Maximum or Minimum Perfect Matchings in Bipartite Graphs

Chan-Ching Hsu
cchsu@iastate.edu

February 10, 2016

Problem: Maximum or Minimum Perfect Matchings in Bipartite Graphs

1 Problem

In these report we consider the following problem essentially:
Maximum Weight Bipartite Matching *Given a bipartite graph* $G = (V, E)$ *bipartite* (X,Y) *and weight function* $w : E \to \mathbb{R}$ *find a matching of maximum weight where the weight of matching of matching* M *is given by* $w(M) = \sum_{e \in M} w(e)$.
In the maximum weighted matching problem a non-negative weight $w_{i,j}$ is assigned to each edge $x_i y_j$ of $W_{w,w}$ and we seek a perfect matching M to maximize the total weight $w(M) = \sum_{e \in M} w(e)$. With these weights, a (weighted) cover is a choice of labels $u_1, \ldots, u_w$ and $v_1, \ldots, v_w$, such that $u_i + v_j \geq w_{i,j}$ for all i, j. The cost $c(u, v)$ of a cover (u, v) is $\sum u_i + \sum v_j$. The minimum weighted cover problem is that of finding a cover of minimum cost.
The equality subgraph $G_{u,v}$ for a weighted cover (u, v) is the spanning subgraph of $W_{w,w}$ whose edges are the pairs $x_i y_j$ such that $u_i + v_j = w_{i,j}$. In the cover, the excess for i, j is $u_i + v_j - w_{i,j}$.

2 Implementation

I ran everything on one of the servers, `linux-5.ece.iastate.edu` [1], which is a Linux platform available to engineering students in ISU. I implemented the augmenting path algorithm for for maximum matching in unweighted bipartite graphs and the Hungarian algorithm for maximum weighted perfect matchings in bipartite graphs. A part of the outputs includes the minimum-weight vertex cover for given problems. As required, I also built a method for solving the minimum weight perfect matching problem for bipartite graphs. The way I did is programming a reduction to the maximum weighted perfect matching problem, additionally, with a maximum-weight vertex under-cover presented for the proof by duality.
Those programs were developed with C [2] programming language. To run our algorithms on a bipartite graph, a square weight matrix $\mathcal{W}$, and the size of this square matrix w are given. For the maximum perfect matching, the first step is to construct the initial excess matrix, accompany with arrays $\mathcal{U}$ and $\mathcal{V}$. For every row in $\mathcal{W}$, every element is subtracted from the largest number in that row to form the initialized excess matrix, $\mathbf{E}$. With $\mathbf{E}$, an unweighted bipartite graph, $\mathbf{B}$, is built. Here, each edge is indicated by the 0's in $\mathbf{E}$. Based on $\mathbf{E}$, starting from the first row, I match the node, if any, with the first 0 found by looking at every column.

The algorithm for finding an M-augmenting path in an unweighted bipartite graph performs next. If no augmenting paths exist, the (unweigthed) vertex cover, $\mathcal{R}$ and $\mathcal{T}$, can be found and the size should be the same with the (unweigther) matching, m. If an augmenting path is found, m will be updated and the procedure of finding augmenting paths will perform again. With the information of the (unweighted) vertex cover, the value of ε is obtained by finding the smallest elements in $\mathbf{E}$ but not in either $\mathcal{R}$ nor $\mathcal{T}$.
$\mathbf{E}, \mathcal{U}$ and $\mathcal{V}$ are updated accordingly in the manner of 1. for the nodes in $\mathcal{R}$ and $\mathcal{T}$, the corresponding elements in $\mathbf{E}$ are increased by ε, 2. for the nodes not either in $\mathcal{R}$ or $\mathcal{T}$, the corresponding elements in $\mathbf{E}$ are decreased by ε, 3. for the nodes in $\mathcal{T}$, the corresponding elements in $\mathcal{V}$ are increased by ε, and 4. for the nodes not in $\mathcal{R}$, the corresponding elements in $\mathcal{U}$ are decreased by ε.
This completes the first iteration of the Hungarian algorithm. At this point, the current matching is stored as m is found, the current (weight) vertex cover is stored as $\mathcal{U}$ and $\mathcal{V}$, and the current excess matrix is stored as $\mathbf{E}$. If m is a perfect matching, the maximum perfect matching and minimum vertex cover will be output; otherwise, the procedure of the Hungarian approach will continue with the current information.
For $\mathbf{B}$ with given m, to find an M-augmenting path, the augmenting path algorithm is implemented. First, all the unmated nodes from $\mathcal{U}$ are put into $\bar{\mathcal{R}}$. Starting from $x \in \bar{R}$, the algorithm puts $y \in Y$ into $\mathcal{T}$ that having edges to x if y has not been visited, then putting $x \notin \bar{\mathcal{R}}$ having edges to y into $\bar{\mathcal{R}}$ if $x \in X$ has not been visited. Then the unmated x is marked as explored. The procedure will goes through other unexplored $x \in \bar{\mathcal{R}}$ with the same actions till an augmenting path is found or there is no unexplored x. The predecessors, if applicable, are stored as the algorithm goes on. Once it finds a y from $x \in \bar{\mathcal{R}}$ with no matched edge incident to other x, then an augmenting path is found. All the matchings along this newly found path will be alternated by following the predecessor information.
For the minimum-weight perfect matching problem, I perform the reduction to the maximum weighted perfect matching problem. The given weight matrix (originally $\mathcal{W}$) now is denoted as $\mathcal{W}'$. The reduction is performed in the following way, the largest number in $\mathcal{W}'$ is discovered first, a; then every element in $\mathcal{W}'$ is subtracted from a to get $\mathcal{W}$ for the developed Hungarian procedure. Next, the maximum perfect matching is able to be disclosed by running the Hungarian program with input $\mathcal{W}$. After reaching the maximum perfect matching, M, we can translate M to the minimum perfect matching by computing $|M'| = w * a - |M|$, which can also be concluded by summing the weights of the matchings based on M or M'. The duality comes from computing $\sum a - u_i - v_j$, where $u_i \in \mathcal{U}, v_i \in \mathcal{V}$.
I didn't encounter a big challenge during my implementation. I would say these algorithms are well developed for the matching problems and easy to implement for practical use. I started the implementation from programming maximum matching in unweighted bipartite graph and then the Hungarian algorithm for maximum weighted perfect matchings in bipartite graphs, followed by realizing method for solving the minimum-weight perfect matching problem. All details can be supplied by request regarding the code of the implementation.

3 Solutions

I now discuss our solutions to the given problem instances. It has been verified that for the maximum weighted perfect matching problems, the value of matching is equal to that of vertex cover; for the minimum weighted perfect matching problems, the value of matching is equal to that

of vertex under-cover.

3.1 Instance I1

The six problems in homework 4 were processed as input to our algorithm to demonstrate the algorithm is able to work correctly.

```
Maximum weighted perfect matching: 82
Minimum-weight vertex cover: 82
Minimum weighted perfect matching: 9
Maximum-weight vertex under-cover: 9
```

3.2 Instance I2

Due to the space limit, the matched pairs for both problems (maximum and minimum weight perfect matching) are omitted but can be provided by request.

```
Maximum weighted perfect matching: 265
Minimum-weight vertex cover: 265
Minimum weighted perfect matching: 31
Maximum-weight vertex under-cover: 31
```

3.3 Instance I3

This instance provides a neat problem size to test our algorithm with 100*100 matrix size. I was able to discover a few minor defects corrected later by taking I3. For example, after not getting matched matching and vertex cover values, I tracked back the excess matrices and epsilon values iteration by iteration, and was able to realize the arrays of u and v (denoted as in class and videos) were not increased and decreased by a right epsilon number accordingly, which was fixed immediately.

```
Maximum weighted perfect matching: 4864
Minimum-weight vertex cover: 4864
Minimum weighted perfect matching: 39
Maximum-weight vertex under-cover: 39
```

3.4 Instance I4

The last instance can be seen as a large case which requires apparently much more computational time to solve the problem as the size of the problem increases. The first I saw the results was surprised since those resulting numbers look nice and perfect. Then I started to retrieve each matching in the solution to see whether every selected edge had the weight of 99 for maximum perfect matching (99*1000) and the weight of 0 for minimum one (0*1000). And it turned out those numbers are generated appropriately.

```
Maximum weighted perfect matching: 99000
Minimum-weight vertex cover: 99000
Minimum weighted perfect matching: 0
Maximum-weight vertex under-cover: 0
```

References

[1] Linux Servers in College of Engineering, Iowa State University, `http://it.engineering.iastate.edu/remote/`

[2] C Programming Language, *Wikipedia*, `http://en.wikipedia.org/wiki/C_(programming_language)`