Massachusetts Institute of Technology
18.433: Combinatorial Optimization
Michel X. Goemans

Handout 4
September 13, 2005

Lecture notes on bipartite matching

We start by introducing some basic graph terminology. A *graph* $G = (V, E)$ consists of a set V of *vertices* and a set E of pairs of vertices called *edges*. For an edge $e = (u, v)$, we say that the *endpoints* of e are u and v; we also say that e is *incident* to u and v. A graph $G = (V, E)$ is *bipartite* if the vertex set V can be partitioned into two sets A and B (*the bipartition*) such that no edge in E has both endpoints in the same set of the bipartition. A matching $M \subseteq E$ is a collection of edges such that every vertex of V is incident to at most one edge of M. If a vertex v has no edge of M incident to it then v is said to be *exposed* (or unmatched). A matching is *perfect* if no vertex is exposed; in other words, a matching is perfect if its cardinality is equal to $|A| = |B|$.

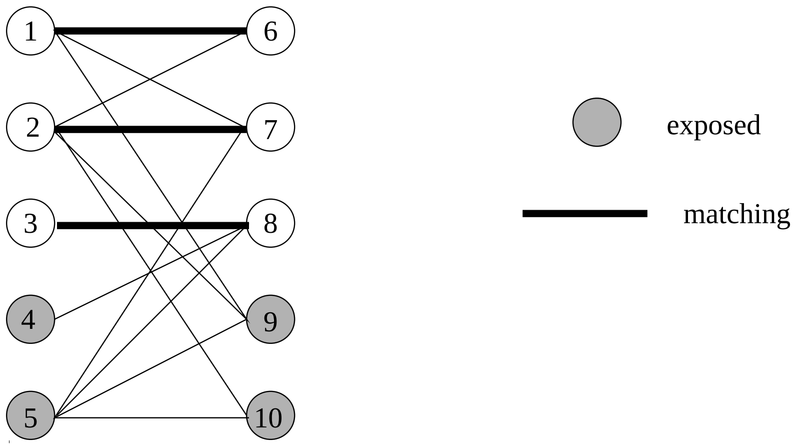

Figure 1: Example. The edges $(1, 6)$, $(2, 7)$ and $(3, 8)$ form a matching. Vertices 4, 5, 9 and 10 are exposed.

We are interested in the following two problems:

Maximum cardinality matching problem: Find a matching M of maximum size.

Minimum weight perfect matching problem: Given a cost c_{ij} for all $(i, j) \in E$, find a perfect matching of minimum cost where the cost of a matching M is given by $c(M) = \sum_{(i,j)\in M} c_{ij}$. This problem is also called the *assignment problem*.

Similar problems (but more complicated) can be defined on non-bipartite graphs.

1 Maximum cardinality matching problem

Before describing an algorithm for solving the maximum cardinality matching problem, one would like to be able to prove optimality of a matching (without reference to any algorithm).

For this purpose, one would like to find upper bounds on the size of any matching and hope that the smallest of these upper bounds be equal to the size of the largest matching. This approach of the problem is identical to the approach we used to derive duality for linear programming. As we'll see, in this case, the dual problem has a strong *combinatorial* flavor.

A *vertex cover* is a set C of vertices such that all edges e of E are incident to at least one vertex of C. In other words, there is no edge completely contained in $V - C$ (we use $-$ to denote the difference of two sets). Clearly, the size of any matching is at most the size of any vertex cover. This follows from the fact that, given any matching M, a vertex cover C must contain at least one of the endpoints of each edge in M. We have just proved *weak duality*: The maximum size of a matching is at most the minimum size of a vertex cover. As we'll prove later in these notes, equality in fact holds:

Theorem 1 (König) *For any bipartite graph, the maximum size of a matching is equal to the minimum size of a vertex cover.*

We shall prove this theorem algorithmically, by describing an efficient algorithm which simultaneously gives a maximum matching and a minimum vertex cover. König's theorem gives a *good characterization* of the problem, namely a simple proof of optimality. In the example above, one can prove that the matching $(1,9)$, $(2,6)$, $(3,8)$ and $(5,7)$ is of maximum size since there exists a vertex cover of size 4. Just take the set $\{1,2,5,8\}$.

The natural approach to solving this cardinality matching problem is to try a *greedy* algorithm: Start with any matching (e.g. an empty matching) and repeatedly add disjoint edges until no more edges can be added. This approach, however, is not guaranteed to give a maximum matching. We will now present an algorithm that does work, and is based on the concepts of *alternating paths* and *augmenting paths*. A path is simply a collection of edges $(v_0,v_1),(v_1,v_2),\ldots,(v_{k-1},v_k)$ where the v_i's are distinct vertices. A path can simply be represented as v_0-v_1-$\ldots$-v_k.

Definition 1 *An* **alternating path with respect to** M *is a path that alternates between edges in* M *and edges in* $E - M$.

Definition 2 *An* **augmenting path with respect to** M *is an alternating path in which the first and last vertices are exposed.*

In the above example, the paths 4-8-3, 6-1-7-2 or 5-7-2-6-1-9 are alternating, but only the last one is augmenting. Notice that an augmenting path with respect to M which contains k edges of M must also contain exactly $k+1$ edges not in M. Also, the two endpoints of an augmenting path must be on different sides of the bipartition. The most interesting property of an augmenting path P with respect to a matching M is that if we set $M' = M \triangle P \equiv (M-P) \cup (P-M)$, then we get a matching M' and, moreover, the size of M' is one unit larger than the size of M. That is, we can form a larger matching M' from M by taking the edges of P not in M and adding them to M' while removing from M' the edges in M that are also in the path P. We say that we have *augmented* M *along* P.

The usefulness of augmenting paths is given in the following theorem.

Theorem 2 *A matching M is maximum if and only if there are no augmenting paths with respect to M.*

Proof: (By contradiction)

($\Rightarrow$) Let P be some augmenting path with respect to M. Set $M' = M \bigtriangleup P$. Then M' is a matching with cardinality greater than M. This contradicts the maximality of M.

($\Leftarrow$) If M is not maximum, let M^* be a maximum matching (so that $|M^*| > |M|$). Let $Q = M \bigtriangleup M^*$. Then:

- Q has more edges from M^* than from M (since $|M^*| > |M|$ implies that $|M^* - M| > |M - M^*|$).
- Each vertex is incident to at most one edge in $M \cap Q$ and one edge $M^* \cap Q$.
- Thus Q is composed of cycles and paths that alternate between edges from M and M^*.
- Therefore there must be some path with more edges from M^* in it than from M (all cycles will be of even length and have the same number of edges from M^* and M). This path is an augmenting path with respect to M.

Hence there must exist an augmenting path P with respect to M, which is a contradiction. $\triangle$

This theorem motivates the following algorithm. Start with any matching M, say the empty matching. Repeatedly locate an augmenting path P with respect to M, augment M along P and replace M by the resulting matching. Stop when no more augmenting path exists. By the above theorem, we are guaranteed to have found an optimum matching. The algorithm terminates in μ augmentations, where μ is the size of the maximum matching. Clearly, $\mu \leq \frac{n}{2}$ where $n = |V|$.

In the example, one would thus augment M along an augmenting path, say 5-7-2-6-1-9, obtain the matching $(1, 9)$, $(2, 6)$, $(3, 8)$ and $(5, 7)$, and then realize that no more augmenting paths can be found.

The question now is how to decide the existence of an augmenting path and how to find one, if one exists. These tasks can be done as follows. Direct edges in G according to M as follows : An edge goes from A to B if it does not belong to the matching M and from B to A if it does. Call this directed graph D.

Claim 3 *There exists an augmenting path in G with respect to M iff there exists a directed path in D between an exposed vertex in A and an exposed vertex in B.*

Exercise: Prove this claim.

This gives an $O(m)$ algorithm (where $m = |E|$) for finding an augmenting path in G. Let A^* and B^* be the set of exposed vertices w.r.t. M in A and B respectively. We can simply attach a vertex s to all the vertices in A^* and do a depth-first-search from s till we hit a vertex in B^* and then trace back our path.

Thus the overall complexity of finding a maximum cardinality matching is $O(nm)$. This can be improved to $O(\sqrt{n}m)$ by augmenting along several augmenting paths simultaneously.

If there is no augmenting path with respect to M, then we can also use our search procedure for an augmenting path in order to construct an optimum vertex cover. Consider the set L (for Labelling) of vertices which can be reached by a directed path from an exposed vertex in A.

Claim 4 *When the algorithm terminates,* $C^* = (A - L) \cup (B \cap L)$ *is a vertex cover. Moreover,* $|C^*| = |M^*|$ *where* M^* *is the matching returned by the algorithm.*

This claim immediately proves König's theorem.
Proof: We first show that C^* is a vertex cover. Assume not. Then there must exist an edge $e = (a, b) \in E$ with $a \in A \cap L$ and $b \in (B - L)$. The edge e cannot belong to the matching. If it did, then b should be in L for otherwise a would not be in L. Hence, e must be in $E - M$ and is therefore directed from A to B. This therefore implies that b can be reached from an exposed vertex in A (namely go to a and then take the edge (a, b)), contradicting the fact that $b \notin L$.

To show the second part of the proof, we show that $|C^*| \leq |M^*|$, since the reverse inequality is true for any matching and any vertex cover. The proof follows from the following observations.

1. No vertex in $A - L$ is exposed by definition of L,
2. No vertex in $B \cap L$ is exposed since this would imply the existence of an augmenting path and, thus, the algorithm would not have terminated,
3. There is no edge of the matching between a vertex $a \in (A - L)$ and a vertex $b \in (B \cap L)$. Otherwise, a would be in L.

These remarks imply that every vertex in C^* is matched and moreover the corresponding edges of the matching are distinct. Hence, $|C^*| \leq |M^*|$. △
Exercise: An edge cover of a graph $G = (V, E)$ is a subset of R of E such that every vertex of V is incident to at least one edge in R. Let G be a bipartite graph with no isolated vertex. Show that the cardinality of the minimum edge cover R^* of G is equal to $|V|$ minus the cardinality of the maximum matching M^* of G. Give an efficient algorithm for finding the minimum edge cover of G.

2 Minimum weight perfect matching

By assigning infinite costs to the edges not present, one can assume that the bipartite graph is complete. The minimum cost (weight) *perfect* matching problem is often described by the following story: There are n jobs to be processed on n machines or computers and one would like to process exactly one job per machine such that the total cost of processing the jobs is minimized. Formally, we are given costs c_{ij} for every $i \in A, j \in B$ and the goal is to find a perfect matching M minimizing $\sum_{(i,j)\in M} c_{ij}$.

In these notes, we present an algorithm for this problem which is based upon linear programming, and we will take this opportunity to illustrate several important concepts in linear programming. The first algorithm given for the assignment problem was given by Kuhn [1955], but he showed only finiteness of the algorithm. A refined version was given by Jim Munkres [1957], and showed a polynomial running time. An algorithm is polynomial-time if its running time (the number of basic operations to run it) is upper bounded by a polynomial in the *size of the input* (i.e. the number of bits needed to represent the input). Munkres' analysis even shows that the algorithm is *strongly polynomial*, and this means that the running time is polynomial in the *number of numbers* involved (i.e. does not depend on the size of the costs c_{ij}). In this algorithm, the number of operations is upper bounded by $O(n^3)$ where $n = |V|$.

The algorithm is often called the Hungarian method, as it relies on ideas developed by Hungarians including König.

We start by giving a formulation of the problem as an *integer program*. We first need to associate a point to every matching. For this purpose, given a matching M, let its *incidence vector* be x where $x_{ij} = 1$ if $(i,j) \in M$ and 0 otherwise. One can formulate the minimum weight perfect matching problem as follows:

$$
\begin{array}{lll}
\text{Min} & \displaystyle\sum_{i,j} c_{ij}x_{ij} & \\
\text{subject to:} & & \\
& \displaystyle\sum_{j} x_{ij} = 1 & i \in A \\
& \displaystyle\sum_{i} x_{ij} = 1 & j \in B \\
& x_{ij} \geq 0 & i \in A, j \in B \\
& x_{ij} \text{ integer} & i \in A, j \in B.
\end{array}
$$

This is not a linear program, but a so-called integer program. Notice that any solution to this integer program corresponds to a matching and therefore this is a valid formulation of the minimum weight perfect matching problem in bipartite graphs.

Consider now the linear program (P) obtained by dropping the integrality constraints:

$$
(P) \qquad
\begin{array}{lll}
\text{Min} & \displaystyle\sum_{i,j} c_{ij}x_{ij} & \\
\text{subject to:} & & \\
& \displaystyle\sum_{j} x_{ij} = 1 & i \in A \\
& \displaystyle\sum_{i} x_{ij} = 1 & j \in B
\end{array}
$$

$$x_{ij} \geq 0 \qquad\qquad i \in A, j \in B.$$

This is the linear programming *relaxation* of the above integer program. In a linear program, the variables can take fractional values and therefore there are many feasible solutions to the set of constraints above which do not correspond to matchings. But we only care about the *optimum* solutions. The set of feasible solutions to the constraints in (P) forms a *polytope*, and when we optimize a linear constraint over a polytope, the optimum will be attained at one of the "corners" or *extreme points* of the polytope.

In general, even if all the coefficients of the constraint matrix in a linear program are either 0 or 1, the extreme points of a linear program are not guaranteed to have all coordinates integral (this is of no surprise since the general integer programming problem is NP-hard, while linear programming is polynomially solvable). As a result, in general, there is no guarantee that the value Z_{IP} of an integer program is equal to the value Z_{LP} of its LP relaxation. However, since the integer program is *more* constrained than the relaxation, we always have that $Z_{IP} \geq Z_{LP}$, implying that Z_{LP} is a lower bound on Z_{IP} for a minimization problem. Moreover, if an optimum solution to a linear programming relaxation is integral then it must also be an optimum solution to the integer program.

Exercise: Prove this last claim.

Exercise: Give an example of an integer program where $Z_{IP} \neq Z_{LP}$.

However, in the case of the perfect matching problem, the constraint matrix has a very special form and one can show the following very important result.

Theorem 5 *Any extreme point of* (P) *is a 0-1 vector and, hence, is the incidence vector of a perfect matching.*

Because of the above theorem, the polytope

$$\begin{aligned} P \quad = \quad \{x : \quad & \sum_j x_{ij} = 1 && i \in A \\ & \sum_i x_{ij} = 1 && j \in B \\ & x_{ij} \geq 0 && i \in A, j \in B\} \end{aligned}$$

is called the *bipartite perfect matching polytope.*

To demonstrate the beauty of matchings, we shall give two completely different proofs of this result, one purely algorithmic and one purely algebraic. The algebraic proof is related to the notion of *totally unimodularity* and is presented in Section 3.

To prove it algorithmically, we describe an algorithm for solving the minimum weight perfect matching problem. The algorithm is "primal-dual". To explain what this means, we need to introduce the notion of duality of linear programs, and let's do it in the specific case of our bipartite matching problem. Suppose we have values u_i for $i \in A$ and v_j for $j \in B$

such that $u_i + v_j \leq c_{ij}$ for all $i \in A$ and $j \in B$. Then for any perfect matching M, we have that

$$\sum_{(i,j)\in M} c_{ij} \geq \sum_{i\in A} u_i + \sum_{j\in B} v_j. \tag{1}$$

Thus, $\sum_{i\in A} u_i + \sum_{j\in B} v_j$ is a *lower bound* on the cost of the minimum cost perfect matching (for bipartite graphs). To get the best lower bound, we would like to maximize this quantity, and therefore we obtain another linear program

$$\begin{array}{lll} & \text{Max} \quad \sum_{i\in A} u_i + \sum_{j\in B} v_j & \\ & \text{subject to:} & \\ (D) & \quad u_i + v_j \leq c_{ij} & i \in A, j \in B. \end{array}$$

This is called the *dual* linear program (D), and it can be shown to be dual to (P) in the sense of linear programming duality. The dual constraints can be interpreted as $w_{ij} \geq 0$ where $w_{ij} = c_{ij} - u_i - v_j$.

If, for any instance, we could always find a feasible solution u, v to (D) and a perfect matching M such that we have equality in (1) (i.e. the cost of the perfect matching is equal to the value of the dual solution) then we would know that the matching found is *optimum*. Given a solution u, v to the dual, a perfect matching M would satisfy equality if it contains only edges (i, j) such that $w_{ij} = c_{ij} - u_i - v_j = 0$. This is what is referred to as *complementary slackness*. However, for a given u, v, we may not be able to find a *perfect* matching among the edges with $w_{ij} = 0$.

The algorithm performs a series of iterations. It always maintains a dual feasible solution and tries to find an "almost" primal feasible solution x satisfying complementary slackness. The fact that complementary slackness is imposed is crucial in any primal-dual algorithm. In fact, the most important (and elegant) algorithms in combinatorial optimization are primal-dual. This is one of the most important tool for designing efficient algorithms for combinatorial optimization problems (for problems which, of course, admit such efficient solutions).

More precisely, the algorithm works as follows. It first starts with any dual feasible solution, say $u_i = 0$ for all i and $v_j = \min_i c_{ij}$ for all j. In a given iteration, the algorithm has a dual feasible solution (u, v) or say (u, v, w). Imposing complementary slackness means that we are interested in matchings which are subgraphs of $E = \{(i, j) : w_{ij} = 0\}$. If E has a perfect matching then the incidence vector of that matching is a feasible solution in (P) and satisfies complementary slackness with the current dual solution and, hence, must be optimal. To check whether E has a perfect matching, one can use the cardinality matching algorithm developed earlier in these notes. If the maximum matching output is not perfect then the algorithm will use information from the optimum vertex cover C^* to update the dual solution in such a way that the value of the dual solution increases (we are maximizing in the dual).

In particular, if L is as in the previous section then there is no edge of E between $A \cap L$ and $B - L$. In other words, for every $i \in (A \cap L)$ and every $j \in (B - L)$, we have $w_{ij} > 0$.

Let

$$\delta = \min_{i \in (A \cap L), j \in (B - L)} w_{ij}.$$

By the above argument, $\delta > 0$. The dual solution is updated as follows:

$$u_i = \begin{cases} u_i & i \in A - L \\ u_i + \delta & i \in A \cap L \end{cases}$$

and

$$v_j = \begin{cases} v_j & i \in B - L \\ v_j - \delta & j \in B \cap L \end{cases}$$

One easily check that this dual solution is feasible, in the sense that the corresponding vector w satisfies $w_{ij} \geq 0$ for all i and j. What is the value of the new dual solution? The difference between the values of the new dual solution and the old dual solution is equal to:

$$\delta(|A \cap L| - |B \cap L|) = \delta(|A \cap L| + |A - L| - |A - L| - |B \cap L|) = \delta(\frac{n}{2} - |C^*|),$$

where A has size $n/2$ and C^* is the optimum vertex cover for the bipartite graph with edge set E. But by assumption $|C^*| < \frac{n}{2}$, implying that the value of the dual solution strictly increases.

One repeats this procedure until the algorithm terminates. At that point, we have an incidence vector of a perfect matching and also a dual feasible solution which satisfy complmentary slackness. They must therefore be optimal and this proves the existence of an *integral* optimum solution to (P). Since, by carefully choosing the cost function, one can make any extreme point be the unique optimum solution to the linear program, this proves Theorem 5.

Of course, as some of the readers might have noticed, the proof is not complete yet since one needs to prove that the algorithm indeed terminates. This can be proved by noticing that at least one more vertex of B must be reachable from an exposed vertex of A (and no vertex of B becomes unreachable), since an edge $e = (i, j)$ with $i \in (A \cap L)$ and $j \in B - L$ now has $w_{ij} = 0$ by our choice of δ. This also gives an estimate of the number of iterations. In at most $n/2$ iterations, all vertices of B are reachable or the matching found has increased by at least one unit. Therefore, after $O(n^2)$ iterations, the matching found is perfect. The overall running time of the algorithm is thus $O(n^4)$ since it takes $O(n^2)$ to compute the set L in each iteration. By looking more closely at how vertices get labelled between two increases of the size of the matching, one can reduce the running time analysis to $O(n^3)$.

Exercise: Check that the running time of the algorithm is indeed $O(n^3)$.

Example: Consider the instance given by the following cost matrix defined on a bipartite graph with 5 vertices on each side of the bipartition:

0	2	7	2	3
1	3	9	3	3
1	3	3	1	2
4	0	1	0	2
0	0	3	0	0

Assume that $u^T = (2, 3, 0, -2, 0)$ and $v^T = (-2, 0, 3, 0, 0)$. The set E of edges with $w_{ij} = 0$ corresponds exactly to the set of edges in Figure 1. The maximum cardinality matching algorithm finds the matching $(1, 9)$, $(2, 6)$, $(3, 8)$ and $(5, 7)$, and the set of labelled vertices is $\{3, 4, 8\}$. We compute δ as

$$\delta = \min_{i \in \{3,4\}, j \in \{6,7,9,10\}} w_{ij} = 1$$

corresponding to the edge $(3, 9)$. The new vectors u and v are $u^T = (2, 3, 1, -1, 0)$ and $v^T = (-2, 0, 2, 0, 0)$. The value of the dual solution has increased from 4 units to 5. The corresponding set E now has a perfect matching, namely $(1, 6)$, $(2, 7)$, $(3, 9)$, $(4, 8)$ and $(5, 10)$ of cost 5. Both the matching and the dual solution are optimal.

3 Total unimodularity

The algebraic proof of Theorem 5 relies on the concept of totally unimodularity.

Definition 3 *A matrix A is totally unimodular if every square submatrix of A has determinant $-1, 0$ or $+1$.*

The importance of total unimodularity stems from the following theorem. This theorem gives a subclass of integer programs which are easily solved. A polyhedron P is said to be *integral* if all its vertices or extreme points are integral.

Theorem 6 *Let A be a totally unimodular matrix. Then, for any integral right-hand-side b, the polyhedron*

$$P = \{x : Ax \leq b, x \geq 0\}$$

is integral.

Before we prove this result, two remarks can be made. First, the proof below will in fact show that the same result holds for the polyhedrons $\{x : Ax \geq b, x \geq 0\}$ or $\{x : Ax = b, x \geq 0\}$. In the latter case, though, a slightly weaker condition than totally unimodularity is sufficient to prove the result. Secondly, in the above theorem, one can prove the converse as well: If $P = \{x : Ax \leq b, x \geq 0\}$ is integral for all integral b then A must be totally unimodular.

Proof: Adding slacks, we get the polyhedron $Q = \{(x, s) : Ax + Is = b, x \geq 0, s \geq 0\}$. One can easily show (see exercise below) that P is integral iff Q is integral.

Consider now any bfs of Q. The basis B consists of some columns of A as well as some columns of the identity matrix I. Since the columns of I have only one nonzero entry per column, namely a one, we can expand the determinant of B along these entries and derive that, in absolute values, the determinant of B is equal to the determinant of some square submatrix of A. By definition of totally unimodularity, this implies that the determinant of B must belong to $\{-1, 0, 1\}$. By definition of a basis, it cannot be equal to 0. Hence, it must be equal to ± 1.

We now prove that the bfs must be integral. The non-basic variables, by definition, must have value zero. The vector of basic variables, on the other hand, is equal to $B^{-1}b$. From linear algebra, B^{-1} can be expressed as

$$\frac{1}{\det B} B^{adj}$$

where B^{adj} is the adjunct matrix of B and consists of subdeterminants of B. Hence, both b and B^{adj} are integral which implies that $B^{-1}b$ is integral since $|\det B| = 1$. This proves the integrality of the bfs. △

Exercise: Let $P = \{x : Ax \leq b, x \geq 0\}$ and let $Q = \{(x, s) : Ax + Is = b, x \geq 0, s \geq 0\}$. Show that x is an extreme point of P iff $(x, b - Ax)$ is an extreme point of Q. Conclude that whenever A and b have only integral entries, P is integral iff Q is integral.

In the case of the bipartite matching problem, the constraint matrix A has a very special structure and we show below that it is totally unimodular. This alongs with Theorem 6 proves Theorem 5. Remember that A corresponds to the system

$$\sum_j x_{ij} = 1 \qquad \text{for all } i \in A,$$

$$\sum_i x_{ij} = 1 \qquad \text{for all } j \in B.$$

Theorem 7 *The matrix A is totally unimodular.*

Proof: Consider any square submatrix T of A. We consider three cases. First, if T has a column or a row with all entries equal to zero then the determinant is zero. Secondly, if there exists a column or a row of T with only one $+1$ then by expanding the determinant along that $+1$, we can consider a smaller sized matrix T. The last case is when T has at least two nonzero entries per column (and per row). Given the special structure of A, there must in fact be *exactly* 2 nonzero entries per column. By adding up the rows of T corresponding to the vertices of A and adding up the rows of T corresponding to the vertices of B, one therefore obtains the same vector which proves that the rows of T are linearly dependent, implying that its determinant is zero. This proves the totally unimodularity of A. △

We conclude with a technical remark. One should first remove one of the rows of A before applying Theorem 6 since, as such, it does not have full row rank and this fact was implicitly used in the definition of a bfs. However, deleting a row of A still preserves its totally unimodularity.

Exercise: If A is totally unimodular then A^T is totally unimodular.

Exercise: Use total unimodularity to prove König's theorem.