Introduction to matching problems

A. Agnetis*

1 Formulation

Matching problems are among the most important and most studied problems in network optimization, and have various applications, both for modeling real-life problems and as a solution tool for complex optimization problems.

Let $G = (N, A)$ be an undirected graph, with n nodes and m arcs. A *matching* M is a subset of A such that each node of G is incident to at most one arc of M. An arc $(i, j) \in M$ is *matched*, an arc $(i, j) \notin M$ is *free*. A node adjacent to an arc of M is *matched* (and the node j such that $(i, j) \in M$ is its *mate*), whereas a node which is not adjacent to any arc of M is *exposed*.

The simplest (and most studied) problem in this class is to determine a matching of maximum cardinality (where the cardinality of M is the number of arcs composing it). Let us consider an ILP formulation of this problem. Associate to each arc (i, j) a binary variable x_{ij} such that $x_{ij} = 1$ if $(i, j) \in M$ and $x_{ij} = 0$ if $(i, j) \notin M$. Recalling that $\delta(i)$ indicates the set of arcs adjacent to node i, one can formulate the problem as follows.

$$\max \sum_{(i,j)\in A} x_{ij} \tag{1}$$

$$\sum_{(i,j)\in\delta(i)} x_{ij} \leq 1, \qquad i \in N \tag{2}$$

$$x_{ij} \in \{0, 1\}, (i, j) \in A \tag{3}$$

Each constraint (2) expresses the fact that a node may be adjacent to at most one arc of a matching. Hence, there is a constraint for each node, and note that each variable x_{ij} appears in exactly two constraints, i.e., those corresponding to nodes i and j. This allows one to recognize that the coefficeint matrix of the problem is the incident matrix of graph G. As a consequence, if G is bipartite, the problem is indeed a linear programming problem, replacing constraints (3) with linear constraints:

$$x_{ij} \geq 0, \tag{4}$$

*Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche - Università di Siena

Note that constraints (2) imply $x_{ij} \leq 1$ for each (i, j). If the graph is not bipartite, the integrality constraint cannot be relaxed. In fact, consider for instance the complete graph with 3 nodes (also called *triangle*). Clearly, the optimal matching has value 1, since one can pick only one of the three arcs of the graph without violating (2). But if we relax integrality constraints, we get an optimal solution in which all three variables have value $1/2$, and hence the objective function has value $3/2 > 1$. Clearly, such solution makes no sense from the viewpoint of the problem we want to solve.

Note that, obviously, if there exists a matching of cardinality $n/2$, it is optimal since all its nodes are matched. We say that this is a *perfect* matching. In general, a perfect matching may not exist, so the optimal solution value may be smaller than $n/2$. (Note that a perfect matching on a bipartite graph may exist only if both node sets have cardinality $n/2$.)

The maximum matching problem is an unweighted problem. If graph G is weighed on the arcs, one may want to determine the maximum weight matching. In the formulation, one only needs to replace the objective function (1) with

$$\max \sum_{(i,j)\in A} w_{ij} x_{ij} \tag{5}$$

where w_{ij} is the weight of arc (i, j). Since the structure of the constraints in unchanged, also this problem can be solved as an LP, if the graph is bipartite. Another very relevant problem is when G is a complete bipartite graph, and one wants to find a minimum weight (or maximum weight) *perfect matching*. Such problem is known as *assignment problem*, and will be analyzed in Section 4.

2 Definitions and fundamentals

The following important definitions are given with respect to a graph G (not necessarily bipartite) *and* a matching M. A path on the graph G is said *alternating* if it alternately consists of matched and free arcs. An alternating path is called *augmenting* if the initial and the final nodes are exposed, i.e., equivalently, if the first and the last arc of the path are free.

Given two arc sets X and Y, their *symmetric difference* $X \oplus Y$ is the arc set formed by the arcs belonging to either X or Y, but not both, i.e., it is the set $(X - Y) \cup (Y - X)$. (From the logic viewpoint, it is the *exclusive or* of the sets X and Y.)

THEOREM 1 *Let M be a matching on G, and let P be an augmenting path. The symmetric difference $M' = M \oplus P$ is a matching of cardinality $|M| + 1$.*

Proof. Consider a matching M on a graph G, and let P be an augmenting path. We compute the symmetric difference between sets M and P, and let M' be the set obtained. We observe that: (i) the arcs of matching M that were not part of P are unchanged, and therefore can be found also in M'; (ii) the arcs of M which are also part of P are excluded from M', but the arcs of P that were not part of M are inserted in M'. The set M' is again a matching, in fact: for nodes that are not affected by P, nothing has changed, intermediate nodes of P were incident an arc of M and now an arc of $P \setminus M$, the extreme nodes of P were exposed, and now are matched. Since P is an augmenting path with respect to M, if it has length $2k+1$, it will contain k arcs of the matching M and $k+1$ free arcs. Performing the symmetric difference, free arcs join the matching while matched arcs leave, and therefore M' has one arc more than M. $\square$

Hence, as long as an augmenting path exists, one can always obtain a larger matching by performing the symmetric difference between the path and the matching. One may therefore ask whether is it possible to characterize a maximum matching similarly to what done for maximum flows. The answer is positive.

THEOREM 2 *Given a graph $G = (N, A)$, a matching M is maximum if and only if there are no augmenting paths.*

Proof. The "only if" part is obvious, in view of Theorem 1. Let us now show that if there are no augmenting paths, with respect to a matching M, then M is optimal. Suppose by contradiction that, even if there are no augmenting paths, M is not optimal, i.e., there exists a matching $\hat{M}$ such that $|\hat{M}| > |M|$. (Actually, the cardinality of $\hat{M}$ may be even larger than $|M| + 1$.) Consider the arc set $Y = \hat{M} \oplus M$, and let $G' = (N, Y)$ be the graph having the same node set as G but only containing the arcs of Y. Let us consider all possible cases concerning the degrees of the nodes of G'.

(i) If a node is matched with the same mate in both matchings M and $\hat{M}$, it is isolated in G'.

(ii) If a node is matched with different mates in the two matchings M and $\hat{M}$, has degree 2 in G'.

(iii) If a node is matched in M and exposed in $\hat{M}$ or viceversa, in G' it has degree 1.

(iv) If a node is exposed in both matchings, it is isolated in G'.

Hence, in G' no node has degree greater than 2. The connected components of G' can be of three different types only: (i) isolated nodes, (ii) paths, consisting of arcs of M and

$\hat{M}$ alternately, (iii) cycles with an even number of arcs, since an odd cycle would have two arcs from the same matching adjacent to the same node, which is impossible. Now, since we are supposing that $|\hat{M}| > |M|$, in Y there must be more arcs from $\hat{M}$ than from M. Where are such surplus arcs? Obviously, not in even cycles, since these contain the same number of arcs from $\hat{M}$ and M. Therefore, there must be at least one path $\tilde{P}$ containing more arcs from $\hat{M}$ than from M. Then, $\tilde{P}$ must start and end with two arcs from $\hat{M}$, and in conclusion $\tilde{P}$ is an augmenting path *with respect to matching* M. This contradicts the hypothesis that no augmenting paths exist. □

This theorem immediately suggests a strategy for searching the matching of maximum cardinality: starting from an initial matching (possibly empty), find an augmenting path. Once found, we can increase the matching by one. Then find another, and so on, until no further augmenting path is found. Theorem 2 ensures that when this occurs, the current matching is optimal.

Hence, given a matching M, the problem is therefore reduced to the problem of finding an augmenting path, or certifying that it does not exist, and hence M is optimal. Although Theorem 2 holds for all graphs, finding an augmenting path has different complexity depending on whether the graph is bipartite or not. Here we limit ourselves to the bipartite case.

3 Bipartite matching and maximum flow

The simplest algorithms for finding a matching of maximum cardinality construct a sequence of augmenting paths, using the result of Theorem 2. The search for augmenting paths can be carried out in various ways. In the following, we use a methodology that, while having the same complexity of others, shows an interesting connection with another optimization problem, namely the *maximum flow problem*.

Given the bipartite graph $G = (U, V, A)$, on which we want to find the maximum matching, we can define an oriented graph $G' = (N, A')$, obtained orienting all the arcs of G from U towards V, adding two nodes s and t (called *source* and *sink* respectively), and connecting s with all nodes of U and all nodes of V to t. In other words, the set A' of arcs is obtained adding to A the arcs $\{(s, i)|i \in U\}$ and $\{(i, t)|i \in V\}$. Interpreting graph G' as a flow network, associate a capacity of 1 to all arcs of the graph. It is fairly simple to show that the following holds.

THEOREM 3 *There is a matching M of cardinality z on G, if and only if in G' there exist a feasible flow from s to t of value z.*

Proof. (Only if.) Given a matching M, we can set to 1 the value of flow in all the arcs corresponding to M, and we also set to 1 the flow for all arcs (s, i) and (j, t) where i and j are nodes of G matched in M. It is immediate to verify that all continuity constraints at the nodes are satisfied, since each matched node $i \in U$ has an inflow of 1 (on the arc (s, i)) and outflow also equal to 1 (on the arc that links i to its mate in M). The same applies to matched nodes of V. Exposed nodes in M are not traversed by any flow, so they also respect continuity. Finally, capacity constraints are also trivially satisfied.

(If.) Due to the total unimodularity of the coefficient matrix in the maximum flow problem, we can limit ourselves to considering solutions of the maximum flow problem in which each arc has integer flow, i.e., in our case, either 0 or 1. Given a feasible flow of value z, consider on G' the cut $(S, \bar{S})$ in which $S = \{s\} \cup U$, and consider the arcs (i, j) traversed by flow with $i \in U$ and $j \in V$. No two arcs of such set have endpoints in common, since otherwise such endpoint would have either an inflow or an outflow greater than one, which is forbidden by the capacity constraints. As a consequence, the arcs in G that correspond to the arcs of G' having flow equal to 1 form a matching of cardinality z. □

The result expressed in this theorem can be stated by saying that the maximum matching problem in bipartite graphs *polynomially reduces* to the maximum flow problem. In other words, if we have a polynomial algorithm for maximum flow, we can use it to solve problems of bipartite matching, also in polynomial time.

If one applies Ford-Fulkerson algorithm to graph G', one can observe that each augmenting path for the maximum flow problem corrisponds exactly to an augmenting path for the maximum matching problem. In fact, at each step of Ford-Fulkerson algorithm, in G' an *integer* feasible flow is given, i.e., in our case, each arc of G' is either empty or full. Now, an augmenting path found by Ford-Fulkerson has a very specific structure, namely $P = \{(s, i_1), (i_1, i_2), (i_2, i_3), \ldots, (i_{2h-1}, i_{2h}),\ (i_{2h}, t)\}$, where $i_1, i_3, \ldots, i_{2h-1} \in U$ and $i_2, i_4, \ldots, i_{2h} \in V$. Note that direct arcs $(i_1, i_2), (i_3, i_4), \ldots, (i_{2h-1}, i_{2h})$ (as well as arcs (s, i_1) and (i_{2h}, t)) are empty, while reverse arcs $(i_2, i_3), (i_4, i_5), \ldots, (i_{2h-2}, i_{2h-1})$ are full. If we augment by 1 the flow along this augmenting path, empty arcs in P become full and viceversa. On the whole, the number of full arcs of type $(p, q), p \in U, q \in V$ increases by 1, and so also the cardinality of the corresponding matching.

Finally, observe that the complexity of augmenting path computation is $O(m)$, since it requires a visit of the arcs. Unlike what happens in general maximum flow problems, in our case it will never be necessary finding more than $O(n)$ augmenting paths (since the maximum flow value cannot exceed $n/2$). In conclusion, the following result holds.

THEOREM 4 *The maximum matching problem on a bipartite graph with n nodes and m*

arcs can be solved in $O(mn)$.

4 The assignment problem

Given a complete bipartite graph $G = (U, V, A)$, in which $|U| = n$ and $|V| = n$, and letting c_{ij} be the cost of assigning node $u_i \in U$ to node $v_j \in V$, the *assignment problem* is the following:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{6}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad u_i \in U \tag{7}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad v_j \in V \tag{8}$$

$$x_{ij} \in \{0, 1\}, (u_i, v_j) \in A \tag{9}$$

(Note that unlike in (2), here we write the constraints concerning nodes of U and V separately.) Since the coefficient matrix is totally unimodular, one can relax integrality constraints and restate the problem as:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{10}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad u_i \in U \tag{11}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad v_j \in V \tag{12}$$

$$x_{ij} \geq 0, \quad (u_i, v_j) \in A \tag{13}$$

The problem can then be solved via the simplex method. However, in this case, the simplex method might not be very efficient, due to the fact that, as can be easily observed, all the vertices of the polytope (11 - 13) are strongly degenerate: in every basic feasible solution, only n basic variables (out of $2n$) are nonzero. In the following, we will then see a specific solution algorithm for the assignment problem, leaving out some demonstration details.

In the following, we let C denote the square matrix $n \times n$ containing all costs. We also suppose that $c_{ij} \geq 0$ for all i and j, but as we will see such assumption is not restrictive. The *Hungarian*[1] algorithm is based on two very simple observations.

[1]The authors are in fact the Hungarian mathematicians König and Egervary, and the algorithm was named *Hungarian* by Kuhn, who disclosed it to Western countries in the fifties.

The first observation is that if, in an assignment problem, we add or subtract an amount θ to all the elements of the same row or the same column of C, we obtain an equivalent problem. In fact, the value of the objective function of any feasible solution will vary by the same amount θ, but an optimal assignment remains optimal.

The second consideration is the following. Suppose that some cost coefficients c_{ij} are zero. Then, if we consider *only* the arcs corresponding to these pairs (u_i, v_j), and with these arcs we can form a matching of cardinality n, such matching is certainly optimal.

The basic idea of the Hungarian algorithm is to manipulate the coefficients cost, always through appropriate operations of addition or subtraction of constants in rows and/or columns of C, in order to bring up more and more zeros, until we are able to find an assignment (i.e., a perfect matching) that makes use of those costs only. At the end, the value of the optimal solution will be given by the sum of the original cost coefficients of the arcs of the perfect matching. In the following we describe this algorithm[2]. For the sake of simplicity, we continue using C for the *current* cost matrix, which is initialized with the matrix of original costs.

The algorithm starts by subtracting from each row of C the smallest element, i.e., for each row i:

$$c_{ij} := c_{ij} - \min_j\{c_{ij}\}$$

notice that, after such updating, at least one c_{ij} becomes null. Thereafter, a similar operation is done for the columns, i.e., for each column j we let:

$$c_{ij} := c_{ij} - \min_i\{c_{ij}\}$$

At this point, let E be the set of arcs (u_i, v_j) such that $c_{ij} = 0$, and consider the problem of finding the maximum cardinality matching on the graph $G = (U, V, E)$, which we address using Ford-Fulkerson algorithm, as seen in the previous section 3. If the resulting matching M is perfect, then we have found the optimal solution of the assignment problem. Otherwise, we proceed as follows. During the last iteration of Ford-Fulkerson algorithm, some nodes in U and V are labeled, let them U^* and V^* respectively (these nodes define the cut $(S^*, \bar{S}^*)$ of minimum capacity). Note that at least one node of U^* is exposed, since otherwise the current matching would be perfect. At this point, *mark* the rows and the columns of the matrix C corresponding to labeled nodes, and let θ be defined as:

$$\theta = \min\{c_{ij} |\ i \in U^*, j \notin V^*\}$$

[2]For a thorough explanation of all details, see for instance Papadimitriou, C., Steiglitz, K., Combinatorial Optimization: Algorithms and Complexity, 1982, Prentice-Hall.

Now we alter the values of C by *subtracting* θ from all marked rows, and by *adding* θ to all marked columns. The situation is the one shown in Table 1, in which marked rows and columns are indicated. Of course, at least one of the values in the areas indicated with "$-\theta$" will become zero. By construction, this operation leaves unchanged all zeros *corresponding to the arcs of the current matching*. In fact, note that the only elements of C which undergo a net increase (by θ) are the elements c_{ij} with $i \notin U^*$ and $j \in V^*$. But if the arc $(i, j) \in M$, then, when Ford-Fulkerson algorithm gets to label j, it would also label i (as an inverse, nonempty arc), which would therefore belong to U^*. Hence, calling E' the new set of arcs, we have that the matching M of maximum cardinality for $G = (U, V, E)$ is, however, a feasible matching on graph $G' = (U, V, E')$, and the search for a matching of maximum cardinality on G' can therefore resume from M.

		•	•	
			$+\theta$	
•		-θ		-θ
•				
			$+\theta$	
•		-θ		-θ
			$+\theta$	

Table 1: Modification of cost matrix.

It is important to note that the set of labeled nodes (during the search for a new augmenting path) on the new graph G' *strictly contains* the set of labeled nodes on G. In fact, the only arcs that can disappear going from E to E' are the arcs (u_i, v_j) with $u_i \notin U^*$ and $v_j \in V^*$. However, such arcs are immaterial with respect to the set of labeled nodes, in the sense that the latter remains the same even if one removes all such arcs. On the other hand, however, the updating of matrix C produces, in E', at least one new arc (u_i, v_j) with $u_i \in U^*$ and $v_j \notin V^*$, which therefore has the effect of labeling node v_j, which was not labeled in G. Then, after $O(n)$ steps, one certainly gets to label an exposed node v_j, and an augmenting path is found. At that point, the matching is updated and the procedure starts again, until the current maximum cardinality matching contains n arcs.

As an example, consider the matrix

$$C = \begin{pmatrix} 3 & 3 & 3 & 7 & 8 \\ 5 & 7 & 8 & 5 & 11 \\ 7 & 8 & 11 & 3 & 10 \\ 7 & 10 & 11 & 4 & 4 \\ 6 & 6 & 10 & 8 & 1 \end{pmatrix}$$

subtracting the minimum element from each row, we obtain

$$C = \begin{pmatrix} 0 & 0 & 0 & 4 & 5 \\ 0 & 2 & 3 & 0 & 6 \\ 4 & 5 & 8 & 0 & 7 \\ 3 & 6 & 7 & 0 & 0 \\ 5 & 5 & 9 & 7 & 0 \end{pmatrix}$$

Since each column contains at least one 0, subtracting the minimum element from each column, the matrix does not change. At this point we can compute a maximum cardinality matching with the arcs corresponding to the elements $c_{ij} = 0$. Such matching is $M = \{(u_1, v_3); (u_2, v_1); (u_3, v_4); (u_4, v_5)\}$, of cardinality 4. During the last iteration of Ford-Fulkerson algorithm, the nodes u_5, v_5, u_4, v_4, u_3 are subsequently labeled.

$$C = \begin{matrix} \\ \bullet \\ \bullet \\ \bullet \end{matrix} \begin{matrix} \begin{matrix} & & & \bullet & \bullet \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 4 & 5 \\ 0 & 2 & 3 & 0 & 6 \\ 4 & 5 & 8 & 0 & 7 \\ 3 & 6 & 7 & 0 & 0 \\ 5 & 5 & 9 & 7 & 0 \end{pmatrix} \end{matrix}$$

Now, compute the smallest element (i, j) such that $u_i \in U^*$ and $v_j \notin V^*$. Such value is $\theta = 3$, obtained for the element $(4, 1)$. Updating hte cost matrix, one must therefore substract 3 from rows 3, 4 and 5, and add 3 to columns 4 and 5:

$$C = \begin{pmatrix} 0 & 0 & 0 & 7 & 8 \\ 0 & 2 & 3 & 3 & 9 \\ 1 & 2 & 5 & 0 & 7 \\ 0 & 3 & 4 & 0 & 0 \\ 2 & 2 & 6 & 7 & 0 \end{pmatrix}$$

Arc (u_4, v_1) is added to E (note that (u_2, v_4) disappears). In the new graph there are no augmenting paths, hence the current matching is still of maximum cardinality. However, the sets U^* and V^* have changed:

$$C = \begin{matrix} \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{matrix} \begin{matrix} \begin{matrix} & \bullet & & \bullet & \bullet \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 7 & 8 \\ 0 & 2 & 3 & 3 & 9 \\ 1 & 2 & 5 & 0 & 7 \\ 0 & 3 & 4 & 0 & 0 \\ 2 & 2 & 6 & 7 & 0 \end{pmatrix} \end{matrix}$$

clearly now $\theta = 2$, which is obtained for the arcs $(u_2, v_2), (u_3, v_2)$ and (u_5, v_2). All such arcs are therefore added to E (while arc (u_1, v_1) disappears). The new C is

$$C = \begin{pmatrix} 2 & 0 & 0 & 9 & 10 \\ 0 & 0 & 1 & 3 & 9 \\ 1 & 0 & 3 & 0 & 7 \\ 0 & 1 & 2 & 0 & 0 \\ 2 & 0 & 4 & 7 & 0 \end{pmatrix}$$

at this point, starting from the same matching, an augmenting path exists consisting of the single edge (u_5, v_2), so one can increase the cardinality of the matching to the value 5. In conclusion, the optimal solution is $x_{13} = 1, x_{21} = 1, x_{34} = 1, x_{45} = 1, x_{52} = 1$ and all other variables are $x_{ij} = 0$. The value of the optimal solution is 21.

We briefly discuss the complexity of the Hungarian algorithm. Whenever the Ford-Fulkerson algorithm "stops", i.e., no further augmenting path exists with the current arc set, it is necessary to update the cost matrix C. The result of this update, as we have seen, is an enlargement of the sets U and V^*. Since the labeling process can resume from the current labeling, in $O(n)$ updates a new augmenting path is found, and therefore to the cardinality of the matching increases by 1. If the update of C is performed in $O(n^2)$, the new path is determined in $O(n^3)$ and therefore the complexity of the algorithm is $O(n^4)$. Actually, using suitable data structures, it is possible to lower the complexity to $O(n^3)$.