2016 IEEE 16th International Conference on Data Mining Workshops

Finding Heaviest k-Subgraphs and Events in Social Media

Matthaios Letsios
Institut Mines Telecom, Telecom Paristech
letsiosm@gmail.com

Oana Denisa Balalau
Institut Mines Telecom, Telecom Paristech
balalau@telecom-paristech.fr

Maximilien Danisch
Institut Mines Telecom, Telecom Paristech
danisch@telecom-paristech.fr

Emmanuel Orsini
Google Inc., Zurich
emmanuel.orsini@polytechnique.edu

Mauro Sozio
Institut Mines Telecom, Telecom Paristech
sozio@telecom-paristech.fr

***Abstract*—In recent years, social media have become a useful tool to stay in contact with friends, to share thoughts but also to be informed about events. Users can follow news channels, but they can be the ones reporting updates, which distinguishes social media from traditional media. In this paper, we use a graph mining approach for finding events in a graph constructed starting from posts of users. We develop an exact algorithm for solving the heaviest k-subgraph problem which is an NP-hard problem. Our experimental analysis on large real-world graphs shows that our algorithm is able to compute the exact solutions for k up to 15 or more depending on the structure of the graph. We also develop an approximation version of our algorithm scaling to larger k. In comparison, for this setting, the classical heuristic based on weighted core decomposition only leads to sub-optimal solutions. Finally, we show that our algorithm can be used to find relevant events in Twitter. Indeed, as an event is usually described by a small number of words, our algorithm is a useful tool to detect them.**

I. Introduction

There is an urgent need to develop efficient algorithms that are able to make sense of the unprecedented amount of data produced daily by social media users, such as users of Facebook, Twitter, etc.

Osborne et al. [17] compared Twitter, Facebook and Google Plus in order to determine which online social network would offer a better coverage of world news. While all three social media perform comparably well in terms of coverage, Osborne et al. [17] observed that in Twitter interesting news are reported in a more timely fashion. Twitter has also the additional advantage of providing free access to the stream of tweets posted by the users through a streaming API. Given these advantages, we will focus on event detection on Twitter.

Algorithms for finding dense subgraphs (i.e. subgraphs with a relative large number of edges) have proved to be an effective tool in data analysis with applications in community detection [9], finding patterns in gene annotation graphs [18], link spam detection [11], as well as event detection in social media [1].

The data is represented as an undirected weighted graph representing the co-occurrence between relevant terms mentioned in tweets. In such a graph, nodes represent relevant terms (such as countries, terms such as 'earthquake', 'shooting', etc.) while each edge indicates whether the two corresponding terms co-occur together in tweets. Each edge is associated with a positive weight measuring the number of co-occurrences of the corresponding terms in tweets.

As events unfold, terms related to the event start to co-occur often in tweets leading to the emergence of a dense subgraph. The average degree of a subgraph is a widely used measure of its density. The problem of finding a subgraph with maximum average degree is called the densest subgraph problem and can be solved in polynomial time using a parametric maximum flow algorithm [12].

However, in some applications such as event detection, densest subgraphs might be large and difficult to analyze. In order to cope with this limitation, we consider the problem of finding dense subgraphs under size restriction. In particular, given a weighted graph we wish to find a subgraph with k nodes with maximum total edge weight. This problem is referred in the literature as the heaviest k-subgraph problem (HkS) and also as the weighted version of the densest k-subgraph problem. The problem is NP-hard and difficult to approximate. In our work we leverage the properties of real-world graphs, so as to develop efficient algorithms.

We summarize our contributions as follows.

- We develop an efficient (exact) branch and bound algorithm to solve the heaviest k-subgraph problem. Our algorithm scales to large weighted real-world networks, for k up to 15 or more depending on the structure of the graph. We also develop an approximated version of our algorithm scaling to even larger values of k. We show that our algorithms are more effective than state-of-the-art heuristics for the same problem. Our code in C is publicly available[1].
- We show that our algorithm is better suited as a subroutine for solving related problems, like finding the top t HkS in a graph. In social media, more than one event is discussed in the same time frame, which might correspond to several heavy subgraphs.
- We include a case study showing that HkS correspond to relevant events in Twitter.

[1]https://github.com/maxdan94/HkS

2375-9259/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDMW.2016.124

The rest of the paper is organized as follows. In Section II we define formally the problems we will attempt to solve, then we present the related work concerning dense subgraph identification applied to event detection in Section III. In Section IV, we present our algorithms for solving the problems. We then evaluate the performance of our algorithm to detect events in Section V. Finally, we conclude and present future work in Section VI.

II. Problem Definition

In this section, we give a formal definition of the problems we study in our work. We also introduce necessary notations and definitions that shall be used in the rest of the paper. We shall assume that we are given an undirected graph $G = (V(G), E(G))$ and a weight function $w : E(G) \rightarrow R^+$.

Problem definition (*Heaviest-k-Subgraph problem*). Given an undirected weighted graph G and an integer $k > 1$, we wish to find a subgraph containing k nodes and such that the sum of the weights its edges is maximum. For this problem we assume the graph has at least k nodes.

In order to treat the problem for large k we also define an approximate version of the problem.

Problem definition (*Heaviest-k-Subgraph α-approximation problem*). Given an undirected weighted graph G, an integer $k > 1$ and a real number $\alpha \geq 1$, we wish to find a subgraph containing k-nodes and such that the sum of the weights on its edges times α is greater or equal to the sum of the weights on the edges of any subgraph of size k.

Problem definition (*Top t heavy k-subgraphs problem*). Given an undirected weighted graph G, an integer $k > 1$, and an integer $t > 0$, find at most t disjoint subgraphs containing k nodes such that the sum of the weights on its edges is maximum. For this problem we assume the graph has at least $k \cdot t$ nodes.

Decomposing an unweighted graph into a hierarchical structure via the core decomposition is a standard operation in any modern graph-mining toolkit. This decomposition, is based on a recursive pruning of a vertex of minimum degree and is used as a subroutine in a large variety of algorithms, in particular it is related to the problem of finding densest subgraph (in unweighted graph and without constraints on the number of nodes in the subgraph) as it leads to a 2-approximation algorithms. It can be straightforwardly generalized to the weighted case where we recursive prune a vertex such that the sum of the weights on its adjacent edges is minimum.

Problem definition (*Weighted core decomposition problem*). Given an undirected weighted graph G, we wish to compute a weighted core decomposition of G.

We will show that an efficient heuristic for the Heaviest-k-Subgraph problem can be derived from the Weighted core decomposition. However, this solution has no fixed parameter approximation guaranteed for our setting.

III. Related Work

Heaviest k-subgraph. Both HkS and DkS (unweighted version of HkS) problems are NP-hard and no polynomial-time algorithms with a fixed performance guarantee are known. In [7] the authors give a polynomial algorithm that computes a solution within a factor of n^{α}, $\alpha < \frac{1}{3}$, from the optimum solution for DkS, with the addition of a factor of $O(\log n)$ for HkS. Note that an algorithm that solves the weighted case will solve the unweighted case without any additional approximation factor. Asahiro et al. [2] describe a greedy algorithm for HkS that has an approximation ratio of $O(\frac{k}{n})$. In [8], using semidefinite programming, the authors get approximation ratios of $\frac{k}{n}$ for HkS. The state-of-the algorithm for DkS is due to Bhaskara et al. [3] and gives a $O(n^{\frac{1}{4}+\epsilon})$ approximation guarantee for any $\epsilon > 0$. In [3], the authors count selected subgraphs of constant size in G, and use these counts to find the vertices of the dense subgraph.

Finding the densest subgraph with at most k vertices $(DamkS)$ or a densest subgraph with at least k vertices $(DalkS)$ are also NP-hard [13]. Khuller et al.[13] give a $1/2$-approximation algorithm for $(DalkS)$ and show that $DamkS$ is as hard as DkS within a constant factor. When the constraint on the number of nodes is removed and the objective is to maximize the average degree of the nodes in a subgraph, then the problem becomes the densest subgraph problem. It is well studied in literature and it can be solved in polynomial time despite the fact that there is an exponential number of subgraphs to consider. Goldberg [12] formally defined the problem in an undirected graph and presented an algorithm that computes a densest subgraph in $O(log(n))$ maximum-flow computations. In [4], Charikar describes a simple heuristic that has a 2-approximation guarantee.

Event detection. Promising research in this field includes the work of Angel et al. [1] based on dense subgraph discovery. The algorithm finds all (possibly overlapping) sub-graphs that have the density above a certain threshold and presents these sub-graphs as corresponding to events. There are also approaches that target a slightly different problem: given an event, the goal is to keep track of all updates and major sub-events concerning that event. In [6] the authors use only non-textual features in order to discover sub-events related to a given an event. They present a model which is based on the intuition that users tend to communicate less with each other while an event is occurring. Using a logistic regression approach they detect goals during the soccer World Cup of 2010. In [16] tweets are represented as a graph and sub-events are identified using the notion of graph degeneracy.

IV. Algorithms

A. Branch and Bound Algorithm for HkS

Branch and bound is a well-known method for solving combinatorial maximization problems. The main idea is to divide the search space into several branches of computation. Intuitively, we can think of the whole process as forming a tree starting from the root which is the set of all possible

solutions, while the children of a node are smaller sets of solutions. Forming the children of a node is called *branching phase*. Each node of the tree is associated to a lower bound (generally a solution of the problem) and an upper bound, while if the upper bound of a node in the tree is lower than the global lower bound (that is, the maximum of the solutions found so far), the children of the node do not need to be explored as they would lead to a worse solution, so the branch can be pruned.

In our approach, the branching phase is based on deciding whether to add a specific edge (and thus its endpoints) in the corresponding solution or not. More specifically, if we are looking for the heaviest subgraph of size k, our branch and bound algorithm consists of the following.

- We start with the root such that all edges are possibly here or not. The upper bound is the sum of the $\binom{k}{2}$ heaviest edges, while the associated solution is the empty subgraph, the lower bound is thus 0.
- Then at each iteration we create two children for the node with maximum lower bound (i.e. density of the associated solution). Suppose the node is at depth i in the tree, we keep the decisions made on the first $i-1$ edges and create two children, one where the i^{th} edge is included and one where it is not.

The lower bound is given by the sum of the weights on the edges of the subgraph induced by the edges that are included.

Assume the number of nodes in the subgraph induced by the edges that are included is s. Then the upper bound is given by the weights of subgraph plus the sum of the next $\binom{k}{2} - \binom{s}{2}$ highest weight edges.

When we add a new edge (u, v) we have to check if both endpoints are part of the current's solution vertex list. If they are not, we add them in the vertex list and we update the weight of this solution by adding the weight of every possible edge between u (resp v) and vertices in S. We thus need a subroutine to check efficiently if two nodes are adjacent or not.

Checking adjacency efficiently. We initially compute a core ordering (or degeneracy ordering) of the unweighted version of the graph. We then keep for each node in the graph a sorted list of its neighbors having higher core ordering (the maximum size of such a truncated neighborhood list is thus c, the core value of the graph). Given 2 nodes x and y (w.l.o.g. we assume that y has a higher core ordering than x) we can efficiently check if they are adjacent by checking if y is in the truncated neighborhood list of x by binary search in $O(\log(c))$ time. Note that this step is in practice the bottleneck of our algorithm taking about 80% of the time. We were not able to make this subroutine faster in practice using a hashtable containing all edges.

The worst case total running time of the process of creating a child is thus in $O(k \cdot \log(c)$ (where c is the core value of the graph), which is the time to check if u and v belongs to the neighborhood of the nodes in the solution subgraph (there are at most $k-1$ of them), note that the maximum size of a neighborhood is $n-1$.

We keep forming the children of the node with the higher lower bound till the branch is pruned. In addition to the pruning using lower and upper bound, if the number of nodes in the subgraph exceeds k we also prune the branch. When we add a new edge (u, v), we have to check for both endpoints if they belong in the current solution. If they don't, we add the weight of the edges between the endpoints of the new edge (u, v) and the vertices of the current solution. However, if such an edge (u, x) has a higher ranking than (u, v) (i.e. the weight of (u, x) is greater or equal to the weight of (u, v)), then we can prune this solution. Indeed, since we make the decisions on edges in non-increasing order according to their weight it means that this edge (u, x) was excluded from the solution, it thus cannot be added at this step.

A key feature of a branch and bound algorithm is the order by which the method iterates over the nodes of the branch and bound tree. This order can be BFS, DFS, based on the lower bound or on the upper bound of a node. After examining those different order, we found that examining the nodes according to the weight of the candidate solutions was the most efficient one. Hence, every time a child is created by the branch and bound algorithm, it is added to a heap, whose head is always the solution of maximum weight.

Intuition behind our method. The intuition behind our method is that the best solution should contain many edges of high weight (that is well ranked in non-increasing order of weight), while possibly containing few edges of low weight. The method should not explore all edges till the ones of low weights as those will be added to the solution through forming the induced subgraph on the edges of high weight.

Branch and bound on nodes. During the branching phase where we make decisions upon the edges, one might ask if it would be better to make decisions upon the vertices. After some experimentation with both approaches, we observed that branch and bound with edges is more efficient. Indeed, a good upper bound for the branch and bound with edges is easy to compute, while we couldn't find any simple way to do so for the branch and bound with nodes. In addition, ranking edges in non-increasing order of weight is rather natural and it is more probable that the best solution contains edges of high weight leading to a quicker termination of the algorithm. While ranking nodes is less natural (even though weighted degree ordering or weighted-core ordering are possible) and nodes in the best solutions can be ranked further leading to a slower termination of the algorithm.

Approximation algorithm for HkS. In order to obtain an α approximation of the HkS problem, we modify our branch and bound algorithm in the following way. If at some point, the current largest upper bound divided by the current best solution is higher than α, we output the solution. As there is no other subgraph of size k such that the sum of the edges is higher than the largest upper bound, we know that our solution

Algorithm 1 Branch and bound

Input: A graph $G(V, E)$ and an integer k
Output: The heaviest k-subgraph of $G(V, E)$
Let $edges$ be the edge list sorted in non-increasing order according to the weight
For a current BnB node, let S be the list of vertices of the subgraph, w its weight, b the upper bound and i the index of the current edge.
Initially $S \leftarrow \emptyset, w \leftarrow 0, i \leftarrow 0, b \leftarrow$ sum of the heavier $\frac{k(k-1)}{2}$edges
HEAP.insert((S, w, i, b))
$bestSolution \leftarrow 0$
while HEAP is not empty **do**
    $(S, w, b, i) \leftarrow$ HEAP.$pop()$;
    if $b \leq bestSolution$ or $|S| > k$ **then**
        Prune current solution
    else
        if $w > bestSolution$ **then**
            $bestSolution \leftarrow w$
        $(S_1, w_1, b_1, i_1) \leftarrow$ createChild((S, w, b, i) ,1)
        HEAP.insert((S_1, w_1, b_1, i_1))
        $(S_0, w_0, b_0, i_0) \leftarrow$ createChild((S, w, b, i) ,0)
        HEAP.insert((S_0, w_0, b_0, i_0))
return best solution found;

Algorithm 2 Procedure createChild((S, w, b, i), int c)

if $c = 1$ **then**
    $e \leftarrow edges[i]$;
    Let (u, v) be the endpoints of edge e
    if $u \notin S$ **then**
        for all $x \in S$ **do**
            if $(u, x) \in E$ **then**
                $w \leftarrow w + (u, x).weight$
        $S \leftarrow S \cup \{u\}$
    if $v \notin S$ **then**
        for all $x \in S$ **do**
            if $(v, x) \in E$ **then**
                $w \leftarrow w + (v, x).weight$
        $S \leftarrow S \cup \{v\}$
$i \leftarrow i + 1$;
Update b;
return (S, w, b, i)

is an α-approximation.

Top heavy k-subgraphs. Due to the hardness of the problem we cannot hope to solve it optimally, so we use a simple greedy heuristic: we find the heaviest k-subgraph, we remove all its vertices and their incident edges from the graph and we iterate on the remaining graph until we have found t subgraphs or the graph is empty. At the end of the computation, we will obtain at most t disjoint subgraphs. Note that this heuristic gives a $\frac{1}{t}$-approximation guarantee as the first found heaviest subgraph is at least as heavy as any subgraph in the optimal solution containing t subgraphs.

B. Heuristic based on weighted core decomposition

In order to have a basis for comparison, we implement the greedy heuristic used in [2]. The algorithm is based on the weighted core decomposition and it consists in repeatedly removing a vertex with the minimum weighted-degree in the currently remaining graph until exactly k vertices are left. We implemented the algorithm using a min-heap as detail in Algorithm 3, so that its asymptotic running time is $O((m + n) \cdot \log(n))$.

Algorithm 3 Weighted core heuristic for DkS

for each node u in G **do**
    $d_w(u) \leftarrow$ weighted-degree of u
    $\Delta_\eta(u) \leftarrow$ list of neighbors of u and associated edge weight $(v, w_{u,v})$
create a min heap HEAP initialized with $(u, d_w(u))$ for each node u in G
while HEAP has more than k elements **do**
    $u \leftarrow$ HEAP.$pop()$
    for each $(v, w_{u,v}) \in \Delta(u)$ **do**
        $d_w(v) \leftarrow d_w(v) - w_{u,v}$
        update HEAP with $(v, d_w(v))$
return the k nodes in HEAP

One essential drawback of this algorithm is that it does not have any fixed parameter approximation guarantee. As proved in [2], for $k < \frac{n}{3}$ it has a $O(\frac{k}{n})$ approximation guarantee, however for our setting k is small (events are described by a small number of entities, in practice we work with k smaller than 20), while n is large and thus the approximation guarantee can be very bad.

Next we show that for any $\alpha < 1$, we can find a graph where the ratio between the solution the weighted core heuristic and the optimal one is smaller than α. This can be easily seen in Figure 1, for $k = 2$, we modify the edge weight between u and v to be an integer δ, satisfying $\delta > \frac{1}{\alpha}$ and we keep a clique of size $(\delta + 2)$ where every edge has unitary weight. This can be extended also for larger k.

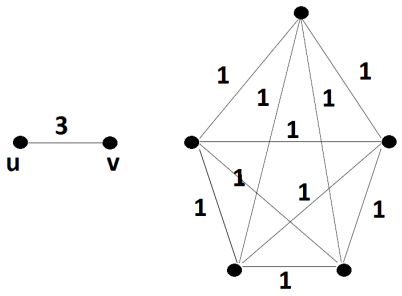

Fig. 1. On this graph for $k = 2$ the best solution is the subgraph induced by the vertices u, v with total weight 3. However, the weighted core decomposition algorithm will exclude first those two vertices since the contribution of every vertex in the clique is 4 and theirs is 3.

Improving the solution by local search. Note that the solution given by Algorithm 3 is a stable local optimum in the sense that adding a node such that the weight of the subgraph of size $k+1$ is maximized and then removing the node such that the size of the subgraph of size k is maximized will lead to a solution having the same value as the initial one. However we can improve the solution by switching a node inside the subgraph of size k and a node outside the subgraph such that the weight is maximized, this can be repeated till the solution becomes a stable local optimum with respect to this switch operation. This can be done efficiently by considering only the nodes inside the subgraph and nodes outside the subgraph having at least one neighbor inside the subgraph. We also add this technical improvement which does not lead to a better approximation but gives, in practice, slightly better results.

V. Experiments

A. Experimental setup

We collected a set of tweets by means of the Twitter Streaming API during the months of November and December 2015. The sample of tweets contains only English and French tweets (language is automatically detected by the Twitter platform).

We construct graphs from each of the dataset as follows. We extract nouns from the tweets using the Stanford POS Tagger [2] and also hashtags and we construct a weighted undirected graph $G = (V, E, w)$, where the set of nodes consists of the words extracted in the previous step, while there is an edge between two nodes if the corresponding words co-occur in at least one tweet, the weight corresponds to the number of co-occurrences.

We obtain four graphs which are detailed Table I.

Name	Nodes	Edges
French-November	220 K	2.9 M
French-December	200 K	2.1 M
English-November	2.5 M	4.5 M
English-December	1.8 M	3.1 M

TABLE I
Our set of graphs of words extracted from Tweets.

The algorithms were implemented in Java and were run on a machine under GNU/Linux with 2.39 GHz clock, while limiting the total amount of memory available to 20 GB.

B. Running time and structural comparisons

Table II shows the running time of our branch and bound algorithm as well as the one based on weighted core decomposition on our set of graphs of words extracted from Twitter. The time of the heuristic based on weighted core does not vary much with k as it computes the weighted core, improve the solution using local search and then outputs the last k nodes we show the average value. As we can see, in most cases, both algorithms take few seconds for $k \leq 15$. However, as we will see later and in Figure 2, the time of the branch and bound algorithm explode when k is larger. This can be explained as when k is large, the probability that the solution contains many edges of high weight and only few edges of low weight becomes lower. This shows a limitation of our approach which is not efficient when k is too large, however for small k we can find the exact solution efficiently. Note that for event detection, only small k values are interesting as events are usually described by a small set of relevant keywords.

Table III shows the sum of the weights on the obtained subgraph. Our branch and bound algorithm always achieve to find the optimal solution. We used the value of the solution of our branch and bound algorithm to compute the ratio of the solution obtained by the heuristic based on weighted core and the optimal solution. As we can see the approximation of the weighted core heuristic is variable: sometimes as good as 0.99, while it can be as bad as 0.67.

Table IV shows the ratio between the number of edges in the found subgraph of size k and the number of edges in a clique of size k. As we can see, in all cases except the one of "Twitter-fr December" the solution is a clique. This shows that the structure of those real-world graphs is special and that, even though we are looking for HkS and not cliques, designing an algorithm to find a clique of size k and of maximum weight is a very similar problem on those real-world graphs. As enumerating cliques can be done efficiently in real-world graphs [5], [15], a heuristic based on enumerating all cliques of size k and returning the one of maximum weight could be considered.

Approximation for larger k. Figure 2 (left) shows the running time of our exact branch and bound algorithm as a function of the input k (number of nodes in the subgraph) for our four datasets. We truncated the curves at 1 hour of computation. As we can see, within one hour our program can solve the problem for $k = 16$ for French-December while it can go up to $k = 36$ for English-December. We also show, Figure 2 (right) the same curves, but for our approximation branch and bound algorithm. We set the wished approximation ratio to $\alpha = 1.5$. This time our algorithm is able to solve the problem for larger k: within one hour our program computes a 1.5 approximation for $k = 20$ for French-December while it can go up to $k = 105$ for English December.

C. Top t heavy k-subgraphs

In this section we present the events that we found using our technique, i.e. computing heavy disjoint subgraphs in the graphs French-November (Table V) and French-December (Table VI). Through experimentation we noticed that the most relevant value for k is 5, meaning that events are usually described using a few important keywords.

In the month of November the heaviest subgraph corresponds to the terrorist attacks in Paris. Two of the places where the attacks occurred are mentioned, the concert hall Bataclan and the stadium Stade de France. Another interesting event found by our method is a police raid in a Paris suburb, which

[2] http://nlp.stanford.edu/software/tagger.shtml

networks	loading time	Weighted core	B&B DkS		
		$k = 5$, 10 and 20	$k = 5$	$k = 10$	$k = 15$
French-November	0.8s	0.7s	0.5s	0.9s	17s
French-December	0.6s	0.5s	0.9s	1.0s	9m58s
English-November	11s	8.9s	17s	17s	18s
English-December	12s	4.8s	7.5s	7.0s	7.0s

TABLE II
RUNNING TIME COMPARISON.

networks	Weighted core			B&B DkS		
	$k = 5$	$k = 10$	$k = 15$	$k = 5$	$k = 10$	$k = 15$
French-November	24669 (0.92)	70528 (0.96)	115341 (0.96)	26824	73432	120419
French-December	18158 (0.95)	22458 (0.67)	36358 (0.81)	19090	33338	45020
English-November	2942467	5836705 (0.99)	8114183 (0.99)	2942467	5913372	8161838
English-December	2386716	5125458 (0.98)	7830637	2386716	5235181	7830637

TABLE III
SUM OF WEIGHTS.

networks	Weighted core			B&B DkS		
	$k = 5$	$k = 10$	$k = 15$	$k = 5$	$k = 10$	$k = 15$
French-November	1.0	1.0	0.98	1.0	1.0	0.98
French-December	0.4	0.27	0.44	1.0	0.44	0.3
English-November	1.0	1.0	1.0	1.0	1.0	1.0
English-December	1.0	1.0	1.0	1.0	1.0	1.0

TABLE IV
EDGE DENSITY.

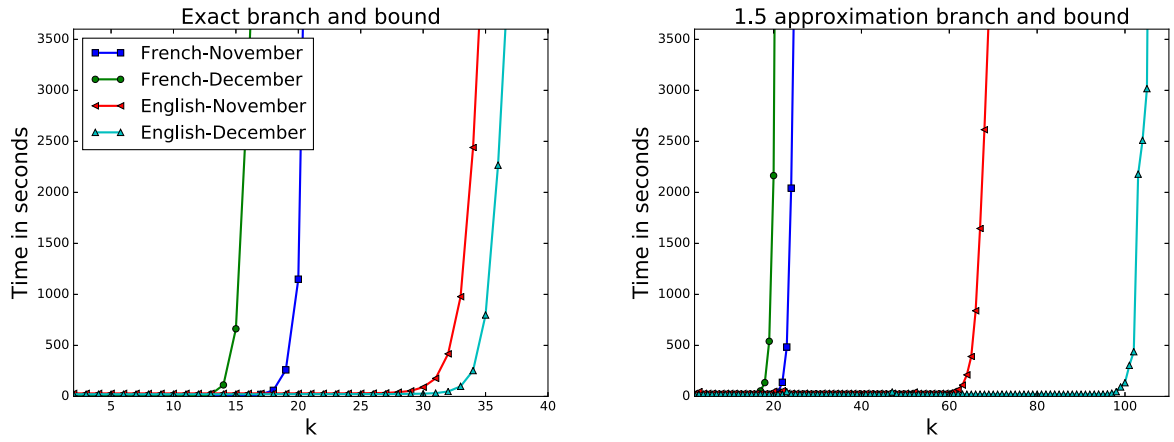

Fig. 2. Running time of branch and bound algorithm as a function of the input k for our four datasets. (left) exact algorithm. (right) approximation algorithm, we set the approximation ratio to $\alpha = 1.5$.

had the objective to find ISIS members responsible for the terrorist attacks.

In the month of December we have more events related to the Twitter platform, like advertising services. An interesting event that we find in this month is connected to the United Nations Climate Change Conference, COP21. The agreement did not include gender equality and supporters argued that the status of women in developing countries makes them more susceptible to climate change. We can see also that our technique grouped non-related events together. One possible solution for alleviating this could be to find the densest subgraph of at most k vertices, but we leave this to future work. We note that our method could be improved by using filtering techniques that have been proposed in literature for removing common words, one example is using the entropy of the distribution of word frequencies over time [14]. However, our goal is to prove that a simple graph mining approach can give meaningful results for the task of event detection.

Graph	Description
familles victimes paris bataclan fusillade	The terrorist attack that took place in Paris at the concert hall Bataclan.
escargophone trecru gameinsight androidgames android	A popular android application that posts automatic messages on social media.
max recherche parisattacks ans rechercheparis	People sharing information on missing friends before the release of the victims names.
france stade direction mtvstars vote	Another terrorist attack occurred at a stadium in Paris, Stade de France. The event is mixed with a popular music contest(MTVStars).
minute silence concours follow tirage	A minute of silence in the memory of the victims. Additional unrelated words are contained.
peuple condoléances monde nouveau religion	People shared a tweet of the king of Saudi Arabia, which expressed his condolence and said terrorism does not have a religion.
salut réponse moment côté buzz	Funny videos circulating of Twitter
nation forces fois l'ordre l'effroi	People talking about a police raid in a Paris suburb in search of ISIS members.
potes followers espace ton technique	Retweet of service promising to provide followers against a cost.
part infos l'histoire spectateur con	Not a clear event.

TABLE V
TOP 10 HEAVY SUBGRAPHS OF SIZE 5 FOR FRENCH-NOVEMBER

Graph	Description
direction mtvstars vote votos videomtv2015	Users are voting for a popular music contest, MTVStars.
gameinsight androidgames android nourriture unités	Automatic posts of an android application on social media.
escargophone trecru photos l'équipage géants	Automatic posts of an android application on social media.
follow concours sort tirage tweet	Contests and games are organized by users for their followers.
réponse salut côté moment buzz	Funny videos circulating on Twitter
serveur offres webhost hebergement cheaphostingoff	Offers for cheap web hosting.
paris travail cop21 offre h/f(homme - man / femme - woman)	One topic of discussion during the COP21 was equality between women and men, as the bad status of women in developing countries makes them more susceptible to climate change.
unfollowers d'aujourd statistiques semaine mois-ci	Retweet of service promising to provide followers against a cost.
2015 regionales2015 2016 missfrance2016 miss	Two events: regional elections and Miss France contest.
place chance followers espace potes	Retweet of service promising to provide followers against a cost.

TABLE VI
TOP 10 HEAVY SUBGRAPHS OF SIZE 5 FOR FRENCH-DECEMBER

The top subgraphs found starting from the English tweets all refer to a popular TV show (called MTV), where users were asked to vote via the Twitter platform. Those results are perhaps less interesting as they focus on one single popular event. This shows the current limitations of our approach in finding interesting events in Twitter. In particular, the diversity of the results should be improved, while noisy or less relevant information should be filtered out. However, on data which is less noisy such as the French tweets our approach delivers good results showing its potential.

The results presented in Table V and Table VI are obtained in the following way. We find a heavy subgraph, we remove its vertices and all edges adjacent to them and we iterate on the remaining graph until we have found t subgraphs. A good heuristic for finding the top t heavy k-subgraphs should give subgraphs which have a large total sum of weights.

In order to compare how well performs (i) the heuristic based on weighted core decomposition (ii) our exact branch and bound algorithm and (iii) our approximation branch and bound algorithm for finding the heaviest t subgraphs, we compute the top 100 subgraphs having as subroutine one of the three algorithms (for the approximation we use $\alpha = 1.5$. Figure 3 shows the sum of weights of the subgraphs as a function of the number of subgraphs for the French datasets. We omit the English datasets as the results are similar. In all datasets our exact algorithm, as well as our approximation algorithm for $\alpha = 1.5$) consistently outperforms the weighted core decomposition algorithm, so we can conclude that it is better suited for solving the top t heavy k-subgraphs problem for small values of k.

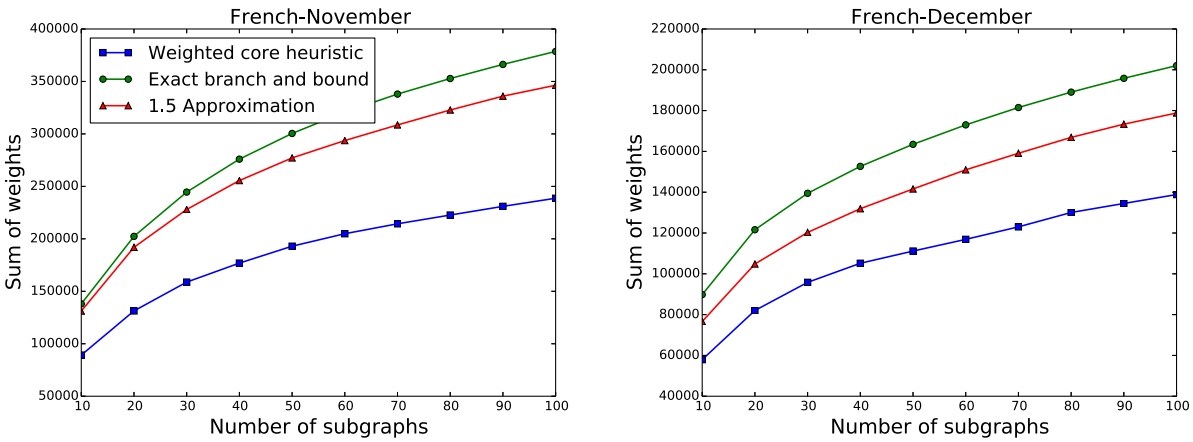

Fig. 3. Sum of the weights of the top t heavy subgraphs of size 5 as a function of t for our French datasets.

VI. Conclusion

We presented a new branch and bound algorithm for solving the densest k-subgraph problem in weighted graphs. The branching phase is based on deciding whether to include an edge in the subgraph or not, edges are examined in non-increasing order of weight in order to maximize the efficiency of the algorithm. The pruning phase is two fold: (i) based on the size of the subgraph and (ii) on the efficient computation of a tight upper bound.

The algorithm scales to large weighted real-world graphs containing millions of edges for up to $k = 15$ or more depending on the structure of the graph. An approximation version of our algorithm can scale to larger k. We show that our algorithm performs better than a state-of-the-art method based on weighted core decomposition of the graph. In addition, we showed that our algorithm is able to detect relevant events in Twitter.

Future work includes improvements of our branch and bound algorithm using parallel computing, as well as its generalization to the detection of other kinds of subgraphs. One possible direction is community detection, where a community is intuitively defined as a set of nodes that are highly connected together, but poorly connected to the outside [10]. Another perspective is in improving the quality of the results for event detection when dealing with noisy data.

References

[1] A. Angel, N. Koudas, N. Sarkas, D. Srivastava, M. Svendsen, and S. Tirthapura. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *VLDB Journal*, 23(2):175–199, 2014.

[2] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203 – 221, 2000.

[3] A. Bhaskara, M. Charikar, E. Chlamtac, and U. Feige. for Densest k -Subgraph. *Organization*, (873):201–210.

[4] M. Charikar. Greedy Approximation Algorithms for Finding Dense Components in a Graph. *Approximation Algorithms for Combinatorial Optimization*, pages 84–95, 2000.

[5] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[6] F. Chierichetti, J. Kleinberg, R. Kumar, M. Mahdian, and S. Pandey. Event Detection via Communication Pattern Analysis. pages 51–60, 2014.

[7] U. Feige. The Dense k -Subgraph Problem 1 Introduction. 1999.

[8] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2):174–211, 2001.

[9] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[10] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[11] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. *International Conference on Very Large Data Bases (VLDB)*, pages 721—-732, 2005.

[12] A. V. Goldberg. Finding a maximum density subgraph, 1984.

[13] S. Khuller and B. Saha. On finding dense subgraphs. *Icalp*, 5555:597–608, 2009.

[14] R. Long, H. Wang, Y. Chen, O. Jin, and Y. Yu. Towards effective event detection, tracking and summarization on microblog data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6897 LNCS(800):652–663, 2011.

[15] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Scandinavian Workshop on Algorithm Theory*, pages 260–272. Springer, 2004.

[16] P. Meladianos, G. Nikolentzos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis. Degeneracy-based real-time sub-event detection in twitter stream. 2015.

[17] M. Osborne and M. Dredze. Facebook , Twitter and Google Plus for Breaking News: Is There a Winner ? *Proceedings of the 8th International AAAI Conference on Weblogs and Social Media*, pages 611–614, 2014.

[18] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X. N. Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6044 LNBI:456–472, 2010.