Data Min Knowl Disc (2012) 25:577–602
DOI 10.1007/s10618-012-0255-0

Mining blackhole and volcano patterns in directed graphs: a general approach

Zhongmou Li · Hui Xiong · Yanchi Liu

Received: 2 May 2011 / Accepted: 20 January 2012 / Published online: 10 February 2012

Abstract Given a directed graph, the problem of blackhole mining is to identify groups of nodes, called blackhole patterns, in a way such that the average in-weight of this group is significantly larger than the average out-weight of the same group. The problem of finding volcano patterns is a dual problem of mining blackhole patterns. Therefore, we focus on discovering the blackhole patterns. Indeed, in this article, we develop a generalized blackhole mining framework. Specifically, we first design two pruning schemes for reducing the computational cost by reducing both the number of candidate patterns and the average computation cost for each candidate pattern. The first pruning scheme is to exploit the concept of combination dominance to reduce the exponential growth search space. Based on this pruning approach, we develop the gBlackhole algorithm. Instead, the second pruning scheme is an approximate approach, named approxBlackhole, which can strike a balance between the efficiency and the completeness of blackhole mining. Finally, experimental results on real-world data show that the performance of approxBlackhole can be several orders of magnitude faster than gBlackhole, and both of them have huge computational advantages over the brute-force approach. Also, we show that the blackhole mining algorithm can be used to capture some suspicious financial fraud patterns.

Responsible editor: Fei Wang, Hanghang Tong, Phillip Yu, Charu Aggarwal.

Z. Li · H. Xiong (✉)
Department of Management Science and Information Systems, Rutgers University, Newark, NJ, USA
e-mail: hxiong@rutgers.edu

Z. Li
e-mail: mosesli@pegasus.rutgers.edu

Y. Liu
University of Science and Technology Beijing, Beijing, China
e-mail: liuyanchi@manage.ustb.edu.cn

Keywords Blackhole pattern · Volcano pattern · Financial fraud detection · Graph mining · Network structure analysis

1 Introduction

Government agencies, such as U.S. Securities and Exchange Commission (SEC), are facing increasing challenges for financial fraud detection. The sophisticated fraud tactics makes detecting and preventing fraud difficult, especially as the number of trading accounts and the volume of transactions grow dramatically. Indeed, the trading networks are vulnerable to these fast-growing accounts and the volume of transactions. In particular, criminals know fraud detection systems are not good at correlating user behaviors across multiple trading accounts. This weakness opens the door for cross-account collaborative fraud, which is difficult to discover, track and resolve because the activities of the fraudsters usually appear to be normal. For instance, consider a trading network with a large number of nodes and directed edges, a trader or a group of traders can perform trading only within several accounts for the purpose of manipulating the market. This kind of illegal trading activities is widely known as trading ring patterns.

In this article, we study a special type of trading-ring patterns, called blackhole patterns. Given a directed graph, a blackhole pattern is a group which contains a set of nodes in a way such that the average in-weight of this group is significantly larger than the average out-weight of the same group. In contrast, a volcano pattern contains a set of nodes where the average out-weight is significantly larger than the average in-weight. In fact, we originate the blackhole and volcano patterns from real-world trading networks. For example, consider a group of traders who are manipulating the market by performing transactions on a specific stock among themselves for a specific time period. In the first stage, these traders produce a large volume of transactions on this stock by purchasing shares from public. During this time period, the average net volume of shares of the target stock per their trading account will increase significantly, and a blackhole pattern can be observed. After the stock price goes up to a certain degree, these traders start selling off their shares to the public to earn profit. In this stage, these trading accounts form a volcano pattern which have the average out-weight significantly larger than the average in-weight.

The blackhole and volcano patterns can also be observed in other application scenarios. For example, in *sina weibo*[1], one of the biggest micro-blog online community in China, a new type of cross-account collaborative fraudulent activity has been observed as follows. To get started, the fraudster needs a very popular account P which may take months to accumulate millions of followers. Then, he creates a number of new accounts and wants to increase the popularity of these accounts in a very short time. Let us take one such account L as an example. First, the fraudster frequently posts lots of interesting tweets under account L on a specific popular subject. Next, the account P retweets (shares) all tweets that L posts to make them visible to P's followers. A part of P's followers will start to follow L. In addition, most of the followers are likely to

[1] http://weibo.com

retweet the tweets L post. Due to the network effect, the number of L's followers will further increase. After a short period of time, L accumulates a considerable number of followers. Then, the process moves to the next stage. Each account L within the group starts to retweet other accounts' tweets in order to maximize the network effect of the whole group in the community. In this way, each L can pile a significant number of followers within a very short period of time. To sum up, the fraudster first uses P to solve the cold-start problem, and then let the accounts form a group to maximize the network effect. From the view of normal accounts, this is a fraudulent activity since it impedes the healthy development of the community. If we consider the community as a directed graph in which all accounts are nodes, and there is a directed edge from node A to node B if account A follows account B. Then, during the period of time, we can observe a blackhole pattern within that group of fraudulent accounts, which can help to prevent such kind of fraudulent activities.

In our preliminary work (Li et al. 2010), we developed a simplified version of blackhole patterns, where the blackhole patterns have the assumption that there is no traffic flow out of the blackhole. Also, there is no weight associated with the edge. In this article, we propose a general definition of blackhole and volcano patterns. Also, we show the problem of detecting blackhole patterns is a dual problem of finding volcano patterns. Therefore, we focus on the problem of identifying blackhole patterns. Along this line, we develop a generalized blackhole mining framework in which there are two pruning schemes to deal with the computational challenges. In the first pruning scheme, we introduce the $additivity$ property of diff-weight to reduce the computational cost of computing diff-weight of a set of nodes. Also, we propose the concept of combination dominance to help reduce the exponentially growing search space significantly. Based on these pruning techniques, we develop the gBlackhole algorithm to find the top-K blackhole patterns. In contrast, the second pruning scheme follows an approximate strategy, which is exploited for developing the approxBlackhole algorithm. Indeed, the average access time to retrieve information from a node or an edge increases rapidly with the number of nodes. This becomes the bottleneck of the performances of gBlackhole for large graphs. Therefore, we improve the computational efficiency by first screening out nodes with small diff-weights to reduce the size of the graph, and then mining the top-K blackhole patterns in the subgraph induced by the rest of the nodes. This approach is correct but not complete, and there is a trade-off between the efficiency and completeness.

Finally, experiments on real-world data sets are provided to evaluate the computational performances of the proposed two algorithms: gBlackhole and approxBlackhole. The results show that the approxBlackhole algorithm can be several orders of magnitude faster than the gBlackhole algorithm, and both of them have a huge computational advantage over the brute-force algorithm in terms of both the number of combinations searched and the average computational cost for each combination. The trade-off effect between efficiency and completeness of the approxBlackhole algorithm is also studied. Moreover, the proposed algorithms have been exploited for identifying some suspicious financial fraud patterns in the simulated E-mini S&P 500 futures contract trading data set from U.S. Commodity Futures Trading Commission. The result shows the effectiveness of the blackhole patterns.

2 Preliminaries

In this section, we introduce the basic concepts and notations that will be used in this article.

Consider a directed graph $G = (V, E)$ (Diestel 2006), where V is the set of all nodes and E is the set of all edges. Assume that G has no self-loops. A directed edge e in G is denoted as $e = (x, y)$, where x and y are nodes of G and an arc is directed from x to y. Each edge e has a positive weight, denoted as ω_e, associated with this edge.

Definition 1 (connected/weakly connected) In an undirected graph G, two nodes u and v are called connected if G contains a path from u to v. An undirected graph is called connected if every pair of distinct nodes in the graph is connected. A directed graph is called weakly connected if the undirected graph produced by replacing all of its directed edges with undirected edges is a connected graph.

Definition 2 (in-weight/out-weight) Given a directed graph $G = (V, E)$, B is a set of nodes and $B \subseteq V$. Let $C = V \setminus B$. The in-weight of B is defined as: $In(B) = \sum_{e=(x,y)\in E} \omega_e$, where $x \in C$ and $y \in B$. And the definition of the out-weight of B is in the opposite direction: $Out(B) = \sum_{e=(x,y)\in E} \omega_e$, where $x \in B$ and $y \in C$.

In the trading network scenario, if we consider the trading network on a specific stock during a period of time as a directed graph, such that the trading accounts are the nodes, the transactions between accounts are the edges, and the transaction volumes are the weights associated with the edges, then for a certain group of trading accounts (nodes) B, the in-weight of B is the total volume of shares that group of traders have purchased from the public during that period of time. Similarly, the out-weight of B is the total volume of shares that group of traders have sold to the public during that period of time.

Figure 1 shows an example of the in-weight and out-weight of a set of nodes. The number associated with each edge is the weight of that edge. In this figure, the in-weight of B is $6 + 5 = 11$, while the out-weight is $3 + 3 + 1 + 2 = 9$.

Now, let us get back to the intuition of blackhole and volcano patterns we have discussed in Sect. 1. In the trading network scenario, we would like to find out the cross-account collaborative fraudulent trading activities. Therefore, there should be at least two trading accounts and these accounts have some sort of interactions among them. In graph terms, it means this set of nodes has at least 2 nodes and the subgraph induced by these nodes is weakly connected.

Also, in Sect. 1, we have described the two stages that a normal trading-ring pattern usually behaves. In the first stage, the average volume of shares that the group of trading accounts have purchased from the public is significantly larger than the average volume of shares that they have sold, and we consider that group of trading accounts as a blackhole pattern. In graph terms, it means the average of the difference between the in-weight and out-weight of that set of nodes is significant. In contrast, another stage for selling off the stock shares can be viewed in a very similar fashion, and we name the group of trading accounts in this stage as a volcano pattern.

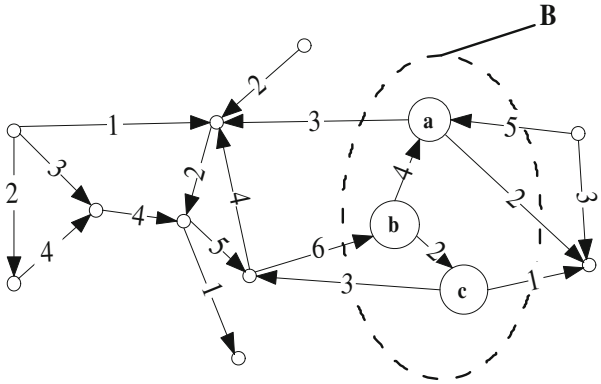

Fig. 1 Illustration: in-weight and out-weight

Since we will refer to the difference between the in-weight and out-weight of a set of nodes a lot in the rest of the article, to make our notations easy and precise, we introduce the definition of *diff-weight*.

Definition 3 (diff-weight) Given a directed graph $G = (V, E)$, B is a set of nodes and $B \subseteq V$. The diff-weight of B is defined as: $Diff(B) = In(B) - Out(B)$.

Based on the above understandings, we have the definition of blackhole pattern and volcano pattern in a directed graph as follows.

Definition 4 (blackhole pattern) Given a directed graph $G = (V, E)$, a set of nodes $B \subseteq V$ form a blackhole pattern, if and only if the following two conditions are satisfied: (1) $|B| \geq 2$, and the subgraph $G(B)$ induced by B is weakly connected, and (2) $Diff(B)/|B| > \theta$, where $|B|$ is the cardinality of B, and θ is a pre-defined positive threshold and is typically a very large value.

Definition 5 (volcano pattern) Given a directed graph $G = (V, E)$, a set of nodes $Vol \subseteq V$ form a volcano pattern, if and only if the following two conditions are satisfied: (1) $|Vol| \geq 2$, and the subgraph $G(Vol)$ induced by Vol is weakly connected, and (2) $Diff(Vol)/|Vol| < -\theta$, where $|Vol|$ is the cardinality of Vol, and θ is a pre-defined positive threshold and is typically a very large value.

Please note that, we used the diff-weight of a set of nodes to define a simplified version of blackhole and volcano patterns in Li et al. (2010). However, by using this definition, we may not end up finding out the groups of nodes that we really want in some cases. Assume that a set of nodes B is a blackhole pattern we have found. By adding another node v such that $Diff(v) > 0$ and v is connected to B, we can find another blackhole pattern B' with $Diff(B') > Diff(B)$. If we keep repeating this process, we will end up with a B' which is a group of nodes with large diff-weight and large size. However, the average diff-weight of B' may be far less than the average diff-weight of B. In real world, we are more likely interested in B rather than B'. Let us consider the trading network scenario again. B' may represent a group of

thousands of trading accounts whose total net volume of shares accumulate rapidly during a certain period, while the average net volume per account may be small. These accounts are more likely to be normal accounts with normal transactions. Meanwhile, B may consist of a couple of trading accounts whose net volume of shares per account increase fast, but the total net volume may not as large as the one of B'. In this situation, B is more likely to perform a trading-ring fraud than B'. In this sense, the average diff-weight of a set of nodes is more important than the diff-weight itself. Therefore, the problem with the simple definitions of blackhole and volcano patterns in Li et al. (2010) is that we only considered the total diff-weight but fail to take the number of nodes in the group into account for normalization. To this end, we will use the average diff-weight of a set of nodes to define generalized blackhole and volcano patterns in this article.

3 Problem formulation

In this section, we formulate the blackhole mining problem. We first show that the problem of detecting blackhole patterns is a dual problem of detecting volcano patterns.

Theorem 1 *The problem of detecting blackhole patterns in a directed graph is a dual problem of detecting volcano patterns in its inverse graph, and vice versa.*

Proof Consider a directed graph $G = (V, E)$. Let $G' = (V, E')$ be the *inverse graph* of G, where all the nodes in G' are the same as in G; while for each edge $e = (x, y) \in E$, there is an edge $e' = (y, x) \in E'$, and the weight associated with e' is exactly the same as the weight associated with e. Therefore, the in-weight of a set of nodes B in G is the same as the out-weight of B in G', and vice versa. Thus, for each blackhole pattern B in G, we have $(In(B) - Out(B))/|B| > \theta$ in G. Then in G', we have $(Out(B) - In(B))/|B| > \theta$, which is equivalent to $Diff(B)/|B| < -\theta$. Also, in G', we still have $|B| > 2$ and $G(B)$ is weakly connected. Therefore, B forms a volcano pattern in G'. On the other hand, we can also prove that for each volcano pattern Vol in G', it is a blackhole pattern in G. Thus, detecting blackhole patterns in a directed graph is equivalent to detecting volcano patterns in its inverse graph. In a similar fashion, we can also prove that detecting volcano patterns in a directed graph is a dual problem of detecting blackhole patterns in its inverse graph. □

According to Theorem 1, we can focus on the problem of detecting blackhole patterns in a directed graph in the rest of the article.

In Li et al. (2010), we formulated the problem of detecting blackhole patterns in a directed graph $G = (V, E)$ as to find out the set of blackhole patterns, denoted as $Blackhole$, such that for each element $B \in Blackhole$, it is a set of nodes in G, and B satisfies the definition of *blackhole*. On the other hand, there is no other set of nodes $B' \notin Blackhole$ such that B' satisfies the definition of *blackhole*. In sum, we would like to find out the complete and correct set of all blackhole patterns in G.

However, there are certain issues to be reconsidered in our previous problem formulation in Li et al. (2010). The main issue is how should we decide the value of θ.

Since the problem has a combinatorial nature such that we would like to find out the complete and correct set of all blackhole patterns in a directed graph, if we set θ too small, most likely we would end up with a set with $O(2^n)$ blackhole patterns in it. On the other hand, if we make θ too large, most likely we would get nothing in the end. It is hard and tricky for us to determine the value of θ without trying many times, and obviously it is very inefficient and inconvenient in practice. Actually, we are much more interested in the "outliers" rather than the ordinary ones in the real world. In other words, we are more interested in the blackhole patterns with larger average diff-weights. Therefore, if we simply identify blackhole patterns with the top-K largest average diff-weights, we will achieve our goal and make things much easier and more efficient. As a result, we have our problem formulation as follows.

Top-K Blackhole pattern mining. Given a directed graph $G = (V, E)$, the goal is to find out the set of top-K blackhole patterns, denoted as $Blackhole$, such that, (1) $|Blackhole| = K$, (2) for each element $B \in Blackhole$, $B \subseteq V$, $|B| \geq 2$, and the subgraph $G(B)$ induced by B is weakly connected, and (3) for any other set of nodes $B' \notin Blackhole$, if B' satisfies condition (2), then for each element $B \in Blackhole$, we have $Diff(B)/|B| \geq Diff(B')/|B'|$.

4 Algorithm design

In this section, we introduce three algorithms to mine the top-K blackhole patterns in a directed graph. To simplify the discussion, let n denote the number of nodes, m denote the number of edges, and $\alpha = m/n$ be the average in-and-out node degree of the graph.

4.1 A brute-force approach

In this subsection, we present a brute-force approach for detecting blackhole patterns in a directed graph. Since the goal is to find out the set of top-K blackhole patterns, the intuition is really simple: we check out all combinations of nodes in V from size 2 to n using the exhaustive search method. For each combination B, if the subgraph $G(B)$ induced by B is weakly connected, we keep a record of B as well as its average diff-weight in a list. Finally, we sort the list and output the blackhole patterns with the top-K largest average diff-weights.

In practice, in order to save space and make the algorithm more efficient, we maintain a priority queue $Blackhole$ of size K to record the current top-K blackhole patterns along with their average diff-weights. It is initialized with K empty sets with key value of $-\infty$ and the elements in it are sorted by the descending order of their key values. During the whole procedure, $Blackhole$ keeps updated to make sure it has recorded the current top-K blackhole patterns among combinations which have already been checked so far. Finally, when the exhaustive search procedure completes, $Blackhole$ will record the set of blackhole patterns with the top-K largest average diff-weights as the result.

Figure 2 shows the pseudo code of the brute-force approach to detect the top-K blackhole patterns in a directed graph G, in which $Subset_i = \{B \mid B \subseteq V, and$

ALGORITHM *Brute-Force* $(G = (V, E), K)$
Input:
G: the input directed graph
V: the set of all nodes
E: the set of all edges
Output:
Blackhole: priority queue of current top-K blackholes

1. Initialize *Blackhole*
2. **for** $i \leftarrow 2$ **to** n **do**
3. **for** each combination of nodes $B \in Subset_i$ **do**
4. **if** *Is_Connected(*$G(B)$*)* **and**
5. $Diff(B)/|B| > Blackhole.extract\_min()$ **then**
6. $Blackhole.delete\_min()$
7. $Blackhole.insert(B, Diff(B)/|B|)$
8. **end if**
9. **end for**
10. **end for**
11. **return** *Blackhole*

Fig. 2 A brute-force approach

$|B| = i\}$ is the set of all combinations of nodes in V of size i, and the function *Is_Connected()* checks whether a directed graph is weakly connected or not.

The computational complexity of the brute-force approach is $T = O(M \cdot N)$, where $N = 2^n - n - 1$ is the number of combinations of size no less than 2, and M is the average computational cost for each combination B. We use depth-first search (DFS) to check whether B is weakly connected or not, and meanwhile we can compute $Diff(B)$. During the DFS, each node in B is accessed once as well as all its edges. Therefore, the computational cost of each combination B is $M = O((\alpha + 1) \cdot |B|)$. Since the average size of B is $n/2$, then the computational complexity of the brute-force approach is $T = O(M \cdot N) = O((m + n) \cdot 2^n)$.

This approach is obviously computationally prohibitive due to its combinatorial nature. As the number of nodes n increases, the number of combinations increases exponentially. This makes the brute-force approach unrealistic to use in practice when n is large. Since $T = O(M \cdot N)$, the computational complexity can be reduced by decreasing either the number of combinations or the average computational cost for each combination. Along this line, we introduce two pruning schemes in the following subsections to help reduce the computational cost.

4.2 The additivity property of diff-weight

In brute-force approach, we compute $Diff(B)$ by calculating $In(B)$ and $Out(B)$ separately first, and then do the subtraction. As mentioned in last subsection, the computational cost of calculating $Diff(B)$ is $O((\alpha+1) \cdot |B|) = O(m+n)$, which is very

time consuming. Consider the high frequency of calculating $Diff(B)$ in the blackhole mining process, we need a more efficient way for that.

The main challenge is that the set of edges between B and C are changing dynamically for different set of nodes. Is it possible that we can pre-compute a value for each node in V separately, and for each set of nodes B, we can compute $Diff(B)$ by adding up the values of each node in B, regardless the set of edges between B and C? The following theorem solves this problem. We name this theorem as *additivity property of diff-weight*.

Theorem 2 *(Additivity property of diff-weight) The diff-weight of a set of nodes B equals to the summation of the diff-weight of each node in B, i.e., $Diff(B) = \sum_{v \in B} Diff(v)$.*

Proof

$$
\begin{aligned}
Diff(B) &= In(B) - Out(B) \\
&= \sum\nolimits_{x \in C, y \in B} \omega_e - \sum\nolimits_{x \in B, y \in C} \omega_e \\
&= \left(\sum\nolimits_{x \in C, y \in B} \omega_e + \sum\nolimits_{x \in B, y \in B} \omega_e\right) - \left(\sum\nolimits_{x \in B, y \in C} \omega_e + \sum\nolimits_{x \in B, y \in B} \omega_e\right) \\
&= \sum\nolimits_{x \in C \bigcup B, y \in B} \omega_e - \sum\nolimits_{x \in B, y \in C \bigcup B} \omega_e \\
&= \sum\nolimits_{x \in V, y \in B} \omega_e - \sum\nolimits_{x \in B, y \in V} \omega_e \\
&= \sum\nolimits_{v \in B} In(v) - \sum\nolimits_{v \in B} Out(v) \\
&= \sum\nolimits_{v \in B} (In(v) - Out(v)) \\
&= \sum\nolimits_{v \in B} Diff(v), where\ e = (x, y) \in E
\end{aligned}
$$

□

Let us illustrate this property using Fig. 1 as an example. In this figure, we have $Diff(a) = (4 + 5) - (3 + 2) = 4$, $Diff(b) = 6 - (4 + 2) = 0$, and $Diff(c) = 2 - (3 + 1) = -2$. Thus, we have $\sum_{v \in B} Diff(v) = 4 + 0 + (-2) = 2$, which equals to $Diff(B) = (6 + 5) - (3 + 3 + 2 + 1) = 2$. This simple example illustrates the *additivity property of diff-weight*.

Based on this property, we can compute $Diff(B)$ in a different way as the following. First, we compute the diff-weight of each node in V, and record the result in a list L. This is a one-time effort and the computational cost is $O(|V| + |E|) = O(m+n)$. Then, for each combination B, $Diff(B)$ can be calculated by adding up the diff-weight of each node in B, which can be retrieved from the list L. By using this method, the computational cost can be reduced to $O(|B|)$. As a result, for the **if** statement in the brute-force approach, we check the condition $Diff(B) > Blackhole.min()$ first, only if it is true, we then check the condition *Is_Connected* $(G(B))$, since the computational cost of the former one is smaller.

4.3 The measure of enumeration

In this subsection, we introduce a measure to fulfill the procedure of enumerating all combinations of nodes in V from size 2 to n. Basically, it consists of two steps, (1) sort the nodes in the graph by a certain criteria, and (2) for each $i = 2, 3, \ldots, n$, find a particular order to list all combinations of nodes with size i.

For step (1), recall that in last subsection, we have calculated the diff-weight of each node, and know that the value of $Diff(B)$ equals to the summation of the diff-weight of each node in B. Since our goal is to find out blackhole patterns with the top-K largest average diff-weight, it implies that nodes with larger diff-weight have higher probability to appear in the top-K blackhole patterns. Therefore, it is very natural to sort the nodes in V by their diff-weights in a decreasing order, and start the enumeration procedure from combinations consisting of nodes with larger diff-weights. We denote the nodes after sorting as $\{v_1, v_2, \ldots, v_n\}$, where v_1 has the largest diff-weight while v_n has the smallest.

For step (2), we use the lexicographic order (a.k.a. dictionary order) to list all combinations of nodes of size i (Knuth 2011), not only because it has been widely used, more importantly, enumerating combinations in this order is consistent with our intuition of starting from nodes with larger diff-weights. Next, we briefly describe the basic idea of the lexicographic order.

Given a general alphabet set $S = \{s_1, s_2, \ldots, s_n\}$, the elements in this set are ordinal, which means each one of them is comparable with other elements by a pre-defined order. We denote this order as the lexicographic order of this general alphabet set S. Without the loss of generality, we let $s_1 \prec s_2 \prec \ldots \prec s_n$, where the notion "$\prec$" here denotes lexicographically less than. In our case, we consider V as the alphabet set, and define the lexicographic order in V as $v_1 \prec v_2 \prec \ldots \prec v_n$, where $Diff(v_1) \geq Diff(v_2) \geq \ldots \geq Diff(v_n)$.

Since the order of elements in a combination does not matter, without the loss of generality, we assume that $\beta_1 \prec \beta_2 \prec \ldots \prec \beta_i$ for the combination $\beta = (\beta_1, \beta_2, \ldots, \beta_i)$ of size i. In the rest of the article, we assume that the elements in a combination are lexicographically ordered if not mentioned otherwise. Next, we give the definition of the lexicographic order of two combinations.

Definition 6 (lexicographic order for combinations) Let $\beta = (\beta_1, \beta_2, \ldots, \beta_i)$ and $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_i)$ be two combinations of the same size i over S. We say that β is lexicographically less than γ, denoted as $\beta \prec \gamma$, if and only if for $j = 1, 2, \ldots, i$, there exist some j, such that $\beta_j \neq \gamma_j$, and for the smallest such j, $\beta_j \prec \gamma_j$.

The lexicographic order generates combinations in a particular order such that for any two combinations β and γ with the same size, if $\beta \prec \gamma$, β is generated earlier than γ. For example, the lexicographic order of the combinations of size 3 over alphabet set $\{1, 2, 3, 4, 5\}$ is (1,2,3), (1,2,4), (1,2,5), (1,3,4), (1,3,5), (1,4,5), (2,3,4), (2,3,5), (2,4,5), (3,4,5), if we define the lexicographic order for this example as $1 \prec 2 \prec 3 \prec 4 \prec 5$. By using the lexicographic order, we can therefore enumerate all combinations of nodes of size i for each $i \geq 2$.

In practice, we employ the recursion of depth i to implement the enumeration process. We use a pointer pt to indicate which node at which level is currently being

visited in the recursion. Level 1 is the first level of the recursion while level i is the deepest. pt initially points to node v_1 at level 1. When pt is pointing to node v_j ($k \leq j \leq n-i+k$) at level k ($k = 1, 2, \ldots, i-1$), we let v_j be the kth element of combination B and move pt to node v_{j+1} at level $k+1$. When pt jumps out of level $k+1$, we erase v_j from the kth element of B first, and if $j < n-i+k$, we move pt to node v_{j+1} at level k and repeat the same process as for node v_j at level k, otherwise, we let pt jump out of level k and return to level $k-1$. If pt is pointing to node v_j at level i, the process is similar to the one for previous levels, except we check out whether the current combination B is a top-K blackhole pattern instead of diving into the deeper level, since we have already reached the last level of the recursion. When pt jumps out of level 1 and returns to level 0, the procedure completes and all combinations of size i have been enumerated in the lexicographic order. More details are available in the next subsection.

4.4 A pruning scheme

In this subsection, we propose a pruning scheme to control the exponential growth of the number of combinations, which can be helpful to reduce the computational cost.

First, we introduce the concept of combination dominance.

Definition 7 (combination dominance) Let $S = \{s_1, s_2, \ldots, s_n\}$ be a general alphabet set. $\beta = (\beta_1, \beta_2, \ldots, \beta_i)$ and $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_i)$ are two combinations of the same size i over S. We say that β dominates γ, denoted as $\beta \vdash \gamma$, if and only if $\beta_j \preceq \gamma_j$ for any $j = 1, 2, \ldots, i$, and for at least one j, $\beta_j \prec \gamma_j$.

Again, we take the alphabet set $\{1, 2, 3, 4, 5\}$ as an example. Among all combinations of size 3 over this set, $(1, 2, 4) \vdash (1, 2, 5)$, $(1, 3, 4) \vdash (2, 3, 5)$, $(1, 2, 3)$ dominates any other combinations, $(3, 4, 5)$ is dominated by any other combinations, $(1, 4, 5)$ and $(2, 3, 4)$ cannot dominate each other.

Next, we introduce several lemmas to help prune the exponentially growing search space.

Lemma 1 *If we consider V as the alphabet set, and define the lexicographic order in V as $v_1 \prec v_2 \prec \ldots \prec v_n$, then for any two combinations of nodes with the same size i over V, denoted as B and B', if $B \vdash B'$, then we have $Diff(B)/|B| > Diff(B')/|B'|$.*

Proof Recall that in Sect. 4.3, we have sorted the nodes in V by their diff-weights in a decreasing order, thus, 1) for any two nodes v and v' in V, if $v \prec v'$, then we have $Diff(v) > Diff(v')$. In addition, since $B \vdash B'$, if we denote $B = (v_{B_1}, v_{B_2}, \ldots, v_{B_i})$ and $B' = (v_{B'_1}, v_{B'_2}, \ldots, v_{B'_i})$, according to Definition 7, we have 2) $v_{B_j} \preceq v_{B'_j}$ for any $j = 1, 2, \ldots, i$, and for at least one j, $v_{B_j} \prec v_{B'_j}$. Therefore, based on 1) and 2), we can get $Diff(v_{B_j}) \geq Diff(v_{B'_j})$ for any $j = 1, 2, \ldots, i$, and for at least one j, $Diff(v_{B_j}) > Diff(v_{B'_j})$. According to the *additivity property of diff-weight* and since $|B| = |B'|$, we have $Diff(B)/|B| > Diff(B')/|B'|$. □

Lemma 2 *Let β and γ be two combinations of size i over the same alphabet set S. If $\beta \vdash \gamma$, then $\beta \prec \gamma$.*

Proof According to Definition 7, if $\beta \vdash \gamma$, then $\beta_j \preceq \gamma_j$ for any $j = 1, 2, \ldots, i$, and for at least one j, $\beta_j \prec \gamma_j$. Therefore, there exists at least one j such that $\beta_j \neq \gamma_j$, and for the smallest j among them, $\beta_j \prec \gamma_j$. According to Definition 6, we have $\beta \prec \gamma$. □

Lemmas 1 and 2 can be very useful to prune the exponentially growing search space. In the brute-force approach, if the average diff-weight of a combination B is no greater than the current minimum key in the priority queue $Blackhole$, we simply skip B and move forward to the next combination. However, according to Lemma 1, for any combination B' which is dominated by B, we have $Diff(B')/|B'| < Diff(B)/|B| \leq Blachole.extract\_min()$. Thus, there is no chance for such B' to appear in $Blackhole$. In addition, according to Lemma 2, $B' \succ B$, which indicates that B' will be checked later than B. Based on the above analysis, we can have the following theorem as the guideline of the pruning scheme.

Theorem 3 *(Pruning strategy) For a combination B of size i, if we skip B since the average diff-weight of B is no greater than the current minimum key in the priority queue $Blackhole$, then for any combination B' which is dominated by B, it can be simply pruned without checking.*

Although we would like to prune all combinations that are dominated by B, we also need to consider the cost of finding them out. If we have to check whether each upcoming combination is dominated by B, then this pruning strategy will have no computational savings. We need to find a simple and systematic way that can help to prune combinations which are dominated by B as many as possible with little cost, even though a few of them are missed and have to be checked out later. Along this line, we have the following pruning rule.

Lemma 3 *(Pruning rule) For a combination B of size i, denoted as $B = (v_{B_1}, v_{B_2}, \ldots, v_{B_i})$, if we skip B for the reason that $Diff(B)/|B| \leq Blachole.extract\_min()$, then we can let pointer pt jump out of level i immediately. If there exists a k $(1 \leq k \leq i-1)$ such that $B_{k+1} - B_k = B_{k+2} - B_{k+1} = \cdots = B_i - B_{i-1} = 1$, then pt can directly return to level $k-1$, otherwise, pt returns to level $i-1$.*

Proof If there exists a k $(1 \leq k \leq i-1)$ such that $B_{k+1} - B_k = B_{k+2} - B_{k+1} = \cdots = B_i - B_{i-1} = 1$, consider combination $B' = (v_{B'_1}, v_{B'_2}, \ldots, v_{B'_i})$, where $B_j < B'_j \leq n - i + j$ $(j = k, k+1, \ldots, i)$ and $B_j = B'_j$ $(j = 1, 2, \ldots, k-1)$ if $k > 1$. It is clear that $B \vdash B'$. Since $Diff(B)/|B| \leq Blachole.extract\_min()$, according to the pruning strategy, we can prune all such B' without checking. Note that these B' are combinations that directly follow B in the lexicographic order. In the recursion, the pruning can be done by simply letting pt jump out of level i, and returning directly to level $k-1$. When there is no such k, the proof is very similar and we skip it here. □

ALGORITHM *Find-Combinations* $(L, n, i, j, k, B, Blackhole)$
Input:
L: list of diff-weights of each node
n: number of nodes
i: size of combination
j: starting node index
k: current level number
B: current combination of nodes
$Blackhole$: priority queue of current top-K blackholes

```
1.  for p ← j to n − i + k do
2.    if k < i then
3.      B[k] = (p, L[p])
4.      Find-Combinations(L, n, i, p + 1, k + 1, B, Blackhole)
5.      if B[k + 1].index − B[k].index == 1 then
6.        break
7.      end if
8.    end if
9.    else
10.     B[k] = (p, L[p])
11.     if Diff(B)/|B| > Blackhole.extract_min() then
12.       if Is_Connected(G(B)) then
13.         Blackhole.delete_min()
14.         Blackhole.insert(B, Diff(B)/|B|)
15.       end if
16.     end if
17.     else
18.       break
19.     end else
20.   end else
21. end for
```

Fig. 3 The find-combinations algorithm

Then, we can incorporate this pruning rule into the enumeration process described in Sect. 4.3 to help reduce the search space. We name this pruning scheme as the *Find-combinations* algorithm. Basically, this pruning scheme follows a general sort-and-prune strategy, which is widely used for the algorithm design (Cormen et al. 2009), such as the development of association rule mining algorithms (Tan et al. 2005). The pseudo code is given in Fig. 3. B is an array of length i which records the current combination of nodes, and is initialized empty. Each element in it is a structure which consists of the index of a node in L after sorting ($B[k].index$), and the diff-weight of the node ($B[k].weight$). All array indices in the pseudo code start from 1.

Figure 4 gives an example to illustrate how our pruning scheme works. In the example, we would like to detect the top-2 blackhole patterns in a directed graph. Figure 4a shows the input graph and the corresponding list of diff-weights of each node after sorting. The procedure of checking out all combinations of size 3 by using our pruning scheme is given in the following subfigures. The priority queue $Blackhole$ in each

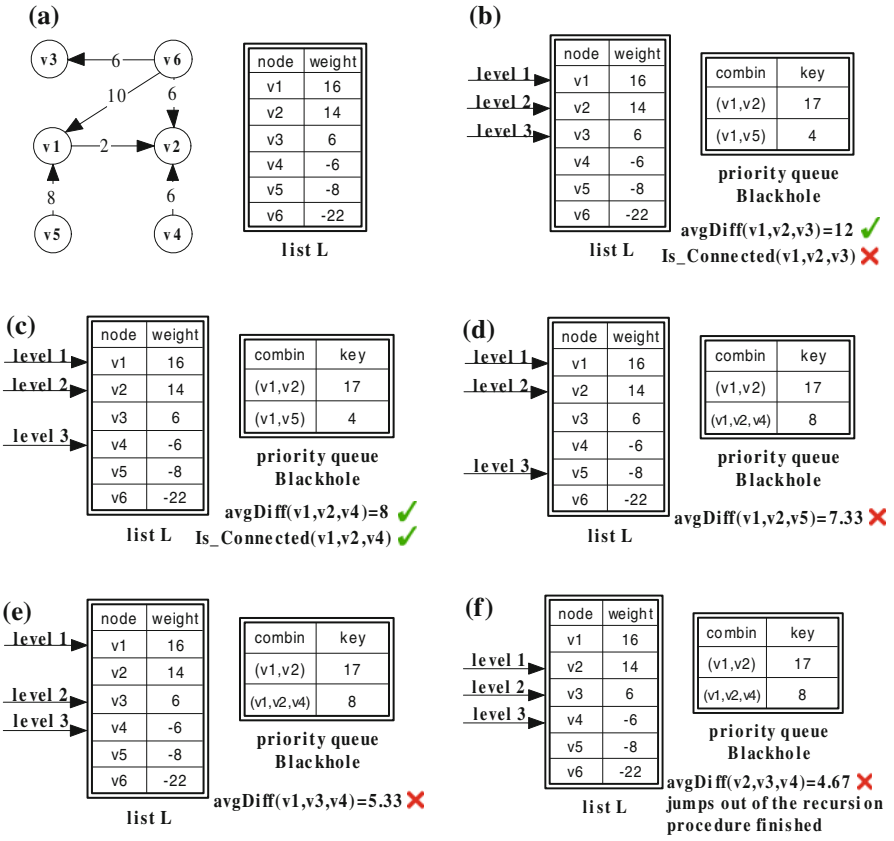

Fig. 4 An example to illustrate the pruning scheme

subfigure records the current top-2 blackhole patterns as well as their diff-weights. Since the procedure is very simple and the figures themselves are quite self-explained, we simply skip the explanation and leave it to the readers. Note that finally we only need to check 5 combinations out of all $C_6^3 = 20$ combinations of size 3 by using this pruning scheme.

Finally, we would like to discuss the reasons for developing this pruning scheme to solve the top-K blackhole mining problem. First, there is no pre-determined "global" order for the average diff-weights among all combinations of nodes in V. For example, let $V = \{v_1, v_2, v_3, v_4, v_5\}$ where $Diff(v_1) \geq Diff(v_2) \geq \ldots \geq Diff(v_5)$. Then for $B = (v_1, v_3, v_5)$ and $B' = (v_1, v_5)$, if $Diff(v_3) > 0$, then $Diff(B)/|B| > Diff(B')/|B'|$, otherwise, $Diff(B)/|B| < Diff(B')/|B'|$. Therefore, for different input graphs, the order of $Diff(B)/|B|$ and $Diff(B')/|B'|$ cannot be determined without checking them out. Even for combinations of the same size, there is still no such "global" order. For example, let $B'' = (v_2, v_3, v_4)$, for case $Diff(v_1, \ldots, v_5) = (10, 6, -4, -5, -7)$, we have $Diff(B)/|B| > Diff(B'')/|B''|$. If $Diff(v_1, \ldots, v_5) = (10, 6, 4, -7, -13)$, we will have the opposite order. Thus, for different input

graphs, the order of $Diff(B)/|B|$ and $Diff(B'')/|B''|$ still cannot be determined until checking them out. Therefore, in order to keep the process simple, organized and systematic, we use the measure introduced in Sect. 4.3 to enumerate all combinations of nodes in V from size 2 to n and check out their eligibility for the top-K blackhole patterns in lexicographic order; while in the same time, we introduce the concept of combination dominance as a "local" order to help develop the pruning scheme to reduce the search space.

4.5 The gBlackhole approach

While the search space has been reduced by using the pruning scheme introduced in last subsection, it is still possible to apply other pruning strategies to the mining process. As we have discussed in Li et al. (2010), for a node $v \in V$, v can only form blackhole patterns with nodes within the same weakly connected component (WCC) in G, since the subgraph induced should be weakly connected. Therefore, if a directed graph has several WCCs, a divide-and-conquer strategy can be exploited by first identifying these WCCs, then applying the pruning scheme on each of them, and finally merging all the results. This pruning strategy can divide a large exponentially growing search space into several smaller ones, and thus can further reduce the computational cost.

Along this line, we combine the pruning scheme based on combination dominance with this divide-and-conquer strategy and develop an algorithm, named *gBlackhole*, to detect the top-K blackhole patterns in a directed graph. Figure 5 shows the pseudo code of this approach. In this approach, we start with the WCC which contains the node with the largest diff-weight, and find out the top-K blackhole patterns in it. Next, we check out all the combinations of nodes in the second largest WCC by comparing them with the current top-K blackhole patterns. The above process repeats until we have found all WCCs.

The theoretical computational complexity of the gBlackhole algorithm is hard to analyze. As discussed in Sect. 4.1, the computational complexity can be expressed as $T = O(M \cdot N)$, where N is the number of combinations searched, and M is the average computational cost for each combination. As shown in the experiment section, the gBlackhole algorithm has a huge computational advantage over the brute-force approach, since the pruning effect in the gBlackhole algorithm is significant in terms of the number of pruned combinations.

4.6 The approxBlackhole approach

The gBlackhole algorithm is shown to be an efficient approach that can reduce the computational complexity dramatically. However, as the number of nodes and edges of the graph increases, the average access time to retrieve information from a node or an edge increases rapidly, especially when the graph is too large to fit into the memory. Thus, the dramatic increase of M, which is the average computational cost for each combination, becomes the bottleneck of the performance of the gBlackhole algorithm for large graphs.

ALGORITHM *gBlackhole* $(G = (V, E),\ K)$
Input:
G: the input directed graph
V: the set of all nodes
E: the set of all edges
Output:
Blackhole: priority queue of current top-K blackholes

1. Initialize $Blackhole$
2. **for** each node $v \in V$ **do**
3. $Diff(v) \leftarrow In(v) - Out(v)$
4. **end for**
5. Sort all nodes descendingly by their diff-weights in L
6. **for** each weakly connected component WCC in G **do**
7. **for** $i \leftarrow 2$ **to** n_{WCC} **do**
8. Initialize B
9. *Find-Combinations*$(L, n, i, 1, 1, B, Blackhole)$
10. **end for**
11. **end for**
12. **return** $Blackhole$

Fig. 5 The gBlackhole approach

The intuition of reducing the average access time to a large graph is by reducing the size of that graph. Since we only interest in blackhole patterns with the top-K largest average diff-weights, it is not necessary for us to search through the entire graph. Most of the top-K blackhole patterns only consist of nodes with very large diff-weights. Therefore, the chance for nodes with small diff-weights to appear in the top-K blackhole patterns is very tiny so that we can filter them out safely in an early stage. In this way, we can reduce the size of the graph and significantly improve the computational efficiency.

We develop a new efficient algorithm according to the analysis above. We only reserve nodes with top $p\%$ largest diff-weights, and filter out the rest of them, where p is a user specified parameter. Then, we identify the top-K blackhole patterns in the subgraph induced by these nodes. Please note that, we only preserve the graph structure of the subgraph in this algorithm; meanwhile, the diff-weight of each node in the subgraph is defined as its corresponding diff-weight in the original graph, rather than its diff-weight in the subgraph. Clearly, this approach is correct but not complete. There is a trade-off between the efficiency and completeness. We name this approximation approach as the *approxBlackhole* algorithm.

Experimental results in the next section will show that the approxBlackhole algorithm can further reduce the computational complexity over the gBlackhole algorithm, in terms of both the number of combinations searched and the average computational cost for each combination. The trade-off effect between efficiency and completeness will also be discussed.

Table 1 Data characteristics

Data set	# nodes	# egdes	avgDegree(α)	# WCC
`Wiki`	7,115	103,689	14.57	24
`Citation`	34,546	421,578	12.20	61
`Slashdot`	77,360	905,468	11.70	1
`Trading`	100	264,600	2646	1

5 Experimental results

In this section, we provide an empirical study to evaluate the performances of brute-force, gBlackhole and approxBlackhole algorithms. Here, we set $K = 100$ in all the experiments.

5.1 The experimental setup

Experimental data. The experiments were conducted on four real-world data sets: `Wiki`, `Citation`, `Slashdot`, and `Trading`. Table 1 shows some basic characteristics of these data sets.

`Wiki` data set. There are 7,115 nodes and 103,689 edges in the `Wiki` data set (Leskovec et al. 2010a,b). The network contains all administrator elections and vote history data from the inception of Wikipedia till January 2008. Nodes in the network represent Wikipedia users and a directed edge from node i to node j represents that user i voted on user j. Since there are no weights associated with the edges in the original data set, we assign the weight of each edge to be 1.

`Citation` data set. There are 34,546 nodes and 421,578 edges in the `Citation` data set (Leskovec et al. 2005; Gehrke et al. 2003). Arxiv HEP-PH (high energy physics phenomenology) citation graph is from the e-print arXiv and covers all the citations within a dataset of 34,546 papers with 421,578 edges. If a paper i cites paper j, the graph contains a directed edge from i to j. If a paper cites, or is cited by, a paper outside the dataset, the graph does not contain any information about this. The `Citation` data set covers papers in the period from January 1993 to April 2003. For this data set, we also assign the weight of each edge to be 1.

`Slashdot` data set. There are 77,360 nodes and 905,468 edges in the `Slashdot` data set (Leskovec et al. 2008). Slashdot is a technology-related news website known for its specific user community. The website features user-submitted and editor-evaluated current primarily technology oriented news. In 2002 Slashdot introduced the Slashdot Zoo feature which allows users to tag each other as friends or foes. The network contains friend/foe links between the users of Slashdot. The network was obtained in November 2008. Also, we assign the weight of each edge to be 1.

`Trading` data set. This data set was generated by U.S. Commodity Futures Trading Commission (CFTC), which simulates the transaction-level data for regular transactions in the E-mini S&P 500 futures contract market (Adamic et al. 2010). The E-mini S&P 500 futures contract is a highly liquid, fully electronic, cash-settled

contract traded on the CME GLOBEX trading platform. It is designed to track the price movements of the S&P 500 Index - the most widely followed benchmark of stock market performances.

This data set is simulated based on the data characteristics of the real-world transaction data. There are 5,163,274 transactions among 100 simulated trading accounts for 20 consecutive trading days. Each transaction has the following data fields: date and time, unique transaction ID, executing trading account, opposite trading account, buy or sell flag (for the executing trading account), price, and quantity. We construct a trading network based on transactions in one particular trading day, which has 264,600 transactions. Each trading account represents a node in the network, and a directed edge from node i to node j represents that there is a transaction between account i and j, such that i is in a short position and j is in a long position. The weight associated with each edge is the trading volume of that transaction.

Experimental platform. All the experiments were performed on a Dell Optiplex 960 Desktop with Intel Core 2 Quad Processor Q9550 (2.83 GHz) and 4 GB of memory running the Windows XP Professional Service Pack 3 OS.

Experimental tool. We used LEDA (Library of efficient data types and algorithms) (Mehlhorn and Naher 1999), which is one of the best resources available to support combinatorial computing, as our experimental tool. LEDA offers a complete collection of well-implemented C++ data structures and types, especially for graphs. It supports all the basic graph operations in an efficient way. The data structure that is used to represent a directed graph in LEDA is the adjacency list.

5.2 The pruning effect on the number of combinations

In this subsection, we provide a comparison of the number of combinations, which have been searched, using different algorithms on different data sets. The results are shown in Table 2. *approx-10%* denotes the approxBlackhole algorithm with parameter $p = 10\%$.

In Table 2, we can see that the pruning effect of both gBlackhole and approxBlackhole algorithms are very significant. The search space can be reduced thousands of orders of magnitude by applying the gBlackhole algorithm to all three data sets. In addition, by using the approxBlackhole algorithm, the search space can be further pruned out by $50 - 95\%$. The experimental results validate our theoretical analysis in

Table 2 The number of combinations searched

Algorithm	Wiki	Citation	Slashdot
brute-force	6.74E+2141	2.41E+10399	4.79E+23287
gBlackhole	25,579	128,513	155,251
approx-20%	11,228	79,431	25,372
approx-10%	9,231	68,147	18,192
approx-5%	8,112	62,791	12,747
approx-2%	7,356	57,600	8,134

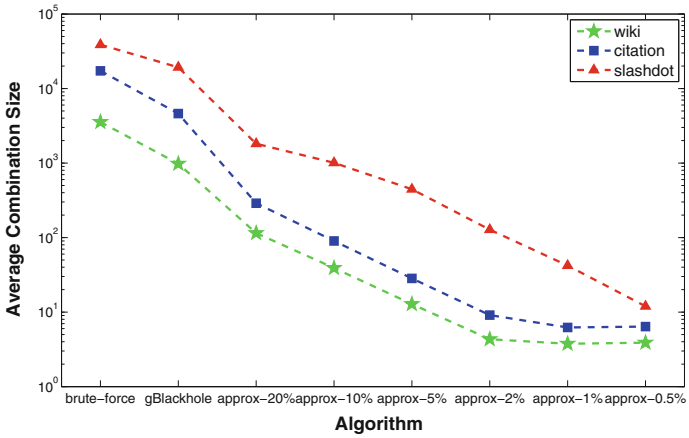

Fig. 6 The pruning effect of the average combination size

previous sections. We will give more detailed comparisons between the gBlackhole and approxBlackhole algorithms in the following subsections.

5.3 The pruning effect on the average combination size

In this subsection, we provide a comparison of the average combination size using different algorithms on different data sets. As shown in Fig. 6, the average combination size in brute-force algorithm is the largest, while the average size in approxBlackhole algorithm is the smallest, and the average size decreases as p gets smaller. This result is consistent with the intuition, since the number of nodes in the induced subgraph is getting smaller as p decreases. The results show that the effect of reducing the average combination size is significant.

5.4 The performances of the gBlackhole algorithm

In this subsection, we provide experiments to study how the running time of the gBlackhole algorithm changes with different input graph sizes. Here, we perform sampling on a large graph to derive a series of graphs with different sizes. In this way, this group of graphs will have the same graph properties, which makes the comparison less biased.

Regarding the sampling method, we refer to Leskovec and Faloutsos's work in 2006 (Leskovec and Faloutsos 2006). We chose the *Random Walk* (RW) sampling method, since it performs the best among all the sampling methods to meet the scale-down criteria, which measure how good the sampled graph inherits the graph properties from the original graph. Next, we sampled a series of graphs from the `Slashdot` data set with different number (percentage) of nodes, and applied the gBlackhole algorithm on each of them. We repeated the procedure for 10 times, and average the corresponding

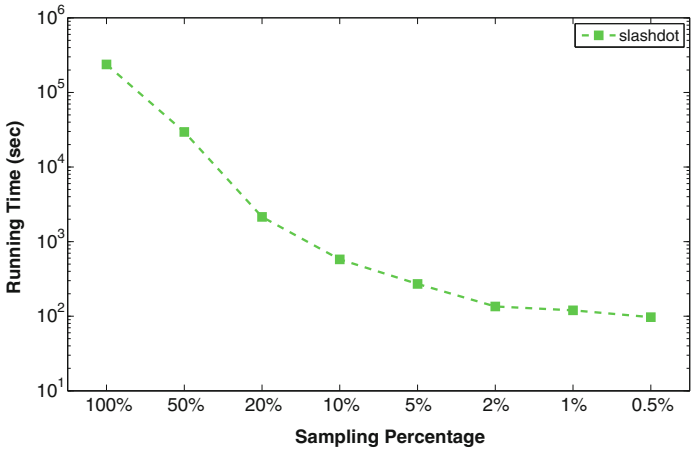

Fig. 7 The running time of gBlackhole on data sets with different sizes

results as the performance of the gBlackhole algorithm on input graphs with different sizes.

The experimental results are shown in Fig. 7. In general, the running time of the gBlackhole algorithm decreases in an approximately exponential way as the size of the graph shrinks. This result proves our theoretical analysis in Sect. 4, because the size of the input graph decreases, both the number of combinations (N) and the average combination size (M) decrease. Especially, the value of N follows an approximately exponential decrease, which results in the exponential decrease of the running time of the gBlackhole algorithm.

5.5 gBlackhole vs. approxBlackhole

In this subsection, we compare the performances of gBlackhole and approxBlackhole algorithms on different data sets. As shown in Fig. 8, the running time of the approxBlackhole algorithm is much smaller than the gBlackhole algorithm over all three data sets, which can save at least 90% of the running time, regardless what the value of p is. The results are consistent with the theoretical analysis in previous sections.

Meanwhile, as shown in Table 3, the completeness rates of detecting the top-K blackhole patterns in different data sets by using the approxBlackhole algorithm are excellent. The completeness rate is calculate as C/K, where C is the number of common blackhole patterns detected by both gBlackhole and approxBlackhole algorithms. For `Wiki` and `Slashdot` data sets, the completeness rates are all 100% for different p. For the `Citation` data set, the results are also very good. Even for the case of $p = 0.5\%$, although the completeness rate is only 69%, the approxBlackhole algorithm finds out all the top 40 blackhole patterns, which are much more important than the missing 31 blackhole patterns.

The performance of the approxBlackhole algorithm depends on the properties of the input graph. Consider Fig. 8 again, we can observe that the running time of the

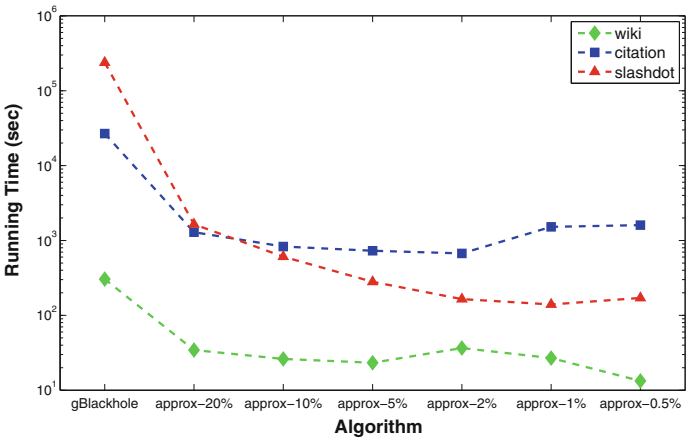

Fig. 8 A comparison of gBlackhole and approxBlackhole algorithms

Table 3 The completeness rate of the approxBlackhole algorithm

Algorithm	Wiki(%)	Citation(%)	Slashdot(%)
Approx-20%	100	100	100
Approx-10%	100	100	100
Approx-5%	100	100	100
Approx-2%	100	100	100
Approx-1%	100	81	100
Approx-0.5%	100	69	100

approxBlackhole algorithm on the Citation data set increases significantly when p becomes very small. This is very different from what we have observed in the other two data sets. Also, the running time of the approxBlackhole algorithm on Citation is much longer than on Slashdot when p is very small, although the size of Slashdot is much larger. Note that Citation is a citation network, while Slashdot is a social network, they have different graph properties. Figure 9 shows how the average node degree of the two networks evolves as p decreases, which we think may be one of the reasons to explain the observations above. In the figure, comparing with Slashdot, we can see that the average node degree of Citation decreases continuously and becomes very small (less than 4) as p decreases, which makes the subgraph induced from Citation less and less connected comparing to the original graph. Though the number of nodes decreases as p decreases, there are considerable combinations, which used to be weakly connected and could be inserted into the *Blackhole* priority queue after they had been checked out, are no longer weakly connected in the subgraph any more when p gets small. Therefore, these combinations are not qualified to become a blackhole pattern and have to be skipped after checking. The number of combinations that have to be checked will increase, which makes the running time becomes longer when p is very small. It may also be the reason why the completeness rate decreases when p gets small.

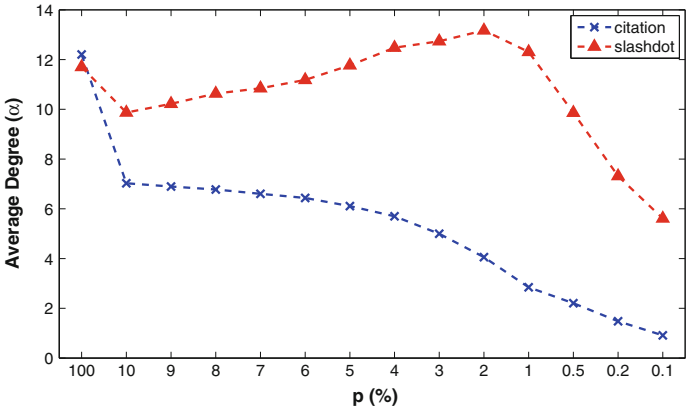

Fig. 9 The average node degree (α) on different data sets with different p

5.6 Blackhole patterns in trading network

In this subsection, we show the results of applying the blackhole mining framework on the `Trading` data set. The running time of the gBlackhole algorithm on this data set is about 1 s.

Here, we have detected a group of blackhole patterns in this `Trading` data set. Due to the page limitation, we only show one suspicious blackhole pattern (in our opinion). In Fig. 10, we can see that this is a blackhole pattern consists of four trading accounts, which forms a very typical trading-ring pattern. During this trading day, account A first places a big amount of sell orders in the market. A small part of them are matched with some buy orders in the market; however, the volume of sell orders highly overwhelms the volume of buy orders during that period, which makes the sell orders still remain a lot. This large volume of sell orders placed in the market makes other people feel that there is an anticipation in the market that the S&P 500 index will fall down in the near future. Therefore, these public accounts start to short the S&P 500 index at a lower price than A. Meanwhile, A's collaborators B, C, and D, place buy orders to match with the sell orders from the public accounts, which get them a large amount of long positions at a low price. A revokes its remaining sell orders in the market, and starts to long the S&P 500 index to offset its early short position, and finally gets a long position. By doing this way, these accounts can accumulate a large amount of long positions at a low price. This is just a simple description about their collaborative strategy. In practice, it would be much more complicated. This detected blackhole pattern can be seen as an alarm for CFTC to oversee the accounts' owners future movements in both the futures contract market and the underlying stock market.

Clearly, there can be other explanations regarding this blackhole pattern, and our concerns can be a false alarm. However, the purpose of our blackhole mining framework is to reduce the workload of people by preliminarily filtering out a huge number of normal patterns automatically, thus people can be more focused on the suspicious patterns that may commit frauds. Also, blackhole patterns can provide us a view of the interactions among some nodes in the network, which can help to detect the

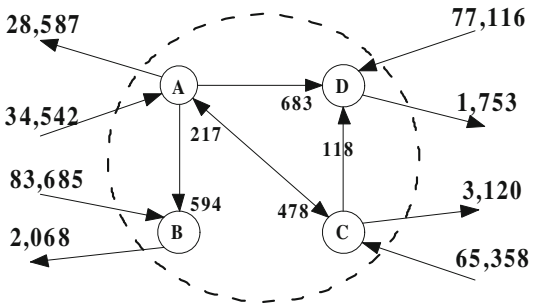

Fig. 10 A suspicious blackhole pattern in the `Trading` data set

collaborative fraud activities. For example, the trading system will trigger an alarm when the trading volume on a specific account exceeds a certain threshold. However, it cannot identify the interactions among these accounts. Our blackhole patterns will not only be able to identify the interactions, but may also able to find out accounts which is under the threshold that collaborate with the suspicious accounts. However, the use of blackhole patterns is still preliminary and more comprehensive studies are expected in the future.

6 Related work

This article is an extended study of our preliminary work (Li et al. 2010). In Li et al. (2010), we proposed a simplified version of blackhole patterns, where the blackhole patterns have the assumption that there is no traffic flow out of the blackhole. Also, there is no weight associated with the edge. In the preliminary work, we provided a heuristic approach to solve the problem. The preliminary solution was based on a set of pattern-size-independent pruning rules and a divide-and-conquer strategy, named as the iBlackhole-DC algorithm, which was designed by exploiting the pruning rules drived from the simplified definition of blackhole patterns.

In general, other related works can be grouped into three categories. The first category includes the work on frequent subgraph mining, which studies how to efficiently find frequent subgraphs in the graph data. For instance, Jiang et al. (2009) proposed a measure for mining globally distributed frequent subgraphs in a single labeled graph. Meanwhile, there are many works in mining frequent subgraphs in multiple labeled graphs, such as Yan and Han (2002); Cook and Holder (1994); Huan et al. (2003); Wang et al. (2006, 2004) and Kuramochi and Karypis (2005). The problems of detecting blackhole and volcano patterns are different from the above works, since their definitions are different. Also, blackhole and volcano patterns may not be frequent subgraphs and vice versa.

The second category includes the works for detecting community structures in large networks. Communities in a network are groups of nodes in which connections are dense, but between which connections are sparse (Newman 2004). There are a lot of works on how to detect communities in a network. For instance, Newman and Girvan (2004, 2002) proposed a betweenness-based method, Hopcroft et al. (2003) proposed a stable method, and Ghosh and Lerman (2008) proposed a global influence based

method to detect community structures. In addition, there are some other methods from a probabilistic view (Pathak et al. 2008; Zhou et al. 2006; Steyvers et al. 2004), rather than based on the links in the network. All the methods of detecting community structures are based on certain definitions and criteria. However, the definitions of blackhole and volcano patterns are different from the above definitions of communities. While the blackhole and volcano patterns are more focusing on small groups which satisfy some specific requirements, the community structures in a network are more vague and flexible. In terms of the scope in the network, blackhole and volcano patterns are more localized, while community structures are more globalized.

Finally, there is a third category of research works related to this study, which is called anomaly or outlier detection (Hawkins 1980; Johnson and Wichern 1998; Barnett and Lewis 1994). Anomaly or outlier detection aims to find out objects different from most other objects in the data. Many efforts have been made to deal with this problem and lots of approaches derived from statistics (Barnett and Lewis 1994), machine learning (Breunig et al. 2000; Papadimitriou et al. 2003), and data mining (Chaudhary et al. 2002; Lazarevic and Kumar 2005) have been developed. Most of these methods focus on traditional data types, such as vectors. Moreover, there are emerging studies on detecting outliers in graphs. Noble and Cook (2003) studied anomalous subgraphs in general graphs using Minimum Description Length (MDL). Autopart (Chakrabarti 2004) applied MDL to find out anomalous edges. Sun et al. (2005) detected anomalous nodes in bipartite graphs using random walk. OutRank also employed random walk and connectivity of nodes to detect outliers (Moonesinghe and Tan 2008). OddBall (Akoglu et al. 2010) focused on detecting anomalous nodes in weighted graphs. For all the approaches mentioned above, they mainly tackle either nodes or edges in undirected graphs. In contrast, our work focuses on detecting a well-defined anomalous structures in a directed graph.

7 Concluding remarks

In this article, we formulated the problem of mining generalized blackhole and volcano patterns in a directed graph. These two patterns could be observed in many application scenarios, such as the trading-ring for stock market manipulation. Indeed, it is essentially a combinatorial problem for mining blackhole or volcano patterns. To reduce the complexity of the problem, we first showed that the problem of finding blackhole patterns is a dual problem of finding volcano patterns. Thus, we could be only focused on mining blackhole patterns. To that end, we proposed two pruning approaches to reduce the computational cost by decreasing both the number of combinations and the average computational cost for each combination. In the first pruning approach, we introduced the concept of combination dominance to help develop a pruning technique to reduce the exponentially growing search space. Based on this pruning technique, we developed the gBlackhole algorithm for finding top-K blackhole patterns. The second pruning approach, named the approxBlackhole algorithm, is an approximate algorithm to further decrease the computational complexity over the gBlackhole algorithm. This approxBlackhole algorithm first filters out nodes with small diff-weights to reduce the size of the graph, and then finds the top-K blackhole patterns in the subgraph induced

by the rest of the nodes. There is a trade-off between the efficiency and completeness of the approxBlackhole algorithm.

Finally, experimental results on real-world data sets showed that the approxBlackhole algorithm could be several orders of magnitude faster than the gBlackhole algorithm, and both of them had a huge computational advantage over the brute-force approach. Finally, we showed the effectiveness of this generalized blackhole mining framework by identifying some suspicious financial fraud patterns in the simulated E-mini S&P 500 futures contract trading data from U.S. Commodity Futures Trading Commission.

Acknowledgements This research was supported in part by National Science Foundation (CCF-1018151) and National Natural Science Foundation of China (NSFC) via project numbers 70890082 and 71028002. The authors would also like to thank the editors and the anonymous reviewers for their valuable and constructive comments on this article.

References

Adamic L, Brunetti C, Harris J, Kirilenko A (2010) Trading networks. SSRN eLibrary. http://ssrn.com/paper=1361184

Akoglu L, McGlohon M, Faloutsos C (2010) Oddball: Spotting anomalies in weighted graphs. In: Proceedings of the 14th pacific-Asia conference on knowledge discovery and data mining (PAKDD'10), Hyderabad, pp 410–421

Barnett V, Lewis T (1994) Outliers in statistical data. Wiley, New York

Breunig MM, Kriegel H-P, Ng RT, Sander J (2000) Lof: Identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD international conference on management of data (ACM SIGMOD'00), Providence, pp 93–104

Chakrabarti D (2004) Autopart: Parameter-free graph partitioning and outlier detection. In: Proceedings of the 8th European conference on principles and practice of knowledge discovery in databases (PKDD'04), Pisa, pp 112–124

Chaudhary A, Szalay AS, Moore AW (2002) Very fast outlier detection in large multidimensional data sets. In: Proceedings of ACM SIGMOD workshop on research issues in data mining and knowledge discovery, Dalas

Cook DJ, Holder LB (1994) Substructure discovery using minimum description length and background knowledge. J Artif Intel Res (JAIR) 1:231–255

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. The MIT Press, Cambridge

Diestel R (2006) Graph theory (Graduate texts in mathematics). Springer, Heidelberg

Gehrke J, Ginsparg P, Kleinberg JM (2003) Overview of the 2003 KDD Cup. In: ACM SIGKDD Explorations 5(2):149–151

Ghosh R, Lerman K (2008) Community detection using a measure of global influence. In: The 2nd SNA-KDD workshop on social network mining and analysis (SNA-KDD'08), Las Vegas

Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99(12):7821–7826

Hawkins D (1980) Identification of outliers. Chapman and Hall, Dordrecht

Hopcroft J, Khan O, Kulis B, Selman B (2003) Natural communities in large linked networks. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining (ACM SIGKDD'03), Washington

Huan J, Wang W, Prins J (2003) Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism. In: Proceedings of the 3rd IEEE international conference on data mining (IEEE ICDM'03), Melbourne

Jiang X, Xiong H, Wang C, Tan AH (2009) Mining globally distributed frequent subgraphs in a single labeled graph. Data Knowl Eng 68:1034–1058

Johnson RA, Wichern DW (1998) Applied multivariate statistical analysis. Prentice Hall, New York

Knuth D (2011) The art of computer programming, Vol 4A: combinatorial algorithms. Addison-Wesley, Boston

Kuramochi M, Karypis G (2005) Finding frequent patterns in a large sparse graph. Data Mining Knowl Discov 11(3):243–271

Lazarevic A, Kumar V (2005) Feature bagging for outlier detection. In: Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining (ACM SIGKDD'05), Chicago, pp 157–166

Leskovec J, Faloutsos C (2006) Sampling from Large Graphs. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining (ACM SIGKDD'06), Philadelphia, pp 631–636

Leskovec J, Huttenlocher D, Kleinberg J (2010a) Predicting Positive and Negative Links in Online Social Networks. In: Proceedings of the 19th international world wide web conference (WWW'10), Raleigh

Leskovec J, Huttenlocher D, Kleinberg J (2010b) Signed Networks in Social Media. In: Proceedings of the 28th ACM conference on human factors in computing systems (CHI'10), Atlanta

Leskovec J, Kleinberg J, Faloutsos C (2005) Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD'05), Chicago

Leskovec J, Lang K, Dasgupta A, Mahoney M (2008) Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. In: arXiv.org:0810.1355

Li Z, Xiong H, Liu Y, Zhou A (2010) Detecting Blackhole and Volcano Patterns in Directed Networks. In: Proceedings of the 10th IEEE International Conference on Data Mining (IEEE ICDM'10), Australia, pp 294–303

Mehlhorn K, Naher S (1999) The LEDA platform of combinatorial and geometric computing. Cambridge University Press, Cambridge

Moonesinghe HDK, Tan P-N (2008) Outrank: a graph-based outlier detection framework using random walk. Int J Artif Intel Tools 17(1):19–36

Newman MEJ (2004) Detecting community structure in networks. Eur Phys J B 38:321–330

Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69:026113

Noble CC, Cook DJ (2003) Graph-based anomaly detection. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining (ACM SIGKDD'03), Washington, pp 631–636

Papadimitriou S, Kitagawa H, Gibbons PB, Faloutsos C (2003) Loci: Fast outlier detection using the local correlation integral. In: Proceedings of the 19th international conference on data engineering (ICDE'03), Bangalore, pp 315–326

Pathak N, DeLong C, Banerjee A, Erickson K (2008) Social topic models for community extraction. In: The 2nd SNA-KDD Workshop on Social Network Mining and Analysis (SNA-KDD'08), Las Vegas

Steyvers M, Smyth P, Rosen-Zvi M, Griffiths T (2004) Probabilistic author-topic models for information discovery. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining (ACM SIGKDD'04), Magdeburg

Sun J, Qu H, Chakrabarti D, Faloutsos C (2005) Neighborhood formation and anomaly detection in bipartite graph. In: Proceedings of the 5th IEEE international conference on data mining (IEEE ICDM'05), Houston, pp 418–425

Tan P-N, Steinbach M, Kumar V (2005) Introduction to data mining. Addison Wesley, Boston

Wang C, Wang W, Pei J, Zhu Y, Shi B (2004) Scalable mining of large disk-based graph databases. In: Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining (ACM SIGKDD'04), Magdeburg

Wang J, Hsu W, Lee M, Sheng C (2006) A partition-based approach to graph mining. In: Proceedings of the 22nd international conference on data engineering (ICDE'06), Atlanta, p 74

Yan X, Han J (2002) gSpan: graph-based substructure pattern mining. In: Proceedings of the 2nd IEEE international conference on data mining (IEEE ICDM'02), Maebashi

Zhou D, Manavoglu E, Li J, Giles CL, Zha H (2006) Probabilistic models for discovering e-communities. In: Proceedings of the 15th international world wide web conference (WWW'06), Edinburgh