Subgraph Mining on Directed and Weighted Graphs

Stephan Günnemann and Thomas Seidl

Data management and data exploration group
RWTH Aachen University, Germany
{guennemann,seidl}@cs.rwth-aachen.de

Abstract. Subgraph mining algorithms aim at the detection of dense clusters in a graph. In recent years many graph clustering methods have been presented. Most of the algorithms focus on undirected or unweighted graphs. In this work, we propose a novel model to determine the interesting subgraphs also for directed and weighted graphs. We use the method of density computation based on influence functions to identify dense regions in the graph. We present different types of interesting subgraphs. In experiments we show the high clustering quality of our GDens algorithm. GDens outperforms competing approaches in terms of quality and runtime.

1 Introduction

Today's complex data can often be described by graphs. The nodes in a graph are objects while the edges illustrate connections between the objects. Examples include biological networks or social networks. A common property of such graphs is the existence of densely connected subgraphs. These clusters or communities are separated by less dense regions in the graph. We can gain a benefit of finding such interesting subgraphs. Based on a biological network, the development of useful drugs deduced from functional modules in protein interaction networks, is one example. In online commercial systems one can use interesting subgraphs for target delivery of customers. Customers in the same interesting subgraph show similar behavior.

Beside the connections between the objects in many cases also the directions and the weights are given. Let us consider a graph that represents the network traffic e.g. of the Internet. Edges that connect the routers of an ISP usually have higher weights/traffic amount than edges to nodes reflecting an end-user PC. These weights are important to identify the core and hence the dense subgraph of the total graph. Second, in general end-users generate more downlink traffic than uplink traffic; thus, the ingoing and outgoing edges are not identical and should be treated separately. Another example is an author graph where the edge weights can be interpreted as the number of co-written papers and the direction as the first author vs. co-author relationship. Overall, the identification of interesting subgraphs based on directed and weighted graphs is an important research field.

M.J. Zaki et al. (Eds.): PAKDD 2010, Part II, LNAI 6119, pp. 133–146, 2010.

Related Work. Several graph mining algorithms have been presented to the community. The identification of optimal dense subgraphs based on some objective function is usually a hard problem [1], so that approximations or simple models are used. Some simple models for the detection of dense subgraphs include the identification of cliques or more meaningful of quasi-cliques [2, 3]. These algorithms usually generate a huge output, even if we retain only the maximal quasi-cliques, and the subgraphs overlap to a very high extend, resulting in marginal differences between subgraphs. Furthermore only undirected and unweighted graphs are used.

Another area is graph partitioning. Algorithms from this area try to divide the graph in flat parts; within these parts the nodes are strongly connected while between different subgraphs only loose connectivity exists. Models based on the maximum flow principle [4] or the k-partite graph partitioning [5] follow this paradigm. Another approach using physical principles is presented in [6]. Further techniques are the spectral clustering [7] or relational clustering [8]. One problem is that usually the number of interesting subgraphs or the size of the groups must be given. Furthermore, each node belongs to or is considered as a cluster even it is not well suited for this. Thus, the SCAN model [9] additionally identifies hubs or outliers; noise nodes are not included in the clusters. Directed and weighted graphs are not considered.

Moreover, several hierarchical/recursive methods exist that split up the graph step by step in smaller subgraphs, e.g. based on the cut-principle [10–12]. Paradigms based on edge betweenness are also used [13] and were extended to allow overlapping subgraphs [14, 15]. However, expensive recalculations are often performed. The modularity [16–18] is another well know measure, which is used for the recursive identification of subgraphs. All methods generate a complete hierarchy of interesting subgraphs. Each cut through this hierarchy represents a more or less meaningful partitioning of the graph in interesting subgraphs. However, many cuts are possible and the user is cluttered with a huge amount of subgraphs. Additionally, the construction of a complete hierarchy results in a high runtime.

Our Contributions. Our model is able to determine the interesting subgraphs on directed and weighted graphs. The direction of edges is a particular aspect, which leads to different definitions of dense subgraphs. Furthermore we present a definition of our interesting subgraphs based on the principle of density calculation. The number of clusters and their sizes are automatically determined and the clusters are not obfuscated by noisy nodes, i.e. nodes that do not belong to any interesting subgraph.

2 Mining Interesting Subgraphs

In this section we present our model for the mining of interesting subgraphs. Section 2.1 starts with some preliminaries. In Section 2.2 the density calculation

on graphs is described, followed by our dense subgraph definitions in Section 2.3. In Section 2.4 algorithmic aspects are discussed and Section 2.5 concludes with some complexity results.

2.1 Preliminaries

A directed and weighted graph G is a tuple (V, E, w) with nodes V, edges $E \subseteq V \times V$, and a weighting function $w : E \rightarrow \mathbb{R}^+$. A path in G is a list of nodes $< p_1, \ldots, p_n >$ with $(p_i, p_{i+1}) \in E$. A node v is reachable from u along a set of nodes M, if there exists a path $< p_1, \ldots, p_n >$ with $p_1 = u$, $p_n = v$ and $p_i \in M$. Usually $M = V$, i.e. one can use all nodes in the graph for the path.

For density calculation, each object influences any other object with a certain value. The sum of influences determines the density of an object/a point in the data space. The influence value is usually based on two criteria. First, one calculates the distance between the objects, e.g. for two vectors o, p the Euclidean distance $d_2(o, p)$. Second, the distances are weighted according to a weighting function $\mathcal{W} : \mathbb{R} \rightarrow \mathbb{R}$. For this purpose often kernel functions are used as the Uniform, the Gaussian or the Epanechnikov kernel. The Epanechnikov kernel is known to be efficient and effective [19]. The overall influence of an object o on another object p is obtained by $influence(o, p) = \mathcal{W}(\frac{d(o,p)}{h})$. The factor h is used to scale the distances. The smaller the distance between two objects the higher is their influence on each other. The overall density of an object p is then calculated as the sum of influences $influence(o, p)$ for each object o in the database.

2.2 Density Computation on Graphs

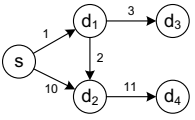
The challenge in our task is to consider the underlying graph structure to define a meaningful density calculation. Thus, in a first step we have to use graph based distances. Nodes that are 'stronger' connected, i.e. are more similar with respect to the selected distance function, should have a higher influence on each other. For ease of presentation we assume that smaller edge weights correspond to higher connectivity of the nodes. If the reverse is true, e.g. the number of co-written papers should be high for a strong connection between two nodes in an author graph, we can simply transform the weights to $1/w(u, v)$. In our approach we use the shortest path distance between two nodes s, d as a representative for a graph based distance function.

Definition 1. *Shortest path distance and influence*
Given a weighted graph $G = (V, E, w)$, the shortest path distance between node s and d is defined as

$$d_{min-path}(s, d) = \min_{path\ <p_1,\ldots,p_n>} \{\sum_{i=1}^{n-1} w(p_i, p_{i+1}) \mid p_1 = s \wedge p_n = d\}$$

The influence based on the Epanechnikov kernel is defined by

$$influence(s, d) = \mathcal{W}(\tfrac{d_{min-path}(s,d)}{h}) \text{ with } \mathcal{W}(x) = \begin{cases} \frac{3}{4}(1 - x^2) & |x| \leq 1 \\ 0 & else \end{cases}$$



$d_{min-path}(s, d_1) = 1$
$d_{min-path}(s, d_2) = 3$
$d_{min-path}(s, d_3) = 4$
$d_{min-path}(s, d_4) = 14$

$influence(x, y) > 0$ iff $d_{min-path}(x, y) < 5$

Fig. 1. Influence region and direct influence

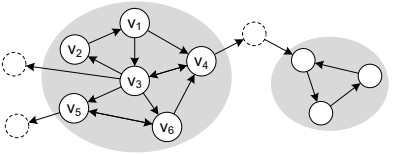


Fig. 2. Cluster cores

By this definition the influence decreases quadratically with the distance and the influence of a node s is not restricted to its direct neighbors but we affect also nodes that are reachable over longer paths. Thus, the connectivity of the nodes is more precisely reflected. In Figure 1 the node d_3 is influenced by s even though it is not a direct neighbor. Furthermore, we do not simply count the number of near located nodes but we weight them according to their distance. Or even stronger, with the non-linear weighting of $\mathcal{W}$ a single very close node results in a higher influence than two not as close nodes. Due to the compact support of the Epanechnikov kernel $\mathcal{W}$ we only have to calculate the shortest path distances up to h, i.e. we do not need to analyze all nodes; in contrast to functions with a non-compact support as the Gaussian kernel. Thereby we increase the efficiency of our method. In contrast to classical density computation our influence function need not to be symmetric, i.e. $influence(s, d) \neq influence(d, s)$ is possible. This non-symmetry is particularly appropriate for directed graphs.

Another important aspect in graphs is the possible influence on nodes that are at a first glance not influenced. In Figure 1 for example the edge between the nodes s and d_2 has a too high weight; thus, s does not influence d_2 based on this edge. However, we can use a 'detour' including other nodes to get an positive influence. Therefore, we distinguish two different node sets: First, the overall set of nodes on which the node s has influence on, the so called influence region. Second, the directly influenced nodes that correspond to closely located neighboring nodes.

Definition 2. *Influence region and direct influence*
Given a graph G and a node s, the influence region $region(s)$ of s is defined by

$$d \in region(s) \Leftrightarrow influence(s, d) > 0$$

The set $direct(s)$ contains all nodes that are directly influenced by s and is defined by

$$d \in direct(s) \Leftrightarrow (s, d) \in E \wedge w(s, d) < h$$

This distinction is based on the intuition that one only interacts with good friends that are directly known to oneself, e.g. to spread a rumor. However, a rumor spreaded by a person can also reach persons not only in its neighborhood, but along a path of good friends. In Figure 1 we get $region(s) = \{s, d_1, d_2, d_3\}$ and $direct(s) = \{d_1\}$ and obviously $direct(s) \subseteq region(s)$ holds. By our definition, the influence region is connected. Even stronger, s can reach all other nodes in its influence region.

Now we are able to calculate the density of a node d. We have to determine all nodes that have an influence on d; in contrast to the nodes which d itself influences, according to the non-symmetry. We call this set of objects the reverse influence region and define $revRegion(d) = \{s \in V \mid d \in region(s)\}$. The density of a node d is the sum of all influences from the objects in this set.

Definition 3. *Density of a node*
The density of a node d is defined by

$$density(d) = \sum\nolimits_{s \in revRegion(d)} influence(s, d)$$

In Figure 1 the reverse influence region of node d_2 is $\{s, d_1, d_2\}$. The actual density can be calculated based on the individual influence values. The higher the density the more interesting is the node in our graph.

2.3 Density Based Clusters on Graphs

Our idea is to identify interesting subgraphs, i.e. subgraph clusters, by dense areas that are separated by sparse areas. For vector spaces similar approaches show good performance even in the presence of noise [20, 21]. We will call a node dense if its density exceeds a threshold τ. All nodes fulfilling this minimal density criterion are the core nodes, as these nodes are the candidates that build the cores of different clusters.

Definition 4. *Core nodes*
Given a graph G and a minimal density τ, the set of core nodes is defined by

$$coreNodes = \{v \in V \mid density(v) \geq \tau\}$$

Starting with a core node, we add further core nodes, which are in a certain range of the node, to our cluster to let it grow. In this step we have to take care of the underlying graph structure. Two aspects are important and have to be considered.

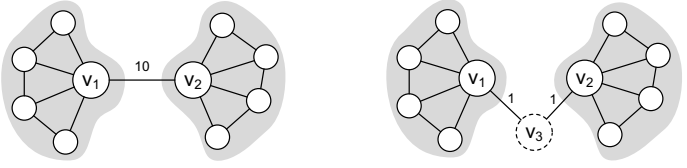
Aspect 1: We have to grow the clusters along our direct influence definition. A node only interacts with the nodes in its direct influence region; thus, if these nodes are core nodes they should belong to the same cluster. In Figure 2 a directed graph is given where the core nodes are marked with solid lines and the non-core nodes with dashed lines. For ease of presentation we do not show the densities or edge weights. If we select v_1, the nodes $\{v_3, v_4\} = direct(v_1) \cap coreNodes$ should correspond to the same dense region and hence to the same interesting subgraph. Because a directed graph is given, also the predecessors $\{s \in coreNodes \mid v_1 \in direct(s)\} = \{v_2\}$ have to be included. This is equivalent to $v_1 \in direct(v_2) \cap coreNodes$. The same procedure is now applied for v_2, v_3 and v_4, so that we add the nodes v_5 and v_6 to the cluster.

Definition 5. *Core of a cluster*
A non-empty subset $C \subseteq coreNodes$ is a core of a cluster iff

$$\forall\, v \in C : \forall\, s \in coreNodes : s \in direct(v) \vee v \in direct(s) \Rightarrow s \in C$$

and C is minimal among these sets.

By this definition, a cluster grows until for each node the required property is fulfilled. In Figure 2 the two minimal sets that fulfill this property are highlighted. These minimal sets are the dense areas that build our clusters cores. In Figure 3 we show examples for the identification of two different cores to point out the advantages of our definition. In Figure 3(a) we get two cores even though the two core nodes v_1 and v_2 are connected. The connection is only very loose; thus, both nodes do not influence each other. In Figure 3(b) we get two cores even if v_1 and v_2 are reachable over a path. This path, however, has to use the non-core node v_3. This non-core node indicates the existence of two different clusters.



(a) (b)

Fig. 3. Examples for the identification of two different cores

Aspect 2: The second aspect we have to consider is the non-symmetry of our direct influence region, i.e. we can have $d \in direct(s) \not\Leftrightarrow s \in direct(d)$. Why are directed graphs a particular challenge? For this, we first consider the Definition 5 for undirected graphs. Consequently with our model we get $d \in direct(s) \Leftrightarrow s \in direct(d)$ and Definition 5 simplifies to

$$\forall\, v \in C : \forall\, s \in coreNodes : s \in direct(v) \Rightarrow s \in C$$
$$\Leftrightarrow\ \forall v \in C : coreNodes \cap direct(v) \subseteq C$$

for a minimal non-empty set C.

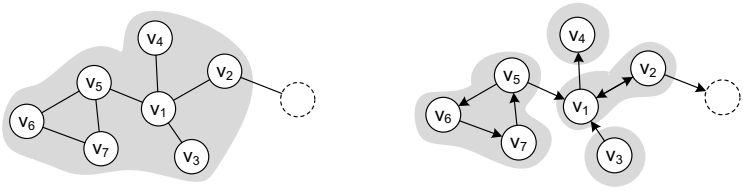
A useful property for interpreting the cores is the core path property. A path (within a cluster core) that uses only direct influence edges and that connects s with d is called a core path for s and d. Formally we define the existence of such a path with respect to a cluster core C and two nodes by:

$$corePath_C(s,d) = TRUE \Leftrightarrow \exists v_1, \ldots, v_n \in C :$$
$$v_1 = s \wedge v_n = d \wedge v_{i+1} \in direct(v_i) \wedge v_i \in coreNodes$$

In Figure 4(a) the node set $C = \{v_1, \ldots, v_7\}$ is a valid cluster core. The nodes v_4 and v_7 for example are connected via a core path. The path $< v_4, v_1, v_5, v_7 >$ uses only nodes within C and each successor is in the direct influence region of its predecessor. If each pair of nodes within the core C is connected via a core path, C fulfills the core path property. In Figure 4(a) this property holds.

Definition 6. *Core path property*
The core C fulfills the core path property iff

$$\forall s, d \in C : corePath_C(s,d) = TRUE$$



(a) undirected graph (b) directed graph

Fig. 4. Core path property and strong cluster cores

One can prove that a cluster core in an undirected graph always fulfills the core path property. Furthermore, a cluster core C is maximal with respect to the core path property, i.e. there exists no $C' \supset C$ that fulfills also the core path property. In Figure 4(a) we cannot add further nodes to C without violating the core path property. Thus, in an undirected graph the cluster cores correspond to the maximal sets that fulfill the core path property.

For a directed graph the Definition 5 and the maximal sets with respect to the core path property do not necessarily lead to the same results. In Figure 4(b) we get the cluster core $C = \{v_1, \ldots, v_7\}$. However, as one can see the node v_4 is not connected to v_7 via a core path; the nodes v_3 and v_5 are not connected at all. In directed graphs the core path property is a more restrictive criterion for a cluster. The Definition 5 is fulfilled by the nodes $\{v_1, \ldots, v_7\}$ in Figure 4(b) while the core path property e.g. only for the nodes $\{v_1, v_2\}$ or $\{v_5, v_6, v_7\}$. These sets are highlighted in Figure 4(b). Thus, for directed graphs we can define another stronger definition for a core of a cluster:

Definition 7. *Strong core of a cluster*
A non-empty subset $C \subseteq coreNodes$ is a strong core of a cluster iff C fulfills the core path property and C is maximal.

Obviously each strong core SC is a subset of a core C, i.e. $SC \subseteq C$. Thus, we have the restrictive strong core property, which yields small cluster cores, and the weaker core property, which yields larger clusters. We want to analyze a further version in between these extremes. In contrast to the strong core property, where each pair of nodes is reachable in both directions, we make a relaxation that only requires a core path in one direction.

Definition 8. *Semi-strong core of a cluster*
A non-empty subset $C \subseteq coreNodes$ is a semi-strong core of a cluster iff

$$\forall s, d \in C : corePath_C(s, d) = TRUE \ \vee corePath_C(d, s) = TRUE$$

and C is maximal.

In Figure 4(b) for example the node set $C = \{v_1, v_2, v_3, v_4\}$ forms a semi-strong core. The node v_3 can reach all nodes in C via a core path, v_2 the nodes $\{v_1, v_4\}$ and v_1 the nodes $\{v_2, v_4\}$. For each pair we get at least one core path in a single direction.

Precluster and postcluster. The cores are the most important nodes that form the clusters. Additionally, two other sets of nodes can be defined. Given a cluster core C, the densities of all nodes within the core exceed a certain threshold. Thus, an interesting node set contains all nodes that account for the density of the core, i.e. removing one of these nodes the densities of the core nodes change. We call this set a precluster. On the other hand we can also define the postcluster of C. This set contains all objects that are influenced by the core.

Definition 9. *Precluster and postcluster*
Given a cluster core C, the precluster $Pre(C)$ and postcluster $Post(C)$ contain the nodes

$$Pre(C) = \left(\bigcup_{d \in C} revRegion(d)\right) \setminus C \text{ and } Post(C) = \left(\bigcup_{d \in C} region(d)\right) \setminus C$$

2.4 Graph-Theoretic View and Algorithmic Aspects

In the following we want to point out a graph-theoretic view of our model that helps us to implement an efficient algorithm. We first transform our graph. We remove all non-core nodes and those edges that do not contribute to a direct influence. Only along the remaining nodes and edges a core could grow. Overall, this so called residual graph is defined by $V' = coreNodes$ and $E' = \{(s,d) \in E | d \in direct(s) \wedge \{s,d\} \subseteq coreNodes\}$. In Figure 5 we show an original graph; non-core nodes are highlighted with dashed lines. The two edges labeled with 'noise edge' should indicate, that v_3 and the other node are not directly influenced by v_2 even if they are connected. If we remove these two edges as well as the nodes n_1 and n_2, the residual graph on the right is obtained.

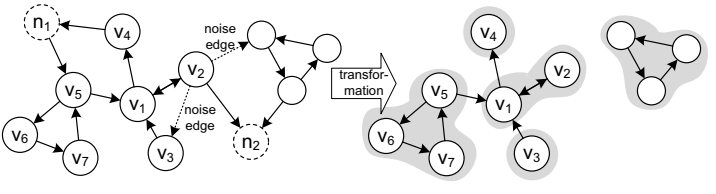


Fig. 5. Graph transformation to the residual graph

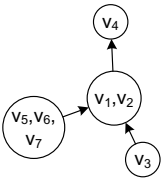


Fig. 6. Quotient graph

The weak components [22] of the residual graph are our cluster cores following Definition 5. The strong components [22] build our strong cluster cores following Definition 7. In Figure 5 (right) we highlighted these node sets. The semi-strong cores obey a more complex structure. Derived from the strong components we can construct a quotient graph. Each node in the quotient graph corresponds to a strong component (cf. Fig. 6). The nodes are connected with a directed edge if at least one edge in the original graph exists between these node sets. This quotient graph is a DAG (directed acyclic graph). Each maximal path from a root node to a leaf node is then a semi-strong component. In Figure 6 we get the

two paths $< \{v_3\}, \{v_1, v_2\}, \{v_4\} >$ and $< \{v_5, v_6, v_7\}, \{v_1, v_2\}, \{v_4\} >$ that define the two semi-strong cores. Thus, the strong, semi-strong and weak components in the residual graph correspond to the definitions of the cluster cores. Due to space limitations we omit the proof.

A comparison of the three definitions yields an interesting conclusion. While the weak and strong components are disjoint sets, the semi-strong node sets can overlap. The cores of our clusters following the semi-strong definition are allowed to share some objects. This overlap or disjointness is already visible in our example in Figure 5 and 6 respectively.

Based on the graph-theoretic properties, our algorithm can efficiently determine the interesting subgraphs. We call our algorithm *GDens*, due to the density calculation in graphs. First, for our density calculation we have to determine the shortest path distances. We use an adaption of Dijkstra's algorithm. By this we can use an early stopping of the density calculation if the distances exceed the maximal distance h. We have to apply Dijkstra's algorithm for each node to determine the overall densities.

After the density calculation step we have to determine the cores of the clusters. For our weak core definition we have to identify the weak components in the residual graph; this can be done by a depth-first search procedure. The strong-components and hence the quotient graphs within each weak component are identified by Tarjan's algorithm. Finally, we generate the maximal paths within the quotient graphs to identify the semi-strong cores.

Summarized, our GDens utilizes efficient graph algorithm methods for identifying the interesting subgraphs. Our beforehand defined core definitions can be reduced to well known properties in the residual graph and hence emphasize the use of these cluster cores. In total, we get the possibility to flexibly identify and interpret different interesting subgraphs based on the density calculation technique in directed and weighted graphs.

2.5 Complexity Analysis

We briefly analyze the complexity of our algorithm with respect to a graph $G = (V, E, w)$. Let us assume that in average the influence is larger than zero for a fraction of x percent of all nodes. Based on the complexity of Dijkstra's algorithm and as we have to use Dijkstra's algorithm for each node to determine the densities, the overall complexity for the density computation step is

$$O\left(|V| \cdot [\ x \cdot |V| \cdot log(x \cdot |V|) + x \cdot |E|\ ]\right) = O(x \cdot |V|^2 \cdot log(x \cdot |V|) + x \cdot |V| \cdot |E|)$$

Obviously in the worst case the influence is larger than zero for all nodes, i.e. $x = 1$, and we have a dense graph, i.e. $O(|E|) = O(|V|^2)$. In this case we can infer the worst case complexity of

$$O\left(|V|^2 \cdot log|V| + V \cdot |V|^2\right) = O(|V|^3)$$

If we assume the positive influence of a node is restricted to a constant number of nodes, i.e. $x = O(1/|V|)$, and a sparse graph with e.g. $O(|E|) = O(c \cdot |V|)$ is given, we get a complexity of

$$O(1/|V| \cdot |V|^2 \cdot log(1/|V| \cdot |V|) + 1/|V| \cdot |V| \cdot c \cdot |V|) = O(c \cdot |V|)$$

In the second step, we have to determine the cores. The depth-first procedure for identifying the weak-components in the residual graph has a complexity of $O(|V| + |E|)$. Tarjan's algorithm for identifying the strong-components has a complexity of $O(V_{max} + E_{max})$, if V_{max} is the maximal number of nodes for all weak components and E_{max} is the maximal number of edges. For the semi-strong core definition we additionally have to generate the maximal paths within the quotient graph. Assuming that we identify k quotient graphs each with n strong components we get an additional complexity of $O(k \cdot e^{n/e})$ (proof skipped). In realistic scenarios we have $k, n \ll |V|$ and hence the term is almost negligible.

3 Experiments

In the next section we analyze the runtime and quality of our *GDens* algorithm.

Setup. We use two variants of our algorithm. *GDens (core)* uses the identified cluster cores as the interesting subgraphs. *GDens (all)* includes the precluster and postcluster to build the interesting subgraphs. For comparison we use the algorithms of [6], called *Voltage*, and [13], called *Edge Betweenness*. All input parameters are optimized. All implementations are in Java. For runtime and quality comparison we generate synthetic data following the methods in [13, 18] and adding uniformly distributed edge weights. In average, edge weights within clusters are two times smaller than edge weights between clusters. As not stated otherwise we hide 20 clusters with 10.000 nodes and 30.000 edges. Clustering quality is measured with the F1 score [23, 24]. The F1 value is the harmonic mean of precision and recall, i.e. an identified subgraph has to detect most of the nodes (recall) but also only the nodes (precision) of a hidden cluster. Keep in mind that for our algorithm two F1 scores can be calculated based on GDens (core) or GDens (all).

Cluster core definitions. In Figure 7 we analyze the effects of our different cluster core definitions. On the left the F1 value is illustrated. The solid bars correspond to the quality for GDens (all). The white bars inside correspond to the quality if only the core is considered, i.e. GDens (core) is used. The quality of GDens (core) decreases slightly with a more restrictive core definition. The

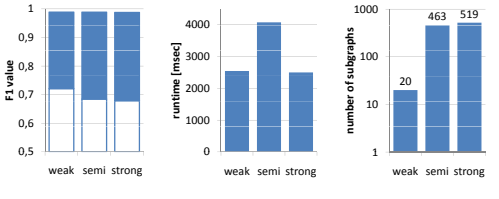


Fig. 7. Cluster core definitions

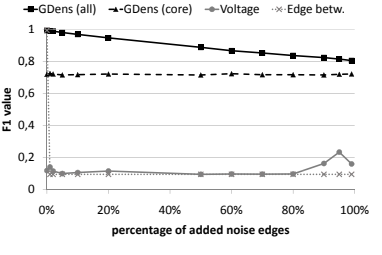


Fig. 8. Quality w.r.t. noise

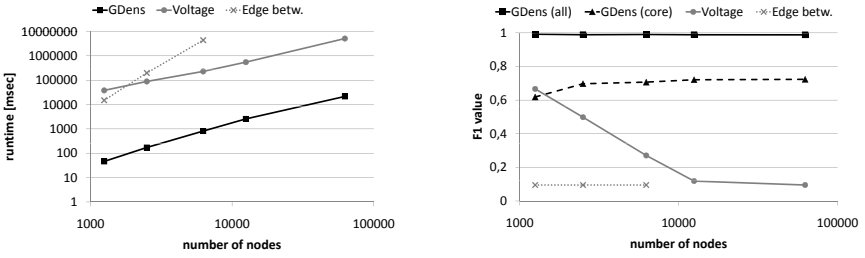


Fig. 9. Runtime and quality w.r.t. number of nodes

cores following the strong cluster core definition are very small and hence we cannot detect all nodes of the clusters. However, if we include the pre/postcluster the quality for all cluster core definitions is high. In the middle the runtime is analyzed. The weak and semi core definitions run in nearly equal time, while the semi-strong core needs more calculations. The determination of the paths within the quotient graph is a more complex task. On the right, the number of subgraphs in the mining result is printed. As expected the weak core definition yields the fewest clusters, while the other two definitions split the weak component in several smaller ones. In the following experiments we focus on the weak core definition as the quality is high and the runtime is low.

Noise. In Figure 8 we increase the number of noise edges in the graph, i.e. we add edges that do not belong to the communities. The higher the value the more difficult is the identification of interesting subgraphs. The quality of GDens (core) is not affected by adding noise; the dense areas are still identified. The high quality of GDens (all) decreases slightly because the pre/postcluster include more and more nodes that do not belong to the cluster. However, even for very high percentages of noise GDens shows in both variants high quality. The quality of the Voltage algorithm remains unchanged with very low quality results. Edge betw. reaches a very high quality for zero percentage of noise but then it rapidly decreases. Both algorithms cannot identify the true hidden clusters.

Number of nodes. In Figure 9 (left) we plot the runtime of the algorithms with respect to the number of nodes. Our GDens algorithm is far more efficient than the other approaches. The Edge betw. method did not even finish within 12 hours for a dataset with 12500 nodes. Additionally in Figure 9 (right) we analyze the quality. While the quality of GDens in both variants stays on a high level or even increases a bit, the quality of Voltage decreases. In large graphs this algorithm cannot identify good patterns. Edge betw. has always low quality.

Hidden clusters. In the next experiment we analyze the effects if the number of hidden clusters in the graph is altered. Due to the high runtime of Edge betw. the number of nodes is set to 2000. As depicted in Figure 10 (left), with increasing number of clusters the quality of GDens is not or only less affected. With increasing number of clusters and fixed number of nodes, the interesting subgraphs get smaller and hence their identification is harder. The Edge betw.

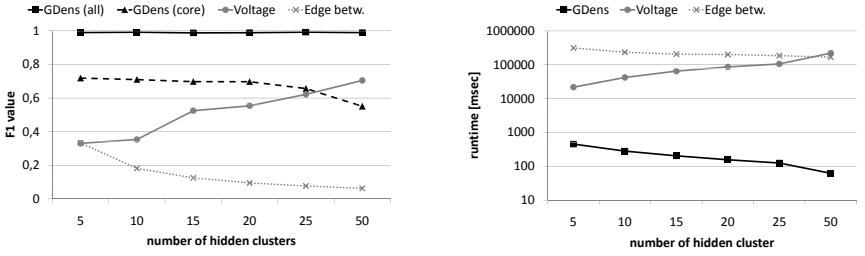


Fig. 10. Quality and runtime w.r.t. number of hidden cluster

algorithm shows a similar decrease but on a much lower quality level. Interestingly the quality of Voltage increases. The algorithm is better in the detection of small clusters. However, the quality of GDens (all) is never reached by this algorithm. Considering the runtime of the algorithms in Figure 10 (right), we see that the quality increase of Voltage is paid with a high and also increasing runtime. The runtime of GDens is orders of magnitudes lower. Furthermore, with more and hence smaller subgraphs the runtime even decreases.

Summarized, our GDens outperforms the competing algorithms in quality as well as in runtime. It is able to detect the interesting subgraphs also in noisy settings and automatically identifies the number of clusters.

Parameter variation. In our next experiment in Figure 11 we vary the parameter τ. On the right y-axis we indicate the number of identified clusters. First, the number increases because the hidden clusters are split-up in several smaller ones due to a higher τ. Afterwards, the number decreases to zero because no more core objects are identified. Correspondingly, the quality of the clustering (left y-axis) based on the core objects decreases. Considering GDens (all) the quality drops but at a later step. The objects that are included in the hidden clusters but not identified by the cores are now contained in the pre/postclusters.

DBLP data. To analyze real world data, we use the DBLP data set where nodes represent authors and edges/edge-weights the number of co-written papers. We generate a graph based on all publications from 2004-2008 and we extracted the largest connected component with 228k nodes and 1458k edges. In Figure 12 we

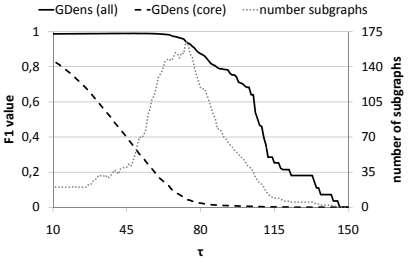


Fig. 11. Results for different τ values

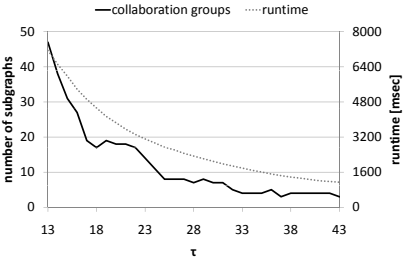


Fig. 12. Results on DBLP data

	GDens (core)			GDens (all)		
	weak	semi	strong	weak	semi	strong
weighted graph	0.72	0.68	0.68	0.99	0.98	0.98
unweighted graph	0.36	0.26	0.23	0.96	0.96	0.96

Fig. 13. Clustering quality (F1 value) for weighted vs. unweighted graphs

present some results with respect to a varying τ value. The F1 value cannot be determined for this data set because the true clustering structure is not known. Instead we plot the number of identified cluster cores with more than 5 nodes, i.e. these clusters correspond to large collaboration groups. Additionally, the runtime of the algorithm is presented. Both measures decrease continuously, i.e. we identify less collaboration groups but increase the efficiency. We want to point out the high efficiency of our GDens algorithm also on this large DBLP data set.

Unweighted graphs. In the next experiment we want to focus on the advantage of our algorithm to handle weighted graphs. In Figure 13 we show the difference in clustering quality if instead of the weighted graph an unweighted one is used, i.e. we ignore the weights by setting these to a constant value. As one can see in all cases the quality of the unweighted clustering is smaller. Especially if we only consider the core of the cluster (middle column) the quality decreases. This experiment supports the need for interesting subgraph mining algorithms that incorporate the weights of edges as our model does.

4 Conclusion

We introduce a novel technique to identify interesting subgraphs using the method of influence functions for calculating the densities of nodes. Our model can handle directed and weighted graphs and we show in experiments that using this information increases the quality of the clustering result. We present three types of dense subgraphs that account for the direction of edges in the graph. Our GDens algorithm identifies the number of clusters automatically and it is robust with respect to noise. In experiments we demonstrate the high quality and low runtime of our GDens algorithm compared to other subgraph mining methods.

Acknowledgment

This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

1. Feige, U., Peleg, D., Kortsarz, G.: The dense k-subgraph problem. Algorithmica 29(3), 410–421 (2001)
2. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 598–612. Springer, Heidelberg (2002)

3. Liu, G., Wong, L.: Effective pruning techniques for mining quasi-cliques. In: ECML/PKDD, vol. (2), pp. 33–49 (2008)
4. Flake, G.W., Lawrence, S., Giles, C.L.: Efficient identification of web communities. In: KDD, pp. 150–160 (2000)
5. Long, B., Wu, X., Zhang, Z.M., Yu, P.S.: Unsupervised learning on k-partite graphs. In: KDD, pp. 317–326 (2006)
6. Wu, F., Huberman, B.A.: Finding communities in linear time: A physics approach. CoRR cond-mat/0310600 (2003)
7. Ruan, J., Zhang, W.: An efficient spectral algorithm for network community discovery and its applications to biological and social networks. In: ICDM, pp. 643–648 (2007)
8. Long, B., Zhang, Z.M., Yu, P.S.: A probabilistic framework for relational clustering. In: KDD, pp. 470–479 (2007)
9. Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.J.: Scan: a structural clustering algorithm for networks. In: KDD, pp. 824–833 (2007)
10. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE TPAMI 22(8), 888–905 (2000)
11. Meila, M., Pentney, W.: Clustering by weighted cuts in directed graphs. In: SDM (2007)
12. Ding, C.H.Q., He, X., Zha, H., Gu, M., Simon, H.D.: A min-max cut algorithm for graph partitioning and data clustering. In: ICDM, pp. 107–114 (2001)
13. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA 99, 8271–8276 (2002)
14. Gregory, S.: An algorithm to find overlapping community structure in networks. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) PKDD 2007. LNCS (LNAI), vol. 4702, pp. 91–102. Springer, Heidelberg (2007)
15. Gregory, S.: A fast algorithm to find overlapping communities in networks. In: ECML/PKDD, vol. (1), pp. 408–423 (2008)
16. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Phys. Rev. E 70, 66111 (2004)
17. Newman, M.E.J.: Modularity and community structure in networks. PNAS USA 103, 8577–8582 (2006)
18. Chen, J., Zaïane, O.R., Goebel, R.: Detecting communities in social networks using max-min modularity. In: SDM, pp. 978–989 (2009)
19. Silverman, B.W.: Density Estimation for Statistics and Data Analysis. Chapman and Hall, London (1986)
20. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: KDD, pp. 226–231 (1996)
21. Kailing, K., Kriegel, H.P., Kröger, P.: Density-connected subspace clustering for high-dimensional data. In: SDM, pp. 246–257 (2004)
22. Godsil, C., Royle, G.: Algebraic Graph Theory. Springer, Heidelberg (2001)
23. Müller, E., Günnemann, S., Assent, I., Seidl, T.: Evaluating clustering in subspace projections of high dimensional data. PVLDB 2(1), 1270–1281 (2009)
24. Van Rijsbergen, C.J.: Information Retrieval. Butterworths, London (1979)