The Community-search Problem and How to Plan a Successful Cocktail Party

Mauro Sozio[*]
Max-Planck-Institut für Informatik
Saarbrücken, Germany
msozio@mpi-inf.mpg.de

Aristides Gionis
Yahoo! Research
Barcelona, Spain
gionis@yahoo-inc.com

ABSTRACT

A lot of research in graph mining has been devoted in the discovery of communities. Most of the work has focused in the scenario where communities need to be discovered with only reference to the input graph. However, for many interesting applications one is interested in finding the community formed by a given set of nodes. In this paper we study a query-dependent variant of the community-detection problem, which we call the *community-search problem*: given a graph G, and a set of *query nodes* in the graph, we seek to find a subgraph of G that contains the query nodes and it is densely connected.

We motivate a measure of density based on minimum degree and distance constraints, and we develop an *optimum greedy* algorithm for this measure. We proceed by characterizing a class of *monotone* constraints and we generalize our algorithm to compute optimum solutions satisfying any set of monotone constraints. Finally we modify the greedy algorithm and we present two heuristic algorithms that find communities of size no greater than a specified upper bound. Our experimental evaluation on real datasets demonstrates the efficiency of the proposed algorithms and the quality of the solutions we obtain.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms

Algorithms, Experimentation, Theory

Keywords

graph mining, community detection, social networks, graph algorithms

1. INTRODUCTION

Graphs is one of most ubiquitous data representations, and they find applications in a wide range of areas including biology, physics, social sciences, and information technology. With the increasing availability of very large networks, there is need for designing algorithmic data-analysis tools and for developing applications that exploit the latent structure in the data.

Discovering communities in graphs and social networks has drawn a large amount of attention in recent years [9, 13, 15, 16, 25]. It has been one of them most well-studied problems of graph mining. Most of the work has focused in the scenario where communities need to be discovered in an a-priori manner, with only reference to the input graph. However, in many application scenarios we are interested in discovering the community defined by a given set of nodes. For instance, if `Bob` and `Alice` take the same tango class, and `Charles` is `Bob`'s boss, the community formed by `Bob` and `Alice` is very different than the community formed by `Bob` and `Charles`.

In this paper we study a query-dependent variant of the community detection problem, which we call the *community-search problem*. Our problem formulation takes as input a graph G, and a set of *query nodes*, and the task is to find a densely connected subgraph of G that contains the query nodes. The problem we study has many interesting applications in areas such as social-network analysis, collaborative tagging systems, query-log analysis, biology, and others, Some motivating examples are the following.

Finding a group or organizing an event: A number of scientists obtain funding to organize a workshop. They figure out that the chances of success of the workshop will be higher if they invite a set of colleagues so that each scientist is well acquainted, and perhaps have even worked together, with a number of other participants.

Tag suggestion: In a social-media environment, a *tag graph* relates similar tags. For instance, in a photo-sharing portal two tags are related if they co-occur frequently in a given set of photos. Given a new photo being uploaded, and a number of initial tags that the user provides for this photo, the system can suggest to the user a number of additional tags. A good set of suggestions is a set of tags related to each other and related to the original tags.

Biology: A biologist has identified a number of proteins that regulate a gene of interest, and she would like to study further a candidate list of other proteins that are likely to participate in the regulation process. Such a candidate set

[*] Part of this work was done while the author was visiting Yahoo! Research, Barcelona.

KDD'10, July 25–28, 2010, Washington, DC, USA.
Copyright 2010 ACM 978-1-4503-0055-1/10/07 ...$10.00.

can be obtained by finding a dense subgraph in the protein-interaction network that contains the original proteins.

A similar problem has been studied by other researchers in data mining, and our paper has been largely inspired by that line of research. For instance, Faloutsos et al. [10], Tong et al. [28], as well as other researchers have studied the problem of finding a subgraph that *connects* a set of query nodes in a graph. The main difference of our approach is that we are not just interested in connecting the query nodes, but also in finding a meaningful community of query nodes. Additionally, we follow a graph-theoretic approach: we first formulate an objective based on the density of the subgraph, and then we devise combinatorial algorithms that optimize this objective. Using such an objective function allows a rigorous approach and it makes applicable a number of techniques and results that have been developed by the community of theoretical computer science.

In order to find densely connected communities that contain the query nodes, one needs to define an appropriate measures of density. Such measures can be the *average* or the *minimum* degree of the nodes in the extracted community. As we discuss in the following section, the average-degree measure has the drawback of being sensitive to *"free-riders"*, namely, irrelevant but dense subgraphs that may be attached to the query nodes and yield unintuitive solutions. For this reason, we focus on the latter measure, the minimum degree. We also give the possibility to exclude nodes that are far from the query nodes, as usually these nodes are less related to the query nodes than those that are nearby. Our goal in this paper, is to find compact communities, containing the query nodes, and whose minimum degree is maximized.

For this problem, we are able to develop a greedy algorithm that computes the optimum solution. An interesting property of the minimum-degree measure and the distance constraint is that they are *monotone*; a formal definition of monotonicity is deferred to Section 4. We then show that the greedy algorithm can be generalized to give the optimal solution for (*i*) any monotone function, and (*ii*) satisfying any number of monotone constraints. We discuss that many natural constraints are monotone.

On the negative side, our greedy algorithm does not work if one requires that the size of extracted communities satisfies an upper bound constraint. In fact, we show that the community-search problem with an upper bound on the size of the extracted community is an NP-hard problem. This is an unfortunate situation since the size requirement is a natural constraint: e.g., for the scientific workshop there is space for inviting at most 100 participants; the biologist has recourses to study at most 50 proteins, and so on.

To handle the size constraint, we modify the greedy algorithm and we present two heuristic algorithms that find communities of size no greater than a specified upper bound. The heuristics are inspired by the optimality of monotone constraints, and attempt to satisfy the size requirement by setting alternative monotone constraints that affect the size of the solution.

Roadmap: The paper is organized as follows. In the next section we review the related work. In Section 3 we define formally the problem of community search, and in Section 4 we present the greedy optimal algorithm, as well as its generalization to monotone constraints and monotone functions. In Section 5 we discuss our heuristic algorithms for finding solutions with upper bound on the community size. Section 6 presents our experimental results, while Section 7 concludes the paper and presents some ideas for future work.

2. RELATED WORK

Our paper is partly inspired by data-mining work on finding communities and connecting subgraphs, and partly by work in theoretical computer science that seek to find dense subgraphs. We review briefly both lines of work below, but given the large amount of literature and our space limitation we cannot hope to be comprehensive.

Connectivity subgraphs. Faloutsos et al. [10] and Tong et al. [28] address the problem of finding a subgraph that connects a set of query nodes in a graph. The focus of those papers is to develop measures of proximity between nodes of the graph that depend on the global graph structure. Such measures are based on *electrical-current flows* [10] and the related notion of *random walks* [28]. Then the task is to extract subgraphs that on one hand are near to the query nodes, according to the developed similarity measure, and on the other hand connect the query nodes. In subsequent work, Koren et al [21] refined the proximity measures using the notion of *cycle-free effective conductance* (CFEC) and proposed a brunch and bound algorithm in order to find a subgraph that optimizes the CFEC measure. More recently, Asur and Parthasarathy [5] suggested the concept of *viewpoint neighborhood* analysis in order to identify neighbors of interest to a particular source in a dynamically evolving network, showing the connection of their measure with *heat diffusion*. Cheng et al [8] address the problem of connecting query nodes in a context-aware framework, where they first partition the graph using the modularity measure, and then they study connectivity at *intra-community* and *inter-community* levels. Finally, Kasneci et al. [19] use a *random walk*-based approach in order to extracting informative subgraphs with respect to query nodes in entity-relationship diagrams.

The main difference of our approach with the above line of research is two-fold. First, we are more interested in extracting the best community that the query nodes define, and not only finding a set of nodes that *connect* the query nodes. Second, we follow an approach that comes closer to theoretical computer, and formulate the problem of community search as a graph theory problem. This allows a more rigorous approach, for example, reasoning about the hardness of the problem, as well as borrowing from the techniques and results that have been developed by the community of theoretical computer science.

Community detection. There is a very large body of work on finding communities in large graphs, social networks, and other kinds of networks. A typical approach for finding communities is by means of optimizing the *modularity* measure [16, 26]. There are many papers studying the modularity measure and developing algorithms for it, for instance, Brandes et al. [6] showed that it is NP-hard to optimize modularity, Fortunato and Barthelemy [14] identify the resolution-limit problem, White and Smyth [29] follow a spectral approach to optimize modularity, Agarwal and Kempe [1] develop a mathematical-programing algorithm, and many more algorithms. Other approaches to community detection include algorithms based on max-flows [12, 13], finding dense subgraphs [9, 15], as well as graph-partitioning algorithms based on spectral methods, such as the METIS algorithm [18], and random walks [3]. All the above methods consider the *static* community detection problem, where the graph needs to be partitioned a-priori with no reference to query nodes.

Densest subgraph problem. The problem of finding the densest subgraph of a given graph has also received a lot of attention in the community of theoretical computer science. A common objective to optimize is the density of the subgraph, which corresponds to two times the average node degree. Without restriction on the subgraph size, the problem can be solved optimally in polynomial time. Charikar [7] showed that the greedy algorithm that we consider in this paper can be used to find a factor-2 approximation. The same simple greedy algorithm was previously studied by Asahiro et al. [4]. When adding size constraints, and asking to find a subgraph of size exactly k the problem becomes NP-hard, while the best algorithm for this problem has an approximation guarantee of $O(n^{\alpha})$, where $\alpha < \frac{1}{3}$ [11]. Recently, Andersen and Chellapilla [2] and subsequently Khuller and Saha [20] studied the case with lower bound and upper bound constraints. In both cases the problem is NP-hard, and they offer approximation results for directed and undirected graphs.

Again the difference of our work with the above papers is that we seek to find dense subgraphs with reference to a set of query nodes. Additionally, we observe that in the case of query nodes, the measure of average degree may lead to non intuitive solutions. Thus we motivate and study using the measure of minimum degree as well as imposing distance constraints.

Team formation. Recently, Lappas et al. [24] studied the problem of team formation: given a social network where nodes are labeled with a set of skills that each node possesses, and given a task that requires a certain set of skills to be satisfied, we are required to find a subgraph in which all skills are present and the communication cost is small. As we will see in Section 4 the constraint to satisfy a set of skills is monotone, and therefore, it can be easily incorporated in our framework. Thus our algorithm can address a variant of the problem of Lappas et al. where the team needs to be formed with respect to a set of initial members, and where the goodness of the team can be measured with a monotone function (such as maximum distance and minimum degree).

3. PROBLEM DEFINITION

In this section we introduce our notation and we define more formally the problem of community search.

We are given an undirected graph $G = (V, E)$ in which the set of edges E represents binary relationships among the items V. Examples of graphs from the introduction are a collaboration network among scientists, a protein-interaction network, social networks, email networks, query graphs, etc. We denote by n the number of nodes and by m the number of edges in G.

When not mentioning explicitly we do not assume any weights and labels on the nodes and edges of the graph, however, many of the concepts we consider in this paper can be generalized to the case of weights and labels. For example, one might consider labels on the nodes indicating the expertise of people to form a team and accomplish a project, as was done by Lappas et al. [24], node weights to indicate importance of nodes, as using PageRank-type of scores, and edge weights to indicate strength of relation among nodes.

In addition to the graph $G = (V, E)$ we are also given as input a set of query nodes Q. These are the nodes that form the seed of the community that we are looking to extract. Given an induced subgraph H of G, consider a function f that measures the goodness of H as an extracted community. We want f to take large values if H is densely connected. We discuss different choices for f shortly after stating the first abstract version of our problem formulation:

Problem 1 (Generic objective function:) *Given an undirected (connected) graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, and a goodness function f, we seek to find an induced subgraph $H = (V_H, E_H)$ of G, such that*

(i) V_H contains Q ($Q \subseteq V_H$);

(ii) H is connected;

(iii) $f(H)$ is maximized among all feasible choices for H.

Notice that the the assumption that the input graph G is connected can be made without any loss of generality: the query nodes Q cannot belong in different connected components in G because then there is no feasible solution, so we can just consider only the connected component that the nodes of Q belong to.

With respect to the choice of the goodness function f, as we have already mentioned, we want to consider functions that capture the density of the subgraph H. One such measure is the number of edges $|E_H|$ in H, divided by all possible possible edges $\frac{|V_H|\,(|V_H|-1)}{2}$. However, even in its simplest form this density definition leads to NP-hard problems (via a simple reduction from MaxClique [17]), so we do not consider it in this paper.

Two other functions that measure the density of the subgraph H are (i) the average degree $f_a(H)$ of the nodes in H, and (ii) the minimum degree $f_m(H)$ of the nodes in H. The average degree in $H = (V_H, E_H)$ can also be expressed as $f_a(H) = \frac{2|E_H|}{|V_H|}$, and indeed the density function f_a has been studied extensively in the community of theoretical computer science [2, 7, 20].

However, in all studies that the measure f_a is considered, the problem is to find the densest subgraph with no reference to a set of query nodes Q. When we require that the extracted community H contains a given set of query nodes Q, the density measure f_a can lead to non intuitive results. This is demonstrated in Figure 1. In this example, the set of query nodes is $Q = \{x, y, z\}$, and an intuitive solution is the community of the nodes $C_1 = \{x, y, z, p, q\}$. However, notice that the average density f_a will increase if in the solution we also include the K_6 clique, that is, if we take $C_2 = \{x, y, z, p, q, r, a, b, c, d, e, f\}$. Intuitively, the nodes of the K_6 clique belong in a different community than the nodes of Q. This problem that is demonstrated in the somehow contrived example of Figure 1 is a serious one: one can easily find to add an unrelated but densely connected community to Q in order to increase the average density.

Thus, in this paper we will focus on the density measure $f_m(H)$ that it is defined to be the minimum degree of any node of V_H in the induced subgraph $H = (V_H, E_H)$. As any measure that seeks to maximize a minimum, f_m has the drawback that it is sensitive to outliers. However, it does

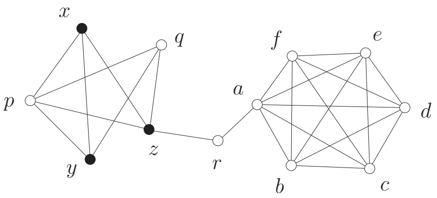


Figure 1: Different definitions of density. The set of query nodes is $Q = \{x, y, z\}$. Should we seek to maximize the *average degree* or the *minimum degree* of the extracted community?

not suffer from the problem of attaching a non-related community, as the measure f_a does. In the example of Figure 1, the optimal solution with respect to the measure f_m is the community $C_1 = \{x, y, z, p, q\}$.

Another way to avoid the pathological situations of attaching communities that are far way from query nodes, is to set a distance constraint, which can be defined as follows. First let $d_G(v, q)$ denote the length of the shortest path between nodes v and q in the graph G. If v and q are in different connected components, then we define $d_G(v, q)$ to be infinity. Now, given a node v in the graph G, we define the distance of v from the query nodes Q to be

$$D_Q(G, v) = \sum_{q \in Q} d_G(v, q)^2, \qquad (1)$$

and we also define

$$D_Q(G) = \max_{v \in V(G)} \{D_Q(G, v)\}, \qquad (2)$$

the distance of the furthest node from the query nodes. For defining $D_Q(G, v)$ other alternatives are possible, for instance, not using squares, or using max instead of $\sum$.

The more concrete problem that we focus in the next section is the following:

Problem 2 *Given an undirected (connected) graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, and a number d to be interpreted as a distance constraint, we seek to find an induced subgraph $H = (V_H, E_H)$ of G, such that*

(i) V_H contains Q ($Q \subseteq V_H$);

(ii) H is connected;

(iii) $D_Q(H) \leq d$; and

(iv) the minimum degree function $f_m(H)$ is maximized among all feasible choices for H.

4. COMMUNITIES WITHOUT SIZE RESTRICTION

In this section we present a family of algorithms for the community-search problem. We start from the specific problem instance of maximizing the minimum degree of the nodes that belong in the community. For the moment we ignore the distance constraint, for instance, in the definition of Problem 2 we may set $d = |V|^3$. Guided by the objective function of minimum degree, we then provide a definition of monotonicity, and we generalize our algorithm for any monotone function and monotone constraint, including the distance constraint.

Throughout this section we assume that there is no restriction on the size of resulting community. The case of communities with size restriction is addressed in the next section.

4.1 Maximizing the minimum degree

We start by present an optimal algorithm for Problem 2. Our algorithm, called GREEDY, is a variant of the greedy algorithm that was proposed and studied by Asahiro et al. [4] and later analyzed by Charikar [7], who showed that it achieves a factor 2 approximation guarantee for the densest-subgraph problem.

The details of the algorithm GREEDY are the following:

- We start by setting $G_0 = G$, the input graph, and we proceed by deleting one node in each step.
- At the t-th step, we consider a node u that has minimum degree in G_{t-1}.
- Consequently, at the the t-th step, we obtain graph G_t by deleting the node u and all the edges incident to u from G_{t-1}.
- The algorithm terminates at the t-th step if either (i) at least one of the query nodes Q has minimum degree in the graph G_{T-1}, or (ii) the query nodes Q are no longer connected.

If G_t is the graph during the t-th step of the algorithm, we denote by G'_t the connected component of G_t that contains all query nodes Q. As defined before, $f_m(G'_t)$ is the minimum degree of the nodes in G'_t. GREEDY returns as a solution G_O the graph G'_t for which $f_m(G'_t)$ is maximized during all steps of its execution. That is, $G_O = \arg\max\{f_m(G'_t) \mid t = 1, \ldots, T-1\}$.

As observed by Charikar [7], GREEDY can be implemented in linear time. The idea is to make a list of nodes with degree d, for $d = 1, \ldots, n$, that is, one separate list for each degree. When removing a node u from G_t, a neighbor of u with degree d needs to be moved from the list d to the list $d-1$. So the total amount of moves is $O(m)$. In addition, since the minimum degree in each step decreases by at most 1, given the node with minimum degree in one step, we can locate a node with minimum degree for the next step in $O(1)$ time. Overall the running time is $O(n + m)$.

Remarkably, GREEDY gives an optimal solution for Problem 2. For the following, we still assume that $d = |V|$, and the case of using a distance constraint is addressed by the generalization of the algorithm in the next section.

Theorem 1 *Let G be a graph and Q be a set of query nodes. The algorithm* GREEDY *returns an optimum solution for the problem of finding the community that contains Q and maximizes the minimum degree of all nodes in the community (Problem 2).*

PROOF. Let G_O be an optimum solution for the problem. Consider the step t of GREEDY, when the first node v of G_O is selected. Let G_t be the graph at step t. From the fact that v is the first node in the optimum that has been chosen, it follows that G_O is a subgraph of G_t. This in turn implies that there must be a connected component G'_t in G_t which is a supergraph of G_O, for otherwise, G_O would not be connected. From this fact it follows that, $\delta_{G'_t}(v) \geq \delta_{G_O}(v)$. Since v has minimum degree in G_t, it follows that for every node u in G'_t it holds:

$$\delta_{G'_t}(u) \geq \delta_{G'_t}(v) \geq \delta_{G_O}(v),$$

which implies that G'_t has optimum minimum degree. □

4.2 Generalization to monotone functions

In this section we generalize the results that we developed in the previous section. We first provide a characterization of *monotone* functions, so that the minimum degree function f_m is a member of this family of functions. Consequently we show that the algorithm GREEDY can be used to find an optimal solution for any monotone function.

Definition 1 (Monotone function) *Let V be an underlying set of nodes, and let $\mathcal{G}_V$ be the collection of all possible graphs defined over subsets of V. Let f be a function that assigns a score value to any graph in $\mathcal{G}_V$, that is, $f : \mathcal{G}_V \to \mathbb{R}$. We say that the function f is* monotone non-increasing *if for every graph G and for every induced subgraph H of G, $f(H) \leq f(G)$. We similarly define f to be* monotone non-decreasing *by requiring $f(H) \geq f(G)$.*

We extend the notion of monotone functions to be defined with respect to a specific node v in V.

Definition 2 (Node-monotone function) *Let $\mathcal{G}_V$ be, as before, the collection of all possible graphs defined over subsets of V. A function $f : \mathcal{G}_V \times V \to \mathbb{R}$ is said to be* node-monotone non-increasing *if for every graph G, for every induced subgraph H of G, and every node v in H, $f(H, v) \leq f(G, v)$. We similarly define* node-monotone non-decreasing *functions.*

An interesting special case is when a function is *boolean*. We refer to such boolean functions as *properties*. We say that the graph G *satisfies* property f if $f(G) = 1$, while it *violates* property f if $f(G) = 0$. Similarly, for node-monotone properties, given a graph G, we say that a node v satisfies the property f in G if $f(G, v) = 1$, while v violates the property f in G if $f(G, v) = 0$,

With respect to the community-search problem, and the minimum degree function that we used in the previous section we have:

Example 1 (Degree) *Let $d(G, v)$ be the degree of node v in graph G. The function d is node-monotone.*

Example 2 (Minimum degree) *Let $f_m(G)$ be the minimum degree of any node in G. The function f_m is monotone.*

Example 3 (Distance) *The functions $D_Q(G, v)$ and $D_Q(G)$, defined by Equations (1) and (2) are node-monotone and monotone, respectively.*

In addition to connectivity and degree there are many interesting monotone and node-monotone functions. In some of the examples below we also consider a set of nodes Q, which can be thought of as the set of query nodes Q for the community-search problem. Also with $d_G(v, q)$ we denote the length of the shortest path distance between nodes v and q in graph G.

- A lower bound on the number of nodes in a graph is monotone.
- Requiring that the set Q must belong to the graph G and must be connected is monotone.
- Other distance functions are monotone. For instance, the function $M_Q(G, v) = \max_{q \in Q}\{d_G(v, q)\}$ is node-monotone, and $M_Q(G) = \max_{v \in V(G)}\{M_Q(G, v)\}$ is monotone.
- The constraint along the lines of Lappas et al [24], where a set of skills specified in input should be present in the extracted community, is monotone.
- The number of disjoint paths between two nodes (which is a popular measure for friendship strength) is node-monotone non-increasing.

Before defining the general problem, we need one more definition for comparing node-functions in different graphs.

Definition 3 *Let $f : \mathcal{G}_V \times V \to \mathbb{R}$ be a function. We say that G maximizes the function f if $\min_{v \in V(G)}\{f(G, v)\} \geq \min_{v \in V(H)}\{f(H, v)\}$ for all H in the collection $\mathcal{G}_V$. The definition for minimizing f is symmetric.*

Now are are ready to define the generalization of the community-search problem.

Problem 3 (Cocktail party) *We are given an undirected graph $G = (V, E)$, a node-monotone non-increasing function $f : \mathcal{G}_V \times V \to \mathbb{R}$, as well as a set of monotone non-increasing properties $f_1, \ldots, f_k$. We seek to find an induced subgraph H of G that maximizes f among all induced subgraphs of G, and satisfies $f_1, \ldots, f_k$. A similar problem can be defined by considering to minimize monotone non-decreasing functions.*

For solving the Problem 3, we generalize the algorithm GREEDY. We call the generalized version GREEDYGEN algorithm. In detail, the GREEDYGEN algorithm is described as follows:

We start from $G_0 = G$ and at each step, we check whether there is a property f_j and a node $v \in V$ such that v violates f_j, $j = 1, \ldots, k$. In such a case we delete v and all the incident edges of v. Otherwise we delete from G a node v such that $f(G, v)$ is minimum. This procedure is iterated until all nodes are deleted. Let G_t be the graph constructed at step t. GREEDYGEN returns as solution the graph H which maximizes f, among all graphs G_t that are constructed throughout the execution of the algorithm and satisfy all the monotone properties.

Theorem 2 *The algorithm GREEDYGEN returns always an optimum solution for Problem 3.*

We omit the proof of Theorem 2 for brevity.

Running time. Note that, as opposed to the algorithm GREEDY, whose execution time is linear, GREEDYGEN can be more expensive. The reason is that after removing each node, it needs to re-evaluate which constraints $f_1, \ldots, f_k$ are violated. The exact running time of the algorithm depends on the constraints employed.

5. COMMUNITIES WITH SIZE RESTRICTION

The drawback of the algorithms proposed in the previous section is that they might return subgraphs with very large size. Unfortunately, if we require that the size of the output subgraph is less than a pre-specified upper bound, the problem becomes NP-hard. This is shown in Theorem 3.

Therefore, we resort to two simple yet effective heuristics inspired by the Greedy algorithm. These heuristics are presented in Section 5.2, while their empirical evaluation is discussed in Section 6.

5.1 Complexity

Formally, the problem of maximizing the minimum degree while having an upper bound on the size of the graph is defined as follows.

Problem 4 *(Minimum degree with upper bound on the size). Given an undirected (connected) graph $G = (V_G, E_G)$, a set of query nodes $Q \subseteq V$, a number d (distance constraint), and an integer k (size constraint), we wish to find an induced subgraph $H = (V_H, E_H)$ of G, such that:*

(i) H contains the query nodes ($Q \subseteq H$);

(ii) H is connected;

(iii) $D_Q(H) \leq d$;

(iv) $|V_H| \leq k$ (H has at most k nodes); and

(v) the minimum degree of H is maximized.

Theorem 3 *Problem 4 is* NP*-hard.*

Proof. We give a simple reduction to the Steiner-tree problem with unit weights. It is well known that the Steiner-tree problem is NP-Hard as shown by Karp (1972) (for a proof see [22, Theorem 20.2]). In the decision version of the Steiner-tree problem, we are given a graph $G = (V, E)$, a set T of nodes ($T \subseteq V$), and an integer k. We are asked to find a subtree of G (a connected subgraph of G with no cycles) containing all nodes in T and having at most k edges.

Given an instance of the Steiner-tree problem, with G being the graph in input, T the set of nodes we are to connect, and k an upper bound on the number of edges, we define an instance of the decision version of our problem as follows. The graph in input is G, the query nodes are the nodes in T, the upper-bound on the number of nodes is $k + 1$, the distance constraint is set to infinity ($d = |V|^3$ suffices), and we are to find a graph with minimum degree at least 1. We show that there is a solution for the Steiner-tree problem if and only if there is a solution for our problem.

First, any Steiner-tree using at most k edges is also a solution for our problem using at most $k + 1$ nodes. On the other hand, given a solution H for our problem containing at most $k + 1$ nodes, we can compute a Steiner tree with at most k edges: simply take any spanning tree of H. □

Note that from the proof it follows that the problem is NP-hard even without the distance constraint, that is, even without the requirement (*iii*) of Problem 4.

5.2 Algorithms

Summarizing our discussion so far, we have seen that the Greedy algorithm cannot be used to find communities that have size within a pre-specified upper bound. And indeed, we saw in the previous section that finding a community with bounded size is an NP-hard problem.

On the other hand, the problem of finding bounded-size communities is very interesting from the application point of view. So, in this section we describe two heuristics that can be used to find communities with bounded size. Our heuristics are inspired by the Greedy algorithm for maximizing the minimum degree. The design principle of the first heuristic is the simple observation that a tighter distance constraint implies smaller communities. In other words, the size of the output graph is a *monotonically decreasing* function of the distance bound d. This monotonicity property can be proved by a simple argument, however, the proof is omitted for brevity.

As one may expect, the solutions provided by our heuristics do not have any provable quality guarantee. Thus, in Section 6 we study the heuristics experimentally, and we show that in practice they give good solutions. Our two heuristics are called GreedyDist and GreedyFast. Both take as input the graph G, the set of query nodes Q, an upper bound k on the size of the output graph, and a distance bound d.

The two heuristics provide a quality–efficiency trade-off: GreedyDist is tries to optimize quality while GreedyFast tries to optimize efficiency.

GreedyDist: As we mentioned above, this heuristic is motivated by the monotone behavior of the size of the output graph with respect to the distance bound. GreedyDist uses the algorithm GreedyGen as a subroutine. GreedyDist starts by executing the algorithm GreedyGen in order to maximize the minimum degree subject to the distance constraint d that is specified in input. If the query nodes are connected and the size constraint is not satisfied in the output graph, then the algorithm GreedyGen is executed again with a tighter distance bound $d' < d$. GreedyDist iterates executing GreedyGen by decreasing at each step the distance constraint, until the size constraint is satisfied or the query nodes become disconnected. In the latter case, GreedyDist returns the smallest graph found among all executions of GreedyGen that is connected.

As we already discussed, GreedyDist provides no guarantee on the quality of the solution that it finds. The reason is that an optimal graph with size less than k may satisfy some distance constraint $d_0 \leq d$, and GreedyGen executed with distance bound d_0 provides no guarantee of finding this optimal graph. However, as we show in the experimental section, the algorithm GreedyDist finds accurate solution.

In order to specify fully the algorithm GreedyDist we need to answer one more question: how is the distance constraint decreased between successive calls of the GreedyGen algorithm? One simple solution is to decrease the distance constraint by exactly one unit at a time. However, note that since the size of the output graph is a monotone function to the distance bound one can perform a binary search on the distance bound. Such a binary search speeds up significantly the GreedyDist algorithm.

GreedyFast: Our second heuristic, GreedyFast, is also a variant of the Greedy algorithm. As the name suggests, we aim at devising a heuristic more efficient than GreedyDist, even if this might worsen the quality of the solution. To this end, there is a preprocessing phase where the input graph is restricted to the k' closest nodes to the query nodes; k' is defined to be minimum such that the resulting graph is connected and contains at least k nodes. The distance of a node to the query nodes is measured using the function D_Q defined in Equation (1).

After this preprocessing phase, we execute Greedy on the restricted graph formed in the preprocessing phase. Our intuition for employing this preprocessing phase, is that the closer a node is to the query nodes the more related the node is to the query nodes, and the more likely it is that the node belongs in their community.

Size and distance bounds: The algorithms GreedyDist and GreedyFast take as input the size upper bound k, and the distance bound d. Both of those bounds depend on the application for which the community-search problem is used. The size upper bound k expresses how many additional nodes we want to include in the community – and as we saw, if such a constraint exists, the problem cannot be solved optimally. For this reason, GreedyDist and GreedyFast might produce a graph whose size is larger than k. The dist ance bound is used for the case that we do not want to include in the community nodes that are far away, so that we obtain communities that are semantically more coherent. If one does not need to specify a distance bound, again the constraint can be ignored, and this leads to a much faster algorithm.

6. EXPERIMENTAL EVALUATION

In this section we present an empirical evaluation of the algorithms presented in the previous sections.

Datasets: We consider the following three datasets:

DBLP: We use a coauthorship graph extracted from a recent snapshot of the DBLP database. The dataset considers publications in all major computer-science journals. There is an undirected edge between two authors if they have coauthored a journal article. Considering the largest connected component of the whole dataset yields a graph of 226K nodes and 1.4M edges.

tag: We consider a tag graph extracted from the flickr[1] photo-sharing portal. The nodes in this graph represent tags that have been used for tagging photos. A tag is considered if it has used by at least 5 different users. An edge between two tags indicates that the two tags appear in at least 100 photos. Again a connected component is extracted, resulting in a graph of 38K nodes and 1.3M edges.

BIOMINE: We use a graph extracted from the database of the Biomine project [27]. The database represents a collection of biological interactions. Interactions are directed and labeled with probabilities. For extracting the BIOMINE graph we use in this paper, we ignore directions on the edges, and we consider one connected component. The resulting graph has 16K nodes, and 491K edges.

To understand better the performance of the proposed heuristics we compare them against a simple and natural baseline algorithm, which we call Baseline. As opposed to the two heuristics, Baseline is based on adding nodes, instead of removing.

Baseline: Assume first that the query nodes Q are connected, and let H_0 be the connectivity graph among Q. The Baseline proceeds in a sequence of steps as follows. At step t, the graph H_{t+1} is defined to be $H_t \cup \{v\}$, where v is the node among all nodes in $G \setminus H_t$ that has largest degree in $H_t \cup \{v\}$. If there is a tie on the degree, then Baseline breaks the tie by selecting the node whose distance D_Q to the query nodes is minimized, and if there is again a tie, a node is selected arbitrarily. The Baseline terminates when no further nodes can be added, and it returns the graph that maximizes the minimum degree among all graphs H_t's that were computed during its execution.

Table 1: Community size vs. distance bound d.

	6	9	11	12	14	17	20	27
BIOMINE	71	76	77	867	870	900	923	1394
DBLP	4	9	9	13	14.5	17	21	160
tag	35	248	248	3316	3554	8287	8305	14256

Of course we cannot assume that the query nodes Q are connected in the input graph G. For this reason, before entering the node-adding phase that we described above, we first compute a Steiner tree on the query nodes. Then we define H_0 to be the Steiner tree we computed. To compute a Steiner tree we use the classic algorithm by Kou et al. [23], which gives an approximation guarantee of $2 - 2/k$, where k is the number of query nodes.

Experimental framework: To evaluate our algorithms we use the three datasets, and we execute the algorithms and the baseline for randomly generated sets of query nodes Q. We study the characteristics of the algorithm as a function of the number of query nodes, which we denote be $q = |Q|$, as well as size bound k, and distance bound d. We also study the dependence of the algorithms on a parameter l that denotes the inter-distance between all query nodes. So, a value of $l = 2$ means that all query nodes are within distance 2 to each other.

For the extracted communities we measure and we report the minimum degree f_m, the average degree (or density) f_a, as well as the size and distance achieved. Each number shown in the graphs is an average over 50 repetitions. We implemented our algorithms in Perl and all experiments run on a dual-core Opteron processor at 3GHz.

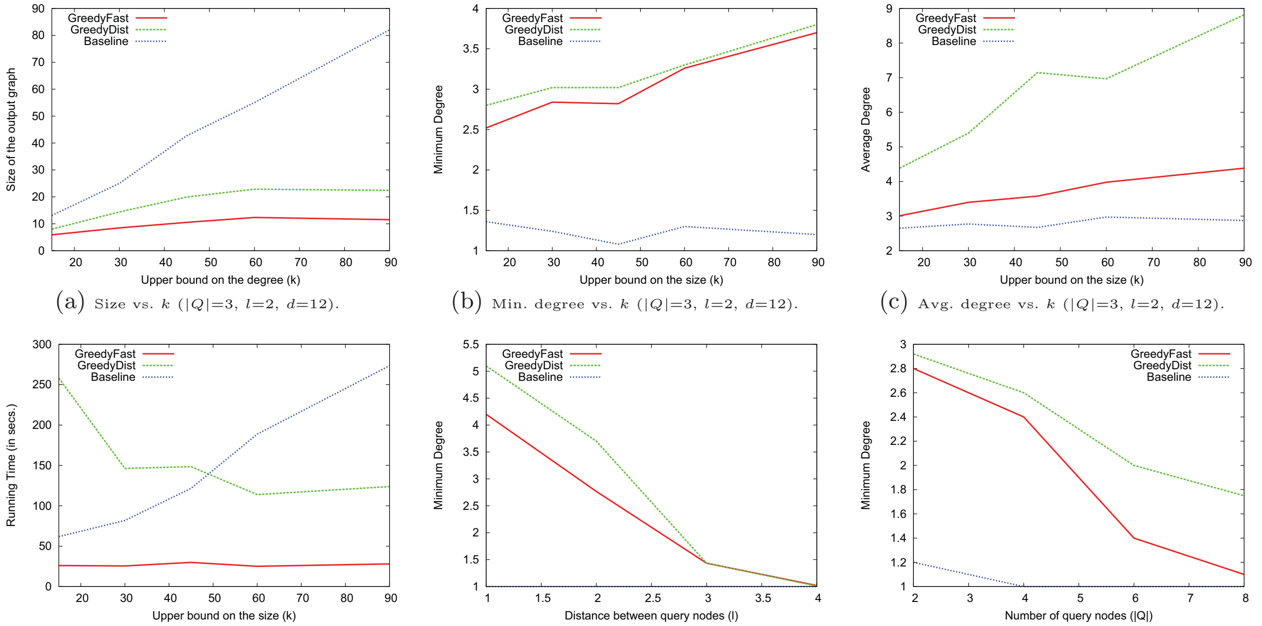
6.1 Quantitative results

We start our empirical evaluation by analyzing Greedy. We can prove that the size of the output graph yielded by Greedy is a monotone non-increasing function of the distance bound d. This monotonicity behavior served as inspiration for the the heuristic GreedyDist. Here we give empirical evidence of this fact aiming at gaining a deeper understanding of the behavior of this function. Our results are shown in Table 1. Each line of this table corresponds to one of the three datasets we consider, while each column corresponds to a possible value of the distance constraint specified in input (that is, an element of the codomain of the function $D_Q(G)$ defined in Section 3). We can see that in all three datasets the size of the output is increasing. The largest increase is observed for tag. This reason is that tag is the dataset with the largest average degree.

Then we move on evaluating our two heuristics, GreedyDist and GreedyFast, and comparing them with Baseline. Our results are shown in Figure 2 for the DBLP dataset, and in Figure 3 for the tag dataset. The behavior of the algorithms for the BIOMINE dataset is similar and we omit the results for space limitations.

We first measure the size of the output graph, the minimum degree, and the average degree, as a function of the size upper-bound k. The first observation is that Baseline

[1] www.flickr.com



(a) Size vs. k ($|Q|$=3, l=2, d=12). (b) Min. degree vs. k ($|Q|$=3, l=2, d=12). (c) Avg. degree vs. k ($|Q|$=3, l=2, d=12).

(d) Running time vs. k ($|Q|$=3, l=2, d=12). (e) Min. degree vs. l ($|Q|$=3, k=30, d=12). (f) Min. degree vs. $|Q|$ (k = 30, l=2, d=12).

Figure 2: Behavior of the three heuristics (GreedyDist, GreedyFast, and Baseline) for the DBLP dataset.

does not perform very well. Both the minimum and the average degree of Baseline are poor in comparison with our heuristics This is shown in Figures 2(b) and 2(c) for the DBLP dataset, and in Figures 3(b) and 3(c)) for the tag dataset. On the other hand, from Figures 2(a) and 3(a) we observe that the size of the graphs yielded by the two heuristics does not increase too much as we increase the upper-bound on the size k, yet, both the minimum and the average degree increases. This is a good property of our heuristics, as they keep the output graphs small, while still succeeding in increasing the objective that they try to optimize.

One may ask how good is a minimum degree of at least 3 and an average degree of 5-8, as those achieved by GreedyDist. We note that for those experiments, we used three query nodes, which are within distance 2 from each other. Thus, given that the query nodes are quite unrelated, and given that the input graph is very sparse, we think that it is quite remarkable that GreedyDist finds subgraph average distance more than 5.

With respect to comparing the two heuristics we can see from indeed GreedyDist delivers the most accurate results. However, GreedyFast still gives very good results and it is much faster, as shown in Figure 2(d) for DBLP and in Figure 3(d) for tag.

In Figures 2(e) and 3(e), respectively for the two datasets, we show the minimum degree as a function of the distance between the query nodes. We can see that for all three algorithms the minimum degree decreases as a function of the distance. The reason is that as the distance increases the query nodes become less related. Therefore, the existence of a dense graph connecting them becomes unlikely. A similar behavior can be observed in Figures 2(f) and 3(f) for increasing the number of query nodes. Since the query nodes are selected randomly, it becomes more difficult to connect a large number of nodes.

6.2 Case study

Evaluating the quality of the results of our algorithm in a way that it does not involve internal measures (such as densities, degrees, and graph distances) but the semantics of an application is a difficult and not entirely well-defined task. Thus a formal quality evaluation is beyond the scope of this paper. Instead we describe a case study that provides a feel of the results that one can obtain with our algorithm.

Christos Papadimitriou is one of the most prolific computer scientists. He has been recognized not only for the importance of his contributions, but also for the extremely wide range of areas that he has worked on. His interests include complexity theory, combinatorial optimization, algorithmic game theory, database theory, economics, biology, the internet, and more.

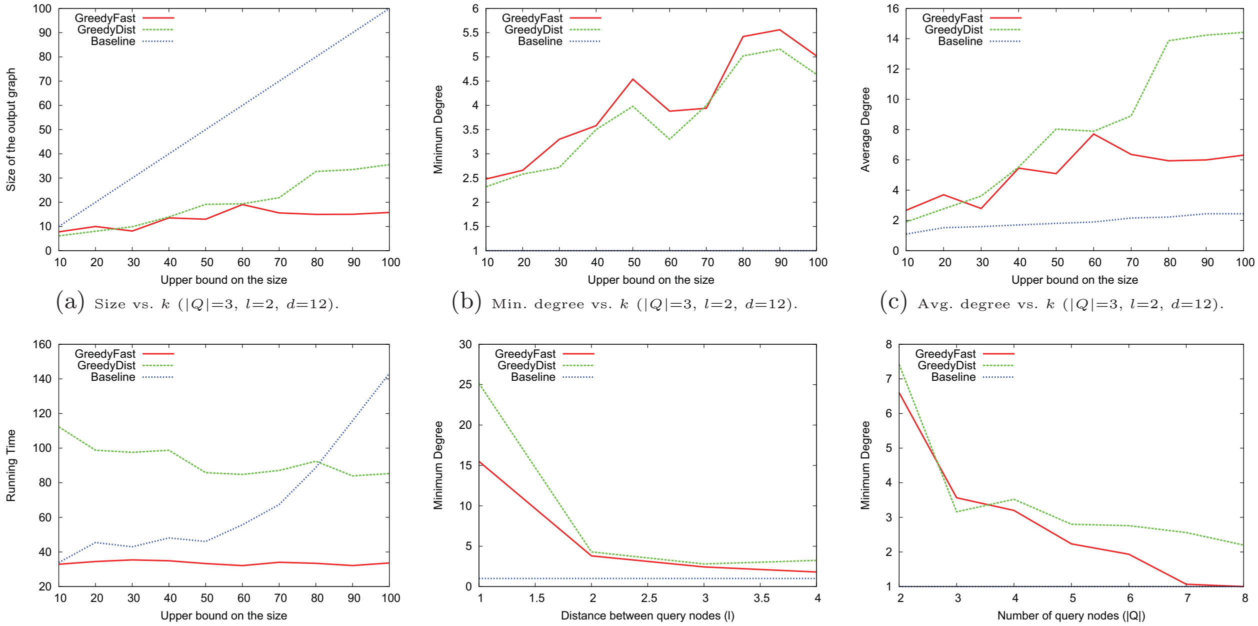
As a manifestation of his multiple research interests, in a network of scientific collaboration, Papadimitriou would be represented by a node that belongs in many overlapping communities. In our case study, we seek to find different communities formed by Papadimitriou and his coauthors.

We use the DBLP dataset, and we make four queries, each one including Papdimitriou and two other computer scientists. The queries are the following

$$Q_1 = \{\text{Papadimitriou, Abiteboul, Kanellakis}\};$$
$$Q_2 = \{\text{Papadimitriou, Widgerson, Fortnow}\};$$
$$Q_3 = \{\text{Papadimitriou, Yannakakis, Karp}\}; \text{ and}$$
$$Q_4 = \{\text{Papadimitriou, Raghavan, Kleinberg}\}.$$

We run our algorithm requesting communities of less than 20 nodes (in order to be able to visualize the results), and setting the sum-of-square-of-distances constraint equal to 10.

The resulting communities are shown in Figure 4. In all cases one can find the theme associated with each community. Query Q_1 corresponds to *database theory*, query Q_2



(a) Size vs. k ($|Q|$=3, l=2, d=12).

(b) Min. degree vs. k ($|Q|$=3, l=2, d=12).

(c) Avg. degree vs. k ($|Q|$=3, l=2, d=12).

(d) Running time vs. k ($|Q|$=3, l=2, d=12).

(e) Min. degree vs. l ($|Q|$=3, l=2, d=12).

(f) Min. degree vs. $|Q|$ ($|Q|$=3, l=2, d=12).

Figure 3: Behavior of the three heuristics (GreedyDist, GreedyFast, and Baseline) for the tag dataset.

Table 2: Statistics of the graphs of Figure 4, showing number of nodes $|C|$, min. degree f_m, and avg. degree f_a.

Community	$\lvert C\rvert$	f_m	f_a
Database theory	13	6	7.38
Complexity theory	12	4	4.81
Algorithms I	12	4	5.16
Algorithms II	14	7	7.84

corresponds to *complexity theory*, query Q_3 corresponds to *algorithms* for work mostly done in 70's and 80's, and query Q_4 corresponds to most recent work in *algorithms*.

Notice that for Q_2 there is only one edge between the three query nodes, while in the other three cases there is a triangle of collaboration. In all cases the resulting communities have high minimum and average degree, as shown in Table 2.

7. CONCLUSIONS

In this paper we studied the problem of finding a community in a graph given a set of query nodes. The aim is to find compact a community that contains the query nodes and it is densely connected. We motivate a measure of density based on minimum degree, and we also suggest pruning nodes based on their distance to the query nodes. For those requirements we study a simple greedy algorithm and we show that it yields the optimal solution. The algorithm generalizes nicely and yields optimal solution if we want to optimize any monotone function, and if we want to satisfy any number of monotone constraints.

For the variant of the problem with size constraint on the size of extracted community, the situation is not so agreeable. We show that the community-search problem with upper-bound constraint on the size of the extracted community is NP-hard – even though the case with lower-bound constraint is still polynomial, but this case is less useful in practice. Thus, to address the problem of community search with upper-bound constraint on the size, we propose two simple heuristics, inspired by the basic greedy algorithm, and we show experimentally that those heuristics outperformed a natural baseline.

For future work, we would like to apply our problem definition in practical applications, such as query suggestion and tag suggestion.

Acknowledgements: We are grateful to Hannu Toivonen for the BIOMINE dataset.

8. REFERENCES

[1] G. Agarwal and D. Kempe. Modularity-maximizing network communities via mathematical programming. *European Physics Journal B*, 66(3), 2008.

[2] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, 2009.

[3] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, 2006.

[4] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. In *SWAT*, 1996.

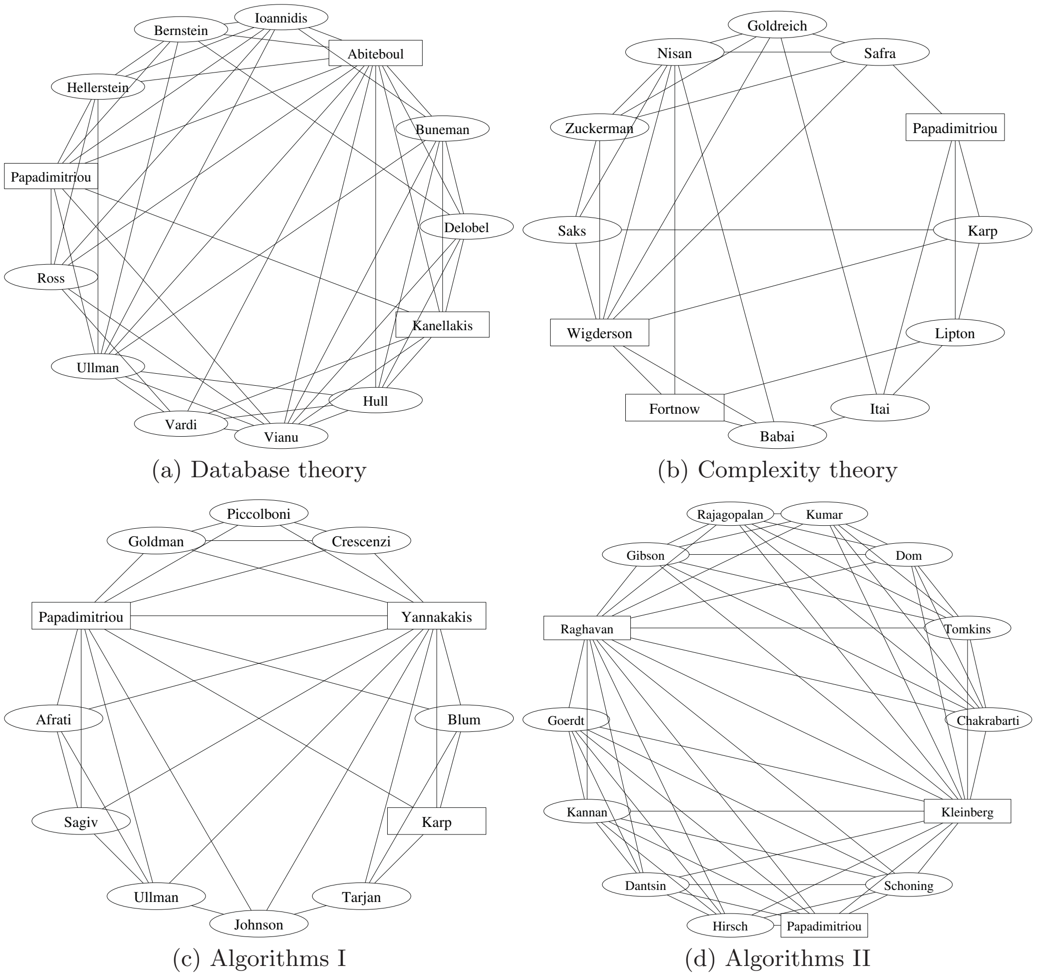
[5] S. Asur and S. Parthasarathy. A viewpoint-based approach for interaction graph analysis. In *KDD*, 2009.

[6] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *TKDE*, 20(2):172–188, 2008.

[7] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, 2000.

[8] J. Cheng, Y. Ke, W. Ng, and J. X. Yu. Context-aware object connection discovery in large graphs. In *ICDE*, 2009.

[9] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense communities in the web. In *WWW*, 2007.



(a) Database theory

(b) Complexity theory

(c) Algorithms I

(d) Algorithms II

Figure 4: Different communities of Christos Papadimitriou. Rectangular nodes indicate the query nodes, and elliptical nodes indicate nodes discover by our algorithm.

[10] C. Faloutsos, K. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, 2004.

[11] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29:2001, 1999.

[12] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *KDD*, 2000.

[13] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–71, 2002.

[14] S. Fortunato and M. Barthelemy. Resolution limit in community detection. *PNAS*, 104(1), 2007.

[15] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005.

[16] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the USA*, 99(12):7821–7826, 2002.

[17] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(38), 1997.

[18] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *JSC*, 20(1), 1998.

[19] G. Kasneci, S. Elbassuoni, and G. Weikum. Ming: mining informative entity relationship subgraphs. In *CIKM*, 2009.

[20] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, 2009.

[21] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity graphs in networks. *TKDD*, 1(3), 2007.

[22] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics)*. Springer, 2007.

[23] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15(2):141–145, 1981.

[24] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[25] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, 2008.

[26] M. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, 2003.

[27] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link discovery in graphs derived from biological databases. In *DILS*, 2006.

[28] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, 2006.

[29] S. White and P. Smyth. A spectral clustering approach to finding communities in graph. In *SDM*, 2005.