Mining Interesting Itemsets in Graph Datasets

Boris Cule, Bart Goethals, and Tayena Hendrickx

Department of Mathematics and Computer Science
University of Antwerp
{firstname.lastname}@ua.ac.be

Abstract. Traditionally, pattern discovery in graphs has been mostly limited to searching for frequent subgraphs, reoccurring patterns within which nodes with certain labels are frequently interconnected in exactly the same way. We relax this requirement by claiming that a set of labels is interesting if they often occur in each other's vicinity, but not necessarily always interconnected by exactly the same structures. Searching for such itemsets can be done both in datasets consisting of one large graph, and in datasets consisting of many graphs. We present novel methods dealing with both possible settings. Our algorithms benefit from avoiding computationally costly isomorphism checks typical for subgraph mining, as well as from a greatly reduced search space, as we consider only itemsets, and not all possible edges and paths that can connect them.

1 Introduction

Traditionally, searching for patterns in graphs has been almost exclusively limited to searching for frequent subgraphs. A frequent subgraph is a structure within a graph where nodes with certain labels are frequently interconnected in exactly the same way. We propose to relax this requirement. We claim that a pattern, a set of node labels, is interesting if these labels often occur in the graph near each other, not necessarily with exactly the same edges between them.

Consider the graph given in Fig. 1. It can be very easily observed that pattern *abcd* clearly stands out. However, traditional approaches, even with a frequency threshold as low as 2, will not find a single subgraph of size larger than 2, since nodes labelled a, b, c and d are always interconnected differently. This demonstrates the need for a new approach.

On top of being more flexible, and thus capable of finding previously undetected patterns, our method also greatly reduces the search space by looking for itemsets alone, rather than considering all possible combinations of edges and paths connecting them. However, if the dataset does contain a frequent subgraph, our method will find the equivalent itemset consisting of the same labels, but without the edges connecting them.

We consider two different problem settings, as the dataset can consist either of a (very large) single graph, or of a set of (smaller) graphs. In the single graph setting, the goal is to find itemsets that reoccur within the graph. We propose two methods to achieve this goal. The first is based on the traditional approaches to

J. Pei et al. (Eds.): PAKDD 2013, Part I, LNAI 7818, pp. 237–248, 2013.

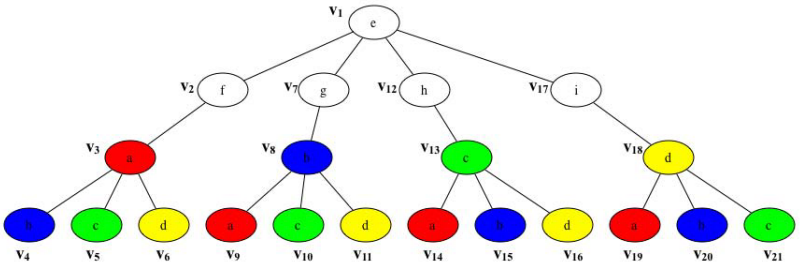

Fig. 1. A graph containing a pattern not discovered by subgraph mining

mining frequent itemsets in transaction databases. Our main contribution here consists of a way to transform a graph into a transaction database, after which any existing itemset mining algorithm can be used to find the frequent itemsets. In our second approach, we look for *cohesive* itemsets, whereby we insist that for an itemset to be considered interesting, it should not only appear often, but its items should also never appear far from each other. In the multiple graph setting, the goal is to find itemsets that occur in many of the input graphs. Here, the dataset cannot be transformed into a typical transaction database. However, the cohesive itemset approach proves perfectly adaptable to this setting.

An interesting property of the cohesive itemset approach is that the proposed interestingness measure is not anti-monotonic, which, at first glance, might represent an obstacle when generating large candidate itemsets. However, in both the single and the multiple graph setting we managed to come up with an efficient pruning method, which enables us to mine cohesive itemsets in a reasonable amount of time.

Even though the problem setting of subgraph mining is very different to that of mining itemsets in graphs, and therefore mostly incomparable, we do note that our algorithm results in a massive reduction in output.

The paper is organised as follows. In Section 2, we present the main related work. In Section 3 we propose two methods of identifying interesting itemsets in the single graph setting. In Section 4, we extend these approaches to be able to handle datasets consisting of multiple graphs. In Section 5, we give a sketch of our algorithms, before presenting the results of our experiments in Section 6. We end the paper with our conclusions in Section 7.

2 Related Work

The problem of discovering subgraphs that occur frequently in a dataset consisting of either one or multiple graphs has been very popular in data mining research. A good survey of the early graph based data mining methods is given by Washio and Motoda [18]. The first studies to find subgraph patterns were conducted by Cook and Holder [3] for a single graph, and by Motoda and Indurkhya [21] for multiple graphs, both using a greedy scheme to find some of the

most prevalent subgraphs. Although this greedy search may miss some significant subgraphs, it avoids the high complexity of the graph isomorphism problem.

Inokuchi et al. [9] and Kuramochi and Karypis [13] proposed the AGM and FSG algorithms, respectively, for mining all frequent subgraphs, using a level-wise approach, similar to that of APRIORI [1]. They suffer from two additional drawbacks, however: costly subgraph isomorphism testing and an enormous number of candidates that are generated along the way (due to the fact that both edges and nodes must be added to the pattern). Yan and Han [19] proposed GSPAN, which performs a depth-first search, based on a pattern growth principle similar to the one used in FP-GROWTH [7] for mining itemsets. Nijssen et al. proposed a more efficient frequent subgraph mining tool, called GASTON, which finds the frequent substructures in a number of phases of increasing complexity [15]. More specifically, it first searches for frequent paths, then frequent free trees and finally cyclic graphs. Further attempts at mining frequent subgraphs, both in one or many input graphs, have been made by Bringmann and Nijssen [2], Kuramochi and Karypis [14], Huan et al. [8], Yan et al.[20] and Inokuchi et al. [10].

Up to now, however, most of the research in graph mining has gone into finding frequent subgraphs. We focus on mining interesting itemsets in graphs, thus relaxing the underlying structure of the pattern. By doing so, we avoid the costly isomorphism testing, and by using a depth-first search algorithm, we avoid the pitfalls of APRIORI-like algorithms. A similar approach has been proposed by Khan et al. [12], where nodes propagate their labels to other nodes in the neighbourhood, according to given probabilities. Labels are thus aggregated, and can be mined as itemsets in the resulting graph. Khan et al. also propose to solve the problem of query-based search in a graph using a similar method [11]. Silva et al. [16, 17] and Guan et al. [6] introduce methods to identify correlation between node labels and graph structure, whereby the subgraph constraint has been loosened, but some structural information is still present in the output.

3 Single Graph Setting

Formally, we define a graph G as the set of nodes $V(G)$ and the set of edges $E(G)$, where each node $v \in V(G)$ has a label $l(v)$. We assume that the graph is connected, and that each node carries at most one label. In this setting, a pattern is an itemset X, a set of node labels that frequently occur in graph G in each other's neighbourhood. In Sections 3.1 and 3.2 we propose two ways of defining and finding interesting patterns in a graph. Note that we can also handle input graphs that contain nodes with multiple labels, by transforming each such node into a clique of nodes, each carrying one label.

3.1 Frequent Itemsets Approach

One way of defining an interesting itemset is by simply looking at its frequency. An interesting itemset is one that occurs often in the dataset. Our challenge therefore consists of taking a graph and converting it into a transaction database.

In essence, we are looking for items that often appear near each other in the graph, so we propose to create a transaction database in which each transaction would correspond to the neighbourhood of a single node in the graph. For a node v, we define the corresponding transaction as $t(v) = \{l(w)|w \in G, d(v,w) \leq n\}$, where $d(v,w)$ is the distance from v to w and n is the neighbourhood size, a user-defined parameter, indicating how far w can be from v and still be considered a part of v's neighbourhood. Typically, n should be relatively small if meaningful results are to be obtained. In a well-connected graph, with a large n, all nodes will be in each other's neighbourhoods.

We define a *transaction database of graph* G as $T(G) = \{t(v)|v \in G\}$. The *frequency* of an itemset X in graph G, $fr_G(X)$, is defined as the number of transactions in $T(G)$ that contain X. An itemset is considered frequent if its frequency is greater than or equal to a user-defined frequency threshold *min_freq*. Once we have converted our original dataset in this way, we can apply any one of a number of existing frequent itemset algorithms to obtain the desired results.

Let us return to our example in Fig. 1. Given a neighbourhood size of 1, the resulting transaction database is given in Table 1. We see that itemset *abcd* will now be discovered as interesting. In fact, only its subsets of size 1 or 2 will score higher.

Table 1. A transaction database obtained from the graph in Fig. 1 with a neighbourhood size of 1

v_{id}	items	v_{id}	items	v_{id}	items
1	$efghi$	8	$abcdg$	15	bc
2	afe	9	ab	16	cd
3	$abcdf$	10	bc	17	edi
4	ab	11	bd	18	$abcdi$
5	ac	12	ech	19	ad
6	ad	13	$abcdh$	20	bd
7	beg	14	ac	21	cd

3.2 Cohesive Itemset Approach

Another possible approach is to look for cohesive, rather than simply frequent, itemsets. This idea was inspired by the approach Cule et al. applied in order to find interesting itemsets in event sequences [5]. The idea is not to find items that only occur near each other often, but items that imply the occurrence of each other nearby with a high enough probability.

Consider, for example, the graph given in Fig. 2. We see that patterns *de* and *bc* are both frequent, though *de* will score higher than *bc*, both in subgraph mining and in frequent itemset mining as defined in Section 3.1. However, it can be argued that the value of itemset *de* should diminish due to the fact that a *d* appears in the graph without an *e* nearby, while each *b* has a *c* right next to it, and vice versa. Here, we present an approach that takes this into account.

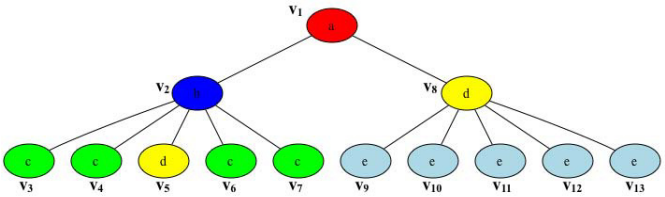

Fig. 2. A graph illustrating the intuition behind cohesive patterns

To start with, we introduce some notations and definitions. In a graph G, the set of nodes is denoted $V(G)$. The number of nodes in G is denoted $|V(G)|$. For an itemset X, we denote the set of nodes labelled by an item in X as $N(X) = \{v \in G | l(v) \in X\}$. The number of such nodes is denoted $|N(X)|$. Finally, we define the probability of randomly encountering a node in G labelled by an item in X as the *coverage* of X in G, or $P(X) = \frac{|N(X)|}{|G|}$.

For each occurrence of an item of X, we must now look for the nearest occurrence of all other items in X. For a node v, we define the sum of all these smallest distances as $W(v, X) = \sum_{x \in X} min_{w \in N(\{x\})} d(v, w)$. We then compute the average of such sums for all occurrences of items making up itemset X, $\overline{W}(X) = \frac{\sum_{v \in N(X)} W(v,X)}{|N(X)|}$. This allows us to define the *cohesion* of an itemset X in G as $C(X) = \frac{|X|-1}{\overline{W}(X)}$. The cohesion is a measure of how near to each other the items in X are in G on average. If they are always right next to each other, the sum of these distances for each occurrence of an item in X will be equal to $|X| - 1$, as will the average of such sums, and the cohesion of X will therefore be equal to 1.

Finally, the *interestingness* of an itemset X is defined as the product of its coverage and its cohesion, $I(X) = P(X)C(X)$. An itemset is considered interesting if its interestingness is greater than or equal to a user-defined interestingness threshold, *min_int*. Unlike with frequent itemsets, where we could tell nothing about the cohesion of the itemsets, we are now able to say that having encountered an item from an interesting itemset, there is a high probability of encountering the rest of the itemset nearby.

Let us now return to the example given in Fig. 2. If we apply these measures to itemsets bc and de, we first note that $P(bc) = 5/13$ and $P(de) = 7/13$. In order to compute the cohesion of the two itemsets, we first have to compute $\overline{W}(bc)$ and $\overline{W}(de)$, which are respectively 1 and $\frac{10}{7}$ (note that all relevant minimal windows are of size 1, except $W(v_5, de) = 4$). Therefore, $C(bc) = 1$ and $C(de) = \frac{7}{10}$, and $I(bc) = 0.385$ and $I(de) = 0.377$. We see that the value of itemset de has indeed diminished due to a d occurring far from any e.

Finally, note that while our method allows us to find more flexible patterns than subgraph mining, we do not miss out on any pattern subgraph mining can discover. If a graph contains many occurrences of a subgraph consisting of nodes labelled a, b and c, then we will find abc as an interesting itemset.

4 Multiple Graph Setting

In many applications, the dataset does not consist of a single graph, but of a collection of graphs. Formally, as before, we define a graph G as a set of nodes $V(G)$ and a set of edges $E(G)$, where each node $v \in G$ has a label $l(v)$. We also define $\mathcal{G}$ as a set of graphs $\{G_1, ..., G_n\}$, and assume that each graph in $\mathcal{G}$ is connected. A pattern is now an itemset X, a set of node labels that frequently occur in the set of graphs $\mathcal{G}$ in each other's neighbourhood. In other words, for a pattern to be interesting, it needs to appear in a cohesive form in many graphs. Unlike the single graph setting, the multiple graph setting does not allow us to transform the dataset into a transaction database. However, the cohesive itemset approach, presented in Section 3.2, can be generalised to the multiple graph setting in a relatively straightforward manner.

We first revisit the notations introduced in Section 3.2. Given a set of graphs $\mathcal{G}$, the number of graphs in $\mathcal{G}$ is denoted $|\mathcal{G}|$. We denote the set of all graphs that contain itemset X as $N_m(X) = \{G \in \mathcal{G} | \forall x \in X \ \exists v \in V(G) \text{ with } l(v) = x\}$. The number of such graphs is denoted as $|N_m(X)|$. Finally, we define the probability of encountering a graph in $\mathcal{G}$ containing the whole of itemset X as the *coverage* of X in $\mathcal{G}$, or $P_m(X) = \frac{|N_m(X)|}{|\mathcal{G}|}$.

Given a graph G_j in $N_m(X)$, we must now look for the most cohesive occurrence of X. To find such an occurrence, we will look for a node in the graph, labelled by an item in X, from which the sum of the distances to all other items in X is the smallest. Given a graph G_j containing an itemset X, we define this lowest sum as $W(X, j) = \min_{v \in N(X, G_j)} W(v, X)$, where $N(X, G_j)$ is the set of nodes in G_j labelled by an item in X, while $W(v, X)$ is defined as in Section 3.2.

We now compute the average of such smallest sums for all graphs in $\mathcal{G}$ containing the whole of itemset X, $\overline{W}_m(X) = \frac{\sum_{j \in N_m(X)} W(X,j)}{|N_m(X)|}$. We can then define the *cohesion* of an itemset X in $\mathcal{G}$ as $C_m(X) = \frac{|X|-1}{\overline{W}_m(X)}$. Once again, a fully cohesive itemset will have cohesion equal to 1.

Finally, the *interestingness* of an itemset X in $\mathcal{G}$ is defined as the product of its coverage and its cohesion, $I_m(X) = P_m(X)C_m(X)$.

5 Algorithms

Mining frequent itemsets in graphs can be done by transforming a graph into a transaction database, and then using an existing itemset miner to generate the output. However, the cohesive itemset approach is less straightforward, and we now present our algorithms, Grit and Mug, for solving this problem in the single graph setting and the multiple graph setting, respectively.

5.1 Single Graph Setting

The fact that our interestingness measure is not anti-monotonic clearly represents a problem. We will sometimes need to search deeper even when we

encounter an uninteresting itemset, as one of its supersets could still prove interesting. Traversing the complete search space is unfeasible, so we will need a different pruning technique to speed up our algorithm. We adapt the approach introduced by Cule et al. for mining itemsets in sequences [5] to our setting.

We approach the problem using depth-first search, and the pseudocode of the main GRIT algorithm is provided in Algorithm 1. The first call to the algorithm is made with X empty and Y containing all possible items. At the heart of the algorithm is the *UBI* pruning function, used to decide when to prune a complete branch of the search tree, and when to proceed deeper. Essentially, we can prune a complete branch if we are certain that no itemset generated within this branch can be interesting. To be able to ascertain this, we compute an upper bound for the interestingness of all these itemsets, and prune the branch if this upper bound is smaller than the interestingness threshold. We begin by noting that, for each Z, such that $X \subseteq Z \subseteq X \cup Y$, it holds that $|N(Z)| \leq |N(X \cup Y)|$, $|Z| \leq |X \cup Y|$ and $\sum_{v \in N(X)} W(v, X) \leq \sum_{v \in N(Z)} W(v, Z)$. Expanding the definition of the interestingness, we get that $I(Z) = \frac{|N(Z)| \times |N(Z)| \times (|Z|-1)}{\sum_{v \in N(Z)} W(v,Z) \times |G|}$. It therefore follows that $I(Z) \leq \frac{|N(X \cup Y)| \times |N(X \cup Y)| \times (|X \cup Y|-1)}{\sum_{v \in N(X)} W(v,X) \times |G|}$. We have thus found an upper bound for the interestingness of all itemsets Z, that can be generated in a branch of the search tree starting off with itemset X, and reaching as deep as itemset $X \cup Y$.

However, while this upper bound is theoretically sound, it is also computationally very expensive. Note that we would need to compute $\sum_{v \in N(X)} W(v, X)$ at each node in our search tree. This would require traversing the whole graph searching for the minimal distances between all items in X for all relevant nodes. This, too, would be infeasible. Luckily, if we express these sums differently, we can avoid these computationally expensive database scans. Adapting the approach introduced by Cule and Goethals [4] to the graph setting, we first note that the sum of the minimal distances between items making up an itemset X and the remaining items in X can also be expressed as a sum of separate sums of such distances for each item individually, $\sum_{v \in N(X)} W(v, X) = \sum_{x \in X} \sum_{v \in N(\{x\})} W(v, X)$. We then note that each such sum for an occurrence of an item $x \in X$ is equal to the sum of individual minimal distances between the same occurrence of x and any other item $y \in X$. For the sum of such distances,

Algorithm 1. GRIT($\langle X, Y \rangle$) finds interesting itemsets

```
if UBI(⟨X, Y⟩) ≥ min_int then
  if Y = ∅ then
    output X
  else
    Choose a in Y
    GRIT(⟨X ∪ {a}, Y \ {a}⟩)
    GRIT(⟨X, Y \ {a}⟩)
  end if
end if
```

it holds that $\sum_{v \in N(\{x\})} W(v, X) = \sum_{v \in N(\{x\})} \sum_{y \in X \setminus \{x\}} W(v, xy)$. Naturally, it also holds that $\sum_{v \in N(\{x\})} W(v, X) = \sum_{y \in X \setminus \{x\}} (\sum_{v \in N(\{x\})} W(v, xy))$. To simplify our notation, from now on we will denote $\sum_{v \in N(\{x\})} W(v, xy)$ by $s(x, y)$. Finally, we see that $\sum_{v \in N(X)} W(v, X) = \sum_{x \in X} \sum_{y \in X \setminus \{x\}} (s(x, y))$, giving us a much more elegant way to compute the sum of distances between an occurrence of an item in X and the rest of X for all nodes labelled by an item in X. Finally, we are ready to define our upper-bound-based pruning function: $UBI(\langle X, Y \rangle) = \frac{|N(X \cup Y)|^2 \times (|X \cup Y| - 1)}{\sum_{x \in X} \sum_{y \in X \setminus \{x\}} (s(x,y)) \times |G|}$.

This pruning function is easily evaluated, as all it requires is that we store $s(x, y)$, the sum of minimal distances between x and y over all occurrences of x, for each pair of items (x, y), so we can look them up when necessary. This can be done as soon as the dataset has been read, and all entries can be computed efficiently. Note that if Y is empty, then $UBI(\langle X, Y \rangle) = I(X)$, so if we reach a leaf node in the search tree, we can immediately output the itemset.

5.2 Multiple Graph Setting

As in the single graph setting, the interestingness measure based on cohesive itemsets in multiple graphs is also not anti-monotonic. Here, however, the coverage alone is anti-monotonic. Given itemsets X and Y, such that $X \subset Y$, it is clear that any graph that contains Y will also contain X. Keeping in mind that the interestingness of an itemset is never greater than its coverage, this allows us to prune even more candidates from our search space. For any itemset Z, such that $X \subseteq Z$, it holds that $I(Z) \leq P(Z) \leq P(X)$. Therefore, if we encounter an itemset X, such that its coverage is lower than the interestingness threshold *min_int*, we can safely discard all its supersets from the search space.

On top of this pruning criterion, we can develop a pruning function *MUBI* in much the same way as we did in the single graph setting above. Once again, it would be infeasible to compute all necessary sums of minimal windows at each node in our search tree. However, this time we cannot express this sum using similar sums for pairs of items as we did in Section 5.1. The reason for this is the fact that we now define $W(X, j)$ as the minimal occurrence of X in graph G_j, while in Section 3.2 we defined $W(v, X)$ as a sum of individual minimal distances between items. Given a graph G_j and an itemset X, knowing the individual minimal distances between the items of X in G_j would bring us no closer to knowing the size of the minimal occurrence of X in G_j. We have therefore decided to perform all our experiments without *MUBI* — pruning less, but faster. As a result, our MUG algorithm for mining interesting itemsets in multiple graphs is exactly the same as the one given in Algorithm 1, with line **if** $P(X) \geq$ *min_int* **then** replacing line 1.

6 Experiments

For our single graph setting experiments, we generated a number of synthetic datasets. To start with, we generated a graph with 10 000 nodes, randomly

allocating labels ranging from 1 to 20. We made sure that labels 0 and 1 were more probable than all the others, followed by labels 2 and 3, while the remaining labels were all equally probable. We build the graph by, at each step, adding either a new node and connecting it to a random existing node, or adding a new edge between two existing nodes. In our first set of experiments, we set the probability of adding a new node to 60%, and the probability of adding a new edge to 40%, resulting in a relatively sparse graph, with around 1.7 edges per node on average. The characteristics of the input graph and the results of the experiments for both the frequent itemset approach and the cohesive itemset approach are presented in the top quarter of Table 2. For the frequent itemset approach, the reported runtime is the sum of the time needed to transform the dataset into a transaction database added to the time needed to run an implementation[1] of the classical FP-GROWTH algorithm on the transformed dataset.

Table 2. Experimental results on four different single graph datasets. The runtime is measured in milliseconds.

Input Graph			Frequent Itemset Approach			Cohesive Itemset Approach		
nodes	edges	items	*min_freq*	runtime	itemsets found	*min_int*	runtime	itemsets found
10 000	16 853	20	2 000	873	4	0.37	2 545	1
			1 000	905	25	0.35	2 796	35
			100	966	737	0.33	3 236	704
			10	1 202	15 644	0.30	4 480	8 193
			1	2 513	250 181	0.20	36 314	430 961
10 000	16 853	30	2 000	853	4	0.37	823 263	1
			500	892	71	0.35	892 480	31
			50	960	1 793	0.33	1 013 627	1 672
			5	1 225	37 349	0.31	1 253 782	28 147
10 000	68 190	20	4 000	1 238	11	0.55	3 327	35
			1 000	1 720	4 449	0.50	4 084	2 228
			250	6 279	462 933	0.35	23 178	245 668
			50	10 953	1 048 575	0.20	61 519	950 411
100 000	166 446	20	20 000	2 994	4	0.37	21 425	1
			5 000	3 192	63	0.35	21 856	17
			500	3 575	1 980	0.30	24 502	8 092
			50	4 137	31 424	0.25	31 559	89 617

To examine how our methods react to different types of input, we created three more graphs, each time changing one of the settings. In the second set of experiments, we introduced some noise into the dataset, by randomly choosing 100 nodes in the original graph and changing their labels to a random item ranging from 21 to 30. For the third set of experiments, we created a denser graph, by setting the probability of adding a new node to 15%, and the probability of adding a new edge to 85%. In the fourth set of experiments, we used a larger graph, creating 100 000 nodes, keeping the other settings as in the original dataset. The characteristics of these three input graphs and the results of our experiments can be seen in the bottom three quarters of Table 2.

In all four sets of experiments, we note that frequent itemset mining works very fast once the transformation of the dataset has been done, while the cohesive itemset approach suffers from having to compute $s(x, y)$ for each pair of

[1] Taken from `http://adrem.ua.ac.be/~goethals/software`

items x and y, regardless of the threshold. GRIT struggles with noisy data, as expensive computations need to be done even for very rare items and itemsets, while transforming the graph into a transaction database takes a lot more time when dealing with both a denser or a larger input graph. Finally, all experiments on the four datasets produced expected results in terms of the discovered itemsets — items 0 and 1 were ranked top amongst items, while itemset $\{0, 1\}$ was the highest ranked pair of items. Both methods discovered the same itemsets of size 3 and 4 (itemsets made up of items 0, 1, 2 and 3), while results differed for itemsets of size 5 or more. However, due to the anti-monotonicity of the traditional frequency measure, FP-GROWTH ranked singletons higher than their supersets, while GRIT considered itemset $\{0, 1, 2, 3\}$, for example, more interesting than each individual item alone. We conclude that both approaches have their respective merits, with the frequent itemset approach being faster, and the cohesive itemset approach more intuitive. GRIT will first find large, informative itemsets, while FP-GROWTH first finds singletons, and the really interesting large itemsets can be buried beneath a pile of smaller itemsets.

In the multiple graph setting, we experimented on a dataset consisting of 340 chemical compounds, made up of 66 different atom types. However, we discovered five unconnected graphs in the dataset and removed them, as the cohesive itemset approach can only be applied to connected graphs, leaving us with 335 input graphs. We compare our algorithm for finding cohesive itemsets in multiple graphs with the GASTON tool developed by Nijssen and Kok [15]. Both the dataset and the implementation are available online[2]. We present the results of our experiments in Tables 3 and 4.

Table 3. Results of the MUG algorithm on the Chemical_340 dataset

min_int	runtime	itemsets
0.50	17	5
0.20	173	17
0.10	772	53
0.05	2 196	172
0.01	21 869	1 122

Table 4. Results of the GASTON algorithm

min_freq	runtime	itemsets
200	10	11
100	16	68
50	26	427
10	376	19 237
5	9 318	447 879

By comparing our MUG algorithm to a frequent subgraph miner, we are in fact comparing apples and oranges. While MUG uses an interestingness measure, GASTON uses a frequency threshold. MUG searches for itemsets, GASTON discovers subgraphs. MUG discards structure and focuses on items in each other's neighbourhoods, while GASTON focuses on structure. Clearly, MUG results in a massive reduction in output, as, for a typical interesting itemset discovered by MUG, GASTON will find a number of combinations of edges (and their subsets) that are frequent. Furthermore, the actual patterns discovered by the two algorithms differ greatly. To name but one example, the most interesting itemset of

[2] `http://www.liacs.nl/~snijssen/gaston/download.html`

size 4 that we discovered was $\{0, 1, 4, 5\}$, while the most frequent subgraph of size 4 was $0-0-0-0$. On the other hand, if we compare only patterns of size 2, GASTON finds graph $1-9$ as the most frequent, while we also found $\{1, 9\}$ as highest ranked itemset of size 2. However, it is also important to note that itemset $\{0, 1, 4, 5\}$ ranked as one of the best itemsets overall in our output, while, due to the nature of frequent subgraph mining, graph $0-0-0-0$ was ranked below all its subgraphs. As with GRIT, MUG has the advantage of ranking the larger interesting itemsets above all other, while GASTON will always rank a large pattern below a number of smaller patterns.

7 Conclusion

In this paper we presented a number of new methods to identify interesting itemsets in one or many input graphs. For one graph, such itemsets consist of items that often occur close to each other. Unlike previous approaches that typically look for structures connecting these items, we only look at the distances between the items themselves. This enables us to avoid the typical pitfalls of subgraph mining — costly isomorphism checks and a huge number of candidates. On top of the classical frequent itemset approach that we adapted to mining itemsets in a large graph, we propose a second method, mining cohesive itemsets, consisting of items that appear close to each other frequently enough. This second approach proved perfectly adaptable to the multiple graph setting, too.

References

[1] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. of the 20th Int. Conf. on Very Large Data Bases, pp. 487–499 (1994)

[2] Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 858–863. Springer, Heidelberg (2008)

[3] Cook, D.J., Holder, L.B.: Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research 1, 231–255 (1994)

[4] Cule, B., Goethals, B.: Mining association rules in long sequences. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) PAKDD 2010, Part I. LNCS (LNAI), vol. 6118, pp. 300–309. Springer, Heidelberg (2010)

[5] Cule, B., Goethals, B., Robardet, C.: A new constraint for mining sets in sequences. In: Proc. of the 9th SIAM Int. Conf. on Data Mining, pp. 317–328 (2009)

[6] Guan, Z., Wu, J., Zhang, Q., Singh, A., Yan, X.: Assessing and ranking structural correlations in graphs. In: Proc. of the 2011 ACM SIGMOD Int. Conf. on Management of Data, pp. 937–948. ACM (2011)

[7] Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proc. of the 2000 ACM SIGMOD Int. Conf. on Management of Data, vol. 29, pp. 1–12 (2000)

[8] Huan, J., Wang, W., Prins, J., Yang, J.: Spin: mining maximal frequent subgraphs from graph databases. In: Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 581–586 (2004)
[9] Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: Zighed, D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 13–23. Springer, Heidelberg (2000)
[10] Inokuchi, A., Washio, T., Motoda, H.: Complete mining of frequent patterns from graphs: Mining graph data. Machine Learning 50, 321–354 (2003)
[11] Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S., Tao, S.: Neighborhood based fast graph search in large networks. In: Proc. of the 2011 ACM SIGMOD Int. Conf. on Management of Data, pp. 901–912. ACM (2011)
[12] Khan, A., Yan, X., Wu, K.-L.: Towards proximity pattern mining in large graphs. In: Proc. of the 2010 ACM SIGMOD Int. Conf. on Management of Data, pp. 867–878. ACM (2010)
[13] Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proc. of the 2001 IEEE Int. Conf. on Data Mining, pp. 313–320 (2001)
[14] Kuramochi, M., Karypis, G.: Finding frequent patterns in a large sparse graph. Data Mining and Knowledge Discovery 11, 243–271 (2005)
[15] Nijssen, S., Kok, J.N.: The gaston tool for frequent subgraph mining. Electronic Notes in Theoretical Computer Science 127, 77–87 (2005)
[16] Silva Jr., A., Meira, W., Zaki, M.J.: Structural correlation pattern mining for large graphs. In: Proc. of the 8th Workshop on Mining and Learning with Graphs, pp. 119–126. ACM (2010)
[17] Silva Jr., A., Meira, W., Zaki, M.J.: Mining attribute-structure correlated patterns in large attributed graphs. Proc. of the VLDB Endowment 5(5), 466–477 (2012)
[18] Washio, T., Motoda, H.: State of the art of graph-based data mining. ACM SIGKDD Explorations Newsletter 5, 59–68 (2003)
[19] Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: Proc. of the 2002 IEEE Int. Conf. on Data Mining, pp. 721–724 (2002)
[20] Yan, X., Zhou, X., Han, J.: Mining closed relational graphs with connectivity constraints. In: Proc. of the 11th ACM SIGKDD Int. Conf. on Knowledge Discovery in Data Mining, pp. 324–333 (2005)
[21] Yoshida, K., Motoda, H., Indurkhya, N.: Graph-based induction as a unified learning framework. Journal of Applied Intelligence 4, 297–316 (1994)