Community Discovery

Communities

Community. Let $S = \{s_1, \ldots, s_n\}$ be a set of entities of the same *type*. A **community** is a pair

$$C = \langle T, G \rangle,$$

where T, referred to as the **community theme** is the uniting property for the community, and $G \subseteq S$ is the set of **community members**.

Properties. This is a *very general definition* of **communities**. Different algorithms use different specializations of this definition.

- Themes. Themes define communities. That is, we expect that if for two communities $C_1 = \langle T_1, G_1 \rangle$ and $C_2 = \langle T_2, G_2 \rangle$, $T_1 = T_2$, then also $G_1 = G_2$ and therefore $C_1 = C_2$.

  (note that the inverse is not the case. It is quite possible for exactly the same set of members to form two different communities, although, in practice, distinguishing such communities may be hard.)

- Themes can be defined arbitrarily... Events, hobbies, concepts, etc...

- Individual entities may be members of multiple communities.

- In some applications, temporal aspect of communities is also important.

Community Discovery. Given a dataset containing (information about) entities, discover (hidden) communities of the entities. For each community discover the community theme and its members.

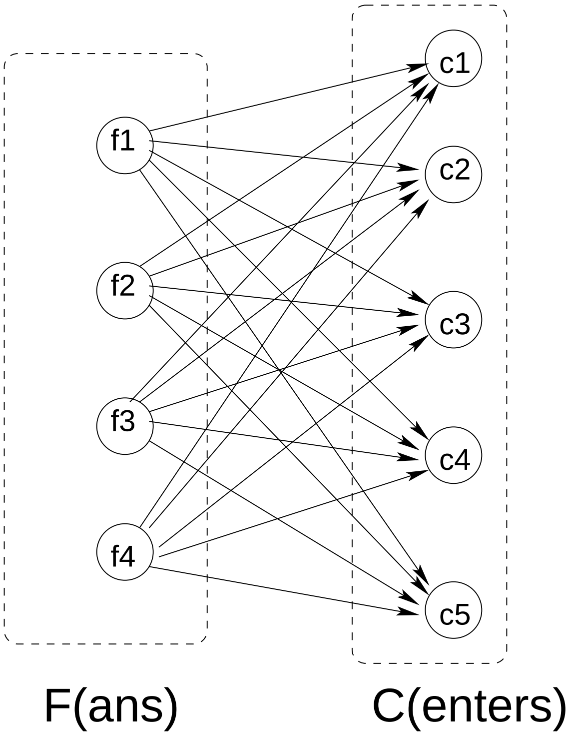

Figure 1: A (4,5) bipartite core.

Bipartite Core Communities

An (i,j) **bipartite core** is a **complete bipartite graph** $G = \langle F, C, E \rangle$, such that:

- $F, C \subseteq S$
- $F \cap C = \emptyset$,
- $E = \{(f,c) | f \in F, c \in C\}$,
- $(\forall f \in F, c \in C)(f,c) \in E$,
- $|F| = i$,
- $|C| = j$.

Elements of set F are referred to as fans, elements of set C are referred to as centers.

Figure 1 shows a (4,5) bipartite core.

Why? Bipartite cores represent a group of entities (fans) that co-cites the same set of entities (centers). One can view a bipartite core as a core of a community that cosists of the fans, and whose theme can be found among the centers.

Bipartite community discovery

Bipartite cores can be discovered using the following procedure:

Input: Graph $G_S = \langle S, E_S \rangle$, $S = \{s_1, \dots, s_n\}$, $E \subseteq S \times S$. i, j - size of the bipartite core.

Step 1. Pruning. Set S is pruned twice:

Step 1.1. Pruning by in-degree. Remove all pages with in-degree greater than some large constant K. (e.g., $K = 50$).

Step 1.2. Iterative pruning of fans and centers. $S_0 = S$. While $S_i \neq S_{i-1}$:

- Prune from S_{i-1} all nodes s with out-degree $d_o(s) < i$.
- Prune from S_{i-1} all nodes s with in-degree $d_i(s) < j$.

Step 2. Bipartite Core generation. For $k = 1 \dots k$ do:

- Recover (k, j) bipartite communities. $(1, j)$ community consists of a single node s with $d_o(s) = j$.

  In general case,

  - For each center of a $k - 1, j$ community, consider any node f' pointing to it, that is not part of the community. If f' is connected to all j centers in the community, it is added to the bipartite core.

Comments. Bipartite cores do not detect full communities, rather, they detect "central" portions of communities, and provide some direction for theme discovery.

Maximum Flow Communities

Maximum Flow Communities. Let $G_S = \langle S, E_S \rangle$ be an interaction graph over a set of entities S. A **maximum flow community** $C \subset S$ is a collection of entities such that

- For each $u \in C$, u has more edges (both in- and out-) connecting it to other members of C than to members of $S - C$.

Idea. Recall the **Maximum Flow** problem on the graphs (See Figure 2 for the Ford-Fulkerson Algorithm). The line marked **(!!!)** is the key to the algorithm: it uses the following property to compute flow:

> The maximum flow going through an path between two nodes is
> the flow of the edge with minimal capacity.

Consider a social network graph G_S where the capacity of each edge $(u, v) \in E_S$ is set to one (1). In such a graph, to find max flow between two

Algorithm FordFulkerson Input: Graph G with flow capacity $c(u,v)$ a source node s and a sink node t
Output: A flow $f(u,v)$ from s to t which is a maximum possible flow.

begin
  $f(u,v) := 0$ for all edges (u,v).
  while there is a path p from s to t in G_f, such that $c_f(u,v) > 0$ for all edges $(u,v) \in p$:
    Find $c_f(p) = \min\{c_f(u,v) \mid (u,v) \in p\}$ **(!!!)**
    for each edge $(u,v) \in p$
      $f(u,v) := f(u,v) + c_f(p)$ (Send flow along the path)
      $f(v,u) := f(v,u) - c_f(p)$ (The flow might be "returned" later)
    end for
  end while end

Figure 2: Ford-Fulkerson Algroithm for maximum flow in newtworks.

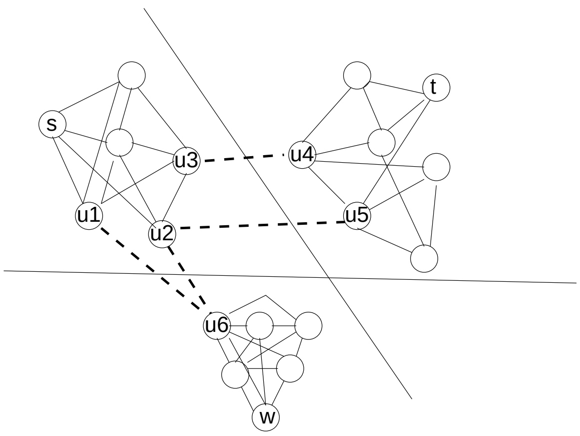

Figure 3: Max-flow in social networks. Cuts separate communities

nodes s and t, one must find **bottleneck nodes**, i.e., nodes with the least number of neighbors. See Figure 3 for an illustration.

The max-flow between nodes s and t on the graph is restrained by the two bottleneck edges $(u3, u4)$ and $(u2, u5)$. Similarly, the max-flow between s and w is restrained by the bottleneck edges $(u1, u6)$ and $(u2, u6)$. If these edges are **cut** (removed from the network), we will have separated three distinct communities.

Algorithm Find-Community. See Figure 4. The algorithm works as follows.

- Input. Social network G_S and a set of seed entities (pages) S^*. The seed pages are known by the user of the algorithm to belong to the same community. The algorithm will find the community boundaries.
- Output. $C \supset S^*$, the list of entities in the community.
- Process. The algorithm works in two steps:
  - Step 1. Crawl. The algorithm crawls the social network starting from the seed pages to collect the "vicinity" of S^*.
  - Step 2. Max-flow. Maximum flow algorithm is applied to separate the community $C \supset S^*$ from the rest of the social network.

Algorithm Find-Community(S^*, K - number of iterations)

begin
  $C := S^*$;
  for $it = 1$ **to** K **do**
    $G :=$ crawlGraph(C);
    $n := |S^*|$;
    $C^* :=$ Max-Flow-Community(G,C,n);
    rank all $v \in C^*$ by the number of edges to other members of C^*.
    $C := C \cup \{v \in C^* | v \text{ at the top of } C^*\}$
  end while   **return** C;
end

Function Max-Flow-Community(G= (V,E), C, k)

begin
  create vertices s, t, add them to V;
  for all $v \in V$ **do** add (s, v) to E, with $c(s, v) = \infty$;
  for all $(u, v) \in E$, $u \neq s$ **do**;

    $c(u, v) = k$;
    if $(v, u) \notin E$ **then**
      add (v, u) to E with $c(v, u) = k$;
  end for
  for all $v \in V$, $v \notin C \cup \{s, t\}$ **do** add (v,t) to E with $c(v, t) = 1$
  Max-Flow(G,s,t);
end

Figure 4: The algorithm for finding max-flow communities.