ICS 260—Fall, 2000–Class Notes 6: Matching Problems in Graphs

Notes Originally Developed by Sandy Irani
Modified by Michael B. Dillencourt

1 Matching

Given an undirected graph $G = (V, E)$, a *matching* is a subset of the edges $M \subseteq E$ such that for each vertex $v \in V$, at most one edge in M is incident to v. We say that a vertex is *matched* if an edge in M is incident to v. A *maximum matching* is a matching M such that for any other matching M', $|M| \geq |M'|$. We will concentrate on finding a maximum matching in a bipartite graph. A *bipartite graph* is a graph whose vertices can be partitioned into two sets $V = L \cup R$, where L and R are disjoint and all edges go between L and R.

There are many applications of bipartite matching. One can consider matching applicants to jobs, where an edge beteen an applicant and a job indicates that the applicant is qualified for the particular job. Alternatively, one can image matching tasks to processors.

We will use an algorithm for the maximum network flow problem to construct a maximum matching in a bipartite graph.

Given a bipartite graph $G = (L \cup R, E)$, we will construct a network G' as follows. $V' = L \cup R \cup \{s, t\}$. The edge set will be

$$\begin{aligned} E &= \{(s, u) \mid u \in L\} \\ &\cup \ \{(v, t) \mid v \in R\} \\ &\cup \ \{(u, v) \mid u \in R, v \in L, (u, v) \in E\} \end{aligned}$$

Each edge in E' has a capacity of one.

An *integer-valued flow* is a flow such that for each node pair (u, v), $f(u, v)$ is in integer.

Lemma 1 *Let G be a bipartite graph, and G' the corresponding flow network as described above. If G has a matching M, then G' has an integer-valued flow f such that $|M| = |f|$. Also, if G' has an integer-valued flow f, then G has a matching M such that $|M| = |f|$.*

Proof. First consider a matching M in the bipartite graph. For each edge (u, v) in the matching, we will set $f(s, u) = f(u, v) = f(v, t) = 1$ and set $f(u, s) = f(v, u) = f(t, v) = -1$. Each edge gives rise to a vertex disjoint path along which one unit of flow can be pushed. Thus, $|M| = |f|$.

Now consider an integral-valued flow in the network. We will let $M = \{(u, v) \mid f(u, v) > 0, u \in L, v \in R\}$. Since it is integer-valued and the capacity of each edge is 1, for each edge (u, v) in the network, $f(u, v) = 1$ or $f(u, v) = 0$ or $f(u, v) = -1$. Thus, for each edge $(u, v) \in M$, $f(u, v) = 1$.

Since each node $u \in L$ can have at most one unit of flow entering it from s, there will be only one edge in M which is incident to u. Similarly, since each node $v \in R$ can have at most one unit of flow leaving it, there will be only one edge in M incident to v.

For each edge (s, u) such that $f(s, u) = 1$, there is exactly one edge (u, v) with $u \in L$ and $v \in R$ such that $f(u, v) = 1$. Thus, $|f| = |M|$. ∎

The lemma suggests that we can use an algorithm for maximum flow to find a maximum matching in a bipartite graph, except we must address happens if our algorithm for maximum flow returns a flow which is not integer valued. We have to establish that whatever method we happen to choose to solve the maximum flow problem on G', it will return an integer-valued flow as long as the capacities are integer valued. This is certainly true for the Ford-Fulkerson algorithm:

Theorem 1 *If the capacity of each arc is an integer, then the Ford-Fulkerson method will return a flow f, such that for every pair of vertices (u, v), $f(u, v)$ is an integer.*

Proof Sketch. This theorem can easily be proved by induction on the number of augmenting paths — if the current flow is integer-valued, then the capacities of all the arcs in the residual network are integer valued. This means that the value of the augmenting path (which is the minimum of all the residual capacities of the arcs in the path) is also integer valued. Since the value of the flow along the augmenting path is integer-valued, the new flow which is the sum of two integer-valued flows will also be integer-valued. ∎

How long does the Ford-Fulkerson algorithm take? Recall that the running time of the Ford-Fulkerson method is $O(mf^*)$, where f^* is the value of the maximum flow in the network. In this case, $f^* \leq n$ because cut $(\{s\}, V - \{s\})$ has value at most n. Thus, Ford-Fulkerson will find a maximum matching in time $O(nm)$.

There is an algorithm due to Even and Tarjan that solves max flow on networks with unit capacity arcs which runs in time $O(\sqrt{n}m)$. This in turn yields an algorithm for unweighted bipartite matching which runs in $O(\sqrt{n}m)$.

Matching in graphs which are not bipartite is a considerably harder problem and requires substantially different techniques. However, the fastest algorithm due to Micali and Vazirani matches the worst case upper bound of $O(\sqrt{n}m)$.

2 Minimum Cost Flow

The shortest path problem and the maximum flow problem are special cases of another central problem involving networks called the Minimum Cost Flow Problem. We will not discuss this problem in detail, but we will define the problem and discuss some of its applications.

In the min cost flow problem, each arc (u, v) has an upper bound $U(u, v)$ and a lower bound $L(u, v)$ on the flow that can be assigned to the edge. That is, the flow must satisfy $U(u, v) \leq f(u, v) \leq L(u, v)$. Furthermore, every node v is assigned a supply/demand $b(v)$. If $b(v) < 0$, then node v has a *demand* of $b(v)$ and $b(v)$ units of flow must flow into v. If $b(v) > 0$, then v has a *supply* of $b(v)$, and $b(v)$ units of flow must leave node v. Finally, each edge (u, v) also has a cost. It costs $c(u, v)$

per unit of flow sent through (u, v). That is, the entire cost of the flow is:

$$\sum_{(u,v)|f(i,v)>0} f(u, v) \cdot c(u, v).$$

The goal is to find a feasible flow (which satisfies all the upper and lower bounds on the edges and all the supply/demands of all the nodes) which has a minimum cost.

Maximum flow can be formulated as a min cost flow problem, by setting all $b(v) = 0$ and all lower bounds to 0. The upper bound for each edge is the capacity of the edge for the max flow problem. The cost for each edge is 0. Finally, we put an arc from t to s with infinite capacity and cost -1. Any feasible flow has to obey the capacity constraints and must have a net zero flow. However, the flow which minimizes the cost is the flow which sends the most amount of flow from s to t in the network.

For the single source shortest path problem, set $b(s) = n - 1$ and $b(v) = -1$ for all other nodes. Each edge has a large capacity ($n - 1$ will suffice) and a lower bound of 0. The cost of every edge is its weight.

2.1 Bipartite Weighted Matching Problem

Given a bipartite graph G with $|L| = |U|$ find the perfect matching with minimum weight. A *perfect matching* is a matching M where $|M| = |L| = |R|$.

This is a special case of the min cost flow problem. For $v \in L$, $b(v) = 1$. For $v \in R$, $b(v) = -1$. The lower bound for each edge (u, v), where $u \in L$ and $v \in R$ is that $f(u, v) \geq 0$.

Of course, it must be established that the optimal solution to the special case of the min cost flow problem will be an integral-valued solution.

The solutions to bipartite weighted matching are very similar to the min cost flow problem, although the special structure of the matching problem can be exploited to give a more efficient solution.

There are a number of algorithms in the literature which solve this problem in time $O(nS(n, m, c_{max}))$, where $S(n, m, c_{max}))$ is the time to solve the single source shortest path algorithm on a graph with non-negative edge weights. There is also an algorithm which solves this problem in time $O(\sqrt{n}m \log(nC))$.

One of the algorithms which give the first time bound solves the single source shorest path problem on the residual network, updates distance labels and augments along a path from a source to a sink. It can be proven that it terminates in $O(n)$ iterations and finds an integral-valued flow.

2.2 Applications

- Optimal Hashing of a static set of strings
- Chinese Postman Problem