Mining Cohesive Itemsets in Graphs

Tayena Hendrickx, Boris Cule, and Bart Goethals

University of Antwerp, Belguim,
firstname.lastname@uantwerp.be

Abstract. Discovering patterns in graphs is a well-studied field of data mining. While a lot of work has already gone into finding structural patterns in graph datasets, we focus on relaxing the structural requirements in order to find items that often occur near each other in the input graph. By doing this, we significantly reduce the search space and simplify the output. We look for itemsets that are both frequent and cohesive, which enables us to use the anti-monotonicity property of the frequency measure to speed up our algorithm. We experimentally demonstrate that our method can handle larger and more complex datasets than the existing methods that either run out of memory or take too long.

1 Introduction

Graph mining is a popular field in data mining, with wide applications in bioinformatics, social network analysis, etc. Traditional approaches have been largely limited to searching for frequent subgraphs, i.e., reoccurring structures consisting of labelled nodes frequently interconnected in exactly the same way. However, the concept of frequent subgraphs is not flexible enough to capture all patterns. First of all, subgraphs are too strict. If we consider the graph given in Figure 1, we see that items a, b and c make up a pattern that visibly stands out. However, this pattern will not be found by subgraph mining since the three items are never connected in the same way. Subgraph mining approaches are also typically computationally complex. To begin with, they are forced to deal with the graph isomorphism problem. For small graphs, isomorphism checking is not really that hard. However, when we want to mine large graphs, like social networks, the isomorphism checks become computationally very expensive. On top of this, due to the fact that both edges and nodes must be added to the pattern, a large number of candidate subgraphs is generated along the way.

In order to avoid these problems, Cule et al. [5] proposed a *Cohesive Itemset Approach* for mining interesting itemsets in graphs. An interesting itemset is defined as a set of node labels that occur often in the graph and are, on average, tightly connected to each other, but are not necessarily always connected in exactly the same way. Although this method could find previously undetected patterns, there are still a number of drawbacks. The proposed method considers an itemset frequent if a large enough proportion of the graph is covered by items making up the itemset. As a result, large itemsets, sometimes partially consisting of very infrequent items, can be found in the output. This undermines

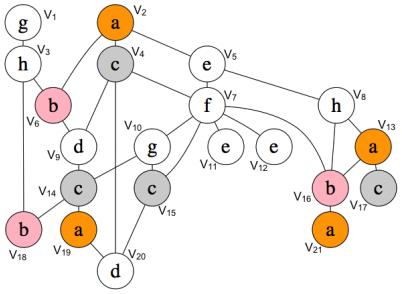

Fig. 1: A graph containing a pattern not discovered by subgraph mining.

the attempts to prune the search space and results in prohibitive run-times and memory usage. To overcome this problem, we propose to look for *Frequent Cohesive Itemsets*, where we only consider itemsets consisting of labels that are all, as individual items, frequent, and look for those that on average occur near each other. In this way, we greatly reduce the search space, resulting in a smaller output and a significant reduction in the time and space complexity of our algorithm. We further explore the possibility of pruning candidate itemsets based on the cohesion measure as well. Cohesion is not anti-monotonic, but we develop an upper bound that allows us to prune whole branches of the search tree if certain conditions are satisfied. We experimentally confirm that our algorithm successfully handles datasets on which the existing method fails. In further experiments on an artificial dataset, with a limited alphabet size, we demonstrate exactly where our algorithm outperforms the existing method.

The rest of the paper is organised as follows. In section 2 we discuss the main related work. Section 3 formally describes our problem setting, and Section 4 presents our algorithm. In Section 5 we present the results of our experiments, before ending the paper with our conclusions in Section 6.

2 Related Work

The problem of discovering patterns in graphs is an active data mining topic. A good survey of the early graph based data mining methods is given by Washio and Motoda [19]. 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.

The first attempts to find subgraph patterns were made by Cook and Holder [4] for a single graph, and by Motoda and Indurkhya [23] for multiple graphs. Both use a greedy scheme that avoids the graph isomorphism problem, but may miss some significant subgraphs. Dehaspe and Toivonen [6] perform a complete search for frequent subgraphs by applying an ILP-based algorithm.

Inokuchi et al. [9] and Kuramochi and Karypis [13] proposed the AGM and FSG algorithms for mining all frequent subgraphs, respectively, using a breadth-

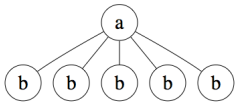

Fig. 2: A graph on which the GRIT algorithm gives counterintuitive results.

first search. These algorithms suffer from two drawbacks: costly subgraph isomorphism testing and an enormous number of generated candidates (due to the fact that both edges and nodes must be added to the pattern). Yan and Han [20] proposed gSPAN, an algorithm that uses a depth-first search. A more efficient tool, called GASTON, was proposed by Nijssen and Kok [15]. Further attempts at subgraph mining have been made by Inokuchi et al. [10], Yan et al. [21, 22], Huan et al. [8], Kuramochi and Karypis [14] and Bringmann and Nijssen [2].

At first glance, it may seem that itemsets, as patterns, are not as expressive as subgraphs. Nevertheless, Karunaratne and Boström [11] showed that itemset mining algorithms are computationally simpler than their graph mining counterparts and are competitive in terms of results. Recently, Cule et al. [5] proposed the GRIT algorithm for mining interesting itemsets in graphs. An itemset is defined as a set of node labels which often occur in the graph and are, on average, tightly connected to each other. Although the method is more flexible than the traditional approaches, it has some drawbacks. The interestingness of an itemset is defined as the product of its coverage and its cohesion, where the coverage measures what percentage of the graph is covered by items making up the itemset, while the cohesion measures average distances between these items. Due to the small world phenomenon, this approach can result in an item that occurs very infrequently in the dataset being discovered as part of an interesting itemset. Consider the graph given in Figure 2. The GRIT algorithm will discover pattern ab as interesting, because, per definition, the coverage of ab will be larger than the coverage of the individual items a and b. Since each a is connected to a b and vice versa, ab will also score well on cohesion. Although item a is not frequent at all, it has made its way into the output thanks to having many neighbours labelled b. However, itemset ab does not represent a reoccurring pattern and should not be considered more interesting than item b on its own.

In this paper, we build on this work, and focus on mining frequent cohesive itemsets, insisting that each item in a discovered itemset must itself be frequent. A separate cohesion threshold ensures that itemsets we discover are also cohesive. By using a two-step approach of first filtering out the infrequent items, and then using the frequent items to generate candidate itemsets, which are then evaluated on cohesion, we greatly reduce the search space and run-time of our algorithm and produce more meaningful results. By searching for itemsets rather than subgraphs, we also avoid the costly isomorphism testing, and by using a depth-first search algorithm, we avoid the pitfalls of breadth-first search.

Another itemset mining approach has been proposed by Khan et al. [12], where nodes propagate their labels to neighbouring nodes according to given

probabilities. Labels are thus aggregated, and can be mined as itemsets in the resulting graph. Silva et al. [16, 17] and Guan et al. [7] introduced methods to identify correlation between node labels and graph structure, whereby the subgraph constraint has been loosened, but not entirely dropped.

3 Frequent Cohesive Itemsets

In this section, we introduce our approach for mining *Frequent Cohesive Itemsets* in a dataset consisting of a single graph. We assume that the graph consists of labelled nodes and unlabelled edges, and we focus on connected graphs with at most one label per node. However, we can also handle input graphs where each node carries a set of labels, as will be shown in Section 5.

To start with, we introduce some notations and definitions. In a graph G, the set of nodes is denoted $V(G)$. Each node $v \in V(G)$ carries a label $l(v) \in S$, where S is the set of all labels. For a label $i \in S$, we denote the set of nodes in the graph carrying this label as $L(i) = \{v \in V(G) | l(v) = i\}$. We define the frequency of a label $i \in S$ as the probability of encountering that label in G, or

$$freq(i) = \frac{|L(i)|}{|V(G)|}.$$

From now on, we will refer to labels as items, and sets of labels as itemsets.

In order to compute the cohesion of an itemset X we first denote the set of nodes labelled by an item in X as $N(X) = \{v \in V(G) | l(v) \in X\}$. In the next step, 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

$$W(v, X) = \sum_{x \in X} min_{w \in N(\{x\})} d(v, w),$$

where $d(v, w)$ is the length of the shortest path from node v to node w. Subsequently, we compute the average of such sums for all occurrences of items making up itemset X:

$$\overline{W}(X) = \frac{\sum_{v \in N(X)} W(v, X)}{|N(X)|}.$$

Finally, we define the cohesion of an itemset X, where $|X| \geq 2$, as

$$C(X) = \frac{|X|-1}{\overline{W}(X)}.$$

If $|X| < 2$, we define $C(X)$ to be equal to 1.

Cohesion measures how close to each other the items making up itemset X are on average. If the items are always directly connected by an edge, 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 be equal to 1.

Given user defined thresholds for frequency (*min_freq*) and cohesion (*min_coh*), our goal is to discover each itemset X if $\forall x \in X : freq(x) \geq min\_freq$ and $C(X) \geq min\_coh$. To allow the user more flexibility, we use two optional size parameters, *minsize* and *maxsize*, to limit the size of the discovered itemsets.

We will now illustrate the above definitions on the graph given in Figure 1. Assume we are evaluating itemsets abc and ef, with thresholds *min_freq* and *min_coh* set to 0.1 and 0.6, respectively. According to our definitions, we first note that $N(abc) = \{v_2, v_4, v_6, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, v_{21}\}$ and $N(ef) = \{v_5, v_7, v_{11}, v_{12}\}$. It follows that $|N(abc)| = 11$ and $|N(ef)| = 4$. To compute the cohesion of itemset abc, we now search the neighbourhood of each node in $N(abc)$ and obtain the following: $W(v_4, abc) = W(v_6, abc) = W(v_{16}, abc) = W(v_{17}, abc) = W(v_{18}, abc) = W(v_{19}, abc) = 3$, $W(v_{15}, abc) = W(v_{21}, abc) = 4$ and $W(v_2, abc) = W(v_{13}, abc) = W(v_{14}, abc) = 2$. The average of the above sums is $\overline{W}(abc) = \frac{32}{11}$. Doing the same for itemset ef we get $W(v_5, ef) = W(v_7, ef) = W(v_{11}, ef) = W(v_{12}, ef) = 1$. Therefore, $\overline{W}(ef) = 1$. We can now compute the cohesion of the two itemsets abc and ef as

$$C(abc) = \frac{|abc|-1}{\overline{W}(abc)} = \frac{3-1}{\frac{32}{11}} = \frac{22}{32} \approx 0.69 \quad \text{and} \quad C(ef) = \frac{|ef|-1}{\overline{W}(ef)} = \frac{2-1}{1} = 1.$$

We see that both abc and ef are cohesive enough, but, for an itemset to be considered a frequent cohesive pattern, each item in the itemset must be frequent. In our dataset, items a, b, c and e are frequent, but f is not. Therefore, although ef is more cohesive than abc, it will not be discovered as a frequent cohesive itemset. Note that we computed the cohesion of ef above only to illustrate the example. Our algorithm, presented in Section 4, first finds frequent items and then considers only itemsets that consist of these items, so itemset ef would not even be considered, and the above computations would not take place.

4 Algorithm

Our algorithm for mining frequent cohesive itemsets in a given graph consists of two main steps. The first step is to filter out the infrequent items. This can be done while loading the dataset into the memory for the first time, counting the frequency of each item as they occur, and then outputting only the frequent ones. In the second step, given in Algorithm 1, candidates are generated by applying depth-first search, using recursive enumeration. During this enumeration process, a candidate consists of two sets of items, X and Y. X contains those items which make up the candidate, while Y contains the items that still have to be enumerated. The first call to the algorithm is made with X empty and Y containing all *frequent* items appearing in the graph.

At the heart of the algorithm is the *UBC* pruning function (discussed in detail below), which is used to decide whether to prune the complete branch of the search tree, or to proceed deeper. If the branch is not pruned, the next test evaluates if there are still items with which we can expand the candidate. If not, we have reached a leaf node in the search tree, and discovered a frequent

Algorithm 1 FCI($\langle X, Y \rangle$) finds frequent cohesive itemsets

```
if UBC(⟨X,Y⟩) ≥ min_coh then
  if Y = ∅ then
    if |X| ≥ minsize then output X
  else
    Choose a in Y
    if |X ∪ {a}| ≤ maxsize then FCI(⟨X ∪ {a}, Y \ {a}⟩)
    if |X ∪ (Y \ {a})| ≥ minsize then FCI(⟨X, Y \ {a}⟩)
  end if
end if
```

cohesive itemset which is sent to the output. Otherwise, the first item in Y, for example a, is selected and the FCI algorithm is recursively called twice: once with item a added to X and once without. In both calls, a is removed from Y.

An important property of the cohesion measure is that it is neither monotonic nor anti-monotonic. For example, consider the graph shown in Figure 3, and assume the *min_coh* threshold is set to 0.6. We can see that $C(ac) < min\_coh < C(abc) < C(a)$, even though $a \subset ac \subset abc$. Therefore, we will sometimes need to go deeper in the search tree, even when we encounter a non-cohesive itemset, since one of its supersets could still prove cohesive. However, traversing the complete search space is clearly unfeasible. In this work, we deploy an additional pruning technique using an upper bound for the cohesion measure, similar to the upper bound for the interestingness measure used by Cule et al. [5]. In a nutshell, we can prune a complete branch of the search tree if we are certain that no itemset generated within this branch can still prove cohesive. To ascertain this, we compute an upper bound for the cohesion measure of all the itemsets in that branch, and if this upper bound is smaller than the cohesion threshold, the whole branch can be pruned.

More formally, recall that a frequent cohesive itemset X, of size 2 or larger, must satisfy the following requirement:

$$C(X) = \frac{(|X|-1)|N(X)|}{\sum_{v \in N(X)} W(v, X)} \geq min\_coh.$$

Assume now that we find ourselves in node $\langle X, Y \rangle$ in the search tree, i.e., at the root of the subtree within which we will generate each candidate itemset Z such that $X \subseteq Z \subseteq X \cup Y$. We know that cohesion is neither monotonic nor anti-monotonic, so we cannot in advance know which such itemset will have the highest cohesion, and we need to develop an upper bound that holds for all of them. What we do know, however, is the cohesion of itemset X, $C(X)$, as shown

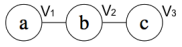

Fig. 3: A small input graph.

above. We can now analyse the extent to which this cohesion can maximally grow as items from Y are added to X. Note that if an item is added to X, both the nominator and the denominator in the above expression will grow. In order for the total value to grow maximally, we therefore need to find the case in which the nominator will grow maximally, and the denominator minimally. Let us first examine the case of adding just a single item y to X. Recall that the denominator contains the total sum, for all occurrences of items in X in the graph, of the sums of all minimal distances from such an occurrence to all other items in X. Each such sum of minimal distances will now have to be expanded to include the minimal distance to the new item y. In the worst case, y will be discovered at a distance of 1 from all these occurrences. These sums will therefore grow by exactly 1. Additionally, the total sum will now also include the sums of minimal distances from each occurrence of y to all items in X. In the worst case, from each y we will be able to find each item in X at a distance of 1. Therefore, these sums will be equal to $|X|$. For the case of adding an item y to X, we thus obtain

$$\sum_{v \in N(X \cup \{y\})} W(v, X \cup \{y\}) \geq (\sum_{v \in N(X)} (W(v, X) + 1)) + |N(\{y\})||X|.$$

By induction, for adding the whole of Y to X, we obtain

$$\sum_{v \in N(X \cup Y)} W(v, X \cup Y) \geq (\sum_{v \in N(X)} (W(v, X) + |Y|)) + |N(Y)|(|X \cup Y| - 1).$$

What the above worst-case actually describes is a case of adding maximally cohesive occurrences of $X \cup Y$ to already known occurrences of X. For the overall cohesion to grow as much as possible, we should add as many such occurrences as possible. Clearly, we will obtain this maximal number of occurrences if we add the whole of Y to X. Therefore, we can update the nominator accordingly, and conclude that for any Z, such that $X \subseteq Z \subseteq X \cup Y$, it holds that

$$C(Z) \leq \frac{(|X \cup Y| - 1)|N(X \cup Y)|}{(\sum_{v \in N(X)} (W(v, X) + |Y|)) + |N(Y)|(|X \cup Y| - 1)}.$$

We also need to consider the user-chosen *maxsize* parameter. If $X \cup Y$ is larger than *maxsize*, the worst case will not be obtained by adding the whole of Y to X, but only by adding items from Y to X until *maxsize* is reached. Before proceeding with our analysis, we first abbreviate the new upper bounds for $|Y'|$ and $|N(Y')|$, where $Y' \subset Y$ and $|X \cup Y'| \leq$ *maxsize*, with

$$UBY' = \min(maxsize - |X|, |Y|)$$

and

$$UBNY' = \min(|N(Y)|, \max_{\substack{Y_i \subset Y, \\ |Y_i| = maxsize - |X|}} |N(Y_i)|).$$

Finally, we develop the upper bound for the cohesion of all candidate itemsets generated in the branch of the search tree rooted at $\langle X, Y \rangle$ as

$$UBC(\langle X, Y \rangle) = \frac{(|X| - 1 + UBY')(|N(X)| + UBNY')}{(\sum_{v \in N(X)} (W(v, X) + UBY')) + (|X| - 1 + UBY')UBNY'}.$$

This upper bound is only defined when $|X| \geq 2$. If X is either empty or a singleton, we define $UBC(\langle X, Y \rangle) = 1$.

At first glance, it seems that in order to compute $UBNY'$, we would need to evaluate all possible sets Y_i, such that $Y_i \subset Y$ and $|X \cup Y_i| = maxsize$, which would be computationally expensive. We avoid this problem by enumerating the items in Y sorted on frequency in descending order. For example, if $Y = \{y_1, y_2, \ldots, y_n\}$, given $X = \{a, b, c\}$ and *maxsize* set to 5, it is obvious that

$$\max_{\substack{Y_i \subset Y, \\ |X \cup Y_i| = maxsize}} |N(X \cup Y_i)| = |N(\{a, b, c, y_1, y_2\})|.$$

Similarly, it may seem that to compute $\sum_{v \in N(X)} W(v, X)$ at each node in our search tree, we would need to traverse the whole graph searching for the minimal distances between all items in X for all relevant nodes, which would be unfeasible. In order to avoid these costly database scans, we adopt the approach that was used in the GRIT algorithm [5] and express the sum of the minimal distances between items making up an itemset and the remaining items in X as a sum of separate sums of such distances for each pair of items individually. Each of these sums between individual items are stored in a matrix which is generated only once at the beginning of the algorithm. Since we are only interested in itemsets consisting of frequent items, it is sufficient to compute the minimal distances only for those frequent items. Consequently, the matrix we generate is of size $|F| \times |F|$, where F is the set of frequent items, which is, depending on the *min_freq* threshold, considerably smaller than the matrix of size $|S| \times |S|$, where S is the complete alphabet, generated by GRIT.

Thanks to the above two optimisations, given a candidate $\langle X, Y \rangle$, we can compute $UBC(\langle X, Y \rangle)$ in constant time.

5 Experiments

In this section, we experimentally compare the FCI and GRIT algorithms. For our experiments, we used two different datasets — a real-life graph dataset, and a synthetic dataset. The first dataset we used in our experiments is a combination of the yeast protein interaction network available in Saccharomyces Genome Database (SGD) [3] and the yeast regulatory network available in YEASTRACT [1]. In the combined network each node represents a yeast protein and each edge represents an interaction between two proteins. The given interaction network consists of 5 811 protein-nodes and 62 454 edges. The node labels are derived from gene ontology assignments [18], i.e., terms that are assigned to proteins that describe their biological process. The mapping of each of these labels to the list of yeast proteins was obtained from the annotation file provided by the SGD database [3]. In total there are 30 distinct labels. Since each protein-node has multiple labels, we first had to transform the given network. More formally, we expanded the graph by replacing each protein-node with a unique dummy node, surrounded by a set of nodes, each carrying one of the original labels. Our resulting graph consisted of 18 108 nodes and 74 751 edges.

min_freq	$\|F\|$	C-Time (s)	*maxsize*	*min_coh*	#itemsets	#candidates	M-Time (ms)
0.001	23	8 974	5	0.40	12	24 774	88
				0.35	154	75 091	224
				0.30	7 989	122 752	586
0.005	15	7 966	5	0.40	9	5 720	37
				0.35	127	9 976	55
				0.30	3 196	13 923	205
0.010	10	6 453	5	0.40	9	1 127	10
				0.35	91	1 290	19
				0.30	612	1 473	70
0.005	15	7 966	∞	0.40	9	23 774	113
				0.35	169	48 916	250
				0.30	25 559	65 518	984

Table 1: Results of the FCI algorithm on the protein interaction network of yeast.

Table 1 reports the number of frequent items $|F|$, the time needed to generate the $|F|\times|F|$ distance matrix (C-Time), the number of discovered itemsets, the number of candidates that were considered, and the time needed for the mining stage (M-Time), for varying values of *min_freq* and *min_coh*. In the first three sets of experiments, *minsize* was set to 2 and *maxsize* to 5, while we set *maxsize* to infinity in the fourth set of experiments. C-Time and M-Time are reported separately because the frequent items are fixed for a given frequency threshold, so the distance matrix needs to be computed just once and can then be reused at various cohesion thresholds. The considerable reduction in mining times as the cohesion threshold grows shows that our pruning function has the desired effect. In Section 4, we presented two crucial elements in our pruning function, using the properties of the cohesion measure, and the *maxsize* parameter. In the fourth set of experiments, we set the *maxsize* parameter to infinity, to entirely eliminate its effect on pruning. The results show that we still manage to produce output quickly, while pruning large numbers of candidates.

The Grit algorithm failed to produce output in all of these settings, as the required matrix and the resulting search space proved far too large. Therefore, we can conclude that filtering out the infrequent items is a crucial step if we wish to handle large real-life datasets.

Finally, let us have a closer look at the discovered itemsets. Having shown the output to biologists, they confirmed that the most cohesive patterns were those that could be expected for this type of network. For example, with *min_freq* set to 0.005 and *min_coh* to 0.2, the highest scoring itemset consists of two terms that are highly related, namely {*cellular metabolic process*, *organic substance metabolic process*}. Indeed, many of the proteins in the studied network are labelled with both terms as they describe overlapping biological processes in yeast. However, besides the expected patterns, biologists discovered some other patterns, such as {*cellular metabolic process, organic substance metabolic process, biosynthetic process, catabolic process, regulation of biological quality*}, an itemset of size 5 with a cohesion of 0.36. This itemset contains three terms that never occur together on a node, namely biosynthetic process, catabolic process and regulation of biological quality. Each of these three terms refers to very dif-

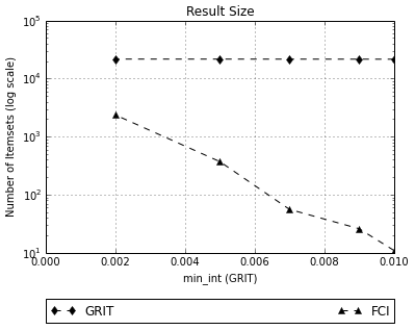

Fig. 4: Comparison of the output size of the GRIT and FCI algorithms.

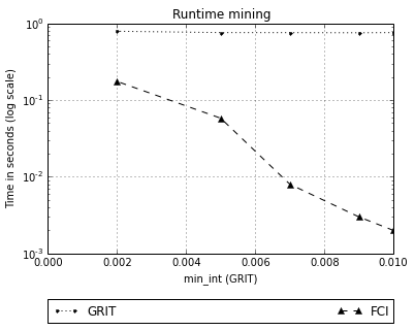

Fig. 5: Comparison of the mining times for the GRIT and FCI algorithms.

ferent, almost opposite, biological processes and thus no proteins exist in yeast that are active in all three categories. However, from a biological point of view, one can expect the nodes with these different terms to be close together in the network due to the regulatory mechanisms that exist in yeast, which propagate through the interactions described in the studied network.

For our second experimental setting, we generated a random graph with 100 000 nodes and 399 495 edges. The labels were randomly allocated and range from 0 to 19. The probability of encountering a label differed for each label, as follows: we defined $p_0 = 1$, $p_1 = 2$ and for $i = 2 \ldots 19$, $p_i = \sum_{j=1\ldots,i} j$. The probability of encountering label i was proportional to p_i. Given that $\sum_{i=0\ldots19} p_i = 1332$, the probability of encountering label 0 was $\frac{1}{1332}$ and the probability of encountering label 19 was $\frac{190}{1332}$. We built 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. For this experiment, we set the probability of adding a new node to 25%, and the probability of adding a new edge to 75%, resulting in an average of approximately 8 edges per node

The main goal of this experiment was to be able to compare GRIT and FCI in terms of ouput size and run-time, since GRIT failed to generate output for the yeast protein interaction network. We applied the FCI algorithm with *min_coh* fixed at 0.1, *minsize* set to 2, *maxsize* to 5 and a varying *min_freq* threshold. The interestingness threshold, *min_int*, for GRIT was set to the product of *min_coh* and *min_freq*, to guarantee that all itemsets found by FCI were also found by GRIT (since GRIT defines interestingness as the product of coverage and cohesion, and coverage is equal to the frequency in case of singletons). Figure 4 shows the number of discovered itemsets for the two approaches. The mining times are reported in Figure 5. The preprocessing stage of GRIT took 2.5 days, while FCI needed between 1 and 50 minutes, depending on the chosen frequency threshold (as reported in Table 2). As we can see in the figures, the output size and the run-time of FCI decrease considerably as we increase the frequency threshold, since more items are filtered out to start with, which results in fewer candidate

min_coh	*maxsize*	*min_freq*	$\|F\|$	C-Time (s)	#itemsets	#candidates	M-Time (ms)
0.1	5	0.10	4	62	11	25	2
		0.05	9	643	372	836	58
		0.02	13	2 853	2 366	6 460	175
0.1	∞	0.10	4	62	11	25	2
		0.05	9	643	502	1 012	64
		0.02	13	2 853	8 178	16 368	427

Table 2: Experimental results of the FCI algorithm on the artificial dataset.

itemsets. On the other hand, as the interestingness threshold, used by GRIT, increases, we see no change in the output size and run-times, since GRIT still generates a huge number of candidates. Finally, Table 2 shows a comparison of using the FCI algorithm with the *maxsize* parameter set to 5 and to infinity, respectively. Once again, we can see that our pruning shows good results, even if we cannot rely on the *maxsize* parameter as a pruning tool.

6 Conclusions

In this paper, we present a novel method for mining frequent cohesive itemsets in graphs. By first filtering out the infrequent items, and only then evaluating the remaining candidate itemsets on cohesion, we achieve much better results than existing algorithms, both in terms of run-times, and the quality of output. Furthermore, by limiting ourselves to itemsets, we avoid the costly isomorphism testing needed in subgraph mining. Using a depth-first search allows us to use an upper bound for the cohesion measure to prune unnecessary candidates, thus further speeding up our algorithm. Experiments demonstrate that the presented method greatly outperforms the existing ones on a variety of datasets.

7 Acknowledgments

We wish to thank Pieter Meysman, bioinformatician at the University of Antwerp, for helping us interpret the results of the experiments on the biological dataset.

References

[1] Abdulrehman, D., Monteiro, P.T., Teixeira, M.C., Mira, N.P., Lourenço, A.B., dos Santos, S.C., Cabrito, T.R., Francisco, A.P., Madeira, S.C., Aires, R.S., Oliveira, A.L., Sá-Correia, I., Freitas, A.T.: YEASTRACT: providing a programmatic access to curated transcriptional regulatory associations in Saccharomyces cerevisiae through a web services interface. Nucleic acids research 39 (2011)

[2] Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: Proc. 12th Pacific-Asia Conf. on Knowledge Discovery and Data Mining. pp. 858–863 (2008)

[3] Cherry, J.: SGD: Saccharomyces Genome Database. Nucleic Acids Research 26(1), 73–79 (1998)

[4] Cook, D.J., Holder, L.B.: Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research 1, 231–255 (1994)
[5] Cule, B., Goethals, B., Hendrickx, T.: Mining interesting itemsets in graph datasets. In: Proc. of the 17th Pacific-Asia Conf. on Knowledge Discovery and Data Mining. pp. 237–248 (2013)
[6] Dehaspe, L., Toivonen, H.: Discovery of frequent datalog patterns. Data Mining and Knowledge Discovery 3, 7–36 (1999)
[7] 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 (2011)
[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: Principles of Data Mining and Knowledge Discovery. pp. 13–23 (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] Karunaratne, T., Boström, H.: Can frequent itemset mining be efficiently and effectively used for learning from graph data? In: Proc. of the 11th Int. Conf. on Machine Learning and Applications (ICMLA). pp. 409–414 (2012)
[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 (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.: The gaston tool for frequent subgraph mining. Electronic Notes in Theoretical Computer Science 127, 77–87 (2005)
[16] Silva, A., Meira, J.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 (2010)
[17] Silva, A., Meira, J.W., Zaki, M.J.: Mining attribute-structure correlated patterns in large attributed graphs. Proc. of the VLDB Endowment 5(5), 466–477 (2012)
[18] The Gene Ontology Consortium: Gene Ontology annotations and resources. Nucleic acids research 41(Database issue), D530–5 (2013)
[19] Washio, T., Motoda, H.: State of the art of graph-based data mining. ACM SIGKDD Explorations Newsletter 5, 59–68 (2003)
[20] 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)
[21] Yan, X., Han, J.: Closegraph: Mining closed frequent graph patterns. In: Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery in Data Mining. pp. 286–295 (2003)
[22] 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)
[23] Yoshida, K., Motoda, H., Indurkhya, N.: Graph-based induction as a unified learning framework. Journal of Applied Intelligence 4, 297–316 (1994)