CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2023**

Lecture 11

October 18, 2023

Instructor: Sepehr Assadi

Topics of this Lecture

1 Bipartite Matching and Vertex Cover Problems

We now see yet another application of LPs to approximating combinatorial optimization problems, this time, a *primal-dual algorithm* for maximum bipartite matching.

Recall that in the **bipartite matching** problem, we are given a bipartite graph $G = (L \sqcup R, E)$ with bipartition L and R of vertices. A *matching* M in G is any collection of edges that do not share any vertices and in the bipartite matching problem, the goal is to find a maximum matching, namely, a one with the largest number of edges.

As we discussed in Lecture 8, this problem can be modeled by the following LP:

$$\begin{aligned} \max_{x \in \mathbb{R}^E} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \forall u \in L: \quad \sum_{e \ni u} x_e \leqslant 1 \\ & \forall v \in R: \quad \sum_{e \ni v} x_e \leqslant 1 \\ & \forall e \in E: \quad x_e \geqslant 0. \end{aligned}$$

Again, as we shown in Lecture 8, the integrality gap of this LP is 1 meaning that we can solve the bipartite matching problem by solving this LP in polynomial time. However, the runtime of such an algorithm, using a generic LP solver, would not necessarily be that low. We are now going to design an *approximation* algorithm for this problem with a much faster running time. We then show that how this algorithm can also be used to obtain a fast algorithm even for the exact version of the problem.

Let us also recall, from Lecture 8, that the *dual* to the bipartite matching LP is the **bipartite vertex cover** problem. Let $G = (L \sqcup R, E)$ be a bipartite graph. A vertex cover in G is a set of vertices S that *cover* all the edges, i.e., any edge has at least one endpoint in S. The following LP for the bipartite vertex cover problem is the dual of the bipartite matching LP:

$$\begin{aligned} \min_{y,z \in \mathbb{R}^L \times \mathbb{R}^R} \quad & \sum_{u \in L} y_u + \sum_{v \in R} z_v \\ \text{subject to} \quad & y_u + z_v \geqslant 1 \qquad \forall\, (u,v) \in E, \\ & y_u, z_v \geqslant 0 \qquad \forall\, u \in L, v \in R. \end{aligned}$$

For simplicity of exposition in later parts, we use y for variables corresponding to vertices in L and z for the ones in R, instead of just using y for all as was done before; this is purely a matter of notation however.

Throughout, we use opt_P to denote the optimum value of the primal LP (which is equal to the maximum matching size) and opt_D to denote the optimum value of the dual LP (which is equal to the minimum vertex cover size). Also, by weak duality, we have $opt_D \geqslant opt_P$ and by strong duality (or in this case, Konig's theorem for bipartite matching/vertex cover), we have $opt_D = opt_P$.

1.1 Warm-Up: A Half-Approximation Algorithm

Let us first see a simple (1/2)-approximation algorithm for these problems using a greedy approach.

Algorithm 1. A (1/2)-approximation algorithm for bipartite matching and vertex cover.

```
1. Let $M = \emptyset$.
2. Iterate over edges $e \in E$ in any arbitrary order:
    (a) If both endpoints of $e$ are unmatched, add $e$ to $M$; otherwise, continue to the next edge.
3. Return $M$ as an approximate matching and $V(M)$, the set of vertices matched by $M$, as an
   approximate vertex cover.
```

There are easier and more direct ways of analyzing this algorithm without using any duality, but let us see what duality can tell us about this problem as a warm-up to our subsequent analysis.

Firstly, we have that

$$|V(M)| = 2 \cdot |M|,$$

as each edge of M contributes two unique vertices to $V(M)$ (uniqueness follows since M is a matching and thus no vertex appears in more than one edge).

Secondly, we have that $V(M)$ is indeed a feasible vertex cover because if there is an edge e not covered by $V(M)$, it means that neither of its endpoints are matched, and thus the algorithm should have picked it in the matching M, a contradiction; see Figure 1 for an illustration.

We can now put these together with what we know from duality theory to have that

$$opt_P \geqslant |M| = \frac{1}{2} \cdot |V(M)| \geqslant \frac{1}{2} \cdot opt_D \geqslant \frac{1}{2} \cdot opt_P,$$

where the first inequality is because M is a feasible matching, the next equality is by what we established above, the next inequality is because $V(M)$ is a feasible vertex cover, and the last one is by the weak duality theorem. Thus, we have $|M| \geqslant 1/2 \cdot opt_P$ and $|V(M)| \leqslant 2 \cdot opt_P \leqslant 2 \cdot opt_D$ and thus both M and $V(M)$ are 1/2-approximation to their respective problems.

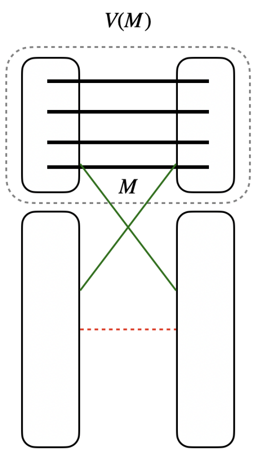

Figure 1: An illustration of the greedy algorithm. The thick (black) edges are M, the solid (green) edges are still possible in G, but the dashed (red) edges cannot exist as the greedy algorithm would have included them. Hence, $V(M)$ is a vertex cover.

2 A Primal-Dual Algorithm for $(1-\varepsilon)$-Approximation

Let us now design our main algorithm for $(1-\varepsilon)$-approximation of both bipartite matching and vertex cover. The general strategy of primal-dual algorithms (with some abuse of their precise definitions in other texts) is as follows. The algorithm starts with a feasible primal solution (here, a matching) and an infeasible dual solution (here, a fractional vertex cover which is infeasible) – throughout the algorithm, we maintain the invariant that the value of the primal is always equal to the value of this infeasible dual solution. Then, we gradually increase the primal value and the corresponding dual solution by focusing on the most violated constraints of the dual (here, the edges that are least covered). Eventually, we show that the dual becomes (approximately) feasible and in that point we can say that the primal-dual pair we maintained are approximately optimal for their respective problems.

The algorithm is formally as follows (see Figure 2 for an illustration of the 'update rule' of the algorithm).

Algorithm 2. A $(1-\varepsilon)$-approximation algorithm for bipartite matching and vertex cover.

1. Let $y_u = 0$ for all $u \in L$, $z_v = 0$ for all $v \in R$, and $M = \emptyset$ initially.
2. Let $U = L$ be the set containing all unmatched vertices in L initially[a].
3. While $U \neq \emptyset$:
   (a) Pick any vertex u from U and remove it from U.
   (b) Find any vertex $v \in \arg\min_{w \in N(u)} z_v$.
   (c) If $z_v = 1$, skip to the next iteration of the while-loop; otherwise, let w be the matched neighbor of v in M (which potentially may not even exist).
   (d) Change the matching M such that u is matched to v instead and w is now unmatched.
   (e) Update $z_v \leftarrow z_v + \varepsilon$ and $y_u = 1 - z_v$ and $y_w = 0$.
4. Output M as the final matching and $(\frac{y}{1-\varepsilon}, \frac{z}{1-\varepsilon})$ as a fractional vertex cover[b].

[a]Although some unmatched vertices later will be removed from U, so U may not contain *all* unmatched vertices.
[b]As we shown in Lecture 8, we can also round this to an integral vertex cover with no decrease in its value.

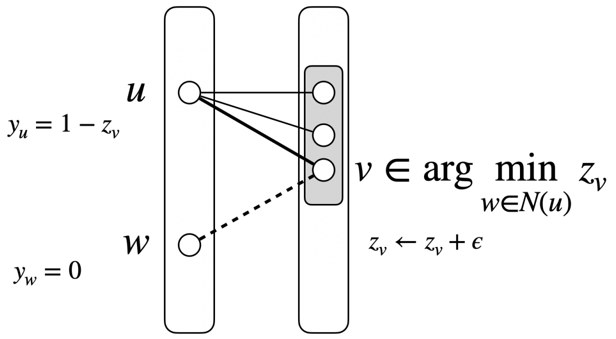
u
$y_u = 1 - z_v$
$v \in \arg\min_{w \in N(u)} z_v$
w
$z_v \leftarrow z_v + \epsilon$
$y_w = 0$

Figure 2: An illustration of the 'update rule' of the algorithm. The neighborhood of u is denoted by the shaded (gray) region. Vertex v has the minimum value z_v in the neighborhood of u. Moreover, v may have been matched to some vertex w (dashed line) but the matching is now updated to include the edge uv (solid line) instead.

Before getting to the analysis of this algorithm, let us provide a quick remark.

Remark. The primal-dual algorithm can also be cast as an **auction algorithm** (cf. [Ber88]). Think of each vertex $u \in L$ as a buyer and each $v \in R$ as an item. Each buyer has a value of 1 for any of the items in its neighborhood but is *unit-demand*, i.e., only wants one item. Then, the above algorithm can be seen as an auction. At each step, M shows a *tentative* assignment, z_v is the *price* of the item v, and y_u is the *utility* or *happiness* of the buyer u (which is equal to 1 (the valuation of the buyer) minus the price it has to pay (z_v)). Thus, at each step, the algorithm finds an unmatched/unhappy buyer, and allows to "trade" for the lowest price item in its neighborhood by increasing the price of that item by ε.

We can indeed analyze the algorithm purely with this point of view (see, e.g., [Ber88]) but for our purpose, it will be easier to stick with the primal-dual interpretation directly. That being said, still seeing y-values as 'happiness' and z-values as 'prices' may provide some more intuition.

2.1 The Analysis

Let us now start the analysis of the algorithm. A key to the analysis of the algorithm is the following *invariants* that hold throughout the algorithm.

Invariant 1. For any vertex $u \in L$:

- If u is matched by M, then $y_u = 1 - z_{M(u)}$ where $M(u) \in R$ is the matched neighbor of u.
- If u is unmatched by M, then $y_u = 0$.

Proof. This follows directly from the update rule in Line (3e) of the algorithm. $\square$

Invariant 2. For any vertex $v \in R$, $z_v > 0$ iff v is matched by M.

Proof. This follows because when we match v for the first time, we increase z_v from zero, and that a matched vertex $v \in R$ always remain matched by the algorithm, although its matched pair in L may change. $\square$

Combining these invariants allows us to prove a key property of the algorithm as a primal-dual algorithm, namely, that M and (y, z) always have the same value.

Claim 1. *At every step of the algorithm,*

$$|M| = \sum_{u \in L} y_u + \sum_{v \in R} z_v.$$

Proof. We have,

$$\begin{aligned}
\sum_{u\in L} y_u + \sum_{v \in R} z_v &= \sum_{(u,v)\in M} y_u + z_v + \sum_{u \in L \setminus V(M)} y_u + \sum_{v \in R\setminus V(M)} z_v \\
&\text{(by pairing up the vertices } u \in L \text{ and } v \in R \text{ that are matched together by } M) \\
&= \sum_{(u,v)\in M} y_u + z_v \\
&\text{(the second sum is zero by Invariant 1 and the third sum is zero by Invariant 2)} \\
&= \sum_{(u,v)\in M} 1 \qquad \text{(as } y_u = 1 - z_v \text{ by Invariant 1)} \\
&= |M|,
\end{aligned}$$

as desired. □

The other next key step is to show that once the algorithm finishes, (y, z) becomes an approximately feasible dual solution.

Claim 2. *At the end of the algorithm, the vectors* $(\frac{y}{1-\varepsilon}, \frac{z}{1-\varepsilon})$ *form a feasible dual solution.*

Proof. To prove the feasibility for the dual LP, given that all $y, z \geqslant 0$ at all times, we only need to show that for every edge $(u, v) \in E$, we have,

$$\frac{y_u}{1-\varepsilon} + \frac{z_v}{1-\varepsilon} \geqslant 1 \qquad \equiv \qquad y_u + z_v \geqslant 1 - \varepsilon.$$

Let us consider the following cases:

- u is unmatched by M: the only way this can happen is if we remove u from U at some point and do not consider it further. But then it means that for every neighbor $w \in N(u)$, we have $z_w = 1$ by the criteria in Line (3c). Thus, for $v \in N(u)$, we have $z_v = 1$ and thus $y_u + z_v \geqslant 1$ in this case.
- u is matched by M to v: by Invariant 1, we have $y_u = 1 - z_v$ and thus $y_u + z_v = 1$ in this case.
- u is matched by M to some other vertex $w \neq v$: At the time that u was matched to w, we had $z_w \leqslant z_v + \varepsilon$ (because w had the minimum z-value in the neighborhood of u and we only increased it by ε after matching it to u). The value of z_w does not change after its last time being matched to u by the construction of the algorithm, and z_v can only increase further (because z-values are increasing in the algorithm). Thus, at the end of the algorithm also, we have $z_w \leqslant z_v + \varepsilon$. But, by Invariant 1, we have $y_u + z_w = 1$ and thus we also have $y_u + z_v \geqslant 1 - \varepsilon$.

This concludes the proof. □

We can now conclude the analysis as follows.

Lemma 3. *Algorithm 2 outputs a $(1 - \varepsilon)$-approximate matching M and a $(1 - \varepsilon)$-approximate fractional vertex cover* $(\frac{y}{1-\varepsilon}, \frac{z}{1-\varepsilon})$.

Proof. We have,

$$opt_D \geqslant opt_P \geqslant |M| = \sum_{u \in L} y_u + \sum_{v \in R} z_v \geqslant (1-\varepsilon) \cdot opt_D \geqslant (1-\varepsilon) opt_P,$$

where the first inequality is weak duality, the next is because M is a feasible primal solution, the next equality is by Claim 1, the next inequality is by Claim 2 since $(\frac{y}{1-\varepsilon}, \frac{z}{1-\varepsilon})$ is a feasible dual solution, and the last is again by weak duality. □

2.2 Runtime

Let us also analyze the runtime of Algorithm 2.

Firstly, see that in each iteration of the while-loop, the z-value of some vertex in R increases by ε, but in total, there can only be $O(n/\varepsilon)$ increments to z-values before they all become 1 and thus the algorithm terminates. This means that the number of iterations is at most $O(n/\varepsilon)$.

Secondly, it is easy to see that we can implement U via a list which allows us to insert and delete to it in $O(1)$ time. Thus, the only step of the while-loop that takes more than $O(1)$ time is Line (3b) for iterating over the neighbors of u. However, this step also takes at most $O(\deg(u)) = O(n)$ time each time.

Putting the above two steps together implies that the algorithm takes $O(n^2/\varepsilon)$ time. However, we are going to make a slightly more careful analysis and implementation of the algorithm to reduce the runtime to $O(m/\varepsilon)$ time.

Faster implementation of Algorithm 2. The faster implementation is as follows. For every vertex $u \in L$, maintain a list $D(u)$ called the *demand list* of u and a value $d(u)$ called the *demand value* of u.

At the beginning of the algorithm, we go over all neighbors of u and place them in $D(u)$ and set $d(u) = 0$. Then, whenever we need to find $v \in \arg\min_{w \in N(u)} z_v$ in Line (3b) we simply iterate over the list $D(u)$ and for each $w \in D(u)$, if $z_w > d(u)$, we remove w from $D(u)$ and if $z_w = d(u)$, we return w as a choice of v. Once the list $D(u)$ becomes empty, we increase $d(u)$ by ε and again insert all neighbors of u back to $D(u)$.

Inductively, and since z-values are only increasing in the algorithm, we have that each vertex returned from $D(u)$ indeed has the minimum price in the neighborhood of u and thus is a valid choice for returning as v. This means that this is a correct implementation of the algorithm still.

Finally, in terms of runtime, we will iterate $O(1/\varepsilon)$ time in total over $D(u)$ as once $d(u)$ becomes 1, we will stop considering u anymore in the algorithm by Line (3c). This means that the total time spent for vertex u throughout the entire algorithm is $O(\deg(u)/\varepsilon)$. Thus, the total runtime of the algorithm is proportional to $\sum_{u \in L} \deg(u)/\varepsilon = m/\varepsilon$, hence, the algorithm takes $O(m/\varepsilon)$ time in total.

3 An Exact Algorithm for Bipartite Matching

Let us conclude this section by showing how to also obtain an exact algorithm for the bipartite matching problem. There are already two easy ways of doing this.

Strategy 1: Run Algorithm 2 with parameter $\varepsilon = 1/(n+1)$, which gives a matching of size

$$(1 - 1/(n+1)) \cdot opt_P > opt_P - 1$$

as $opt_P \leqslant n$ always. But since opt_P only takes integral values, we have that the matching is of size opt_P itself, namely, is an optimal matching.

The runtime of this algorithm is $O(mn)$ time.

Strategy 2: We can simply run an **augmenting path** algorithm (similar to the Ford-Fulkerson algorithm for maximum flows that I hope many of you have seen already). For a bipartite graph G and a matching M, an augmenting path is any path that starts from an unmatched vertex and goes by using one unmatched edge, followed by a matched edge of M, followed by an unmatched edge, followed by a matched edge of M, and so on, until it ends at another unmatched vertex of G. Whenever we find an augmenting path, we can switch the matched edges with unmatched edges in the path to increase our matching size by one. See Figure 3 for an example.

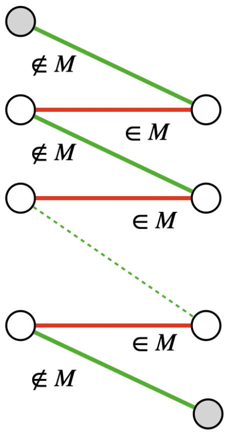

Figure 3: An illustration of the augmenting paths. The gray vertices are unmatched, the green edges are not-matching edges and the red ones are matching edges. By switching red edges to green edges in our matching, we can increase its size by one.

There is one question however; is there always an augmenting path to be found? The following claim shows that yes.

Claim 4. *Let M be any matching in a bipartite graph G which is not a maximum matching. Then, M admits at least one augmenting path.*

Proof. Let M^* be a maximum matching. Consider the graph $M \cup M^*$. The degree of every vertex in this graph is at most two which means this graph is a collection of paths and cycles. Moreover, by bipartiteness, all these cycles are of even length. Now, notice that every even-length cycle and every even-length path has the same number of edges in both M and M^*. But, since $|M^*| > |M|$ by our assumption, we need to have at least one odd-length path also with one more edge from M^* in it. This is now an augmenting path. $\square$

An augmenting path can be found using a simple graph search (DFS or BFS algorithms) by directing unmatched edges in one direction, and matched edges in the other direction in the bipartite graph. This takes $O(m)$ time. We can start with an empty matching M and by Claim 4, as long as we do not have a maximum matching, we can always find an augmenting path to increase its size by one. As size of the matching cannot go beyond n, this means we need to find at most n augmenting paths.

The runtime of this algorithm is also $O(mn)$ time.

Strategy 3: Interestingly, even though both strategies above have $O(mn)$ time in the worst-case, we can combine them in a simple way to obtain a faster algorithm!

Set $\varepsilon = 1/\sqrt{n}$ and run Algorithm 2 first. This takes $O(m\sqrt{n})$ time and at the end of it, we find a matching M of size

$$(1 - 1/\sqrt{n}) \cdot opt_P \geqslant opt_P - \sqrt{n},$$

as $opt_P \leqslant n$ always. This means that at this point we have at most $\sqrt{n}$ unmatched vertices. Switch to running the augmenting path algorithm from now on. As we can only find $O(\sqrt{n})$ augmenting paths (before matching all vertices), this step also takes another $O(m\sqrt{n})$ time.

As a result, the runtime of this combined algorithm is $O(m\sqrt{n})$ time.

Remark. The $O(m\sqrt{n})$ time obtained via the algorithm outlined above matches the runtime of the celebrated *Hopcroft-Karp Algorithm* for bipartite matching [HK73] from 1973, but via a simpler algorithm and analysis.

It is also worth mentioning that while the $O(m\sqrt{n})$ time for bipartite matching was improved over the years for various ranges of parameters, only the very recent breakthroughs on almost-linear time algorithms for maximum flow resulted in completely improving this bound for all ranges of parameters. Discussing this line of work is beyond the scope of our course, and you are referred to [vdBLN$^+$20, CKL$^+$22, vdBCK$^+$23] to learn more about these exciting developments.

References

[Ber88] Dimitri P Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of operations research*, 14(1):105–123, 1988. 4

[CKL$^+$22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022. 8

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. 8

[vdBCK$^+$23] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. *CoRR*, abs/2309.16629, 2023. 8

[vdBLN$^+$20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 919–930. IEEE, 2020. 8