets

	#Vertices	#Edges
Protein	2361	7182
Cond-mat	23133	93497
DBLP	317080	1049866
Rmat1	10K	100K
Rmat2	100K	1M
Rmat3	1M	3M

5.2 Benchmark methods

Two benchmark methods, Tale[10] and Ness[16], are used which represent two recent state of the art approximate sub-graph matching methods, which were described in Section 2. The software of these two systems was kindly supplied by the authors, upon request.

The Tale system required the following software and systems to be installed and configured: Linux OS; C++; PostgreSQL (http://www.postgresql.org/); LEDA (Library of Efficient Data types and Algorithms), a C++ class library ; GUESS (http://graphexploration.cond.org/); Java Version 6. PostgreSQL is used to store and index graph data, LEDA is used for a bipartite graph matching computation, GUESS is used for visualizing the graphs and Java 6 is used for running the graphical user interface.

The Ness system required the following software and systems to be installed and configured: Fedora Linux OS release 8 (Werewolf); C++, gcc; LEDA.

GED[5] is used as the 'gold standard' method for calculating the "ground truth" quality of the results for precision/recall, and for training ASGM. That is, for a given sub-graph query it returns the top *k* sub-graphs which are the closest match (smallest distance) to being isomorphisms of the sub-graph query. The Java code of the GED (Graph Edit Distance Finder) was adapted from the online source https://code.google.com/p/ged-finder/, authored by Roman Tekhov. It is observed that the cost for all types of edit operations (vertex addition/deletion, edge addition/deletion edge, and so on) is set to 1.

The ASGM matcher requires Eclipse Standard/SDK Kepler Service Release 2 (2014), Java Version 7, PostgreSQL for storing and indexing the pre-calculations and GED. The following parameters were assigned for the simulating annealing procedure: initial temperature, 1.e+6; cooling rate, 20; iterations, 20; tolerance, 1.e-5. The type of simulated annealing was downhill simplex.

5.3 Hardware

The hardware used is a PC with an Intel quad-Core i5-3470S processor at 2.9Ghz and 4Gb of RAM. The software used is Eclipse Kepler R2 with Java SE 7 and Windows / 7 (32 bits).

6. EMPIRICAL TESTING AND RESULTS

In this Section the performance results are presented in terms of computational cost for the training phase and precision/recall and computational cost for the apply phase. The ASGM method is benchmarked against the comparison methods Tale[10] and Ness[16] for six datasets: three real world and three synthetic. It is observed that the GED matcher is used as the gold standard for precision/recall, as described previously. An effort has been made to replicate equal test conditions in terms of hardware and software, although Tale and Ness use the LEDA library and ASGM does not, which may give the former methods some advantage.

6.1 Training Phase

The results of elapsed time are shown for different Rmat generated graph datasets and the real datasets. In the training phase cost, the corresponding index building times are included for Tale and Ness. In the case of ASGM, the pre-calculations and the simulated annealing calibration of the attribute weights are taken into account. In Fig. 5 the datasets have been grouped into real datasets (on the left) and synthetic datasets (on the right). Also, within each group they have been arranged, from left to right on the *x*-axis, in approximate order of size and complexity (that is the number of vertices and edges, see Table 3). The *y*-axis uses a logarithmic scale given the large difference in processing time between graph datasets. From this, it can be seen that ASGM and NESS consume a similar amount of training time, with ASGM being slightly faster, and TALE is consistently the fastest of the three.

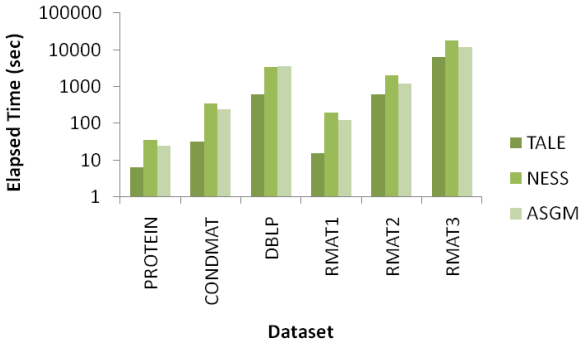

Fig. 5. Pre-processing times for methods: ASGM (calibration); Tale and Ness (index build)

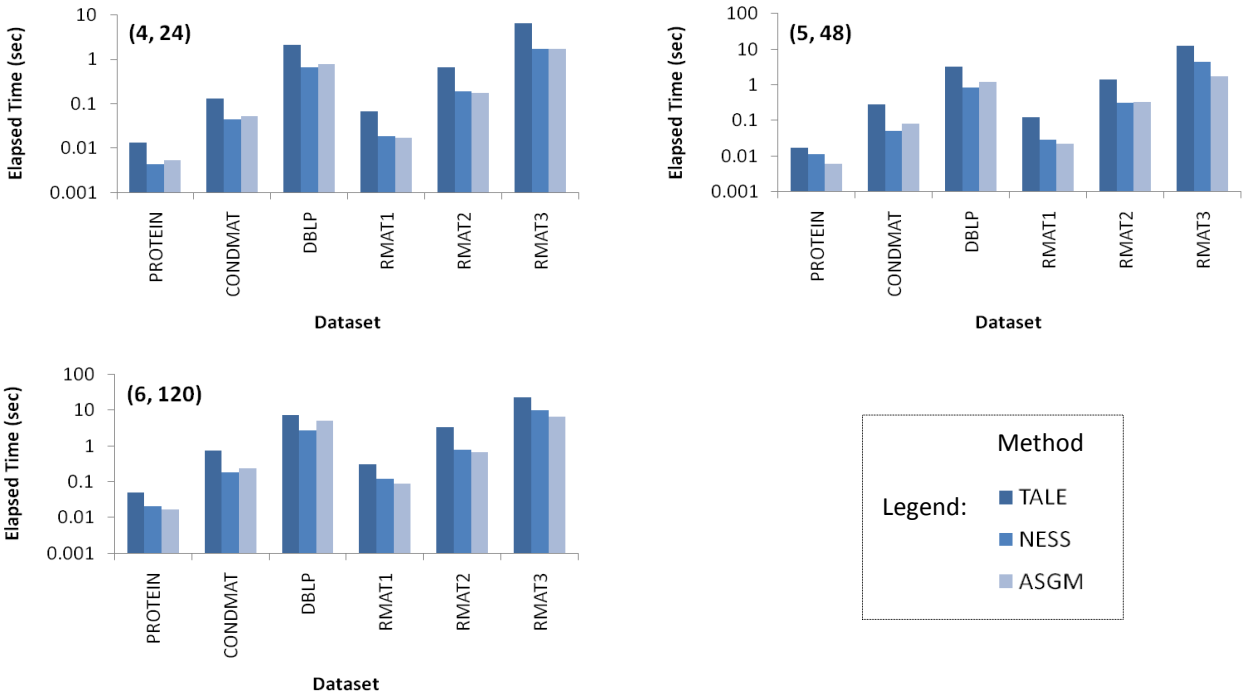

Fig. 6. Execution times for methods and sub-graph queries (vertices, edges): (a) (4, 24) (b) (5, 48) (c) (6, 120)

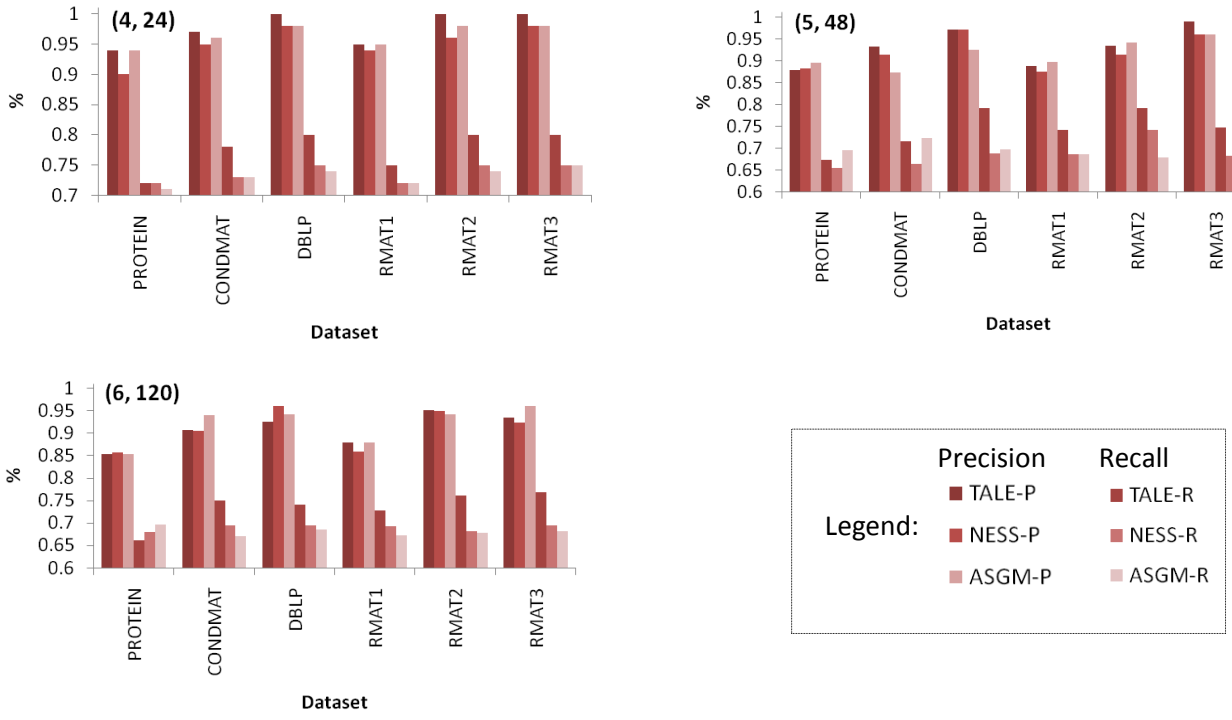

Fig. 7. Precision and Recall for methods and sub-graph queries (vertices, edges): (a) (4, 24) (b) (5, 48) (c) (6,120)

6.2 Apply Phase

In the following the results are shown in terms of elapsed execution time (Figs. 6), precision and recall (Figs. 7) for all methods and datasets. For the precision and recall, the relevant sub-graph set ('gold standard') is defined as the set recovered by the GED matcher. See Section 3 for a description of the quality metrics: precision, recall and F_1.

In Figs. 6 the runtime computation cost results are shown for three different queries, for each method and dataset. The datasets are organized on the *x*-axis in the same manner as previously for Fig. 5. Each query is designated by two numbers (V, E) which signify the number of vertices V and edges E in the sub-graph query. The three queries are (4, 24), (5, 48) and (6, 120). In Figs. 6 it can be seen that TALE consistently shows a higher runtime than the other two

methods. Ness and ASGM, on the other hand, display a very similar performance. It is observed that a logarithmic scale is used on the *y*-axis given the large difference in processing time between graph datasets.

In Figs. 7 the results are shown for the quality of the query responses for each query, method and dataset. The datasets are organized on the *x*-axis in the same manner as previously for Figs. 5 and 6. The quality is calculated as described previously, using the information retrieval metrics of precision and recall and GED as the 'gold standard' for the retrieved result set. In general, it can be seen that the precision is consistently higher than recall, which is to be expected and is consistent with other results in the literature. This is because, it is relatively easier that most of the returned results are relevant (precision) and more difficult that all the possible relevant results are returned (recall), with respect to a given threshold (see Section 3).

In terms of the methods, it can be seen that overall NESS has a slightly lower precision score than TALE and ASGM. TALE has stronger overall recall for the datasets, and better precision for the bigger datasets and the smaller query. ASGM in general has a recall which is similar to NESS, and precision which competes with TALE and NESS as the best result. However, each of the methods has specific queries and datasets in which they performed better/worse. This could suggest the possible utility of building an ensemble system integrating different methods, with a consensus as output.

7. SUMMARY AND CONCLUSIONS

A novel sub-graph matcher has been presented which employs an optimization process to calibrate sub-graph statistic weights and use them in a similarity distance. It has been shown to be competitive with state of the art methods in terms of precision, recall and computational cost in both the pre-calculation/training and execution phases. As future work, other types of data related to the graph could be incorporated into the similarity matching, such as categorical and numerical data associated with the nodes and edges. Also, an ensemble approach could be considered.

8. ACKNOWLEDGEMENTS

This research is partially supported by the Spanish MEC project HIPERGRAPH TIN2009-14560-C03-01.

9. REFERENCES

[1] Ewing, T.J.A, Kuntz I.D. 1997. Critical evaluation of search algorithms for automated molecular docking and database screening. J. Comput. Chem., 18(9) (1997) 1175-1189.

[2] Raymond, J.W., Willett, P. 2002. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. J. Computer-aided Molecular Design, 16(7) (2002) 521-533.

[3] Inokuchi, A., Washio, T., Motoda, H. Complete mining of frequent patterns from graphs: Mining graph data. Mach. Learn. 50(3) (2003) 321-354.

[4] Nettleton, D.F. 2013. Data Mining of Social Networks Represented as Graphs, Comp. Sci. Rev., 7 (2013) 1-34.

[5] Bunke, H. 1999. Error correcting graph matching: on the influence of the underlying cost function. IEEE Trans. Patt. Anal. Mach. Intell. 21, 917-922.

[6] Nettleton, D. F. and Dries, A. 2014. Local Neighbourhood Sub-Graph Matching Method. European Patent application number: 13382308.8. Presented: 30th July 2013. PCT application number: PCT/ES2014/065505. Presented 18th July 2014.

[7] Nettleton, D.F., Torra, V., Dries, A. 2014. A Comparison of Clustering and Modification based Graph Anonymization Methods with Constraints. International Journal of Computer Applications (0975 – 8887), Vol. 95– No.20, June 2014.

[8] McKay, B.D. 1981. Practical graph isomorphism, Congressus Numerantium 30, 45-87.

[9] Cordella, L.P., Foggia, P., Sansone, C. and Vento, M. 2001. An Improved Algorithm for Matching Large Graphs. In: Proc. 3rd IAPR-TC-15 Int. Workshop on Graph based Representations, pp. 149-159.

[10] Tian, Y. and Patel, J.M. 2008. TALE: A Tool for Approximate Large Graph Matching. In: Proc. IEEE ICDE 2008, 24th Int. Conf. on Data Engineering, pp. 963-972.

[11] Wei, F. 2010. Tedi: Efficient shortest path query answering on graphs. In: Proc. SIGMOD 2010, pp. 99–110.

[12] Zou, L., Chen, L., Özsu, M.T. and Zhao, D. 2012. Answering pattern match queries in large graph databases via graph embedding. J. VLDB 2012; 21(1): 97-120:

[13] Zhang, S., Yang, J. and Jin, W. 2010. SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. In: Proc. VLDB, 3(1), 2010.

[14] Zhao, P. and Jiawei Han, J. 2010. On Graph Query Optimization in Large Networks. In: Proc. VLDB 3(1): pp. 340-351.

[15] He, H. and Singh, A.K. 2008. Graphs-at-a-time: query language and access methods for graph databases. In: Proc. SIGMOD'08, pp. 405-418.

[16] Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S. and Tao, S. 2011. Neighborhood based fast graph search in large networks. SIGMOD '11.

[17] Sun, Z., Wang, H., Wang, H., Shao, B. and Li, J. 2012. Efficient subgraph matching on billion node graphs. In: Proc. VLDB, 5(9):788-799.

[18] Zhang, S., Li, S. and Yang, J. 2009. Gaddi: distance index based subgraph matching in biological networks. In: Proc. EDBT '09, 12th Int. Conf. on Extending Database Technology: Advances in Database Technology, pp. 192-203.

[19] Walters, W.H. Comparative Recall and Precision of Simple and Expert Searches in Google Scholar and Eight Other Databases. 2011. The Johns Hopkins University Press, Baltimore. Portal: Libraries and the Academy, vol. 11, no. 4, pp. 971–1006.

[20] Yang, J. and Leskovec, J. 2012. Defining and Evaluating Network Communities based on Ground-truth. In: Proc. ICDM 2012, pp. 745-754.

[21] Leskovec, J., Kleinberg, J. and Faloutsos, C. 2007. Graph Evolution: Densification and Shrinking Diameters. ACM Trans. on Knowledge Discovery from Data (ACM TKDD), 1(1).

[22] Sun, S., Ling, L., Zhang, N., Li, G. and Chen, R. 2003. Topological structure analysis of the protein-protein interaction network in budding yeast. Nucleic Acids Research, Vol. 31, No. 9 2443-2450.

[23] Chakrabarti, D., Zhan, Y. and Faloutsos, C. 2004. R-mat: A recursive model for graph mining. In Proc. SDM (Secure Data Management), Orlando, Florida, USA, pp. 442-446.

10. APPENDIX

PROCEDURE 1 **PreCalculations**(graph *G*)
Input: complete graph G = (V, E)
Output: graph statistics for each sub-graph, assign closest sub-graphs using GED

1. FOR each (v) ∈ (*G*)
2. Define SG_v as the sub-graph for vertex v.
3. For SG_v calculate the following statistics: degree of reference vertex, clustering coefficient, number of edges, average degree of neighbors of reference vertex, standard deviation of degrees of reference vertex.
4. FOR each (v') ∈ (*G*), v' ≠ v
5. Calculate the GED between SG_v and $SG_{v'}$ as $GED_{vv'}$
6. END FOR
7. Assign to vertex v, the *k* sub-graphs $SG_{v'}$ which are closest to SG_v*
8. END FOR
9. Normalize all statistics

END PROCEDURE
*GED is calculated for all sub-graphs for purposes of runtime benchmarking. Otherwise, it would only be necessary to calculate the GED for the sample vertex set S.

PROCEDURE 2 **Train**(graph *G*, sample vertices *S*)
Input: complete graph G, sample vertices S
Output: optimal weights for Equation (5)

1. WHILE **optimum fitness** NOT FOUND or **maxiterations**
2. Run simulated annealing
3. Weights assigned by simulated annealing
4. Execute **fitness function** with weights assigned by simulated annealing, for sample vertices S on graph G
5. Simulated annealing evaluates if **optimum fitness**.
6. END WHILE
7. RETURN optimum weight assignments

END PROCEDURE

PROCEDURE 3 **Fitness**(graph *G*, sample vertices *S*, weights *W*)
Input: complete graph *G*, sample vertices *S*, current weights *W*
Output: fitness value F1 for current weights *W*

1. FOR each (s) ∈ (*S*)
2. Define SG_s as the sub-graph for s
3. Find top k sub-graphs L_E corresponding to vertices *i* in G identified by *GED* as being minimum distance to SG_s
4. Find top k sub-graphs L_U, corresponding to vertices *i* in G whose *A-SGM* distance is closest to that of SG_s.
5. Calculate fitness for s by calculating F1 measure for sets L_E and L_U
6. END FOR
7. RETURN average fitness

END PROCEDURE

PROCEDURE 4 **Apply**(graph *G*, sub-graph query *Q*, optimum weights *W'*)
Input: complete graph *G*, sub-graph query *Q*, optimum weights *W'*
Output: returned sub-graphs set corresponding to top matched sub-graphs, precision statistics

1. Find top *k* sub-graphs L_E corresponding to vertices *i* in G identified by *GED* as being minimum distance to *Q*
2. Find top *k* sub-graphs L_U, corresponding to vertices *i* in G whose *A-SGM* distance is closest to that of *Q*
3. Calculate precision and recall for sets L_E and L_U
4. RETURN precision and recall for query *Q*; top *k* sub-graphs L_E

END PROCEDURE