Bipartite Matching via Max Flow

1 Project description

In this project we will look at the problem of finding the maximum matching in a given bipartite graph G (also called bipartite matching). As a point of departure here we will consider another problem, namely the problem of finding the maximum flow in a network (which can be efficiently solved using the well-known Ford-Fulkerson algorithm) and show that the problem of finding the maximum matching in bipartite graphs can be efficiently solved by reducing it to an instance of the maximum flow problem[1].

Your task is to write the algorithm for finding the maximum matching in a given bipartite graph G. In sections 1.1 – 1.2 below we will describe such an algorithm more in detail.

1.1 Maximum flow

We start by describing the network flow problem informally. A flow in a network is a specification of how to route "stuff" from a source node s to a sink node t so that no link is used beyond its capacity, and so that every link, except the sender s and the receiver t, relays out "stuff" at exactly the same rate at which it receives from other vertices. An illustrating example of this is a communication network: (1) we know that if nodes where sending out less data than they receive then there would be data loss in the network. Moreover, (2) the nodes cannot send out more data than they receive because they simply forwarding incoming data.

Formally, a network and a network flow are defined as follows.

Definition 1. A **network** is a directed graph $G = (V, E)$, in which

- a vertex $s \in V$ and a vertex $t \in V$ are specified as being the source node and the sink node, respectively,
- every directed edge $(u, v) \in E$ has a positive capacity $c(u, v) > 0$ associated to it.

Definition 2. A **flow** in a network (G, s, t, c) is an assignment of a non-negative number $f(u, v)$ to every edge $(u, v) \in E$ such that

1. For every edge $(u, v) \in E$, $f(u, v) \leq c(u, v)$; (capture constraint (1) above)
2. For every vertex $v \in V$, $\sum_{u \in V} f(u, v) = \sum_{w \in V} f(v, w)$ (capture constraint (2) above), where we follow the convention that $f(u, v) = 0$ if $(u, v) \notin E$.

[1] A number of other problems can also be cast into a maximum flow, and so knowing how to solve a network flow problem can come in very handy.

The **cost of the flow** (the throughput of the communication, in the communication network example above) is

$$\sum_v f(s,v)$$

The following example (se Fig. 1) illustrates the aforementioned definition of the cost of a flow in a network. In this example, we are given the network depicted in Fig. 1 (a), where each edge going from node u to node v is labelled with its capacity $c(u,v)$ (for example, the edge going from node s to a has capacity $c(s,a) = 2$, and etc.).

In Fig. 1 (b), we are sending 3 units of flow from s to t (i.e., we are sending 2 units along the $s \to a \to t$ path, and 1 unit along the $s \to b \to t$). Is this flow optimal? We are only sending 3 units of flow from s to t, while we see that we can send 2 units along the $s \to a \to t$ path, and another 2 units along the $s \to b \to t$ path, for a total of 4, so the above solution is not optimal.

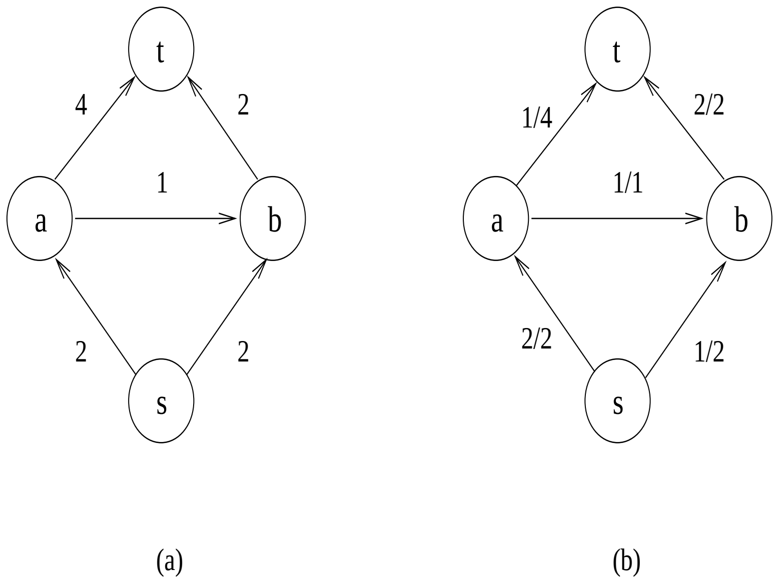

Figure 1: An illustrating example of a flow in a network. Here, a label x/y on an edge (u,v) means that the flow $f(u,v)$ is x and the capacity $c(u,v)$ is y.

Definition 3. The flow of maximum cost in the network is called the *maximum flow* in the network.

In the **maximum flow problem**, we want to find a flow of maximum cost in the network. The well-known Ford-Fulkerson algorithm described below can be used in order to find a maximum flow in a graph G.

```
Algorithm   Ford-Fulkerson
    /* Input: Graph $G$ with flow capacity $c$, a source node $s$, and a sink node
    $t$.  */
    /* Output: A flow $f$ from $s$ to $t$ which is a maximum. */
1   for each edge $(u,v) \in E$ do
1.1       $f(u,v) \leftarrow 0$
1.2       $f(v,u) \leftarrow 0$
    endfor
2   while there exists a path $p$ from $s$ to $t$ in the residual network $G_f$ do
2.1           $c_f(p) \leftarrow \min\{c_f(u,v) | (u,v) \text{ is in } p\}$
2.2           for each edge $(u,v)$ in $p$ do
2.2.0.1           $f(u,v) \leftarrow f(u,v) + c_f(p)$
2.2.0.2           $f(v,u) \leftarrow -f(u,v)$
              endfor
    endwhile
3   return $f$
End Ford-Fulkerson
```

1.2 Bipartite matching

We will now continue by describing how the problem of finding a maximum matching in a bipartite graph (i.e., the bipartite matching problem) can be solved by reducing this problem to an instance of the maximum flow problem described in the aforementioned section.

We start by providing the formal definition of a **bipartite matching**.

Definition 4. A **bipartite graph** $G = (V, E)$ is a graph in which the vertex set V can be divided into two disjoint subsets X and Y such that every edge $e \in E$ has one end point in X and the other end point in Y. A matching M in G is a subset of the edges such that each node in V appears in at most one edge in M.

Definition 5. A **maximal matching** is a matching to which no more edges can be added without increasing the degree of one of the nodes to two; it is a local maximum.

Definition 6. A **maximum matching** is a matching with the largest possible number of edges; it is globally optimal.

Fig. 2 shows an illustrating example of a matching in the bipartite graph G.

Question 1. Is the matching depicted in Fig. 2 a maximum matching? Motivate your answer.

1.2.1 The reduction

The problem of finding the maximum matching in a graph can be reduced to maximum flow in the following manner. Let $G = (V, E)$ be the bipartite graph where V is divided into X and Y. We will construct a directed graph $G = (V, E)$, in which V contains all the nodes of V along with a source node s and a sink node t. For every edge in E, we add a directed

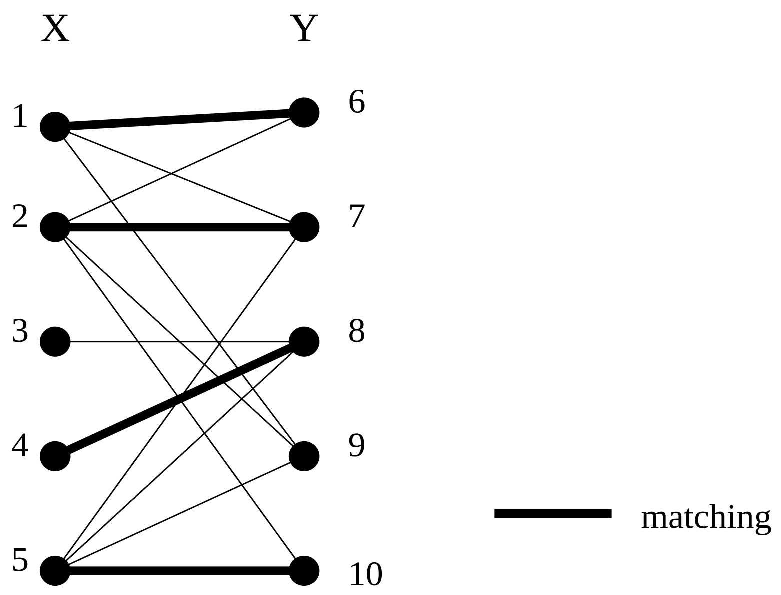

Figure 2: An illustrating example of a matching in the bipartite graph $G = (X, Y, E)$.

edge in E from X to Y. Finally we add a directed edge from s to all nodes in X and from all nodes of Y to t. Each edge is given unit capacity. See Fig. 3 for an illustrating example of the reduction.

Let f be an integral flow of G of value k. Then we can make the following observations:

1. There is no node in X which has more than one outgoing edge where there is a flow.
2. There is no node in Y which has more than one incoming edge where there is a flow.
3. The number of edges between X and Y which carry flow is k.

By these observations, it is straightforward to conclude that the set of edges carrying flow in f forms a matching of size k for the graph G. Likewise, given a matching of size k in G, we can construct a flow of size k in G. Therefore, solving for maximum flow in G gives us a maximum matching in G. Note that we used the fact that when edge capacities are integral, Ford-Fulkerson produces an integral flow.

Question 2. What is the running time of this algorithm? Motivate your answer in detail.
Question 3. Can the running time of such an algorithm be improved? If so, motivate your answer in detail.

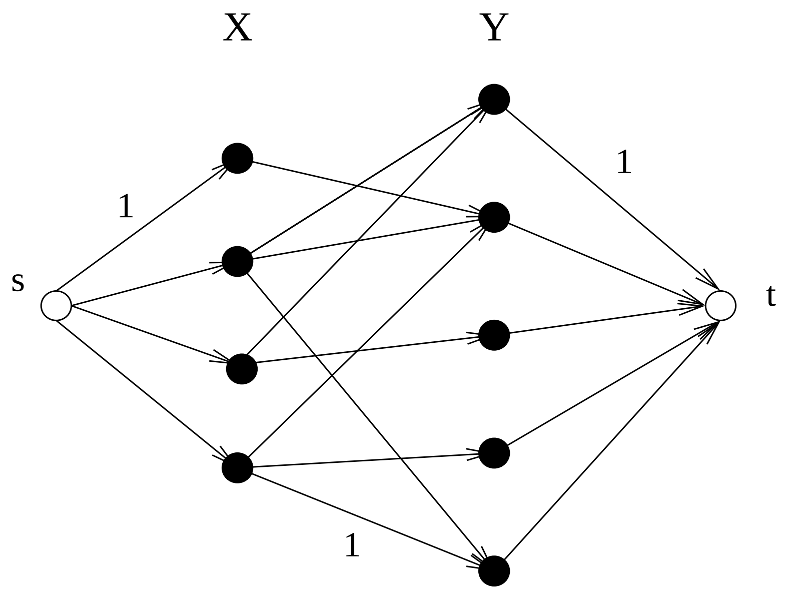

Figure 3: An illustrating example of the reduction.

2 Implementation – Some hints

2.1 Input

The input to your program should be as follows.

- The number of edges in the bipartite graph $G = (X, Y, E)$ (i.e., the bipartite graph where we want to find a bipartite matching).
- The edges in the bipartite G, where each edge is represented as a pair (u, v), where $u \in X$ och $v \in Y$. Note that your program should handle also the case when a user provides the edge (a, a), where $a \in X$ and $a \in Y$, and a and a are two nodes and they are copies.
- Each edge is unique.

2.2 Output

The output of your program should be as follows.

- The number of edges in the bipartite matching in G (i.e. the number of edges in the maximum matching in the bipartite graph G).
- The edges in the bipartite matching in G, on the format u –> v where $u \in X$ and $v \in Y$.