Title

Computer Science 604
The Marriage Problem

David Juedes

School of EECS

`juedes@cs.ohiou.edu`

An interesting problem

Consider the following simple problem.

We have a collection of n men and n women. We would like it to be the case that each person can find a suitable partner and that all $2n$ people can be happily married. Each person has a list of suitable marriage candidates from which to chose. We want to determine if it is possible to pair up each person with a suitable marriage candidate so that all $2n$ people can be married. This is the *marriage problem*.

This is somewhat related to "Refrigerator Madness" (see last slides).

Cont'd

While the applications of this problem to marriage are usually quite limited, related problems have important applications in industrial applications where, for example, it may be desirable to match tasks with qualified employees so that each task is completed and each employee has a job.

These types of problems are referred to "assignment problems."

Ideas

How can we solve this type of problem?

Modeling

We can model this type of problem with a graph.

Every person is a node in the graph, and there are edges between people who would be acceptable pairs. (I.e., Jane likes Jack, and Jack likes Jane.)

Furthermore, we can break this graph into two pieces: the men and the women.

Bipartite Graphs

Such a graph that can be split into two pieces V_1 and V_2 where all edges go from V_1 to V_2 is called a *bipartite graph*.

Formally, a bipartite graph $G = ([V_1, V_2], E)$ is a graph where for every edge $\{u, v\} \in E$, it must be the case that $u \in V_1$ and $v \in V_2$ or vice-versa.

If we are given a bipartite graph, then we can solve the marriage problem by finding a *maximum matching*.

Maximum Matchings in Bipartite Graphs

Formally, a *matching* in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that no two edges share an endpoint.

A vertex c is said to be "covered" by the matching if

$$c \in C(M) = \bigcup_{e \in E} e$$

The set of uncovered vertices is $U(M) = V - C(M)$.

(Notice that we are treating each edge as a set of vertices.)

Example: See the board.

Finding Maximum Matchings

The marriage problem can be cast as the problem of finding a *perfect* matching in a bipartite graph G, i.e., a matching where every vertex is covered. The matching in this case corresponds to the marriage coupling.

We can determine whether such a graph has a perfect matching by finding the *maximum cardinality matching* in G.

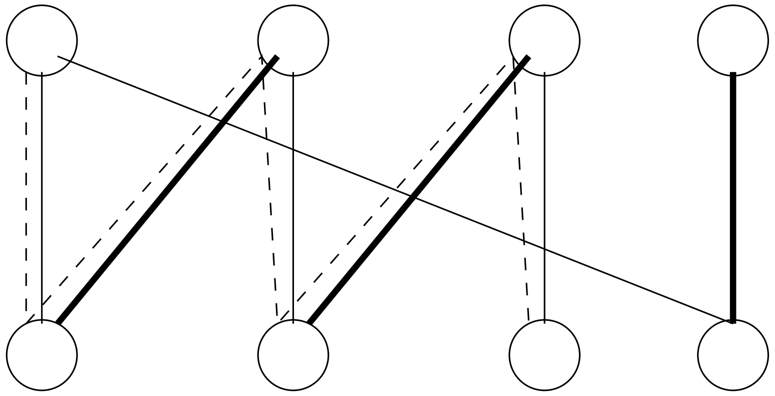
Notice that finding some matching is easy.... Simply pick edges until you can't add any. Finding the maximum cardinality matching is harder.

It can be done by finding *augmenting paths*.

Alternating/Augmenting Paths

Given a bipartite graph G and a matching M in G, an M-alternating path $e_1, \ldots, e_i$ (given as a list of edges) in G is a path in G where $e_i \in M$ for every even value of i, and $e_i \notin M$ for every odd i.

An M-augmenting path is an M-alternating path where the path begins at an uncovered vertex $u \in U(M)$ and ends at an uncovered vertex $v \in U(M)$.



Cont'd

If G is a graph and M is a matching, then a larger matching M can be constructed from any augmenting path — simply replace the edges in M in the augmenting path with the edges not in M.

This gives a matching M' of size $|M| + 1$. Hence, the existence of an augmenting path in a graph indicates that the given matching is not a maximum matching. A matching M contains the maximum number of edges if there is no augmenting path in G for M.

Notation

Let M be a matching in a graph $G = (V, E)$, i.e., $M \subseteq E$, where edges in M do not share endpoints. The matching covers the set $V' = \{u | \{u, v\} \in M\}$. The set of exposed vertices is $V - V'$.

An M-alternating path in G consists of a simple path in G where the edges in the path alternate between edges in M and edges in $E - M$. Notice that if u an exposed vertex, then an M-alternating path starting at u must begin with an edge that is not in the matching. An M-augmenting path is an M alternating path that begins and ends with an exposed vertex.

It is easy to see that if there exists an M-augmenting path in G, that M is not a maximum matching since we can create a larger matching by swapping the edges in the M-augmenting path.

An Important Theorem

Most algorithms for finding maximum matchings depend on the following theorem due to Berge (1957) and Norman and Rabin (1959).

Theorem 1 A matching M has maximum cardinality if and only if there is no M-augmenting path.

We can use this result to give a polynomial-time algorithm for maximum cardinality matching in bipartite graphs.

Using Theorem 1

How can we use Theorem 1 to give an algorithm to find the maximum cardinality matching? Well, M is a matching of maximum cardinality if and only if for each pair (u, v) of exposed vertices (vertices that are not endpoints in the matching), there does not exist an augmenting path from u to v.

If there exists an M-augmenting path, then we can simply swap the edges in the augmenting path to create a larger matching. This can happen at most $n/2$ times since the size of the largest matching is at most $n/2$.

Finding Augmenting Paths in Bipartite Graphs

Finding augmenting paths in bipartite graphs of the form $G = ([V_1, V_2], E)$ is easier than finding augmenting paths in general graphs because if there is an M-alternating path that begins at a vertex $u \in U(M) \cap V_1$, then the terminal vertex v in the path must be either in V_1 (if the path length (# of edges) is even), or in V_2 (if the path length is odd). Hence, once you have determined that there exists a M-alternating path to a vertex, you can mark that vertex as "visited."

Using Breadth-First Search

We can use breadth-first search in a straightforward manner to find augmenting paths.

```
INPUT: a bipartite graph $G = ([V_1, V_2], E)$ and a matching $M$
$Visited = U(M) \cap V_1$; .
**for** each vertex $v \in U(M) \cap V_1$ **do**
   place $v$ in the queue *next_v*
**end for**
```

Cont'd

```
while next_v is not empty do
    $new = next\_v.front()$;
    $next\_v.pop()$;
    $Visited = Visited \cup \{new\}$;
    if $new \in V_2$ then
        if $new \in U(M)$ then
            FOUND AN AUGMENTING PATH!
        else
            Let $\{new, v\} \in M$;
            if $v \notin Visited$ then
                place $v$ in next_v;
            end if
        end if
    else
        for each neighbor $v$ of $new$ do
            if $v \notin Visited$ then
                place $v$ in next_v
                $Visited = Visited \cup \{v\}$;
            end if
        end for
    end if
end while
```

Finding Maximum Matchings

The algorithm to find maximum matchings is as follows:

1. Find some matching M.
2. While there is a M augmenting path P do
   (a) Augment M by P.
3. Return M.

What's the running time of this algorithm?

Stable Marriages

Sometimes we want to solve a marriage problem where we have preferences, i.e., it's not just that A is willing to marry B, but A may prefer B_1 over B_2.

According the associate Wikipedia page
(Stable Marriage Problem)

A matching is stable whenever it is not the case that both a) some given element A of the first matched set prefers some given element B of the second matched set over the element to which A is already matched and b) B also prefers A over the element to which B is already matched in other words, when there does not exist any alternative pairing (A, B) in which both A and B are individually better off than they would be with the element to which they are currently matched.

Stable Marriage Problems

The stable marriage problem is to find a matching that is stable.

Finding Stable Matchings

Finding a stable matching is a bit harder than finding a maximum cardinality matching in a bipartite graph.

The Gale-Shapley algorithm (1962) is the standard for finding stable marriages if they exist.