On Speeding up Frequent Approximate Subgraph Mining

Niusvel Acosta-Mendoza, Andrés Gago-Alonso, and José E. Medina-Pagola

Advanced Technologies Application Center (CENATAV), Havana, Cuba.
{nacosta,agago,jmedina}@cenatav.co.cu

Abstract. Frequent approximate subgraph (FAS) mining has become an interesting task with wide applications in several domains of science. Most of the previous studies have been focused on reducing the search space or the number of canonical form (CF) tests. CF-tests are commonly used for duplicate detection; however, these tests affect the efficiency of mining process because they have high computational complexity. In this paper, two prunes are proposed, which allow decreasing the label space, the number of candidates and the number of CF-tests. The proposed prunes are already used and validated in two reported FAS miners by speeding up their mining processes in artificial graph collections.

Keywords: Approximate graph mining, approximate graph matching, frequent approximate subgraphs, labeled graphs.

1 Introduction

In recent years, the necessity to use approximate matching in graph mining tasks has increased [7]. In fact, there are concrete problems where exact matching could not be applicable with positive outcome [5]. Sometimes, interesting subgraphs show slight differences throughout datasets. This means that we should tolerate certain level of geometric distortion, slight semantic variations, vertices or edges mismatch in frequent pattern search. For this reason, it is required to evaluate the similarity between graphs allowing some structural differences, i.e. considering approximate matching.

Taking into account these facts, several algorithms have been developed for FAS mining which use different approximate graph matching techniques in different domains of science [1, 4–6, 8–11]. However, only *APGM* [6] and *VEAM* [1] detect the FASs in a graph collection, where the approximation consists in considering some variations of the data through the substitution probability, keeping the topology of the graphs. These algorithms specify which vertices, edges or labels can replace others. Thus, the idea that not always a vertex label or an edge label can be replaced by any other is defended. APGM only deals with the variations between the vertex labels while VEAM perform the mining process using the vertex and edge label sets.

On the other hand, candidates are represented by a unique code called CF for frequency counting. In order to obtain the occurrences of the candidates are performed CF-tests to each of them. However, the computational complexity of a CF-test is very high [3]. In this paper, we introduce several prunes for improving FAS mining in an approximate framework related to APGM and VEAM approaches. Using these prunes it is possible to reduce the label space and the search space, as well as number of candidates. This fact also allows us to reduce the number of CF-tests.

The basic outline of this paper is as follows. Section 2 provides some basic concepts. The prunes for speeding up approximate methods are provided in section 3. The experimental results are discussed in section 4. Finally, conclusions of the research and some ideas about future directions are exposed in Section 5.

2 Basic Concepts

This work is focused on simple undirected labeled graphs. Henceforth, when we refer to graph we assume this type of graph. Before presenting their formal definition, we define the domain of labels.

Let L_V and L_E be label sets, where L_V is a set of vertex labels and L_E is a set of edge labels. The domain of all possible labels is denoted by $L = L_V \cup L_E$.

A *labeled graph* in L is a 4-tuple, $G = (V, E, I, J)$, where V is a set whose elements are called *vertices*, $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ is a set whose elements are called *edges* (the edge $\{u, v\}$ connecting the vertex u with the vertex v), $I : V \to L_V$ is a *labeling function* for assigning labels to vertices and $J : E \to L_E$ is a *labeling function* for assigning labels to edges.

Let $G_1 = (V_1, E_1, I_1, J_1)$ and $G_2 = (V_2, E_2, I_2, J_2)$ be two graphs, we say that G_1 is a *subgraph* of G_2 if $V_1 \subseteq V_2$, $E_1 \subseteq E_2$, $\forall u \in V_1, I_1(u) = I_2(u)$, and $\forall e \in E_1, J_1(e) = J_2(e)$. In this case, we use the notation $G_1 \subseteq G_2$ and we say that G_2 is a *supergraph* of G_1.

Given two graphs $G_1 = (V_1, E_1, I_1, J_1)$ and $G_2 = (V_2, E_2, I_2, J_2)$, where $G_1 \subseteq G_2$, we say that $e = \{u, v\} \in E_2$ is an *extension* of G_1 if: $V_2 = V_1 \cup \{v\}$ and $E_1 = E_2 \setminus \{e\}$. This fact can be denoted by $G_2 = G_1 \diamond e$. We say that e is a *backward extension* if $v \in V_1$, otherwise we say that it is a *forward extension* (it extends the vertex set of G_1).

Given G_1 and G_2, we say that f is an *isomorphism* between these graphs if $f : V_1 \to V_2$ is a bijective function, where $\forall u \in V_1, f(u) \in V_2 \wedge I_1(u) = I_2(f(u))$ and $\forall \{u, v\} \in E_1, \{f(u), f(v)\} \in E_2 \wedge J_1(\{u, v\}) = J_2(\{f(u), f(v)\})$. When there is an isomorphism between G_1 and G_2, we say that G_1 and G_2 are *isomorphic.*

Let Ω be the set of all possible labeled graphs in L, the *similarity* between two elements $G_1, G_2 \in \Omega$ is defined as a function $sim : \Omega \times \Omega \to [0, 1]$. We say that the elements are very different if $sim(G_1, G_2) = 0$, the higher the value of $sim(G_1, G_2)$ the more similar the elements are and if $sim(G_1, G_2) = 1$ then there is an isomorphism between these elements.

Let $G_1 = (V_1, E_1, I_1, J_1)$, $G_2 = (V_2, E_2, I_2, J_2)$ and $T = (V_T, E_T, I_T, J_T)$ be three labeled graphs in L, where $T \subseteq G_2$. Using an isomorphism threshold τ, we say that T is an *embedding* of G_1 in G_2 if $sim(G_1, T) \geq \tau$. The *embedding set* of G_1 in G_2 is denoted by $O(G_1, G_2)$.

Let T be an embedding of G_1 in $G_2 = (V_2, E_2, I_2, J_2)$, using an isomorphism threshold τ. The *extension set* of T is denoted by $ExtSet(T) = \{e \in E_2 \mid e$ is an extension of $T\}$.

Let $D = \{G_1, \ldots, G_{|D|}\}$ be a graph collection and let G be a labeled graph in L, the *support* value of G in D is obtained through the following equation:

$$supp(G, D) = \sum_{G_i \in D} sim(G, G_i)/|D| \tag{1}$$

If $supp(G, D) \geq \delta$, then the graph G is approximately frequent in the collection D, saying that G is a *FAS* in D. Notice that when we refer to a graph collection we assume that it is the representation built from a real graph collection. The value of the support threshold δ is in $[0, 1]$ assuming that the similarity is normalized to 1. *FAS mining* consists in finding all the FASs in a collection of graphs D, using a similarity function *sim* and a support threshold δ.

3 Prunes for Mining

In this section, we introduce two prunes for FAS mining. These prunes can be used in any FAS miner which uses substitution matrices and they are based on the downward closure property [2]. VEAM and APGM are used as basis for showing the improves.

The candidate generation process consists of extending a subgraph pattern by an edge. This is done searching the possible approximate label set of the new edge e and for every possible label of e seeking the possible label set of the new vertex if e is a forward extension. Finally, candidates are generated using those labels that satisfy definitions *"Vertex Approximate Sub-isomorphism*[1]*"* and *"Approximate Sub-isomorphism*[2]*"* presented by Acosta-Mendoza *et al.* [1]. In the above mentioned label sets, there are labels that could replace another ones with a similarity less than the threshold τ. Using these labels it is not possible to obtain candidates; however, the algorithms lose time checking these extensions. To formalize the previous assertions the following definition is presented.

Definition 1 (Useful label set). *Let $l_v \in L_V$, $l_e \in L_E$ be a vertex label and an edge label respectively, we say that the label sets, for a vertex label l_v and an edge label l_e, are useful label sets if they are obtained by the functions $U_V^\tau : L_V \to P_{L_V}$ and $U_E^\tau : L_E \to P_{L_E}$ such that:*

- $U_V^\tau(l_v) = \{l \in L_V | \frac{MV_{l,l_v}}{MV_{l,l}} \geq \tau\}$,
- $U_E^\tau(l_e) = \{l \in L_E | \frac{ME_{l,l_e}}{ME_{l,l}} \geq \tau\}$;

where, P_{L_V} and P_{L_E} are the power sets of L_V and L_E respectively[3].

[1] Definition used as similarity function in APGM
[2] Definition used as similarity function in VEAM
[3] For a set X, the power set of X is $P_X = \{Y | Y \subseteq X\}$.

Notice that, for each $l_v \in L_V$, $U_V^\tau(l_v) \neq \emptyset$, since at least $l_v \in U_V^\tau(l_v)$. In the same way, for each $l_e \in L_E$, $U_E^\tau(l_e) \neq \emptyset$, since at least $l_e \in U_E^\tau(l_e)$.

Theorem 2. *Let G be a labeled graph, let MV and ME be a substitution matrix indexed by vertex labels and a substitution matrix indexed by edge labels respectively; let v, v' be two vertices with labels l_v and $l_{v'}$ respectively, and let $e = \{u, v\}, e' = \{u, v'\}$ be two edges with labels l_e and $l_{e'}$ respectively. Then the following statements are true:*

1. *If $l_{v'} \notin U_V^\tau(l_v)$, then $G \diamond e' \neq_A G \diamond e$ (in the same way $G \diamond e' \neq_a G \diamond e$).*
2. *If $l_{e'} \notin U_E^\tau(l_e)$, then $G \diamond e' \neq_A G \diamond e$.*

Proof. In these statements, the definitions of approximate sub-isomorphism of APGM and VEAM are used. These definitions use a product of substitution probabilities with the requirement that its result should be greater than or equal to τ. These substitution probability values are in the interval $[0, 1]$. Therefore, if a factor of this product is lesser than τ then its value is less than τ too and does not generate an approximate subgraph candidate. $\square$

On the other hand, in APGM and VEAM, the approximation is based on the semantic of labels. For this reason, we proposed a heuristic to reduce the search space, according to some rules that can be applied by these FAS miners. These rules are: (1) The vertices (and its corresponding edges), which are not used as an embedding of a single-vertex FAS, are removed from each graph in D; (2) The edges, which are not used as an embedding of a single-edge FAS, are removed from each graph in D.

Theorem 3. *Let $L_U^V \subseteq L_V$ be the label set used by the single-vertex FASs in a collection D and u be a vertex with label $l_u \in L_V$. The following statements are true:*

1. *The vertex u is not used as an embedding by any single-vertex FAS if and only if $L_U^V \cap U_V^\tau(l_u) = \emptyset$.*
2. *If $L_U^V \cap U_V^\tau(l_u) = \emptyset$ then a non-frequent subgraph is obtained when $e = \{v, u\}$ extends an existing FAS G in D, where u is the new vertex of this forward extension.*

Proof. First, we will prove statement 1. Suppose that the vertex u is used as an embedding by a single-vertex FAS $Y = (\{w\}, \emptyset, I_Y, J_Y)$. If this occurs, then the label $l_w \in L_U^V$ of the vertex w replaces the label l_u with a value greater than or equal to τ, i.e. $l_w \in U_V^\tau(l_u)$, and $L_U^V \cap U_V^\tau(l_u) \neq \emptyset$. Therefore, if $L_U^V \cap U_V^\tau(l_u) = \emptyset$, then the vertex u is not used as an embedding by any single-vertex FAS. On the other hand, assuming that $L_U^V \cap U_V^\tau(l_u) \neq \emptyset$, then u is not used as an embedding by any single-vertex FAS because their labels replace l_u with values less than τ. Therefore, if u is used as an embedding by a single-vertex FAS, then $L_U^V \cap U_V^\tau(l_u) = \emptyset$.

Next, we will prove the second statement. Suppose that we have a forward extension $e = \{v, u\}$, with u as the new vertex, of an existing FAS G in D.

If $L_U^V \cap U_V^\tau(l_u) = \emptyset$ then the vertex u is not used as an embedding by any single-vertex FAS, according to the first part of the Theorem and, as a directly consequence, the single-vertex subgraph $T = (\{u\}, \emptyset, I_T, J_T)$ is non-frequent. As $T \subseteq_A G \diamond e$, applying the downward closure, we have $supp(G \diamond e, D) \leq supp(T, D)$, then $G \diamond e$ is a non-frequent pattern because T is a non-frequent. Therefore, to prune the vertices with label l_u of D does not change the frequent pattern output. $\square$

Theorem 4. *Let $L_U^E \subseteq L_E$ be the label set used by the single-edge FASs in a collection D and an edge e with label $l_e \in L_E$, the following statements are true:*

1. *The edge e is not used as an embedding by any single-edge FAS if and only if $L_U^E \cap U_E^\tau(l_e) = \emptyset$.*
2. *If $L_U^E \cap U_E^\tau(l_e) = \emptyset$ then a non-frequent subgraph is obtained if e extends a FAS G in D.*

Proof. First, we will prove statement 1. Suppose that the edge e is used as an embedding by a single-edge FAS $Y = (V_Y, \{e'\}, I_Y, J_Y)$. If this occurs, then the label $l_{e'} \in L_U^E$ of the edge e' replaces the label l_e with a value greater than or equal to τ, i.e. $l_{e'} \in U_V^\tau(l_e)$, and $L_U^E \cap U_E^\tau(l_e) \neq \emptyset$. Therefore, if $L_U^E \cap U_E^\tau(l_e) = \emptyset$, then the edge e is not used as an embedding by any single-edge FAS. On the other hand, assuming that $L_U^E \cap U_E^\tau(l_e) \neq \emptyset$, then e is not used as an embedding by any single-edge FAS because their labels replace l_e with values less than τ. Therefore, if e is used as an embedding by a single-edge FAS, then $L_U^E \cap U_E^\tau(l_e) = \emptyset$.

Next, we will prove the second statement. Suppose that we have an extension e of an existing FAS G in D. If $L_U^E \cap U_E^\tau(l_e) = \emptyset$ then the edge e is not used as an embedding by any single-edge FAS, according to the first part of the Theorem and, as a directly consequence, the single-edge subgraph T with l_e as edge label is non-frequent. As $T \subseteq_A G \diamond e$, applying the downward closure, we have $supp(G \diamond e, D) \leq supp(T, D)$, then $G \diamond e$ is a non-frequent pattern because T is a non-frequent. Therefore, to prune the edges with label l_e of D does not change the frequent pattern output. $\square$

VEAM and APGM are shown through three pseudo-codes where the common main algorithm consists in finding the single-vertex FAS and single-edge FAS sets, then the first set is stored in set F and only the second one is stored in set C (see Algorithm 1 in Figure 1). A prune using Theorem 3 as basis is performed in line 2. After obtaining the single-edge FAS set, a prune using Theorem 4 as basis is performed in line 5. As we can see, in order to apply the last prune, a breadth-first search (BFS) is required to obtain the approximate frequent edges in the initial process. Later, for each pattern in C the algorithm *Search* is invoked. When all single-edge FASs in C have been extended, then the set F of all FASs in D is returned.

The other common pseudo-code of VEAM and APGM performs the extension of subgraph patterns on one edge using DFS (see Algorithm 2 in Figre 1). Thus, all candidate subgraphs are created using the label set obtained through the "*appLSetVEAM*" algorithm, in the VEAM case, or "*appLSetAPGM*" algorithm, in the APGM case, in line 2. In Algorithm 3 (see lines 1 and 4) and in Algorithm 4 (see line 3), a prune using theorem 2 as basis is included, where only the labels in $U_E^\tau(J(e))$ and $U_V^\tau(I(v))$ are used.

```
Algorithm 1: Main
  Input: $D$ : A graph collection, $MV$ : Substitution matrix indexed by $L_V$,
         $ME$ : Substitution matrix indexed by $L_E$, $\tau$ : Isomorphism
         threshold, $\delta$ : Support threshold.
  Output: $F$ : Frequent approximate subgraph set.
1 $F \leftarrow$ the frequent approximate single-vertex set in $D$;
2 Remove from $D$ the vertices with label $l_v$ such that $L_U^V \cap U_V^\tau(l_v) = \emptyset$;
3 $C \leftarrow$ the frequent approximate single-edge set in $D$;
4 $F \leftarrow F \cup C$;
5 Remove from $D$ the edges with label $l_e$ such that $L_U^E \cap U_E^\tau(l_e) = \emptyset$;
6 foreach $T \in C$ do
7     Search($T, D, MV, ME, \tau, \delta, F$);
```

```
Algorithm 2: Search
  Input: $T = (V_t, E_t, I_t, J_t)$ : A frequent approximate subgraph, $D$ : Graph
         collection, $MV$ : Substitution matrix indexed by $L_V$, $ME$ :
         Substitution matrix indexed by $L_E$, $\tau$ : Isomorphism threshold,
         $\delta$ : Support threshold, $F$ : Frequent approximate subgraph set.
  Output: $F$ : Frequent approximate subgraph set.
1  foreach $o_j \in O(T, G_i)$, where $G_i \in D$ do
2      foreach $e = ExtSet(o_j)$ do
3          $CL \leftarrow$ appLSet($T, MV, ME, G_i, o_j, e, \tau$);
4          foreach $(elabel, vlabel) \in CL$ do
5              The candidate $X$ is built using the tuple $(elabel, vlabel)$;
6              The code CAM of $X$ is computed and stored in $codeCAM(X)$;
7              $C \leftarrow C \cup \{(X, codeCAM(X), score)\}$;
8  foreach $T_1 \in C$ do
9      if $sup_G(T_1, D) \geq \delta$ and $codeCAM(T_1) \notin F$ then
10         Insert $T_1$ in $F$;
11         Search($T_1, D, MV, ME, \tau, \delta, F$);
```

```
Algorithm 4: appLSetAPGM
  Input: $T$ : A candidate graph, $MV$ : Substitution matrix indexed by $L_V$,
         $G = (V, E, I, J)$ : A graph of the collection, $G'$ : Embedding of $T$
         in $G$, $e = \{u, v\}$ : An extension of $G'$, $\tau$ : Isomorphism threshold.
  Output: $CL$ : A set of candidate 2-tuples $(elabel, vlabel)$.
1 $l_{uv} = J(e)$;
2 if $e$ is a forward extension of $G'$ then
3     foreach $i \in U_V^\tau(I(v))$ do
4         $score \leftarrow S_{max}(T, G') * \frac{MV_{i,I(v)}}{MV_{i,i}}$;
5         if $score \geq \tau$ then $CL \leftarrow CL \cup \{(l_{uv}, i)\}$;
6 else $CL \leftarrow CL \cup \{(l_{uv}, \emptyset)\}$;
```

```
Algorithm 3: appLSetVEAM
  Input: $T$ : A candidate graph, $MV$ : Substitution matrix indexed by $L_V$,
         $ME$ : Substitution matrix indexed by $L_E$, $G = (V, E, I, J)$ : A
         graph of the collection, $G'$ : Embedding of $T$ in $G$, $e = \{u, v\}$ :
         An extension of $G'$, $\tau$ : Isomorphism threshold.
  Output: $CL$ : A set of candidate 2-tuples $(elabel, vlabel)$.
1 foreach $j \in U_E^\tau(J(e))$ do
2     $scoreE \leftarrow S_{max}(T, G') * \frac{ME_{j,J(e)}}{ME_{j,j}}$;
3     if $e$ is a forward extension of $G'$ then
4         foreach $i \in U_V^\tau(I(v))$ do
5             if $i$ is less than or equal to the largest of the vertex labels of $T$
              then
6                 $score \leftarrow scoreE * \frac{MV_{i,I(v)}}{MV_{i,i}}$;
7                 if $score \geq \tau$ then $CL \leftarrow CL \cup \{(j, i)\}$;
8     else if $scoreE \geq \tau$ then $CL \leftarrow CL \cup \{(j, \emptyset)\}$;
```

Fig. 1. Pseudo-code of VEAM and APGM using the proposed prunes

4 Experimental Results

All our experiments were carried out using a personal computer ($x64$ platform) Intel (R) Core (TM) 2 Quad CPU $Q9450$ @ 2.66 GHz with 4 Gb main memory. We used ANSI C language and we compiled the algorithms using gcc compiler of GNU/Linux.

In this paper, the mining process is performed in the several image collections used by Acosta-Mendoza *et al.* [1]. The impact of our prunes in FAS mining on image collections is evaluated with the aim of showing the usefulness of these prunes. The performance of VEAM and APGM[4] with and without our prunes in several collections are compared as follows. Note that in Table 1 are shown only the results of the algorithms in two collections ($D700$ and $D600$) due to space restrictions.

First, VEAM and APGM with and without the prunes proposed are compared regarding the number of exhaustive CF-tests that they perform in their mining strategies (see subtable a) in Table 1). Note that the original algorithms are denoted by VEAM and APGM, and the algorithms with our prunes are denoted by *VEAMwP* and *APGMwP*. These CF-tests become much more complex to great extent with the increase of the candidate size. In this comparison, the number

[4] The APGM version used in our experiments was implemented by us to detect all frequent vertex approximate subgraphs and not just the clique subgraphs as proposed [6].

of such expensive CF-tests, in most of the cases, were reduced in 30% when our prunes are used. The fact that these prunes reduce the number of candidates to be processed impacts in a positive manner in the algorithm's performance.

Second, the performance, in terms of runtime, of APGM and VEAM with and without the prunes proposed are compared. In this comparison the runtimes were reduced, in most of the cases, in 15%, when our prunes are used. Notice that we use the same support threshold and isomorphism threshold values presented by Acosta-Mendoza *et al.* [1] with the purpose of showing the improvement achieving the same accuracies obtained by them.

Table 1. Comparison between VEAM and VEAMwP, and between APGM and APGMwP in image collections using $\tau = 0.4$

a) Number of CF-tests computed

	Support (δ)								
Algorithm	**20%**	**25%**	**30%**	**35%**	**40%**	**45%**	**50%**	**55%**	**60%**
				D700					
APGM	153524	48437	11927	3555	1407	992	992	904	829
APGMwP	**67197**	**21709**	**4478**	**1131**	**303**	**187**	**187**	**180**	**174**
				D600					
APGM	58300	24453	8278	2515	1081	963	963	883	808
APGMwP	**28950**	**12133**	**3245**	**848**	**257**	**186**	**186**	**179**	**173**
				D700					
VEAM	350589	114907	33423	11105	5212	2600	2302	2118	1687
VEAMwP	**259034**	**67405**	**17264**	**5505**	**2522**	**477**	**453**	**428**	**371**
				D600					
VEAM	157751	79555	26160	8241	4374	2526	2236	2056	1641
VEAMwP	**113005**	**47699**	**13662**	**4167**	**2049**	**475**	**452**	**427**	**370**

b) Runtimes (s)

				D700					
APGM	58.61	9.95	2.13	0.76	0.42	0.36	0.35	0.33	0.30
APGMwP	**31.24**	**6.61**	**1.60**	**0.59**	**0.29**	**0.17**	**0.17**	**0.16**	**0.15**
				D600					
APGM	11.00	3.69	1.27	0.46	0.29	0.28	0.28	0.26	0.23
APGMwP	**7.24**	**2.71**	**0.96**	**0.36**	**0.19**	**0.13**	**0.13**	**0.12**	**0.12**
				D700					
VEAM	133.78	25.50	6.51	2.43	1.41	0.94	0.88	0.81	0.67
VEAMwP	**114.93**	**20.34**	**5.53**	**2.15**	**1.24**	**0.42**	**0.40**	**0.39**	**0.34**
				D600					
VEAM	29.91	12.69	4.19	1.57	0.98	0.72	0.68	0.63	0.52
VEAMwP	**25.47**	**10.37**	**3.53**	**1.34**	**0.83**	**0.33**	**0.30**	**0.29**	**0.26**

In summary, we can conclude that our prunes positively impact the performance of APGM and VEAM. The prune based on Theorem 2 is very effective when the number of dissimilarities between labels or the number of candidates are increased. The prune using Theorems 3 and 4 is considered effective when the collection has many non-frequent vertices and edges, and its graphs are dense. In general, the new prunes help us to reduce runtimes of VEAM and APGM. These results ensure the usefulness of the prunes proposed in this paper.

5 Conclusions and Future Work

In this paper, we introduced a prune which is useful to reduce the number of CF-tests during the FAS mining. This prune allows decreasing the search space of collection's graphs. Another prune, which uses only the label set that comply with the isomorphism threshold, was proposed. These prunes are implemented and tested in already reported FAS miners, APGM and VEAM. As can be seen in the experimental results, the number of CF-tests is notably reduced, and the improvement in time is appreciable.

As future work, we are going to develop new ways for taking advantage of the candidate reduction in order to achieve better performance in FAS mining. This allows us to accelerate the support calculation by the reduction of the number of test sub-isomorphism, and in turn, the number of embedded to keep.

References

1. Acosta-Mendoza, N., Gago-Alonso, A., Medina-Pagola, J.E.: Frequent Approximate Subgraphs as Features for Graph-Based Image Classification. Knowledge-Based Systems 27, 381–392 (2012)
2. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: In Proceedings of the 1994 International Conference on Very Large Data Bases (VLDB 1994), Santiago, Chile, pp. 487–499 (1994)
3. Borgelt, C.: Canonical Forms for Frequent Graph Mining. In: Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation *e.V.*, pp. 8–10. Universitat Berlin (2006)
4. Chen, C., Yan, X., Zhu, F., Han, J.: gApprox: Mining Frequent Approximate Patterns from a Massive Network. In: IEEE International Conference on Data Mining, ICDM 2007, pp. 445–450 (2007)
5. Holder, L.B., Cook, D.J., Bunke, H.: Fuzzy substructure discovery. In: Proceedings of the 9th International Workshop on Machine Learning, San Francisco, CA, USA, pp. 218–223 (1992)
6. Jia, Y., Zhang, J., Huan, J.: An Efficient Graph-Mining Method for Complicated and Noisy Data with Real-World Applications. Knowledge Information Systems 28(2), 423–447 (2011)
7. Jiang, C., Coenen, F., Zito, M.: A Survey of Frequent Subgraph Mining Algorithms. To appear: Knowledge Engineering Review (2012)
8. Song, Y., Chen, S.: Item Sets Based Graph Mining Algorithm and Application in Genetic Regulatory Networks. In: Proceedings of the IEEE International Conference on Granular Computing, Atlanta, GA, USA, pp. 337–340 (2006)
9. Xiao, Y., Wu, W., Wang, W., He, Z.: Efficient Algorithms for Node Disjoint Subgraph Homeomorphism Determination. In: Haritsa, J.R., Kotagiri, R., Pudi, V. (eds.) DASFAA 2008. LNCS, vol. 4947, pp. 452–460. Springer, Heidelberg (2008)
10. Zhang, S., Yang, J.: RAM: Randomized Approximate Graph Mining. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 187–203. Springer, Heidelberg (2008)
11. Zou, Z., Li, J., Gao, H., Zhang, S.: Mining Frequent Subgraph Patterns from Uncertain Graph Data. IEEE Transactions on Knowledge and Data Engineering 22(9), 1203–1218 (2010)