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

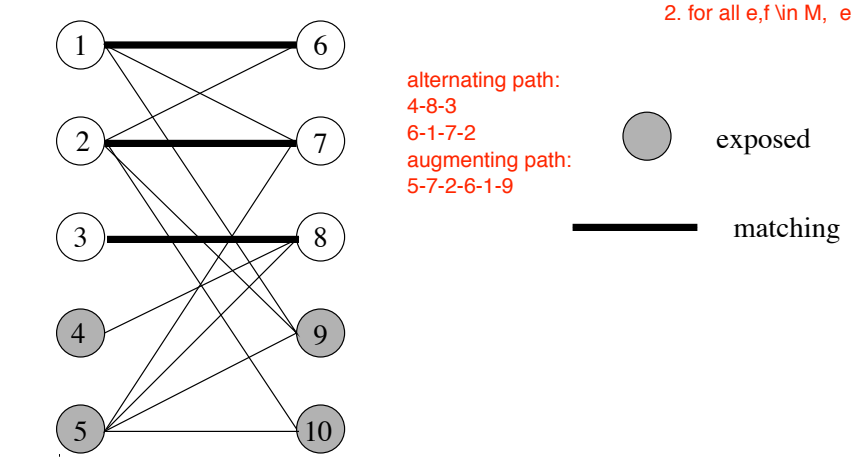
Handout 3
February 9th, 2009

1. Lecture notes on bipartite matching

Matching problems are among the fundamental problems in combinatorial optimization. In this set of notes, we focus on the case when the underlying graph is bipartite.

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 $|A| = |B|$.

matching as a subset of vertex-disjoint edges
1. M is a subset of E
2. for all e,f \in M, e intersect f = emptyset



max matching {(1,9), (2,6), (3,8), (5,7)} min vertex cover {1,2,5,8}

Figure 1.1: 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:

might have no perfect matching

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.

1.1 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 is a *duality* concept that will be ubiquitous in this subject. In this case, the dual problem will itself be a combinatorial optimization problem.

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 both $-$ and $\setminus$ 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 1.1 (König 1931) *For any bipartite graph, the maximum size of a matching is equal to the minimum size of a vertex cover.*

We shall prove this *minmax* relationship 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 (convince yourself). 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-...-v_k.

Definition 1.1 *An* **alternating path with respect to** M *is a path that alternates between edges in* M *and edges in* $E - M$.

Definition 1.2 *An* **augmenting path with respect to** M *is an alternating path in which the first and last vertices are exposed.*

1. since first and last vertex in path exposed
2. since 2k+1 is odd, so must be on opposite sides of bipartition

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

Note in an alternating path, consecutive vertices are in different partition, i.e. p: A-B-A-B-A-B-A-B-A
consecutive edges in the path are in M and N = E - M by definition, the first/last edge always not in M, i.e. {n, e, n, e, …, n}
idea is for edges in M' that P is involved. {n, e, n, e, …} >> {e, n, e, n, …}. all vertices in the path now in some edge in M', additionally, IM'I = IMI + 1

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 1.2 *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 $M' = M \triangle P$. 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 $Q = M \triangle M^*$. Then: by contradiction: idea is to find an augmenting path…

- 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 $M^* \cap Q$.

<= 1 edge because definition of matching. So each vertex <= 2 incident edges. So induced graph of Q consists of disjoint union of paths/cycles

- Thus Q is composed of cycles and paths that alternate between edges from M and M^*.

alternating. 2 edges incident to 1 edge is in M contradicts assumption that M is a matching

- 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 1.3 *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 1-1. Prove claim 1.3.

(=>) exists P: m=(a,b), n, m, …, m, n. first edge unmatched, so goes from A to B. last edge not in M, so it is an edge going from A to B. so first and last vertex on diff partition, i.e. A and B
(<=) directed path consists edges that alternate between M (A -> B) and E\M (B -> A) because graph bipartite and no edges (A->A, B->B), directed path is augmenting path since first/last vertex unmatched

algorithm, simply starts with all vertex in A, see if there is a path start with exposed vertex in A and ends in exposed vertex in B

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.

L: always starting at some unmatched vertex in A and ends up in either A or B that is either matched or unmatched

Claim 1.4 *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.

because some edge always connect some vertex in A to some vertex in B

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, length zero path, consists of a single exposed vertex in A ?
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. idea is edges in M always go (B -> A)

These remarks imply that every vertex in C^* is matched and moreover the corresponding edges of the matching are distinct. Hence, $|C^*| \leq |M^*|$. △

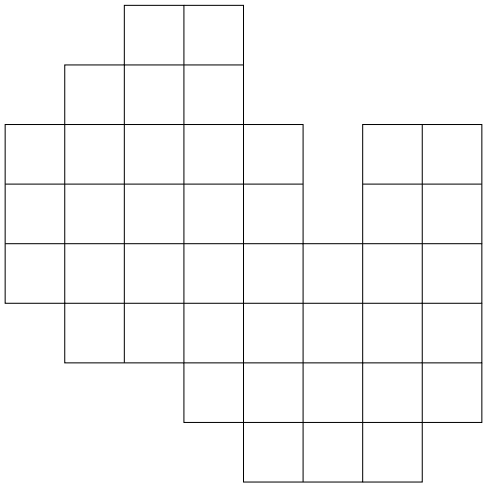
Although the concepts of maximum matchings and minimum vertex covers can be defined also for general (i.e. non-bipartite) graphs, we should remark that König's theorem does not generalize to all graphs. Indeed, although it is true that the size of a maximum matching is *always* at most the minimum size of a vertex cover, equality does not necessarily hold. Consider indeed the cycle C_3 on 3 vertices (the smallest non-bipartite graph). The maximum matching has size 1, but the minimum vertex cover has size 2. We will derive a minmax relation involving maximum matchings for general graphs, but it will be more complicated than König's theorem.

Exercises

Exercise 1-2. 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. Is this true also for non-bipartite graphs?

Exercise 1-3. Show that in any graph $G = (V, E)$ (not necessarily bipartite), the size of *any* *maximal* matching M (i.e. a matching M in which one cannot add an edge while keeping it a matching) is at least half the size of a *maximum* matching M^*.

Exercise 1-4. Can the following figure be tiled by dominoes (a domino being 2 adjacent squares)? Give a tiling or a short proof that no tiling exists.



Exercise 1-5. Consider a bipartite graph $G = (V, E)$ with bipartition (A, B): $V = A \cup B$. Assume that, for some vertex sets $A_1 \subseteq A$ and $B_1 \subseteq B$, there exists a matching M_A covering all vertices in A_1 and a matching M_B covering all vertices in B_1. Prove that there always exists a matching covering all vertices in $A_1 \cup B_1$.

Exercise 1-6. Consider the following 2-person game on a (not necessarily bipartite) graph $G = (V, E)$. Players 1 and 2 alternate and each selects a (yet unchosen) edge e of the graph so that e together with the previously selected edges form a simple path. The first player unable to select such an edge loses. Show that if G has a *perfect* matching then player 1 has a winning strategy.

1.1.1 Hall's Theorem

Hall's theorem gives a necessary and sufficient condition for a bipartite graph to have a matching which saturates (or matches) all vertices of A (i.e. a matching of size $|A|$).

Theorem 1.5 (Hall 1935) *Given a bipartite graph $G = (V, E)$ with bipartition A, B ($V = A \cup B$), G has a matching of size $|A|$ if and only if for every $S \subseteq A$ we have $|N(S)| \geq |S|$, where $N(S) = \{b \in B : \exists a \in S \text{ with } (a, b) \in E\}$.*

Clearly, the condition given in Hall's theorem is necessary; its sufficiency can be derived from König's theorem.

Exercise 1-7. Deduce Hall's theorem from König's theorem.

1.2 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 $c_{ij} \in \mathbb{R} \cup \{\infty\}$ 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. These concepts will be formalized and generalized in a subsequent chapter.

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, and especially König and Egerváry. In 2006, it was discovered that the method had actually been discovered in the 19th century by Jacobi and this was posthumously published in 1890 in Latin, see
`http://www.lix.polytechnique.fr/~ollivier/JACOBI/bibliographieEn.html`.

We start by giving a formulation of the problem as an *integer program*, i.e. an optimization problem in which the variables are restricted to integer values and the constraints and teh objective function are linear as a function of these variables. 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:

$$\text{Min} \quad \sum_{i,j} c_{ij} x_{ij}$$

for each edge: x_e = 1 if e is in M, 0 o/w

subject to:

$$\sum_j x_{ij} = 1 \qquad i \in A$$
$$\sum_i x_{ij} = 1 \qquad j \in B$$
$$x_{ij} \geq 0 \qquad i \in A, j \in B$$
$$x_{ij} \text{ integer} \qquad i \in A, j \in B.$$

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:

$$\text{Min} \quad \sum_{i,j} c_{ij} x_{ij}$$

subject to:

$$(P) \qquad \sum_j x_{ij} = 1 \qquad i \in A$$
$$\sum_i x_{ij} = 1 \qquad j \in B$$
$$x_{ij} \geq 0 \qquad i \in A, j \in B.$$

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. An extreme point x of a set Q is an element $x \in Q$ which cannot be written as $\lambda y + (1-\lambda)z$ with $0 < \lambda < 1$, $y, z \in Q$ with $y \neq z$. (This will be formalized and discussed in more details when we discuss polyhedral theory.)

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 1-8. Prove this last claim.

if optimal solution to LP relaxation integral, then solution is feasible for IP. Just so happens the solution is also optimal, since otherwise Z_IP >= Z_LP not true for minimization problem

Exercise 1-9. 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 1.6 *Any extreme point of (P) is a 0-1 vector and, hence, is the incidence vector of a perfect matching.*

incidence vector for each edge determines if the edge is in the matching or not

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 here and one purely algebraic in the chapter on polyhedral theory. The algebraic proof is related to the notion of *totally unimodularity.*

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

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

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.

Text

idea is: 1. x feasible for primal 2. y feasible for dual. 3. x,y satisfies complementary slackness
pick 2 of 3 to maintain at all time, and work towards achieving the third. when it happens —> x,y optimal

w_ij = 0 -> x_ij != 0 so (i,j) edge is in the matching

primal's constraint basically enforces incident vector to be a perfect mathing

max matching output not perfect -> E_y does not have perfect matching -> need to change y

no augmenting path -> no edge of E between A cup L and B - L
proof sketch: by contradiction, assume there is such edge, show such edge must be in the match and not in the match

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 complementary 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 1.6.

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 1-10. 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 1.1. 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.

Exercises

Exercise 1-11. Consider a bipartite graph $G = (V, E)$ in which every vertex has degree k (a so-called k-regular bipartite graph). Prove that such a graph always has a perfect matching in two different ways:

1.by using König's theorem,

2.by using the linear programming formulation we have derived in this section.

Exercise 1-12. Using the previous exercise 11, show that the edges of a k-regular bipartite graph G can be partitioned into k matchings (i.e. the number of colors needed to color the edges of a k-regular bipartite graph such that no two edges with a common endpoint have the same color is precisely k).

(Optional: Are this result and the one in the previous exercise also true also for non-bipartite graphs?)

Exercise 1-13. We have shown that there always exists a solution x to the linear program (P) with all components integral. Reprove this result in the following way.

Take a (possibly non-integral) *optimum* solution x^*. If there are many optimum solutions, take one with as few non-integral values x^*_{ij} as possible. Show that, if x^* is not integral, there exists a cycle C with all edges $e = (i, j) \in C$ having a non-integral value x^*_{ij}. Now show how to derive another optimum solution with fewer non-integral values, leading to a contradiction.

Exercise 1-14. In this exercise, you will do a little experiment with the (minimum cost) assignment problem. Take a complete bipartite graph with n vertices on each side of the bipartition, and let us assume that all c_{ij} (for $i, j \in \{1, \cdots, n\}$) are all independent uniform random variables between 0 and 1. Take 5 different values for n (the largest being a few hundreds) and for each compute the *minimum* cost assignment value for 5 instances. Any guess on how this value increases as n goes to infinity. Does it seem to converge? To what value? Surprised? (Yes, you should be, and it is normal that you do not understand why.)

To solve the assignment problem, you can use MATLAB (it is available on athena). To access MATLAB on athena, you need to type:

```
>> add matlab
>> matlab
```

If you have never used MATLAB before, you can find some tutorial on the 18.06 webpage `http://web.mit.edu/18.06/www/`. There is a MATLAB m-file at `http://www-math.mit.edu/~goemans/bghungar.m` which implements the Hungarian method (just put it in the directory from which you run MATLAB). If C is an $n \times n$ matrix, then

```
>> a=bghungar(C)
```

gives a vector a such that $(1, a(1)), (2, a(2)), \cdots$ is the *maximizing* matching. So to get the value of the *minimum* assignment, you can just do:

```
>> a=bghungar(-C);
>> v=sum(diag(C(1:n,a(1:n))))
```