CMPUT 675: Topics in Algorithms and Combinatorial Optimization (Fall 2009)

Lecture 2: Cardinality Matching, Maximum Weighted matching

Lecturer: Mohammad R. Salavatipour **Scriber:** Siamak Ravanbakhsh
Date: Sept 8 and 10, 2009

This lecture continues the discussion on bipartite matching, by analyzing the algorithm for construction of maximum bipartite matching. Afterwards, construction of minimum vertex cover from maximum matching is discussed, and finally we consider generalization to maximum weighted bipartite matching and introduce three different algorithms for it.

1 Maximum Cardinality Bipartite Matching

From previous lecture we came up with the following algorithm for construction of a maximum matching M for graph $G = (A \cup B, E)$:

Maximum Bipartite Matching

```
M ← ∅
while there is an M-augmenting path P do
    let M be MΔP
return M
```

Figure 1: Cardinality bipartite matching algorithm

We proved in the previous lecture that this algorithm is guaranteed to find a maximum matching. The number of iterations for this algorithm is in the order of n (at most $\frac{n}{2}$). A question we have not answered yet is: How to find an M-augmenting path? To find an M-augmenting path P, we construct a digraph $D = (A \cup B, E')$, where E' has a directed edge e' for every edge $e \in E$; e' is directed from B to A if $e \in M$ and copies of edges of $E - M$ are from A to B in D.

Lemma 1.1 *There is an M-augmenting path in G* iff *there is a directed path in D from and exposed node in A to an exposed node in B.*

Proof: An easy exercise. ■

To find such a path we can add an extra node r and connect it to all the exposed nodes of A with directed edgs. Now it is sufficient to use a search algorithm such as Breadth First Search (BFS) starting from r to find a path to an exposed node in B. Since BFS takes $O(m + n)$, the total time-complexity of the matching algorithm will be $O(n(m + n))$. The above lemma, together with Theorem 2.8 in Lecture 1 implies that

Theorem 1.2 *Algorithm of Figure1 returns a maximum matching in a bipartite graph in time $O(n(m+n))$.*

However one may be able to improve the time complexity by finding several disjoint M-augmenting paths at each iteration. The following theorems are due to Hopcraft and Karp (1971).

Theorem 1.3 *Let M and $\hat{M}$ be matchings. If $|M| = r$, $|\hat{M}| = s$ and $s > r$, then $M \Delta \hat{M}$ contains at least $s - r$ vertex-disjoint augmenting paths relative to M.*

Given these M-augmenting paths are disjoint, there exists an M-augmenting path of length $\leq 2\lfloor r/(s-r)\rfloor + 1$.

Theorem 1.4 *Let M be a matching, P a shortest M-augmenting path, and P' a $(M\Delta P)$-augmenting path. Then*

$$|P'| \geq |P| + |P \cap P'|$$

Therefore if we index the order in which an augmentation may happen using shortest augmenting paths we have

Corollary 1.5 *$|P_i| \leq |P_{i+1}|$ and for all i and j such that $|P_i| = |P_j|$, P_i and P_j are vertex disjoint.*

This means that we can augment M with all disjoint M-augmenting paths in each iteration. The following theorem limits the number of necessary iterations.

Theorem 1.6 *Let $|M^*| = s$. The number of distinct integers in the sequence $|P_0|, |P_1|, \ldots, |P_i|, \ldots$ is less than or equal to $2\lfloor\sqrt{s}\rfloor + 2$.*

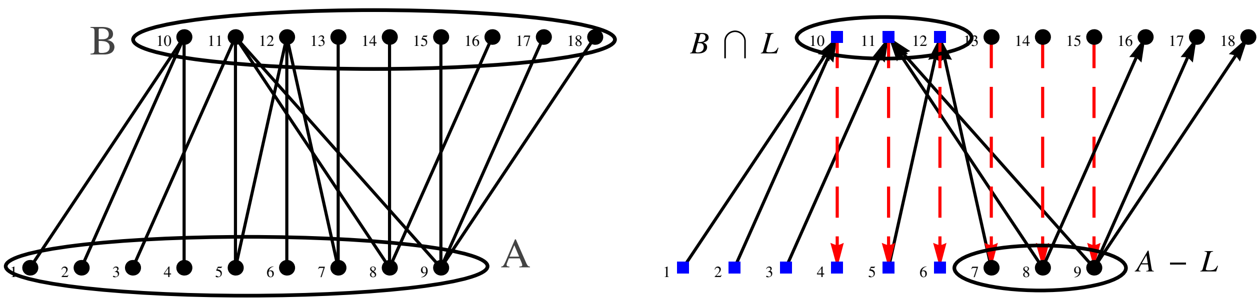
Therefore using single BFS to find all disjoint augmenting paths in each iteration the new algorithm will have complexity of $O((m+n)\sqrt{s}) = O((m+n)\sqrt{n})$.

2 Deriving Minimum Vertex Cover from Maximum Cardinality Matching in Bipartite Graphs

So far we have shown that once the algorithm terminates we have found a maximum matching. In this section we show that we can also find a minimum vertex cover easily. Consider digraph $D = (A \cup B, E')$, where E' is the union of edges of M directed from B to A and edges of $E - M$, directed from A to B. Let L be the set of vertices in D, that can be reached from any exposed node of A (including exposed nodes themselves).

Lemma 2.1 *When the algorithm of figure 1 terminates $C^* = (A - L) \cup (B \cap L)$ is a vertex cover and $|C^*| = |M|$*

Example: Consider the graph G of figure 2. The edges of maximum matching M is in dotted (red) lines and the vertices in L are painted as (blue) squares. Two circles isolated the vertices in the minimum vertex cover. ■



(a) graph G (b) graph D

Figure 2: An example of derivation of vertex-cover from maximum matching. In graph D, square nodes belong to L, and dotted edges belong to M

Proof: If C^* is not a vertex cover, then there is an edge $ab \in E$ that is not covered. Such an edge must have its end-point $a \in A\mathcal{L}$ and $b \in B - L$ because all the other edges are by definition of C^* covered.

We claim that $ab \notin M$. Otherwise, since $a \in L$, can be reached only by ba (directed from b to a by construction of D) from an exposed node, b should be reached from the exposed node as well, which implies $b \in L$, a contradiction with our assumption $b \in B - L$. This means $ab \in E - M$ and so is directed from a to b by construction of D. But since by assumption $a \in L$, existence of ab implies $b \in L$ as well, which is in contradiction with assumption, $b \in B - L$. Since ab is neither in M nor in $E - M$, such edge does not exist and C^* is a vertex cover.

We know show that $|C^*| \leq |M|$ which together with the fact that any vertex cover is lower bounded by any matching implies that equality must hold. First, no vertex in $A - L$ is exposed, because exposed vertices of A are in L by definition. Also no vertex in $B \cap L$ is exposed, because otherwise a path from an exposed node in A to such node exists (since both belong to L) that makes an M-augmenting path. Also, we proved that there is no edge $e = ab$ with $a \in A - L$ and $b \in B \cap L$. Therefore, $|M| \geq |(A - L) \cup (B \cap L)| = |C^*|$, which completes the proof that C^* is a minimum vertex cover. ■

This lemma together with Theorem 1.2 completes the proof of König's theorem form last lecture.

Although matching and vertex cover are still dual problems for general graphs too the min-max theorem we proved (König's theorem) does not hold for general graphs. For example, for cycle of size three, maximum matching has size 1 whereas the min vertex cover has size 2. However, we can still derive a min-max theorem for general graphs that will be seen in the next few lectures.

Another theorem which gives a characterization of bipartite graphs with perfect matching. This theorem can be proved vita König's theorem as well.

Theorem 2.2 (Hall 1935) *A given bipartite graph $G = (A \cup B, E)$ has a matching that saturates all vertices of A iff $\forall S \subseteq A \quad |N(S)| \geq |S|$, where $N(S) = \{b \in B \quad | \quad \exists\, a \in S \quad s.t. \quad ab \in E\}$ is the set of neighbors of a vertex set.*

Exercise 2.3 *Prove this theorem using Königs theorem.*

3 Maximum Weighted Bipartite Matching

Maximum weighted bipartite matching is a generalization of maximum cardinality bipartite matching defined as follows.

Definition 3.1 *Given a bipartite graph $G = (A \cup B, E)$, and edge weights $w_{i,j}$, find a matching of maximum total weight.*

In the following we may assume

- G is a complete bipartite graph. For any non-existing edge add an edge with zero weight.
- $|A| = |B|$. If not, add proper dummy vertices and corresponding zero weight edges.
- $w_{i,j} \geq 0$. If not, choose a sufficiently large W (say $W = \max_{ij} |w_{ij}|$) and add that to all edge weights.

We can alternatively formulate the problem as a minimiation by considering the costs $c_{ij} = W - w_{ij}$ with $W = \max_{ij} w_{ij}$.

Let $W(M) = \sum_{(i,j)\in M} w_{i,j}$ denote the weight of a matching. We present three algorithms for maximum weighted bipartite matching.

3.1 First Algorithm: Negative cycles

The first algorithm is an extension of the algorithm for maximum cardinality bipartite matching (figure 1). Let $D = (A \cup B, E')$, where E' is the union of edges of M directed from B to A and edges of $E - M$ directed from A to B with negative weights, $w_{i,j} \leftarrow -w_{i,j}$. Start from any perfect matching M and build the directed graph D as follows: Let $D = (A \cup B, E')$, where E' is the union of edges of M directed from B to A and edges of $E - M$ directed from A to B with negative weights, $w_{i,j} \leftarrow -w_{i,j}$. Suppose there is a matching M^* with $w(M^*) > w(M)$. Then consider the graph $H = M \cup M^*$. This graph is the union of some even cycles C(note that both are perfect matchings). Furthermore:

$$w(H) = \sum_{c\in C} w(c) = w(M) - w(M^*).$$

Since $w(M^*) > w(M)$, there must be a negative cycle in H. Such negative cycles may be detected using algorithms such as Floyd-Warshal($O(n^3)$) or Bellman-Ford($O(mn)$). Thus we can find a negative cycle in D and improve M by replacing the edges of M with those not in M in the negative cycle. Figure 3.1 summarizes this algorithm.

Maximum Weighted Bipartite Matching, 1^{st} **Algorithm**

> $M \leftarrow$ any perfect matching
> Build graph D from M
> **while** there is a negative cycle C in D**do**
>   let M be $M\Delta C$
>   update D
> **return** M

Theorem 3.2 *A perfect weighted bipartite matching M is maximum* iff *there is no negative cycle in D.*

Proof: It is easy to see that if there is a negative cycle in D then M is not a maximum matching. Now we consider the other way; if there is no negative cycle M is maximum. Suppose there is no negative cycle in D and M is not maximum. Therefore there exists a maximum matching M'. Consider the graph D' a sub-graph of D that only contains the edges in $M \cup M'$. This graph is made up of single disjoint edges ($e \in M \cap M'$) and alternating cycles ($C \subset (M \cup M') - (M \cap M')$). Since $W(M') > W(M)$ and $W(e)$ for $e \in M \cap M'$ is equal for both matchings, we have $W(M' - M) > W(M - M)$, which means there exists negative cycle $C \subset (M \cup M') - (M \cap M') = (M - M') \cup (M' - M)$ in D', and therefore in D, with a weight at most $W(M - M) - W(M' - M)$. ■

This method is not very efficient because there is no guarantee in the amount of improvement in each iteration and when $w_{\max} = \max_{i,j}\{w_{i,j}\}$ is large compared to $w_{\min} = \min_{i,j}\{w_{i,j}\}$ the time complexity becomes $O(mn \times n(w_{max} - w_{min})$, which is not polynomial in the inputs.

Goldberg and Tarjan (1989) showed that if one finds negative cycles with minimum average weight $W(C)/|C|$, the time complexity will be strongly polynomial.

3.2 Second Algorithm: Hungarian method

This method, known as Hungarian method, was first introduced by Kuhn(1955) using Egervàry's idea, showing the finiteness of the assignment. The method was later improved by Munkres(1957) showing its polynomial running time, and later by Iri(1960) and Edmond/Karp(1970). The method progresses in iteration such that in iteration k, it has the maximum weighted matching of size k.

For this, starting from $M = \emptyset$, in each iteration construct digraph $D = (A \cup B, E')$, where E' is the union of edges of M directed from B to A and edges of $E - M$, directed from A to B. We also let the weight of edges $b_j a_i$ from B to A to be negative of their original weight, i.e. $-w_{ij}$.

Let P be an M-augmenting path and $M' = M \Delta P$. Then we have $W(M') = W(M) - W(P \cap M) + W(P - M) = W(M) - l(P)$, where $l(P) = W(P \cap M) - W(P - M)$ is the length of path P. Since $|M'| = |M| + 1$, we obtain a matching whose size is one larger. The idea of this method is to augment M with the shortest augmenting path–*i.e.,* the negative length path with largest absolute value ($l(P)$), in each iteration.

Definition 3.3 *A matching M of size k is extreme if it has the largest weight among those of size k.*

Theorem 3.4 *Augmenting an extreme matching M of size k by a shortest augmenting path produces an extreme matching, M' of size $k+1$.*

Proof: We prove the theorem by induction on k.

- **Base Case:** For $k = 1$, the shortest augmenting path is of length one, and is the edge with the highest original weight. It is obvious that this is an extreme matching for $k = 1$.

- **Induction Step:** Let P be the shortest M-augmenting path. Suppose $M' = M \Delta P$ is not an extreme matching of size $k+1$. Let N be the extreme matching of size $k+1$. We therefore should have.

$$W(N) > W(M') \tag{1}$$

Let $H = (A \cup B, N \cup M)$ be a sub-graph of D with the same weighting. Since $|N| > |M|$, there is an M-augmenting path P' in H. We already know P is the shortest augmenting path in D, and P' is an augmenting path in H (as a subset of D) should be longer than P:

$$l(P) \leq l(P') \tag{2}$$

Consider the matching $N' = N \Delta P'$, obtained by applying P' in reverse to N[1]. Since M is by assumption the maximum matching of size k, we have

$$W(N') \leq W(M) \tag{3}$$

Combining Eq(2) and Eq(3)

$$W(N') - l(P') \leq W(M) - l(P)$$

using definitions of N' and M' we get

$$W(N) \leq W(M')$$

which contradicts our assumption Eq(1). Therefore M' is an extreme matching of size $k+1$.

[1]by applying in 'reverse' we mean P' is not augmenting N but decreasing its size by one. However the symmetric difference operation is performed the same.

■

To find the shortest M-augmenting path we can use Bellman-Ford algorithm which runs in $O(mn)$ time. Since the maximum matching has at most $n/2$ edges, the algorithm is in the order of $O(n^2m)$. This may be improved to $O(n(m + n\log(n)))$. Figure 3.2 summarizes this algorithm. We have proved the following:

Theorem 3.5 *We can solve the weighted bipartite matching problem in $O(mn^2)$ time.*

Maximum Weighted Bipartite Matching, 2^{nd} Algorithm; Hungarian Method

$M \leftarrow \{\max_{i,j} w_{i,j}\}$
Construct Digraph D
while $|M| \leq n/2$ **do**
    find shortest path P in D
    let M be $M\Delta P$
    update D
return M

3.3 Third Algorithm: Primal Dual method

To introduce this algorithm first we will have an overview of the linear programming and the concept of duality in an example.

3.3.1 Duality in Linear Programs

Consider the following constraint minimization, which is a linear program (LP)

$$\begin{aligned} \min \quad & 10x_1 + 6x_2 + 4x_3 & & \\ s.t \quad 2x_1 + x_2 - x_3 & \geq & 2 \\ x_1 + x_2 + x_3 & \geq & 3 \\ x_i & \geq & 0 \end{aligned}$$

Let z^* be the optimum value of this LP. We may ask questions about the lower and upper bounds for z^*

Q: is $z^* \leq 100$? **A:** Since z^* is less than or equal to all feasible solutions, we may answer this question easily by finding a feasible solution for which the condition holds. $x = (1, 1, 1)$ is such an example.

Q: is $z^* \geq 10$? **A:** To answer such questions we should be able to find good lower bounds for z^*. By looking at second constraint we can say $z^* \geq 3$ since the coefficients of all variables in the constraint are smaller than those in the objective function. In fact using different linear combinations of the constraints (with positive weights so that the combination is convex) give us different lower-bounds on z^*. For example just by adding the two constraints we get $3x_1 + 2x_2 \geq 5$. But since $3x_1 + 2x_2 \leq 10x_1 + 6x_2 + 4x_3$ we have $z \geq 5$.

Therefore we may look for a combination y_1 factor of the first constraint and y_2 factor of the second:

$$\begin{aligned} y_1(2x_1 + x_2 - x_3) \geq 2y_1 \\ y_2(x_1 + x_2 + x_3) \geq 3y_2 \end{aligned}$$

that gives us the tightest lower-bound on z^*. That is if we have:

$$y_1(2x_1 + x_2 - x_3) + y_2(x_1 + x_2 + x_3) \leq 10x_1 + 6x_2 + 4x_3 \tag{4}$$

then it follows that

$$z^* \geq 2y_1 + 3y_2.$$

Therefore $2y_1 + 3y_2$ is the value that we want to maximize to get the tightest bound subject to the constraints that the coeficients of x_1, x_2, and x_3 are no larger than those in the objective fucntion. This gives us the following *dual* linear program:

$$\begin{array}{rrcl} \max & 2y_1 + 3y_3 & & \\ s.t. & 2y_1 + y_2 & \leq & 10 \\ & y_1 + y_2 & \leq & 6 \\ & -y1 + y2 & \leq & 4 \\ & y_i & \geq & 0 \end{array} \tag{5}$$

in which the conditions are representing the constraint of inequality Eq(4), for each individual variable x_i. On the other hand for each constraint of the *primal* program (Eq(4)), we have a variable in the dual (Eq(5)). For z_P and z_D as feasible solutions to primal and dual solutions respectively, when the primal is a

Primal-Dual correspondence

Constraints in Primal $\Leftrightarrow$ Variables in Dual
Constraints in Dual $\Leftrightarrow$ Variables in Primal

minimization, we always have $z_P \geq z_D$. The equality holds for optimal solutions.

3.3.2 Duality of Maximum Weighted Bipartite Matching and Minimum Weighted Vertex Cover

Here we show the duality of weighted vertex cover and and weighted maximum matching and exploit this duality to find the optimal solution for corresponding programs.

Recall that a vertex cover is defined as a function $y : V \to \{0, 1\}$ such that for all edge $e = uv$: $y_u + y_v \geq 1$ We can generalize this to weighted graphs.

Definition 3.6 *A weighted vertex cover for a graph with weighted edges is a function $y : V \to \mathbb{R}^+$ such that forall edges e $= uv$: $y_u + y_v \geq w_{uv}$.*

Fact 3.7 *By this definition for any matching M and vertex cover y*

$$\sum_{e \in M} W(e) \quad = \quad W(M) \quad \leq \quad C(y) \quad = \quad \sum_{r \in V} y_r$$

Now we formulate the maximum weighted matching as an Integer Program (IP), which we then relax to a Linear Program (LP). Let $x_{i,j}$ be an indicator variable for each edge ij such that:

$$x_{i,j} = \begin{cases} 1 & \text{if the edge} \quad a_i b_j \in M \\ 0 & \text{otherwise} \end{cases}$$

when the following constraints enforce a matching:

$$\forall \quad a_i, \quad \sum_j x_{i,j} \leq 1$$
$$\forall \quad b_j, \quad \sum_i x_{i,j} \leq 1$$

then for $x_{i,j}$ to represent the maximum matching, we want $\sum_{i,j} x_{i,j} w_{i,j}$ to be maximized. Therefore maximum weighted matching has the following Integer Program formulation:

$$\begin{aligned} \max \quad & \sum_{i,j} x_{i,j} w_{i,j} & & \\ s.t. \quad \forall \quad a_i, \quad & \sum_j x_{i,j} & \leq & \quad 1 \\ \forall \quad b_j, \quad & \sum_i x_{i,j} & \leq & \quad 1 \\ & x_{i,j} & \in & \quad \{0, 1\} \end{aligned} \tag{6}$$

By relaxing the last constraint of IP (6) to $x_{i,j} \geq 0$ we have a linear program. If Z_{IP} denotes the optimal solution of the IP and Z_{LP} denotes the optimal solution of the corresponding LP relaxation then it is easy to see that $Z_{LP} \geq Z_{IP}$. The inequality for general IP/LP's can be strict even if the coefficients of the variables are all integer; the optimum solution of an LP can have fractional values. However, for this specific problem (matching) there is always an optimal solution to the linear program that is integer; so it is also a solution to the integer program. We prove this property in two different ways. One is by giving an algorithm below that finds both an optimal solution to the matching problem and an optimum solution to the vertex cover problem with the same cost. Later on, we'll see a different proof.

The dual program to this linear program is the formulation of weighted vertex cover problem:

$$\begin{aligned} \min \quad & \sum_i y_i & & \\ s.t. \quad \forall e = a_i b_j : \quad & y_{a_i} + y_{b_j} & \geq & \quad w_{i,j} \\ & y_i & \geq & \quad 0 \end{aligned} \tag{7}$$

3.3.3 Primal-Dual Method for Maximum Bipartite Matching

The idea of the *primal-dual* method is to maintain a dual feasible and a primal (not necessarily feasible) solution. At each iteration we try to make the primal solution closer to a feasible solution and also improve the dual (making it closer to an optimum one). In the end we have a primal feasible solution, whose cost is the same as the dual and therefore both are optimal. We will use the following simple lemma.

Lemma 3.8 *For a perfect matching M and a weighted vertex cover y:*

$$C(y) \geq W(M)$$

Also $C(y) = W(M)$ iff *M consists of edges $a_i b_j$ such that $y_i + y_j = w_{i,j}$. In this case M is optimum.*

Exercise 3.9 *Prove this lemma*

The algorithm starts with $M = \emptyset$ and a trivial feasible solution for the weighted vertex cover which is the following one:

$$\forall \quad a_i \in A; \quad y_{a_i} = \max_j \quad w(a_i b_j)$$
$$\forall \quad b_j \in B; \quad y_{b_j} = 0$$

At any iteration of the algorithm we build an equality graph defined below. The *equality graph*, $G_y = (A \cup B, E_y)$ is built based on the y values such that it only contains *tight* edges

$$a_i b_j \in E_y \quad \Leftrightarrow \quad y_i + y_j = w_{i,j}$$

Let us call $y_{a_i} + y_{b_j} - w_{i,j}$, the *excess* of $a_i b_j$.

Observation 3.10 *If M is a perfect matching in G_y then $W(M) = \sum_{a_i \in A} y_{a_i} + \sum_{b_j \in B} y_{b_j}$ and by previous lemma that matching in G is an optimum solution.*

Based on this observation, the goal of the algorithm is to find a perfect matching in the equality graph. For this we update y to make more edges tight to be added to E_y (until it contains a perfect matching) while keeping y a vertex cover. Now we present an algorithm to add an edge to equality graph.

Suppose we are at some iteration of the algorithm and M is a maximum matching in G_y but is not perfect. Construct digraph D as before[2]. Let L be a set of nodes accessible from any exposed node in A. Recall that $C^* = (A - L) \cup (B \cap L)$ is a vertex cover. Therefore there is no edge between $A \cap L$ and $B - L$ (otherwise that edge is not covered by C^*). However, we know that we start with a complete graph G. Thus there are edges in G between $A \cap L$ and $B - L$ but they are not in G_y; which means all those edges have positive excess (i.e. are not tight). We update the y values to make one of these edges go tight. Let

$$\epsilon = \min\{y_{a_i} + y_{b_j} - w_{i,j} \quad s.t., \quad a_i \in A \cap L, \quad b_j \in B - L\}$$

be the minimum excess value of all such edges (these are the edges that could be added to G_y). Then we update vertex y values to tighten the edges with ϵ excess value by defining:

$$y_{a_i} = \begin{cases} y_{a_i} & a_i \in A - L \\ y_{a_i} - \epsilon & a_i \in A \cap L \end{cases}$$

and

$$y_{b_j} = \begin{cases} y_{b_j} & b_j \in B - L \\ y_{b_j} + \epsilon & b_j \in B \cap L \end{cases}$$

Note that by this change every edge that was in G_y remains tight. Also by the choice of ϵ, no edge constraint is going to be violated, so y remains a vertex cover. Furthermore, at least one edge between $A \cap L$ and $B - L$ goes tight and therefore is added to G_y. We can repeat this operation until either G_y has a perfect matching, or there is no edges left between $A\mathcal{L}$ and $B - L$. The latter happens only if both these sets are empty, which implies that we have a perfect matching. Thus, eventually we find a perfect matching in G_y which by the previous lemma corresponds to an optimum matching in G. At this point the solution y is also an optimum vertex cover. Algorithm of figure 3.1 summarizes this method.

Next lecture we will see the analysis of the running time of this algorithm.

Example: Figure 3 demonstrate the expansion of G_y in the final iteration of the algorithm and before adding the edges with $w_{i,j} = 0$[3]. ■

[2] $D = (A \cup B, E')$, where E' is the union of edges of M directed from B to A and edges of $E - M$ directed from A to B.

[3] Since we assumed a complete bipartite graph the edges with zero weight should be added too. These are not shown in the graph of figure 3

Maximum Weighted Bipartite Matching Algorithm; Primal-Dual Method

For each $a_i \in A$ let $y_{a_i} = \max_{b_j \in B} w(a_i b_j)$;
For each $b_j \in B$ let $y_{b_j} = 0$;
Build graph G_y and let M be a maximum matching in G_y
Construct Digraph D
repeat
  let L be the set of nodes (in D) accessible from any exposed node in A.
  Let $\epsilon = \min\{y_{a_i} + y_{b_j} - w(ij) : \quad a_i \in A \cap L, \quad b_j \in B - L\}$
  Decrease y_{a_i} for each $a_i \in A \cap L$ by ϵ and increase y_{b_j} for each $b_j \in B - L$ by ϵ
  Add the tight edges to G_y and recompute matching M.
Until M is a perfect matching.

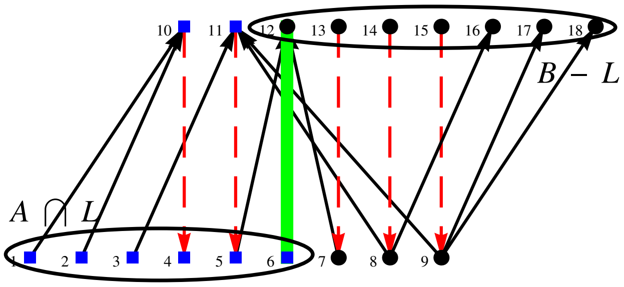


Figure 3: Augmenting the equality graph (with dotted red edges) with an edge (in green) of D connecting $A \cap L$ and $B - L$

References

1. Combinatorial Optimization, Schrijver, (Volume 1) Springer-Verlag, 2003.
2. Combinatorial Optimization: Algorithms and Complexity, by Christos H. Papadimitriou, Kenneth Steiglitz, Dover Publications, 1998.
3. Lecture notes by Lap Chi Lau for Combinatorial Optimization and Approximation Algorithms, 2008. http://www.cse.cuhk.edu.hk/ chi/csc5160/index.html
4. Lecture notes by M.X. Goemans for Combinatorial Optimization, 2007. http://www-math.mit.edu/ goemans/18433S07/18433.html