Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM), Chania, Greece, 2012

# Discovery of Top-k Dense Subgraphs in Dynamic Graph Collections

Elena Valari Maria Kontaki Apostolos N. Papadopoulos

Data Engineering Lab., Department of Informatics, Aristotle University
54124 Thessaloniki, Greece, {evalari,kontaki,papadopo}@csd.auth.gr

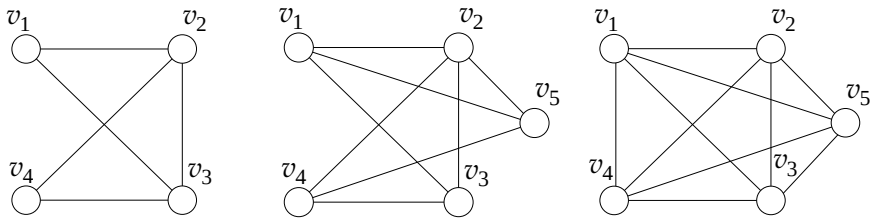
**Abstract.** Dense subgraph discovery is a key issue in graph mining, due to its importance in several applications, such as correlation analysis, community discovery in the Web, gene co-expression and protein-protein interactions in bioinformatics. In this work, we study the discovery of the top-$k$ dense subgraphs in a set of graphs. After the investigation of the problem in its static case, we extend the methodology to work with dynamic graph collections, where the graph collection changes over time. Our methodology is based on lower and upper bounds of the density, resulting in a reduction of the number of exact density computations. Our algorithms do not rely on user-defined threshold values and the only input required is the number of dense subgraphs in the result ($k$). In addition to the exact algorithms, an approximation algorithm is provided for top-$k$ dense subgraph discovery, which trades result accuracy for speed. We show that a significant number of exact density computations is avoided, resulting in efficient monitoring of the top-$k$ dense subgraphs.

## 1 Introduction

Many modern applications require the management of large volumes of *graph data*. Graphs are very important in scientific applications such as bioinformatics, chemoinformatics, link analysis in social networks, to name a few. Dense subgraph discovery is a fundamental *graph mining* task [1, 5] with increasing importance. Density is a significant property of graphs, because it is highly related to how well a graph is connected and can be used as a measure of the graph coherence. Usually, the densest the graph the more likely that the connectivity among the graph nodes will be higher.

Among the various density definitions, we adopt the one that relates the graph density to the average degree [7]. More formally, for a graph $G(V, E)$, where $V$ is the set of nodes and $E$ the set of edges, the density of $G$, denoted as $den(G)$, is given by the number of edges over the number of nodes, i.e., $den(G) = |E|/|V|$. For the rest of the work, we focus on undirected and unweighted graphs. Generalizations to other graph classes are performed easily with appropriate modifications. Figure 1 depicts some examples of density computation.

Algorithms for dense subgraph discovery that have been proposed in the literature require one or more *constraints* to be defined by the user [2]. For example,




(a) $den(G) = 5/4$ (b) $den(G) = 8/5$ (c) $den(G) = 10/5$

**Fig. 1.** Densities of various graphs.

all subgraphs having a density value higher than a threshold are reported back to the user. The major limitation of such an approach is that the determination of threshold values is difficult: *i*) if the threshold is too low, then we risk a cumbersome answer set and *ii*) if the threshold is too high then an empty answer set may be returned. A meaningful threshold value may be difficult to define without at least a limited a priori knowledge regarding the nature of the data (e.g., distribution of densities). To overcome this limitation, in this work, we focus on dense subgraph discovery by taking a top-$k$ oriented approach. More specifically, the only input required by our methods is the number $k$ of the dense subgraphs. The basic benefit of this approach is that the cardinality of the answer set is always known and is equal to $k$, thus, no surprises are expected regarding the number of answers. In addition, no "wild guesses" regarding the density threshold need to be performed by the user.

The second novelty of our research is that we focus on dynamic graph collections that take the form of a *stream of graphs*. More specifically, computation is performed on a *count-based sliding window* of size $w$, defined over a stream of graphs $G_1$, $G_2$, ..., $G_w$. New graph objects may arrive whereas old ones expire. The value of $w$ depends on the application. The challenge is to monitor the $k$ densest subgraphs induced by the graphs that are currently active in the sliding window. This translates in performing the necessary actions when a new graph arrives or an old one expires. Evidently, computation must be as efficient as possible to avoid significant delays during updates. Previously proposed methods working in graph streams do not consider deletions.

There are many applications that benefit from the support of top-$k$ dense subgraph discovery. In *web usage mining*, the links followed by users form a directed graph. Many such graphs may be available in a streaming fashion, corresponding to different time instances or different geographical areas. Dense subgraphs are useful in this case because they enable community discovery. The use of the sliding window enables to center our focus at the most recent data available, rather than base the discovery process on the complete history. Mining these graphs on-the-fly is extremely important towards continuous knowledge discovery. As a second example, consider a large social network, where users communicate by means of message exchange. Each graph in this case, corresponds to the inter-

actions among users for a specific time period, e.g., a day. By collecting these graphs for a period of time, we can monitor the evolution of interactions among users. Dense subgraphs in these graphs correspond to communities formed at certain instances. If these communities grow in size and density, then this could mean that an interesting topic emerges.

We note that, to the best of our knowledge, this is the first work that addresses the problem of top-$k$ dense subgraph discovery and consequently, the first work that employs this concept over a stream of graphs. The rest of the paper is organized as follows. The next section, contains a brief discussion of related work in the area. Section 3 presents our methods in detail. A performance evaluation study is contained in Section 4 and finally, Section 5 concludes our work and motivates for further research in the area.

## 2 Related Work and Contributions

There is a significant number of research works studying the problem of dense subgraph discovery. Material in this subject can be found in [5], whereas an excellent survey chapter in the area is contained in [1]. The problem is considered important due to the numerous applications interested in dense subgraphs, such as, community discovery, genes with significant coexpression, highly correlated items in market basket data and many more.

The majority of the algorithms proposed so far in the literature, assume a fairly static case. For example, in [6] shingling is used to discover dense parts in large bipartite graphs. Dense subgraphs are also used in [8] to discover important subgraphs in a database of graphs corresponding to gene networks. Furthermore, [13] defines a generalization of the densest subgraph problem by adding an additional distance restriction (defined by a separate metric) to the nodes of the subgraphs. This method was applied to a data set of genes and their annotations.

More recently, some efforts have been performed to enable graph mining over evolving graphs [15]. In particular, to complicate things further, in many cases it is assumed that the graph (or graphs) form a data stream. In such a case, usually we are allowed to see the data only once, without the ability to perform random access. In these lines, the authors in [2] discover dense subgraphs when the graph appears in the form of an edge stream. A similar idea is studied in [4], where there are multiple streams, one for each graph. Our work differs from the previous approaches in several points:

(i) We are the first to provide algorithms for dense subgraph discovery in a top-$k$ fashion. This is very important since the user may control the answer set and may terminate the execution at will.

(ii) There are no magic thresholds used in the algorithms. This means that there are no risks that the answer set will be too small or too big, since the $k$ best subgraphs are returned, without any explicit reference to their density. However, our techniques may enforce constraints on the density by simply executing the top-$k$ algorithm and reject any subgraph that does not satisfy the density constraint.

(iii) We support dense subgraph discovery in a set of graphs, rather than in a single graph. In particular, graphs are presented to the system in the form of a *stream of graphs*. Since it is natural to focus on the most recent data, we apply a count-based sliding window of size $w$ [11, 16]. This means that at any time instance there are $w$ active graphs that are mined in a continuous manner and thus, both insertions and deletions must be supported.

## 3 Dense Subgraphs in Graph Collections

### 3.1 Preliminaries

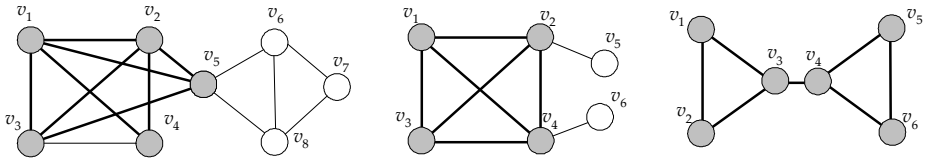
In this section, we present some fundamental concepts necessary for top-$k$ dense subgraph discovery. Table 1 summarizes some frequently used symbols. As mentioned previously, in this work, the density of a graph is defined as the average degree over its nodes [7]. In that work, an algorithm is given to determine the *densest subgraph* of a graph $G$. The densest subgraph $G^{(1)}$ of $G$ is simply an induced subgraph with the maximum possible density among all subgraphs of $G$. More formally:

$$G^{(1)} = \arg\max\{den(g) : g \sqsubseteq G\}$$

The algorithm proposed by Goldberg in [7] (from now on this algorithm will be denoted as GOLD) for the computation of the densest subgraph of a graph $G$ requires a logarithmic number of steps, where in each step a maxflow computation is performed. The maxflow computations are performed on an augmented graph $G'$ and not on the original graph $G$. More specifically, $G$ is converted to a directed graph by substituting an edge between nodes $u$ and $v$ by two directed edges from $u$ to $v$ and backwards. These edges are assigned a capacity of 1. In addition, two artificial nodes are inserted, the source $s$ and the sink $t$. Node $s$ is connected to all nodes in $G$ by using directed arcs emanating from

| Symbol | Description |
|---|---|
| $G_i$ | the $i$-th graph in the stream |
| $V_i$, $E_i$ | set of vertices and set of edges of $G_i$ |
| $n_i$, $m_i$ | order and size of $G$ ($N_G = \|V_G\|$, $M_G = \|E_G\|$ ) |
| $d(v)$ | the degree of a vertex $v$ |
| $g \sqsubseteq G$ | $g$ is an induced subgraph of $G$ |
| $den(G)$ | density of graph $G$ |
| $C(G)$ | the maximum core subgraph of $G$ |
| $G^{(j)}$ | the $j$-th densest subgraph of $G$ |
| $w$ | the size of the sliding window |
| $k$ | number of densest subgraphs monitored |
| $TOPK$ | the (current) result of top-$k$ dense subgraphs |

**Table 1.** Basic notations used.




(a) $den(G^{(1)})$=9/5 (b) $den(G^{(1)})$=6/4 (c) $den(G^{(1)})$=7/6

**Fig. 2.** Densest subgraphs and their density values.

$s$, whereas $t$ is connected by adding directed arcs emanating from each node and ending at $t$. The capacities of these edges are carefully selected (details in [7]) in order to guarantee that the mincut computed after at most a logarithmic number of steps, will separate the densest subgraph from the rest of the graph. Assuming that the push-relabel maxflow algorithm is being used enhanced with dynamic trees, the complexity for the computation of the densest subgraph is $O(\log n \cdot (2n + m) \log(n^2/(n + m)))$.

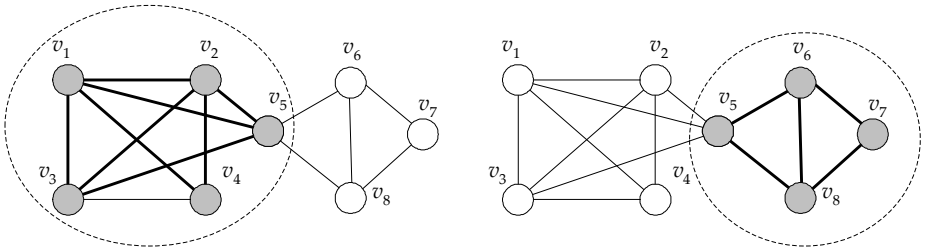
Figure 2 depicts the densest subgraph for some graphs. The edges of the densest subgraph is shown bold and the nodes are gray-filled. If the density of every subgraph of $G$ is less than the density of $G$ then the whole graph $G$ is its densest subgraph. Such a case is shown in Figure 2(c). Next, we state explicitly the problem investigated in this paper:

*Problem Definition*: Given a dynamic stream of $w$ graphs $G_1, ..., G_w$ and an integer $k$, monitor the top-$k$ most dense edge-disjoint subgraphs continuously, taking into account arrivals and expirations of graph objects. ■

### 3.2 Dense subgraph discovery in a set of graphs

In this section, we study a progressive process to determine the $k$ densest subgraphs of a graph $G$. The usefulness of such a mechanism raises from the fact that the user may require more dense components of the graph in a *get-next* fashion. Consequently, the search process may terminate at any time, if adequate and satisfactory results have been computed. This technique generalizes easily for a set of graphs.

To compute the $k$ densest subgraphs we proceed as follows: $i$) first, the densest subgraph of $G$ is computed, $ii$) the edges comprising the densest subgraph are removed from $G$, together with any nodes that become isolated, $iii$) if the number of answers reported is less than $k$ we repeat the process. According to this process, two dense subgraphs cannot share edges. However, they may share some nodes. To illustrate the idea, an example is given in Figure 3, where the top-2 dense subgraphs are computed. The densest subgraph $G^{(1)}$ of $G$ is shown in Figure 3(a) and its density is 9/5. By removing the edges contained in $G^{(1)}$ from $G$ (along with the isolated nodes $v_1, v_2, v_3, v_4$) and by applying again the




(a) densest subgraph (b) second densest subgraph

**Fig. 3.** The two densest subgraphs of a graph $G$.

densest subgraph discovery algorithm in the reduced graph, we arrive at the situation depicted in Figure 3(b). Therefore, the second best (densest) subgraph is composed of the vertices $v_5, v_6, v_7, v_8$ and its density is $5/4$.

For simplicity in the presentation, we define the operation $\ominus$ between a graph $G$ and a subgraph $g \sqsubseteq G$. The result $G \ominus g$ is computed by removing from $G$ all edges in $g$ and also the nodes that become isolated after edge removal. This process may be applied iteratively, until $k$ dense subgraphs are reported. According to this method, the densities corresponding to the densest subgraphs are reported in a non-increasing order. This is very important towards progressive computation of dense subgraphs since it enables an early termination (e.g., before $k$) if the density values become too low to be of interest to the user. Note, however, that after a graph reduction process, the resulting graph may be disconnected. In such a case, the process is applied to each connected component separately, until $k$ results are obtained.

The only available tool we have to determine the $k$ densest subgraphs is the GOLD algorithm of [7]. This means that so far we do not have a pruning mechanism at hand in order to discard a connected component before applying the expensive sequence of maxflow computations. If this was possible, then a significant number of maxflow operations could have been avoided, resulting in a more efficient computation. To enable pruning, we will use the concept of the *maximum core* of a graph [10, 14].

**Definition 1.** *The maximum core $C(G)$ of a graph $G$ is a subgraph of $G$ containing vertices with a degree at least $\beta$, where the value of $\beta$ is the maximum possible.*

To illustrate the idea, an example is shown in Figure 4. A simple algorithm to compute the maximum core performs a sequence of vertex removals, starting with the vertices with the smallest degree. According to this process, vertices $v_9$, $v_{10}$ and $v_{11}$ will be removed first, since their degree is 1. The resulting graph is the 2-core of $G$, since all vertices have a degree at least 2. Next, we remove vertex $v_7$, and consequently vertices $v_6$ and $v_8$ are also removed, because their degree

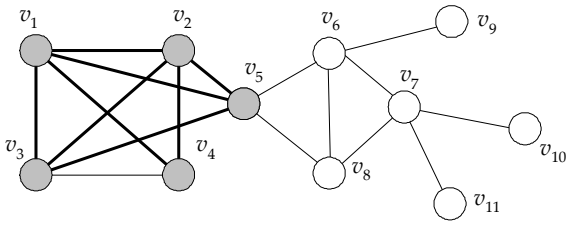



**Fig. 4.** The 1-core is composed of all vertices of $G$. The 2-core contains the vertices $v_1$ through $v_7$. Finally, the 3-core, which is also the maximum core, contains the vertices $v_1$, $v_2$, $v_3$, $v_4$ and $v_5$ (the vertices of maxcore are shown gray).

has been reduced due to the removal of $v_7$. The resulting graph, composed of the vertices $v_1$, $v_2$, $v_3$, $v_4$ and $v_5$ is the 3-core of $G$ since every vertex has a degree at least 3. At this point, if we continue this process, we will result in an empty graph. Therefore, the maximum core value of $G$ is 3.

The question is how can we use the maximum core to enable pruning during top-$k$ dense subgraph discovery. A very interesting result has been reported in [9], stating that the density of the maximum core of a graph $G$ is a (1/2)-approximation of the density of the densest subgraph of $G$. This means that the density of the densest subgraph of $G$ is at most twice the density of the maximum core of $G$. In addition, since $C(G)$ is an induced subgraph of $G$, its density is less than or equal to the density of the densest subgraph. More formally, we have the following inequality:

$$2 \cdot den(C(G)) \geq den(G^{(1)}) \geq den(C(G)) \tag{1}$$

According to the previous inequality, we may use the density of the maximum core to define an upper and a lower bound on the density of the densest subgraph of $G$. The next important issue, is how fast can we compute the maximum core. By using a binary heap enhanced with a hash table for fast decrease-key operations in the heap, the previous algorithm which is based on repetitive removal of vertices with the lowest degree requires $O(m \log n)$ comparisons, resulting in slow computation, especially for large graphs. A more efficient algorithm has been studied in [3], which is based on count-sort. The algorithm requires linear additional space but runs in $O(m)$ worst case time, resulting in a very fast maximum core computation.

Figure 5 depicts the outline of TopkDense algorithm, which computes the $k$ densest subgraphs of an input graph $G$ by using pruning based on the concept of maximum cores. The algorithm requires a priority queue $PQ$ which accommodates the current connected components produced. $PQ$ is implemented as a binary maxheap data structure which stores entries of the form $< g$, $C(g)$, $den(C(g)) >$, prioritized by the density of the maxcore (the last attribute). The $k$ best answers are stored in $A$. The algorithm uses Inequality 1 in Line 6 in

**Algorithm** TopkDense ($G$, $k$)
**Input**: $G$ initial input graph, $k$ number of results
**Output**: $A$, set of $k$ densest subgraphs of $G$

```
1.  initialize answer set A ← ∅;
2.  compute the maxcore C(G) of G;
3.  initialize priority queue PQ ←< G, C(G), den(C(G) > ;
4.  while (PQ not empty)
5.      < g, C(g), den > ← PQ.deheap(); /* get the first element of the heap */
6.      if (2 · den > k-th density in A) then
7.          g^(1) ← call FindDensest(g); /* compute densest subgraph of g */
8.          if (den(g^(1))> k-th density in A) then
9.              remove subgraph with the k-th density from A;
10.             insert g^(1) in A;
11.             remove g^(1) from g;
12.             for each component h of g ⊖ g^(1)
13.                 C(h) ← maxcore subgraph of h;
14.                 den(C(h))) ← density of C(h);
15.                 PQ.enheap(< h, C(h), den(C(h)) >);
16. return A;
```

**Fig. 5.** Outline of TopkDense algorithm.

order to decide if a subgraph is promising or not. In case where the upper bound of its density is lower than the current $k$-th best density, obviously it must be discarded without any further consideration. Otherwise, we should compute the density of its densest subgraph (invocation of FindDensest() algorithm in Line 7) and then proceed accordingly (Lines 8-15).

**Lemma 1.** *Let $g_1$ and $g_2$ be induced subgraphs of $G$ such that $den(g_1) = den(g_2) = d$. Then, if $g_1$ and $g_2$ have at least a common vertex, the density $den(g_1 \cup g_2)$ of the subgraph composed of the vertices and edges of $g_1$ and $g_2$ is strictly larger than $d$.*

*Proof.* Let $n_i$ and $m_i$ denote the number of vertices and edges of $g_i$, where $i = 1, 2$. Based on the definition of the density, it holds that $den(g_1)=m_1/n_1$ and $den(g_2)=m_2/n_2$. Let $h$ be the graph composed by the union of $g_1$ and $g_2$, i.e., $h = g_1 \cup g_2$. For the density of $h$ we have that $den(h) = (m_1 + m_2)/(n_1 + n_2 - \epsilon)$, where $\epsilon$ is the number of common vertices of $g_1$ and $g_2$, which is strictly larger than zero because we have assumed that the number of common vertices is at least one. Therefore, $den(h) > d$ and this completes the proof. □

Based on the previous discussion, it is not hard to verify that by using this algorithm only *maximal* dense subgraphs are returned. This is an important property, because we avoid repetitive computations to discover a large dense

subgraph. It is guaranteed that the next dense subgraph returned will be maximal with respect to the number of vertices.

### 3.3 Dense subgraphs in a stream of graphs

In this section, we extend the idea of top-$k$ dense subgraph discovery in order to handle dynamic graph collections and more specifically, a *stream of graphs*. We center our attention in the case of a *count-based sliding window*, where we are interested only in the $w$ most fresh graph objects. Therefore, if a new graph object is inserted in the collection, the oldest graph object must be deleted. Notice that the top-$k$ dense subgraph set may contain subgraphs of different graphs. Without lost of generality, we assume that at each time instance only one new graph object arrives and only the oldest one expires. Arrivals and expirations of more graphs are handled similarly.

The monitoring process of top-$k$ dense subgraphs consists of $i$) the *initialization phase* and $ii$) the *maintenance phase*. The initialization involves the computation of the top-$k$ dense subgraphs for the first $w$ graphs. This is performed only once, and the maintenance phase can be used to achieve this by disabling expirations. For this reason, we focus only on the maintenance phase.

Assume that at time instance $t$ we have $w$ graphs denoted as $G_{t-w+1}$, $G_{t-w+2}$, ..., $G_t$. We store the active graphs (i.e., graphs belonging to the current window) in a FIFO list with respect to their timestamps. In addition, we keep the top-$k$ dense subgraph set separately. When an update occurs, the current time increases by one and a new graph $G_{t+1}$ is inserted in the window. Since a count-based sliding window is used, the graph $G_{t-w+1}$ expires. The maintenance phase should update the top-$k$ set in order to reflect the changes of the window.

The naive approach begins with the discovery of the densest subgraph of the new graph $G_{new}$ and if its density $den(G^{(1)}_{new})$ is greater than the current $k$-th best density then $G^{(1)}_{new}$ is inserted into $TOPK$ and the method continues similarly for each connected component $h$ of $G_{new} \ominus G^{(1)}_{new}$. Next, it deletes the expired graph $G_{old}$. Notice that the removal of $G_{old}$ may reduce the number of densest subgraphs. Assume that $k' \leq k$ is the cardinality of the answer set after the removal of $G_{old}$ and its subgraphs. If $k' = k$ no more operations are required. On the other hand, if $k' < k$, the naive approach should scan all active graphs to find the remaining $k - k'$ densest subgraphs in order to retain the size of the answer set. Notice that, during this process the examined subgraphs may have been computed during the insertion of the corresponding original graph, but this does not hold for all of them. Thus, it is possible that additional densest subgraph computations may be required.

The naive approach invokes too many unnecessary densest subgraph computations. The proposed method tries to reduce the number of time consuming operations. We examine separately the insertion of the new graph and the expiration of the oldest one. Thus, our proposed method can handle different number of insertions/expirations in each update, with minimal modifications. We can modify the TopkDense algorithm to enable the insertion of $G_{new}$. Instead of initializing the answer set (Line 1 of Figure 5), we give the current answer set as a

parameter in TopkDense. The proposed method reduces the number of densest subgraph computations, because it uses as a pruning criterion the density of the maximum core subgraph (Line 6 of Figure 5). The method is correct, i.e. if there exists a subgraph of $G_{new}$ that has one of the highest $k$ densities, it will be inserted in the answer set.

For the expiration of a graph, we distinguish two cases: $i$) the expired graph $G_{old}$ has at least one of the $k$-th densest subgraphs and $ii$) none of its subgraphs is part of $TOPK$. For the latter case, it is sufficient to remove $G_{old}$ from the list with the active graphs. However, if $G_{old}$ has at least one subgraph belonging to $TOPK$, further operations are needed to update correctly the answer set. First, we remove the subgraphs of $G_{old}$ from the answer set. Assume that $k - k'$ densest subgraphs are deleted. A simple approach is to scan the active graphs to find the substitute subgraphs. Remember, we reduced the number of densest subgraph discovery during the insertion of a new graph with the invocation of TopkDense algorithm. The cost to compute the densest subgraphs that are missing is prohibitive. The proposed method uses again Inequality 1 to handle this case efficiently. For a graph $G$, if $2 \cdot den(C(G)) < k$-th density, $G$ can be omitted from further consideration. More specifically, the method forces the insertion of the first $k'$ subgraphs into $TOPK$. Next, the method tries to improve the answer set by scanning the remaining graphs. For each graph $G$, if $G^{(1)}$ is not available, we check if $2 \cdot den(C(G)) < k$-th density, then we omit $G$ and we proceed with the next graph. Otherwise, we compute $G^{(1)}$. If $den(G^{(1)}) \geq k$-th density, we insert $G^{(1)}$ into $TOPK$ and we proceed with the component of the residual graph.

To clarify the proposed method, we give an example. Assume the stream of graphs of Table 2. Moreover, assume that $w = 5$, $k = 4$ and the current time is 5. The current window contains the first five graphs. The $TOPK$ consists of $G_4^{(1)}$, $G_2^{(1)}$, $G_2^{(2)}$ and $G_5^{(1)}$. Assume now, that graph $G_6$ arrives. The densities of the three densest subgraphs of $G_6$ are given in Table 2. The proposed method computes the density of the maximum core subgraph $den(C(G_6)) = 20.2$. It holds that $2 \cdot den(C(G_6)) > k$-th density, therefore the method computes the densest subgraph $G_6^{(1)}$ of $G_6$ and its density $den(G_6^{(1)}) = 26.2$. Thus $G_6^{(1)}$ is included in $TOPK$. We proceed with the residual graph. For each component $h_i$, we compute the density of the maximum core subgraph. Since $2 \cdot den(C(h_1)) = 2 \cdot 10.5 < k$-th density, $h_i$ is not further considered. The removal of $G_1$ does not affect the $TOPK$ set and therefore it is straight-forward.

| | $G_1$ | $G_2$ | $G_3$ | $G_4$ | $G_5$ | $G_6$ | $G_7$ |
|---|---|---|---|---|---|---|---|
| $den(G_i^{(1)})$ | 18.0 | 25.2 | 20.2 | 30.8 | 22.2 | 26.2 | 17.6 |
| $den(G_i^{(2)})$ | 16.1 | 23.1 | 18.7 | 21.4 | 16.4 | 20.6 | 14.5 |
| $den(G_i^{(3)})$ | 14.5 | 18.4 | 16.3 | 15.6 | 13.2 | 18.5 | 12.7 |

**Table 2.** Last seven graphs of a stream and the density of their three densest subgraphs.

Now, assume that graph $G_7$ arrives and $den(C(G_7)) = 10.0$. The method computes the maximum core subgraph and its density. Since $2 \cdot den(C(G_7)) < k$ no further action are needed. However, due to arrival of $G_7$, graph $G_2$ expires. We should update $TOPK$ because now has only two densest subgraphs. The proposed method examines the first graph $G_3$ and includes the first two densest subgraphs, i.e. $G_3^{(2)}$ and $G_3^{(3)}$, to $TOPK$. Next, for each available graph (initial or component of a residual) is examined the density of the maximum core subgraph. If this density is less than the half of the $k$-th density, the graph is omitted. On the other hand, the naive approach computes the densest subgraphs of every available graph. The proposed method is denoted as StreamTopkDense. In the sequel, we propose two enhancements in order to further improve efficiency.

**Examination order of candidate graphs.** The most time consuming part of StreamTopkDense is the case where an element of $TOPK$ expires, because multiple densest subgraph discovery operations may be invoked. The first enhancement tries to reduce the number of examined graphs by means of a suitable examination order. In the proposed method, we favor graphs that have a maximum core subgraph with high density. We expect that during the answer set improvement, the number of examined graphs will be reduced. In order to achieve this, we use a priority queue to define the examination order of active graphs.

The maximum possible density $maxden(G)$ of a graph $G$ is used as the key to insert $G$ in the priority queue. We define $maxden(G)$ as $den(G^{(1)})$ if $G^{(1)}$ is available or $2 \cdot den(C(G))$ otherwise. The priority queue contains entries of the form $< G, maxden(G) >$. For two graphs $G_1$ and $G_2$, if $maxden(G_1) > maxden(G_2)$, then it is not necessary that $den(G_1^{(1)}) > den(G_2^{(1)})$. Therefore, if $G$ is at the top of the priority queue and it holds that $maxden(G) \geq k$-th density, then we should further examine $G$ before we insert it into $TOPK$. If $G^{(1)}$ is available (i.e., $maxden(G) = den(G^{(1)})$), we insert $G$ to the answer set immediately. Otherwise, we compute the subgraph $G^{(1)}$ and then we check its density to determine if its inclusion in $TOPK$ is necessary. The method extracts and examines the top of the priority queue, until a graph $G$ is found for which it holds that $maxden(G)$ is less than the current $k$-th best density.

**Graph pruning.** The second enhancement of StreamTopkDense algorithm identifies graphs that cannot be included in $TOPK$ and discards them in order to reduce processing time and memory requirements. The key observation is that a graph can be part of top-$k$ if it belongs to the answer set of the ($k$-1)-skyband query in the 2-dimensional space (time, density of densest subgraph). A similar approach has been followed in [11].

A $\delta$-skyband query reports all the objects that are dominated by at most $\delta$ other objects [12]. In our case, the maximization of the expiration time and the maximization of the density, determine the domination relationship between graph objects, i.e., a graph $H$ dominates another graph $G$ if $H$ has larger expi-

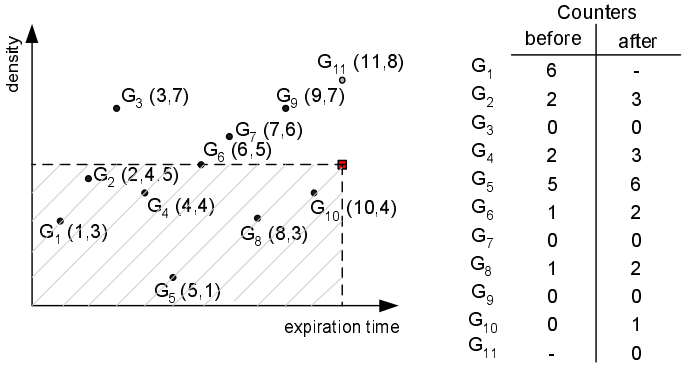



**Fig. 6.** Graph pruning example.

ration time and $H$ has a more dense subgraph than $G$. The use of graph pruning does not introduce false dismissals, according to the following lemma.

**Lemma 2.** *A graph $G$ could not be part of $TOPK$, if there are at least $k$ subgraphs which have greater density than that of the densest subgraph of $G$ and their expiration times are greater than the expiration time of $G$.*

*Proof.* Assume that there are $k$ subgraphs with density greater than that of the densest subgraph $G^{(1)}$ of a graph $G$ and these subgraphs expire later than $G$. Even in the case where all the current top-$k$ graphs are expired before $G$, any densest subgraph of $G$ will never be part of $TOPK$ since during its lifetime always exist $k$ subgraphs with greater density. □

It is evident that Lemma 2 can also be used to prune computed densest subgraphs from further consideration. In order to enable the use of $\delta$-skyband, we keep a counter $G.c$ for each graph in the active window. When the initial graph is inserted for the first time in the window, we initialize its counter to zero. Then, we scan all active graphs and for each graph $G$ we increase by one $G.c$, if the density of the densest subgraph of the new graph is greater than that of $G$. We preserve a graph $G$ as long as $G.c$ is less than $k$. If $G.c = k$ then we can safely discard $G$ from the active window, since it does not belong to the ($k$-1)-skyband set and therefore cannot be part of the answer throughout its lifetime.

Due to the use of Inequality 1 we do not have all the densest subgraphs and therefore their corresponding densities. The question is how we can use the max-core subgraph and its density, if the densest subgraph has not been computed? There are two cases to study: the densest subgraph is not available either for the new graph or for an existing active graph. To preserve the precision of the proposed method, we use the minimum possible density of the new graph and

the maximum possible density $maxden$ of the other existing graphs. The minimum possible density, $minden(G)$, is defined as $den(G^{(1)})$, if $G^{(1)}$ is available, or $den(C(G))$ otherwise.

Figure 6 gives an example of $\delta$-skyband pruning. Assume that the current time is 10, thus the most recent graph is $G_{10}$. We transform each active graph, which is not part of $TOPK$, to (density, expiration time)-space by using the pair $< maxden, exptime >$. The values of these attributes are shown in parentheses in Figure 6. When a new graph arrives (graph $G_{11}$), we use the pair $< minden, exptime >$ to update the counters of the other active graphs. The left column shows the counters $c$ of the graphs before update while right column shows them after update. Moreover, we discard $G_1$ and its counter, we set $G_{11}.c = 0$ and we use the pair $< maxden, exptime >$ to transform $G_{11}$ to (density, expiration time)-space. For $k = 1$, assume that $G_9$ contains the most densest subgraph. Due to Lemma 2, it is sufficient to store graphs $G_3$, $G_7$ and $G_{11}$ (i.e., graphs with $G.c \leq 0$). The remaining graphs can be discarded.

Recall that the density of the densest subgraph is used to prune graphs, since it affects the value of the counters of existing graphs. By using the minimum possible density of the new graph and the maximum possible density for all other existing graphs, we ensure that we do not prune graphs which are part of the ($k$-1)-skyband and therefore the proposed method does not introduce any false dismissals. We integrate both enhancements in StreamTopkDense algorithm. The new algorithm is denoted as StreamTopkDense*.

## 4 Performance Evaluation Study

In the sequel, we report some representative performance results showing the efficiency of the proposed techniques. All algorithms have been implemented in JAVA and the experiments have been conducted on a Pentium@3.2GHz.

To study the performance of the algorithms we have used both synthetic and real-life data sets. The synthetic graphs have been generated by using the GenGraph tool [17]. This generator produces graphs according to power law degree distributions. In particular, GenGraph generates a set of $n$ integers in the interval $[d_{min}, d_{max}]$ obeying a power law distribution with exponent $a$. Therefore, according to the degree distribution produced, a random graph is generated. The default values for the parameters of the generator are: $a \in [1.5, 2.5]$, $d_{min} = 1$, $d_{max} = 1\%$ of the number of vertices. The synthetic data set contains 10,000 graphs each with at most 1500 vertices. The real-life data set, AS-733, contains 733 daily instances (graphs) representing autonomous systems from University of Oregon. The real data set is available for download by the SNAP website at http://snap.stanford.edu/data/as.html.

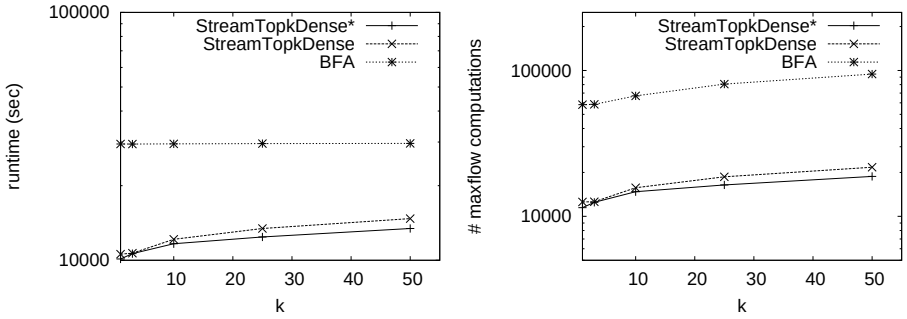
We study the performance of the algorithms by varying the most important parameters, such as the window size ($w$) and the number of results ($k$). We examined several features of the algorithms, such as computational time, the time required for maxflow computations, number of maxcore computations, etc. The computational cost is represented by the running time. All measurements

correspond to the total number of updates required, and this translates to $w$ updates required to shift the whole window $w$ times. The default values for the parameters, if not otherwise specified, are: $w = 2000$ and $k = 10$. The straightforward approach, which performs GOLD invocations directly, is denoted as BFA (brute-force algorithm).

### 4.1 Performance of Exact Algorithms

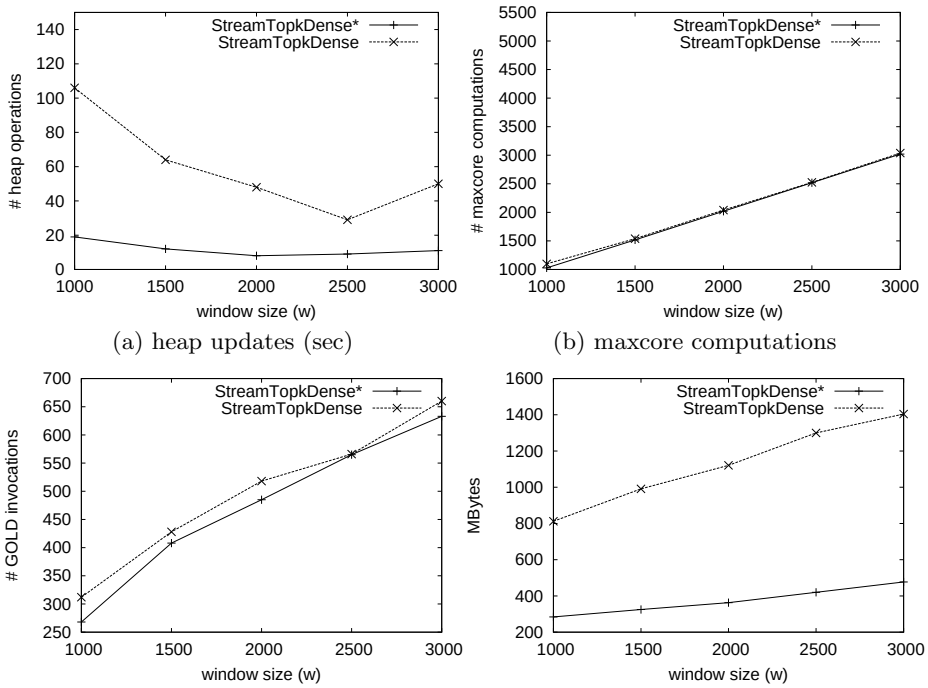
Figure 7 depicts the performance of the algorithms for the synthetic data set, for several values of $k$. As expected, the straight-forward solution (BFA) shows the worst performance due to the excessive number of GOLD invocations. Since GOLD requires $O(\log n)$ maxflow computations, the cost is dominated by the method implementing the maxflow. In particular, it is observed that the runtime of BFA does not heavily depend on the number of results ($k$). BFA performs a significant number of maxflow computations, as shown in 7(b). However, many of these maxflow computations are executed on small graphs and thus, they are fast. In contrast, a maxflow computation over a larger graph is more costly. Note also, that the $y$ axis is in logarithmic scale and thus, small changes are not easily detected. On the other hand, StreamTopkDense and StreamTopkDense$^*$ perform much better. In particular, StreamTopkDense$^*$ shows the best overall performance since the two optimizations applied have a significant impact in cost reduction, as shown later.

This is also depicted in Figure 8 which shows the performance of the two stream-based algorithms. Algorithm BFA is excluded from any further experiment since its performance is by orders of magnitude inferior. Again, it is evident that the optimizations applied to StreamTopkDense manage to reduce the number of maxcore computations, the number of heap operations and the number of invocations of the GOLD algorithm. Another important feature of StreamTopkDense* is that it requires significantly less storage than the other algorithms. This is illustrated in Figure 8(d) which shows the memory requirements




(a) runtime (sec) (b) maxflow computations

**Fig. 7.** Comparison of algorithms for different values of $k$ (synthetic data set).




(a) heap updates (sec) (b) maxcore computations

(c) GOLD invocations (d) memory requirements

**Fig. 8.** Comparison of algorithms wrt window size (synthetic data set).

| **Parameter k** | **StreamTopkDense** | **StreamTopkDense$^+$** | **StreamTopkDense$^*$** |
|---|---|---|---|
| $k = 1$ | 120 | 91 | 89/2040 |
| $k = 3$ | 26 | 6 | 5/1986 |
| $k = 10$ | 48 | 14 | 8/1984 |
| $k = 25$ | 156 | 32 | 30/1961 |
| $k = 50$ | 341 | 240 | 190/1962 |

**Table 3.** Evaluation of pruning.

in MBytes vs. the window size. Note that in comparison to the simple version of the stream-based algorithm, the advanced one manages to keep storage requirements low. Since many graphs are pruned due to the application of the skyband technique, the number of graphs that must be kept in memory is reduced.

In addition, Table 3 shows the performance of the pruning techniques. Specifically, we can see the number of graphs which we consider to find dense subgraphs that can participate in the Top $k$ for each algorithm over the total number of updates. The first column shows the pruning results of the StreamTopkDense algorithm. In this algorithm, the computations are reduced by using the density

of the maximum core subgraph as a pruning criterion. The second column shows the pruning capabilities of StreamTopkDense$^+$, which is the basic algorithm enhanced by a priotity queue to determine the examination order of active graphs. As expected, StreamTopkDense$^+$ examines less graphs than the basic algorithm. Finally, the third column presents the pruning power of StreamTopkDense$^*$ which uses the skyband pruning technique in addition to the priority queue. As expected, the number of examined graphs (i.e., the number of graphs that we must apply the exact density computation) is smaller for StreamTopkDense$^*$ than the other two methods. In addition, the third column represents also the number of graphs which are pruned by the skyband pruning technique. A graph that is pruned is also removed from the priority queue, resulting in reduced memory requirements (as shown in Figure 8(d)).

In conclusion, StreamTopkDense$^*$ shows the best performance both in terms of running time and memory requirements. The reason for this is threefold: *i*) the reduction of the number of GOLD computations (which consequently reduces the number of maxflow computations), *ii*) the use of an appropriate examination order for the graphs resulting in a more effective pruning and *iii*) the use of the skyband to exclude graphs and subgraphs from further consideration.

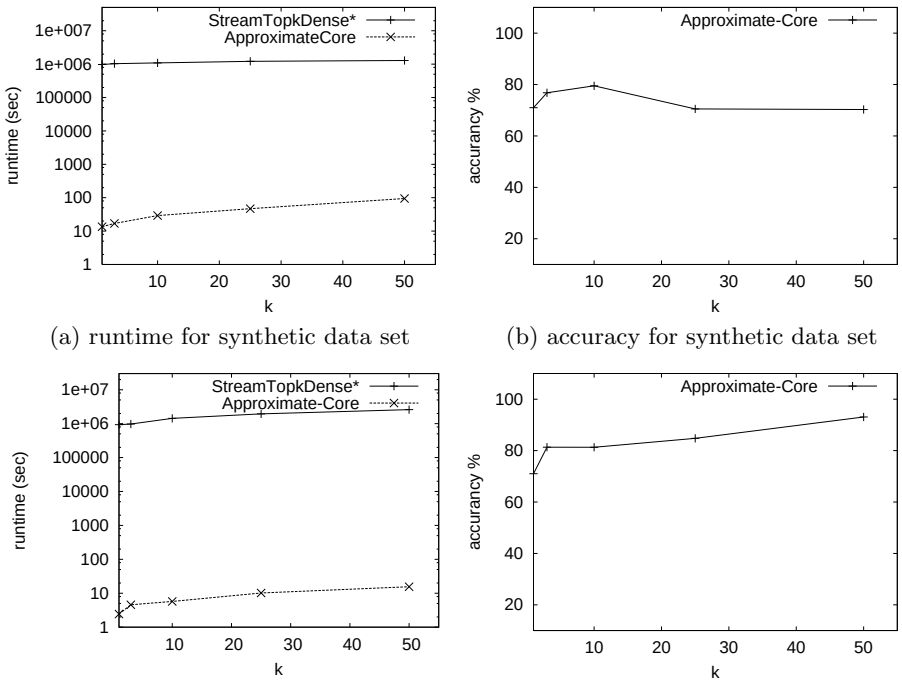
### 4.2 Trading Accuracy for Speed

In many cases, it is important to get the answers as quickly as possible, even if the accuracy of the result is not perfect. The importance of quickly answers is more obvious when the data is from real applications.Toward this direction, we study the performance of an approximation algorithm, which uses only the core computation to determine the set of graphs containing the top-$k$ dense subgraphs. The accuracy of the algorithm is defined as the percentage of the graphs containing the $k$ densest subgraphs that have been reported over the set of the correct subgraphs.

The results of this study are given in Figure 9, where both the runtime and the accuracy are reported for the synthetic as well as the real-life data set. As shown, the runtime of the approximation algorithm is several orders of magnitude smaller than that of the exact one, because the approximation algorithm does not perform any maxflow computations, whereas each core computation is performed in linear time with respect to the number of edges. The performance of the algorithms is similar for the synthetic and the real-life data set. Based on the runtime comparison, the approximate algorithm can be applied when the arrival rate is large, and thus, each new graph must be processed as efficiently as possible. The quality of the approximation is at least 70% for all experiments conducted, whereas it reaches 90% in the real-life data set as $k$ grows to 50.

## 5 Concluding Remarks

Dense subgraph discovery is considered an important data mining task. Although similar to clustering, dense subgraph discovery has evolved as a separate problem because the requirements are fairly different than clustering. In this paper,




(a) runtime for synthetic data set (b) accuracy for synthetic data set

(c) runtime for real data set (d) accuracy for real data set

**Fig. 9.** Runtime-accuracy trade-off for synthetic and real data sets.

we presented stream-based algorithms for continuous monitoring of the $k$ densest subgraphs from a dynamic collection of graphs. The proposed algorithm, StreamTopkDense$^*$ is the most efficient variation, because it has the lowest running time and shows the lowest memory consumption. The key issues in the design of our proposal have as follows: *i*) the maxcore is used to define an upper bound on the density of a subgraph, *ii*) a priority queue is used to enforce a particular examination order of the graphs and *iii*) the skyband concept is applied (as in [11]) to reduce the number of graphs that should be considered.

In addition to the study of exact algorithms, we studied the performance of an approximation algorithm, which uses solely the concept of maxcore to determine the set of graphs containing the top-$k$ dense subgraphs. This algorithm offers an accuracy of at least 70%, whereas its running time is orders of magnitude better than that of the exact algorithms.

We point out that this is the first work that performs dense subgraph discovery in a top-$k$ fashion. This technique alleviates the requirement for posing density constraints which are sometimes difficult to provide, especially when the graph collection evolves and graph properties change over time.

There are several interesting directions for future work. The first one, involves the use of graph summaries in order to reduce the size of the graphs. This means that we are willing to sacrifice accuracy in favor of a more efficient computation. A second direction is the adaptation of the methods in [2, 4] to work in a top-$k$ scenario, without the requirement of density constraints.

## References


1. C. Aggarwal, H. Wang. *Managing and mining graph data*, Springer, 2010.
2. C. Aggarwal, Y. Li, P. S. Yu, R. Jin. On dense pattern mining in graph streams. *Proceedings of the 36th VLDB Conference*, pp.975-984, 2010.
3. V. Batagelj, M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.
4. L. Chen, C. Wang. Continuous subgraph pattern search over certain and uncertain graph streams. *IEEE Transactions on Knowledge and Data Engineering*, 22(8), pp.1093-1109, 2010.
5. D. J. Cook, L.B. Holder (eds). *Mining graph data*, Wiley, 2007.
6. D. Gibson, R. Kumar, A. Tomkins. Discovering large dense subgraphs in massive graphs. *Proceedings of the 31st VLDB Conference*, pp.721-732, 2005.
7. A. V. Goldberg. Finding a maximum density subgraph. Technical Report CSD-84-171, University of Berkeley, 1984.
8. H. Hu, X. Yan, Y. Huang, J. Han, X.J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(1), pp.i213-i221, 2005.
9. G. Kortsarz, D. Peleg. Generating sparse 2-spanners. Journal of Algorithms, 17(2), pp.222-236, 1994.
10. T. Luczak. Size and connectivity of the k-core of a random graph. *Discrete Mathematics*, 91(1), pp.61-68, 1991.
11. K. Mouratidis, S. Bakiras, D. Papadias. Continuous monitoring of top-k queries over sliding windows. *Proceedings of the ACM SIGMOD Conference*, pp.635-646, 2006.
12. D. Papadias, Y. Tao, G. Fu, B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1), pp.41-82, 2005.
13. B. Saha, A. Hoch, S. Khuller, L. Raschid, X.-N. Zhang Dense subgraphs with restrictions and applications to gene annotation graphs. *Proceedings of the 14th Annual International Conference on Research in Computational Molecular Biology*, pp.456-472, 2005.
14. S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5, pp.269-287, 1983.
15. J. Sun, C. Faloutsos, S. Papadimitriou, P. S. Yu. GraphScope: parameter-free mining of large time-evolving graphs. *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.687-696, 2007.
16. Y. Tao, D. Papadias. Maintaining sliding window skylines on data streams, *IEEE Transactions on Knowledge and Data Engineering*, 18(3), pp.377-391, 2006.
17. F. Vigen, M. Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. *Proceedings of the 11th International Conference on Computing and Combinatorics*, pp.440-449, 2005.