Computers Math. Applic. Vol. 25, No. 5, pp. 65–71, 1993
Printed in Great Britain. All rights reserved

FINDING A COMPLETE MATCHING WITH THE MAXIMUM PRODUCT ON WEIGHTED BIPARTITE GRAPHS

FRANK S. C. TSENG AND WEI-PANG YANG
Department of Computer Science and Information Engineering, National Chiao Tung University
Hsinchu, Taiwan 30050, Republic of China
ARBEE L. P. CHEN*
Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan 30043, Republic of China

(Received June 1992)

Abstract—The traditional bipartite weighted matching problem is to maximize the largest possible *sum* of weights. In this paper, we define a bipartite matching problem which maximizes the largest possible *product* of weights and develop an algorithm to solve it. Although this problem corresponds to a non-linear program, we show this problem can be easily solved by modifying the *Hungarian method*. Finally, we present an application of this problem.

1. INTRODUCTION

Graph theory [1] is a useful mathematical tool to model systems involving discrete objects. A *matching* in a graph is a set of edges, no two of which are adjacent. Given a bipartite graph, it is a well-known problem to find a matching that has as many edges as possible. Many interesting algorithms for solving such a problem have been developed. For example, in [2], Hopcroft and Karp presented an efficient algorithm to solve such a problem. Even and Tarjan [3] pointed out that such a problem can be solved by employing the max-flow algorithm [4].

A more generalized version of the matching problem is to consider a weighted bipartite graph and find a matching with the largest possible *sum* of weights. This generalized problem has been solved by the *Hungarian Method* [5, 6]. This problem can be formulated as a 0-1 integer linear program as follows.

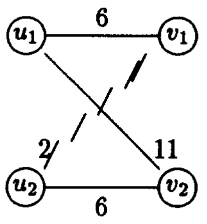
Let x_{ij} be a set of variables for $i = 1, \dots, n$ and $j = 1, \dots, n$, where n is the number of nodes in each node set of the complete bipartite graph $G = (U \cup V, E)$, $U = \{u_1, u_2, \dots, u_n\}$ and $V = \{v_1, v_2, \dots, v_n\}$. Here, $x_{ij} = 1$ means that the edge (u_i, v_j) is included in the matching, whereas $x_{ij} = 0$ means that it is not. Therefore, for a set of such values to represent a complete matching, we have the following constraints:

$$\begin{aligned} &\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \dots, n, \\ &\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \dots, n, \\ &x_{ij} = 0 \text{ or } 1. \end{aligned}$$

The goal of this problem is to maximize $\sum_i \sum_j w_{ij} x_{ij}$, where w_{ij} is the weight of the edge (u_i, v_j).

However, how to solve the matching problem with the largest possible *product* of weights (i.e., maximizing $\prod_i \sum_j w_{ij} x_{ij}$ with the same constraints) has not been established. The solution of maximizing $\sum_i \sum_j w_{ij} x_{ij}$ is not necessarily the same as that of $\prod_i \sum_j w_{ij} x_{ij}$. An example is shown as follows. Consider the following weighted bipartite graph.

*Author to whom all correspondence should be sent.



The maximum matching with the largest possible *sum* of weights is $\{(u_1,v_2),(u_2,v_1)\}$ with their sum of weights

$$\sum_i \sum_j w_{ij}\, x_{ij} = 11 + 2 = 13.$$

The other matching is $\{(u_1,v_1),(u_2,v_2)\}$ with their sum of weights

$$\sum_i \sum_j w_{ij}\, x_{ij} = 6 + 6 = 12.$$

However, the maximum matching with the largest possible *product* of weights is $\{(u_1,v_1),(u_2,v_2)\}$ with their product of weights

$$\prod_i \sum_j w_{ij}\, x_{ij} = 6 \times 6 = 36.$$

The other matching is $\{(u_2,v_1),(u_1,v_2)\}$ with their product of weights

$$\prod_i \sum_j w_{ij}\, x_{ij} = 11 \times 2 = 22.$$

In this paper, we will provide a solution to the matching problem with the objective function $\prod_i \sum_j w_{ij}\, x_{ij}$. We call such problem as the bipartite maximum product matching problem (MPM). Although the problem formulation is a non-linear program, we find the *Hungarian Method* can be modified to solve it.

2. THE MODIFIED HUNGARIAN METHOD FOR MPM

In such a weighted matching problem on a bipartite graph $G = (U \cup V, E)$, we can assume the underlying graph is a complete bipartite graph, $K_{n,n}$. Otherwise, without loss of generality, if $\mid U \mid < \mid V \mid$ then we can add $(\mid V \mid - \mid U \mid)$ new nodes to U with edges of weight one incident from all nodes in V to each of them. Furthermore, if there are missing edges in G we can assign the weights of these edges to be zero.

To attack this problem, it is more convenient to consider it as a *minimization* problem by simply considering the weight of (u_i, v_j) to be $\frac{1}{w_{ij}}$. That is, we want to find a matching M which minimizes $\prod_i \sum_j x_{ij}/w_{ij}$ under the following constraints:

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \dots, n, \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \dots, n, \tag{2}$$

$$x_{ij} = 0 \text{ or } 1. \tag{3}$$

We now present an example to sketch how to find such a solution. Then we will modify the *Hungarian* algorithm presented in [7], in which it was used to find a complete matching M that maximizes $\sum_i \sum_j w_{ij} x_{ij}$. For formal treatment of that algorithm, refer to [8].

EXAMPLE 2.1. We begin by representing the bipartite graph in matrix form, $[m_{ij}]$, where $m_{ij} = \frac{1}{w_{ij}}$ is the inverse of the weight of the edge (u_i, v_j). An example matrix is given below.

$$\begin{bmatrix} 2 & 5 & 4 \\ 3 & 8 & 7 \\ 1 & 4 & 6 \end{bmatrix}.$$

Following Theorem 2.1 (to be discussed later), our solution remains unchanged if we divide all members of some row or some column by the smallest number of the row or the the column. This follows since only one entry will be selected from any row or any column. Therefore, the value of $\prod_i \sum_j x_{ij}/w_{ij}$ for any matching M will be divided by the same amount.

By dividing each row by the smallest member of that row, the example matrix thus becomes

$$\begin{bmatrix} 1 & \frac{5}{2} & 2 \\ 1 & \frac{8}{3} & \frac{7}{3} \\ 1 & 4 & 6 \end{bmatrix}.$$

Analogously, by dividing each column by the smallest member of that column, the example matrix becomes

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \frac{16}{15} & \frac{7}{6} \\ 1 & \frac{8}{5} & 3 \end{bmatrix}.$$

Our problem now is to select n (here, 3) entries from the matrix, such that there are no two in the same row or column (we call these entries *independent*), with as small a product as possible.

Since our entries are all greater than or equal to one, the smallest product we could hope for is one. Therefore, if n independent entries of value one can be found, then an optimal solution can be obtained. Now, since all the ones are contained in the first column and the first row, such a solution is not present. This can be seen by crossing with a line on the first row and a line on the first column. Notice that the entry in the left-upper corner is crossed twice. We now adjust the matrix by the following procedure.

1. Let k be the smallest number that is not included in any of our crossed rows or columns. In our example, $k = \frac{16}{15}$.
2. Divide all uncrossed numbers by k.
3. Leave entries which are crossed once unchanged.
4. Multiply k to all numbers which are crossed twice.

This procedure produces at least one or more one in the uncrossed position of our matrix and leaves all the ones unchanged, unless they are crossed twice. It is impossible for an optimal solution to include those entries which are crossed twice.

By following this procedure until n independent ones are obtained, our example matrix becomes

$$\begin{bmatrix} \frac{16}{15} & 1 & 1^* \\ 1 & 1^* & \frac{35}{32} \\ 1^* & \frac{3}{2} & \frac{45}{16} \end{bmatrix}.$$

The optimal solution entries are all marked with an asterisk (*). Referring to the original matrix, $\prod_i \sum_j x_{ij}/w_{ij} = 1 \times 8 \times 4 = 32$. That is, $\prod_i \sum_j w_{ij}\, x_{ij} = \frac{1}{32}$. ∎

This algorithm can now be summarized by the following steps.

ALGORITHM 2.1. *An algorithm for MPM.*

Input: A Matrix $[m_{ij}]$, $m_{ij} = \frac{1}{w_{ij}}$ representing a weighted bipartite graph.

Output: A Matching with the largest possible *product* of weights.

1. (***Reduce row and column.***) Divide each row by the smallest number of that row. Do the same for each column.
2. (***Check for n independent ones.***) If there are n independent ones, then we are done; stop. Obtain the corresponding edges as the resultant matching.
3. (***Find a minimal cover to adjust the matrix.***) Otherwise, find the minimum number of lines that cross all ones. Let k be the smallest number of those uncrossed entries. Divide those uncrossed numbers by k and multiply k to those numbers that are crossed by two lines. Go to Step 2. ∎

Now, we show that if $[m'_{ij}]$ is the matrix obtained from $[m_{ij}]$ by Step 1 of Algorithm 2.1, then the solution matchings of $[m'_{ij}]$ and $[m_{ij}]$ are identical.

THEOREM 2.1. *If $[m'_{ij}]$ is the matrix obtained from $[m_{ij}]$ by Step 1 of Algorithm 2.1, then the solution matchings of $[m'_{ij}]$ and $[m_{ij}]$ are identical.*

PROOF. Suppose that α_i and β_j are the smallest numbers used to divide the ith row and jth column of $[m_{ij}]$, respectively, for each row i and each column j. Then,

$$m'_{ij} = \frac{m_{ij}}{\alpha_i\,\beta_j}.$$

Let X and X' be the objective functions associated with the old MPM and the new MPM problems represented by $[m_{ij}]$ and $[m'_{ij}]$, respectively; i.e.,

$$X' = \prod_i \sum_j m'_{ij}\, x_{ij} \quad \text{and} \quad X = \prod_i \sum_j m_{ij}\, x_{ij}.$$

Since constraints (1), (2), and (3) can be regarded as: for each permutation

$$\pi : \{1,\dots,n\} \to \{1,\dots,n\},$$

$$x_{ij} = \begin{cases} 1, & \text{if } j = \pi(i), \\ 0, & \text{else}, \end{cases} \qquad i = 1,\dots,n \quad \text{and} \quad j = 1,\dots,n, \tag{4}$$

the new problem that maximizes $X' = \prod_i \sum_j m'_{ij}\, x_{ij}$ under constraints (1), (2), and (3) is equivalent to

$$\begin{aligned}
&\text{maximize} \textstyle\prod_i \sum_j m'_{ij}\, x_{ij} && \text{under constraint (4)},\\
\equiv\ &\text{maximize} \quad \textstyle\prod_i m'_{i\pi(i)} && \text{for all possible permutations } \pi,\\
\equiv\ &\text{maximize} \quad \textstyle\prod_i \frac{m_{i\pi(i)}}{\alpha_i\,\beta_{\pi(i)}} && \text{for all possible permutations } \pi,\\
\equiv\ &\text{maximize} \frac{\prod_i m_{i\pi(i)}}{\prod_i \alpha_i \prod_i \beta_{\pi(i)}} && \text{for all possible permutations } \pi,\\
\equiv\ &\text{maximize} \frac{\prod_i \sum_j m_{ij}\, x_{ij}}{\prod_i \alpha_i \prod_i \beta_i} && \text{under constraint (4)},\\
\equiv\ &\text{maximize} \frac{X}{\prod_i \alpha_i \prod_i \beta_i} && \text{under constraints (1), (2), and (3)}.
\end{aligned}$$

That is, X and X' differ only by the total amount divided, which is a constant. Therefore, their solution matchings are identical. ∎

An efficient way for performing Step 2 in Algorithm 2.1 is necessary to pronounce whether or not a set S of independent ones exists, and if it does, which entries belong to it. We observe that any such S has the property that its ones in $[m'_{ij}]$ can be transformed into a leading diagonal of ones by interchanging some rows. For example, suppose $[m'_{ij}]$ is

$$\begin{bmatrix} 1^* & 2 & 3 \\ 4 & 5 & 1^* \\ 6 & 1^* & 7 \end{bmatrix},$$

then, upon the interchange of rows 2 and 3 it becomes:

$$\begin{bmatrix} 1^* & 2 & 3 \\ 6 & 1^* & 7 \\ 4 & 5 & 1^* \end{bmatrix}.$$

We need exactly n (in this example 3) lines to cross out all the marked ones. No smaller number of lines will suffice.

Because the interchange of rows does not affect the minimum number of crossing lines, it is easy to see that if the minimum number of lines necessary to cross out all the ones equals n, a minimal S can be identified. If the minimum number of lines is strictly less than n, a minimal S is not yet at hand.

To find the smallest possible number of crossing lines, [9] provides the following rules of thumb, which can be repeatedly applied until none of them are satisfied.

1. If there is a row (column) with exactly one uncrossed one, then draw a vertical (horizontal) line through this one.

2. If all rows or columns with ones have two or more uncrossed ones, then choose the row (column) with the least number of uncrossed ones and draw a vertical (horizontal) line through one of the uncrossed ones.

3. Break ties arbitrarily.

The following example illustrates these rules:

EXAMPLE 2.2. Consider the following matrix:

$$\begin{bmatrix} 1 & 1 & 3 \\ 4 & 1 & 1 \\ 6 & 1 & 1 \end{bmatrix}.$$

The first column has exactly one uncrossed one (i.e., m_{11}), by Rule 1, we cross out the first row as follows:

$$\begin{bmatrix} \cancel{1} & \cancel{1} & \cancel{3} \\ 4 & 1 & 1 \\ 6 & 1 & 1 \end{bmatrix}.$$

Now, we must apply Rule 2. By arbitrarily choosing m_{22} and cross out the second row, we obtain:

$$\begin{bmatrix} \cancel{1} & \cancel{1} & \cancel{3} \\ \cancel{4} & \cancel{1} & \cancel{1} \\ 6 & 1 & 1 \end{bmatrix}.$$

Finally, since column 3 has only one uncrossed one (i.e., m_{33}), we apply Rule 1 and cross out the third row:

$$\begin{bmatrix} \cancel{1} & \cancel{1} & \cancel{3} \\ \cancel{4} & \cancel{1} & \cancel{1} \\ \cancel{6} & \cancel{1} & \cancel{1} \end{bmatrix}.$$

This requires three lines to cross out all ones. Therefore, there is a minimal S. ∎

Each application of Step 3 in Algorithm 2.1 will produce at least one more one in the uncrossed entries in the matrix and leave all the ones unchanged, unless they are crossed by two lines. Therefore, Step 3 will always yield a set of n independent ones in a finite number of repetitions.

3. JUSTIFICATION OF OUR ALGORITHM

We justify Algorithm 2.1 by the following theorems.

THEOREM 3.1. *If c, $c < n$, is the minimum number of lines that cover all ones, then each application of Step 3 in Algorithm 2.1 will divide the product of all entries by k^{n^2-cn}.*

PROOF. The net effect of Step 3 can be regarded as the following two steps.

1. Divide all entries by k.
2. Multiply k to all entries covered by a line. Do this line by line.

That will leave all entries which are crossed once unchanged, then multiply k to all numbers which are crossed twice.

Assume the product of all entries is P, then after Step 3 the product of all entries will be $P/k^{n^2} \times k^{cn} = P/k^{n^2-cn}$. ∎

Note that k is always greater than 1. Therefore, by $n > c$, we have $k^{n^2-cn} > 1$, which implies $P/k^{n^2-cn} < P$.

THEOREM 3.2. *Algorithm 2.1 will terminate in finite steps.*

PROOF. By Theorem 3.1, each application of Step 3 will reduce the product of all entries by $k^{n^2-cn} > 1$. If the algorithm loops forever, then the product of all entries will be reduced to zero, which is impossible—since all entries are kept to be greater than or equal to one after Step 1. Therefore, the algorithm will terminate in finite steps. ∎

In fact, the algorithm can be performed in $O(n^3)$ for a complete bipartite graph with $2n$ nodes [8].

4. AN APPLICATION OF MPM

In this section, we present an application of the MPM problem.

EXAMPLE 4.1. Suppose we have a tennis team of three netters, named a, b, and c, and we want to pairwise play single games with another team of three netters, named x, y, and z. By analyzing the past records of our netters and opponents, we may obtain the following probability information of each of our netters to beat the opponents.

vs.	x	y	z
a	0.6	0.3	0.8
b	0.5	0.9	0.3
c	0.2	0.7	0.8

For example, the probability for a to beat x is 0.6. Now, suppose we want to find an arrangement for our netters and their corresponding opponents, such that we have the best chance to beat all the opponents.

This problem can be regarded as an MPM problem by transforming it to the complete bipartite graph $G = (\{a, b, c\} \cup \{x, y, z\}, E)$ with the entries in the above table as the weights of the edges. That is, we want to find a matching in G with the largest *product* of weights.

To apply our algorithm, we regard it as a minimization problem by considering the following matrix:

$$\begin{bmatrix} \frac{1}{0.6} & \frac{1}{0.3} & \frac{1}{0.8} \\ \frac{1}{0.5} & \frac{1}{0.9} & \frac{1}{0.3} \\ \frac{1}{0.2} & \frac{1}{0.7} & \frac{1}{0.8} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} & \frac{10}{3} & \frac{5}{4} \\ 2 & \frac{10}{9} & \frac{10}{3} \\ 5 & \frac{10}{7} & \frac{5}{4} \end{bmatrix}.$$

By Step 1 of Algorithm 2.1, we obtain

$$\begin{bmatrix} 1 & \frac{8}{3} & 1 \\ \frac{27}{20} & 1 & 3 \\ 3 & \frac{8}{7} & 1 \end{bmatrix}.$$

By Step 2, we can find 3 independent ones as follows, which are marked by asterisks.

$$\begin{bmatrix} 1^* & \frac{8}{3} & 1 \\ \frac{27}{20} & 1^* & 3 \\ 3 & \frac{8}{7} & 1^* \end{bmatrix}.$$

Therefore, the solution of this example is:

$$\begin{array}{lll} a & \text{versus} & x, \\ b & \text{versus} & y, \quad \text{and} \\ c & \text{versus} & z. \end{array}$$

If we arrange the single games according to this solution, then the probability to beat all the opponents is $0.6 \times 0.9 \times 0.8 = 0.432$. ∎

5. DISCUSSION

In this paper, we define a bipartite matching problem which maximizes the largest possible *product* of weights. The solution for a bipartite matching which maximizes the *sum* of weights is not necessarily the same as that for maximizing the *product* of weights. Fortunately, we show this problem can be easily solved by modifying the *Hungarian method*. Besides, we show an example, which is usually invoked for decision making, to illustrate an application of our problem.

REFERENCES

1. J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, Macmillan Press, NY, (1976).
2. J.E. Hopcroft and R.M. Karp, An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, *SIAM J. Computing* **2** (4), 225–231 (1973).
3. S. Even and R.E. Tarjan, Network flow and testing graph connectivity, *SIAM J. Computing* **4** (4), 507–512 (1975).
4. L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, (1962).
5. D. König, Graph and matrices, *Mat. Fig. Lapok (Hungarian)* **38**, 116–119 (1931).
6. H.W. Kuhn, The Hungarian method for the assignment problem, *Naval Research Logistics Quarterly* **2**, 83–97 (1955).
7. R. Gould, *Graph Theory*, The Benjamin/Cumming Publishing Comp., (1988).
8. C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, pp. 221–226, (1982).
9. L.R. Foulds, *Optimization Techniques: An Introduction*, Springer-Verlag, NY, pp. 87–88, (1981).
10. P. Hall, On representatives of subsets, *J. London Math. Soc.* **10**, 26–30 (1935).