Princeton Univ. F'22 COS 521: Advanced Algorithm Design

Lecture 2: Linear Programs and LP Rounding

Lecturer: *Matt Weinberg* Last updated: September 15, 2022

One of the running themes in this course is the notion of *approximate solutions*. Of course, this notion is tossed around a lot in applied work: whenever the exact solution seems hard to achieve, you do your best and call the resulting solution an approximation. In theoretical work, approximation has a more precise meaning whereby you *prove* that the computed solution is close to the exact or optimum solution in some precise metric.

1 Quick Refresher on Linear Programming

A linear program has a set of variables (in the example below, $x_1, \ldots, x_n$), a linear objective (in the example below, $\vec{c} \cdot \vec{x}$), and a system of linear constraints (in the example below, $A_{ji} \cdot \vec{x} \leq b_j$, for all j, and $x_i \geq 0$ for all i). A linear program in "standard form" therefore takes the following form:

$$\begin{aligned} \max \quad & \sum_i c_i x_i \\ \text{s.t.} \quad & \sum_i A_{ji} x_i \leq b_j, \qquad \forall j \\ & x_i \geq 0, \qquad \forall i. \end{aligned}$$

Recall that it is OK to have variables which aren't constrained to be non-negative, equalities instead of inequalities, min instead of max, etc. (and all such linear programs are equivalent to one written in standard form — if you're unfamiliar with LPs, you may want to prove this as a quick exercise). Linear programs can be solved in weakly polynomial time via the Ellipsoid algorithm (which we'll see later in class). "Weakly polynomial time" means the following:

- You are given as input an n-dimensional vector $\vec{c}$, and an $m \times n$ matrix A. Each entry in $\vec{c}$ and A will be a rational number which can be written as the ratio of two b-bit integers.
- Therefore, the input is of size poly(n, m, b). A weakly polynomial time algorithm is just an algorithm which terminates in time poly(n, m, b) (and the Ellipsoid algorithm is one such algorithm).
- A stronger stance might be to say that the input is really of size poly(n, m), but you acknowledge that of course doing numerical operations on b-bit integers will take time poly(b). A strongly polynomial-time algorithm would be one which performs poly(n, m) numerical operations (and then the algorithm will also terminate in time poly(n, m, b), because each operation terminates in time poly(b). A major (major, major) open problem is whether a strongly poly-time algorithm exists for solving linear programs. Note that the Ellipsoid algorithm does *more numerical operations* if the input numbers have more bits, it's not just that each operation takes longer.

2 Integer Programs

In discrete optimization problems, we are usually interested in finding 0/1 solutions. Using LP one can find *fractional* solutions, where the relevant variables are constrained to take real values in $[0,1]$. Sometimes, we can get lucky: you write an LP relaxation for a problem, and the LP happens to produce a 0/1 solution. Now, you know that this 0/1 solution is clearly optimal: not only is it the best 0/1 solution, it's even the best $[0,1]$ solution. We will see on example of this phenomenon in PSet 1, where we use a linear program to find the minimum s-t cut in a graph.

Another important polynomial-time problem that admits a linear program which exactly solves the integral problem is *max-weight bipartite matching*. Given a bipartite graph $G = ((A,B),E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$ (i.e., the vertices in G can be partitioned into sets A and B and each edge in E is of the form (a,b) for some vertex $a \in A$ and $b \in B$), the max-weight bipartite matching problem is to find a subset of edges $M \subseteq E$ that do not share a vertex while maximizing $\sum_{e \in M} w(e)$. We won't prove it in class but the optimal value of the following linear program returns the max-weight matching:[1]

$$\begin{aligned}
\max \sum_{(a,b)\in E} & w((a,b)) \cdot x_{(a,b)} & \\
& 0 \leq x_{(a,b)} \leq 1 & \forall (a,b) \in E \\
& \textstyle\sum_{b:(a,b)\in E} x_{(a,b)} \leq 1 & \forall a \in A \\
& \textstyle\sum_{a:(a,b)\in E} x_{(a,b)} \leq 1 & \forall b \in B.
\end{aligned}$$

Needless to say, we don't expect this magic to repeat for NP-hard problems. So the LP relaxation yields a fractional solution in general. Then we give a way to *round* the fractional solutions to 0/1 solutions. This is accompanied by a mathematical proof that the new solution is provably approximate.

The rest of the lecture discusses different LP rounding schemes.

3 Deterministic Rounding (Weighted Vertex Cover)

First we give an example of the most trivial rounding of fractional solutions to 0/1 solutions: round variables $< 1/2$ to 0 and $\geq 1/2$ to 1. Surprisingly, this is good enough in some settings.

Definition 1. *The Weighted Vertex Cover Problem is the following:*

- *Input: a graph, $G = (V,E)$ and a weight w_i for each node $i \in V$.*
- *Output: a* vertex cover*, which is a subset $S \subseteq V$ such that every edge $e \in E$ contains at least one vertex of S (that is, there does not exist an $e = (u,v) \in E$ such that $u \notin S$ and $v \notin S$).*

[1] There are (at least) two ways to see this, both of which we won't prove. But these are some buzzwords in case you want to look it up yourself. One way is to use the Birkhoff-Von Neumann Theorem and to treat the fractional matching as a doubly-stochastic matrix, and the integral matching as a permutation matrix. The other way is to consider writing a flow network where the max-flow is equal to the maximum fractional matching, and then using the flow integrality theorem.

- *Goal: Output a set S minimizing $\sum_{i \in S} w_i$.*

We first observe that Weighted Vertex Cover can be solved by an Integer Program.

Observation 1. *The following Integer Program is equivalent to Weighted Vertex Cover.*

$$\begin{array}{lll} \min & \sum_i w_i x_i & \\ & x_i \in \{0,1\} & \forall i \\ & x_i + x_j \geq 1 & \forall \{i,j\} \in E. \end{array}$$

Proof. The first constraint guarantees that every i is either in S ($x_i = 1$) or not in S ($x_i = 0$). The second constraint guarantees that every edge e is covered (because at least one of its endpoints is in S). The objective computes the weight of nodes in S. □

We now want to consider the following LP relaxation of this Integer Program:

$$\begin{array}{lll} \min & \sum_i w_i x_i & \\ & 0 \leq x_i \leq 1 & \forall i \\ & x_i + x_j \geq 1 & \forall \{i,j\} \in E. \end{array}$$

Let OPT_f denote the optimum value of this linear program, and let $\text{VC}_{\min}$ denote the weight of the optimum vertex cover.

Observation 2. *$OPT_f \leq VC_{\min}$.*

Proof. This immediately follows as every feasible solution to the integer program defining $\text{VC}_{\min}$ is also a feasible solution to the LP relaxation. Therefore, the LP can only be better. □

We now consider the following simple rounding algorithm:

Definition 2 (Deterministic VC Rounding)**.** *Solve the LP relaxation. For each i such that $x_i \geq 1/2$, add i to S. For each i such that $x_i < 1/2$, keep i out of S.*

Lemma 1. *Deterministic VC Rounding outputs a vertex cover.*

Proof. By definition of the LP relaxation, we know that $x_i + x_j \geq 1$ for every edge $\{i,j\}$. Therefore, at least one of x_i or x_j is $\geq 1/2$, and therefore at least one of $\{i,j\}$ must be in S. Therefore, every edge is covered. □

Lemma 2. *The weight of the set output by Deterministic VC Rounding is at most $2OPT_f \leq 2\,VC_{\min}$.*

Proof. Every element i of S contributed at least $w_i/2$ to OPT_f, and contributes w_i to the weight of S. OPT_f can only be larger than $\sum_{i \in S} w_i/2$ because maybe other i not in S have $x_i > 0$. □

Thus we have constructed a vertex cover whose cost is within a factor 2 of the optimum cost. In particular, observe that we can guarantee that our vertex cover is a 2-approximation, even though we don't know the quality of the optimum.

Exercise: Show that for the complete graph, Deterministic VC Rounding indeed computes a set of size no better than $2 \cdot \text{OPT}_f$.

Remark: This 2-approximation was discovered a long time ago, and despite myriad attempts we still don't know if it can be improved. Using the so-called PCP Theorems, Dinur and Safra showed (improving a long line of work) that 1.36-approximation is NP-hard. Khot and Regev showed that computing a $(2-\epsilon)$-approximation is UG-hard, which is a new form of hardness popularized in recent years. The bibliography mentions a popular article on UG-hardness.

4 Simple randomized rounding: MAX-2SAT

Simple randomized rounding is as follows: if a variable x_i is a fraction then toss a coin which comes up heads with probability x_i. If the coin comes up heads, make the variable 1 and otherwise let it be 0. The expectation of this new variable is exactly x_i. Furthermore, linearity of expectations implies that if the fractional solution satisfied some linear constraint $c^T x = d$ then the new variable vector satisfies the same constraint *in the expectation*. But, we may need to do more work to understan

Definition 3. *The MAX2SAT Problem is the following:*

- *Input: n boolean variables $x_1, \ldots, x_n$, and m clauses. j clauses are in J_1, and contain a single literal of the form x_i or $\bar{x}_i$. The remaining $m - j$ clauses are in J_2 and are in the form $y \vee z$, where both y and z are equal to some literal or its negation (we are guaranteed that y and z are from different variables).*[2]
- *Output: An assignment of each variable to either TRUE or FALSE.*
- *Goal: Maximize the number of satisfied clauses (i.e. the clauses that evaluate to true).*[3]

Observation 3. *The following Integer Program is equivalent to MAX2SAT. We have a variable z_j for each clause $j \in J_1 \cup J_2$, where the intended meaning is that it is 1 if the assignment decides to satisfy that clause and 0 otherwise. Below, y_{j1} is shorthand for the literal in clause j (Similarly for y_{j2}.)*

$$\begin{aligned}
\max \quad & \sum_{j \in J} z_j & \\
& t_i, f_i \in \{0,1\} & \forall i \\
& t_i = 1 - f_i & \forall i \\
& z_j \leq 1 & \forall j \in J_1 \cup J_2 \\
& y_{j1} \geq z_j & \forall j \in J_1 \\
& y_{j1} + y_{j2} \geq z_j & \forall j \in J_2
\end{aligned}$$

[2] If not, then either both literals are the same, in which case it is just in J_1, or the clause is always true.

[3] Random aside: if instead we wish to ask whether it is possible to satisfy *all* clauses, then there is a simple poly-time algorithm. But satisfying the maximum number of clauses is NP-hard.

Proof. The first constraint guarantees that each x_i is either true ($t_i = 1$) or false ($t_i = 0$). The second guarantees that each clause can be satisfied at most once. The third constraints guarantee that each clause can only be satisfied if at least one of its literals is true. □

Now, we again want to consider the LP relaxation:

$$\begin{aligned}
\max \quad & \sum_{j\in J} z_j & \\
& 1 \geq t_i, f_i \geq 0 & \forall i \\
& t_i = 1 - f_i & \forall i \\
& z_j \leq 1 & \forall j \in J_1 \cup J_2 \\
& y_{j1} \geq z_j & \forall j \in J_1 \\
& y_{j1} + y_{j2} \geq z_j & \forall j \in J_2
\end{aligned}$$

Definition 4 (M2S Randomized Rounding)**.** *The M2S Randomized Rounding algorithm first solves the LP relaxation. Then, independently for each i, it sets variable x_i to true with probability t_i.*

Again, let OPT_f denote the optimal solution to the LP relaxation. We claim that M2S guarantees a 3/4-approximation:

Theorem 3. *The expected number of clauses satisfied by the output of M2S is at least $3OPT_f/4$.*

Proof. We will analyze each clause j separately, and show that clause j is satisfied with probability at least $3z_j/4$. The theorem will then follow by linearity of expecation. We handle the cases of clauses in J_1 and J_2 separately.

Lemma 4. *Let $j \in J_1$. Then the probability that clause j is satisfied in M2S at least z_j.*

Proof. Because $j \in J_1$, it contains only one literal. If that literal is x_i, then x_i is set to true with probability $t_i \geq z_j$. If that literal is $\bar{x}_i$, then x_i is set to false with probability $f_i \geq z_j$. Therefore, the lemma holds. □

Lemma 5. *Let $j \in J_2$. Then the probability that clause j is satisfied in M2S is at least $3z_j/4$.*

Proof. Wlog, say that clause j is $x_r \vee x_s$ (idential reasoning holds if one/both of these variables is a negation, swapping t for f below as necessary). Then the probability that clause j is satisfied is $1 - (1 - x_r)(1 - x_s) = x_r + x_s - x_r x_s \geq x_r + x_s - (x_r + x_s)^2/4$.[4]

Now, consider the case when $x_r + x_s \leq 1$. Then we have:

- $z_j \leq x_r + x_s$ (directly from the LP).
- $x_r + x_s - (x_r + x_s)^2/4 \geq x_r + x_s - (x_r + x_s)/4 = 3(x_r + x_s)/4$.

[4]To see this last inequality, observe that $(x_r+x_s)^2-(x_r-x_s)^2 = 4x_s x_r$, and therefore $x_s x_r \leq (x_r+x_s)^2/4$.

These two facts together imply that the clause is satisfied with probability at least $3z_j/4$.

Consider now the case when $x_r + x_s \geq 1$. Then we have:

- $z_j \leq 1$ (directly from the LP).
- $x_r + x_s - (x_r + x_s)^2/4 \geq 3/4$, as $x_r + x_s \leq 2$.[5]

These two facts together imply that the clause is satisfied with probability at least $3z_j/4$.

We've now shown that for all clauses, the probability it is satisfied is at least $3z_j/4$.

□

This completes the proof, by linearity of expectation. □

Remark: This algorithm is due to Goemans-Williamson, but the original 3/4-approximation is due to Yannakakis. The 3/4 factor has been improved by other methods to 0.94.

5 More Clever Rounding: Job Scheduling

Here, we'll consider a more clever rounding scheme that also starts from an LP relaxation due to Shmoys and Tardos. Consider the problem of scheduling *jobs* on *machines*. That is, there are n jobs and m machines. Processing job i on machine j takes time p_{ij}. Your goal is to finish *all* jobs as quickly as possible: that is, if $x_{ij} = 1$ whenever job i is assigned to machine j (and 0 otherwise), minimize $M(\vec{x}) = \max_j\{\sum_i x_{ij}p_{ij}\}$, $M(\vec{x})$ refers to the *makespan* of $\vec{x}$, and we will keep this definition even when $\vec{x} \in [0,1]^{nm}$ (instead of $\{0,1\}^{nm}$). This lends itself to a natural LP relaxation:

$$\begin{aligned}
\min \quad & T \\
& x_{ij} \in [0,1] \quad \forall i,j \\
& \sum_j x_{ij} \geq 1 \quad \forall i \\
& T \geq \sum_i p_{ij}x_{ij} \quad \forall j
\end{aligned}$$

That is, we want to minimize the maximum load on any machine, subject to every job being assigned (at least) once. Unfortunately, this LP has a huge *integrality gap*. That is, the best fractional solution might be significantly better than the best integral solution. Why? Maybe there's only one job with $p_{1j} = 1$ for all machines j. Then the best fractional solution will set $x_{1j} = 1/m$ for all machines and get $T = 1/m$. But clearly the best integral schedule takes time 1. The problem is that we're asking for too much: if there's a single job that itself takes time $t \gg T$ to process on every machine, we can't possibly hope to get

[5] To see this last claim, take the derivative with respect to $(x_r + x_s)$. The derivative is $1 - (x_r + x_s)/2$, which is 0 at $x_r + x_s = 2$, and positive on $[1, 2]$. Therefore, the minimum on $[1, 2]$ is achieved at $x_r + x_s = 1$, which is $3/4$.

a good approximation to T with an integral schedule. Instead, we'll consider the following modified relaxation, which is parameterized by $t > 0$, and we'll refer to as $LP(t)$.

$$\begin{aligned} \min \quad & T \\ & x_{ij} \in [0,1] \quad \forall i,j \\ & \sum_j x_{ij} \geq 1 \quad \forall i \\ & T \geq \sum_i p_{ij} x_{ij} \quad \forall j \\ & x_{ij} = 0 \quad \forall i,j \text{ such that } p_{ij} > t \end{aligned}$$

The problem with the previous example was that a single job had processing time 1, but $T = 1/m$ and we asked for a new schedule with processing time $O(1/m)$. Instead, we'll ask for one of time $T + t$. Note that if the optimal schedule has total processing time P, then the maximum time it takes to process any job is some $t \leq P$. So if we solve the above LP with this given t, the optimal schedule will be considered, and we'll have $T \leq P$ and $t \leq P$ for a 2-approximation. This is the main idea for why this approach works, but we'll specify everything in more detail below.

Definition 5 (ST Rounding Algorithm)**.** *Given as input a fractional solution $\vec{x}$ to $LP(t)$: For each machine j, let $w_j = \lceil \sum_i x_{ij} \rceil$. Make a bipartite graph with jobs on the left and machines on the right. Make $\lceil w_j \rceil$ copies of the machine j node, call them $j_1, \ldots, j_{w_j}$. Make a single node on the right for each job.*

For each machine j, sort the jobs in decreasing order of p_{ij}, so that $p_{(1)j} \geq p_{(2)j} \cdots \geq p_{(n)j}$. Place edges from jobs to machine j in the following manner:

1. *Initialize current-node $c := 1$. Initialize current-job $i := 1$. Initialize job-weight $w := x_{(1)j}$. Initialize node-weight-remaining $r := 1$.*
2. *While $(i \leq n)$:*
   (a) *If $w \leq r$, add an edge from job (i) to j_c of weight w. Update $r := r - w$, update $i := i + 1$, $w := x_{(i)j}$ (the newly updated i). Keep $c := c$.*
   (b) *Else, add an edge from job (i) to c of weight r. Update $w := w - r$, update $r := 1$, update $c := c + 1$. Keep $i := i$.*

In other words, starting from the slowest jobs, we put edges totalling weight x_{ij} from job i to (possibly multiple) nodes for machine j. We do so in a way such that the slowest jobs are on the earliest-indexed copies, and that each copy has total incoming weight at most 1 (actually all but the last copy have incoming weight exactly one, and the last copy has weight at most one). Now our rounding algorithm simply takes any matching with n edges, ignoring the weights (i.e. matches every job somewhere) in this graph. We first need to claim that such a matching exists, then claim that the total processing time is not too large.

Proposition 6. *In the bipartite graph defined by ST Rounding Algorithm, there exists a matching of size n.*

Proof. Because the total edge weight coming out of job i into a copy of machine j is x_{ij} for all i, j, the total edge weight coming out of job i in total is 1. Moreover, the total edge weight coming into each copy of machine j is at most 1. Therefore, we have constructed a fractional matching of size n. Therefore, there is also an integral matching of size n (this is the same fact discussed in Section 2, which we didn't prove). □

The above argues that the algorithm is well-defined (note that the proof is not "complete" in the sense that we didn't prove that fractional matchings imply integral matchings. But it's "formal" in the sense that the proof is complete with this outside theorem). Now we need to argue that the total processing time is good.

Theorem 7. *The integral solution output by ST Rounding Algorithm has makespan at most* $M(\vec{x}) + t$.

Proof. We'll show that for all machines j, the total processing time of jobs assigned to j is at most $M(\vec{x}) + t$ (which is equivalent to the proposition statement). Note first that *every* job with an edge to node j_c has a lower processing time than *any* job with an edge to node j_{c-1}. So let T_c denote the processing time of the slowest job with an edge to j_c. Then we have $M(\vec{x}) \geq \sum_i x_{ij} p_{ij} \geq \sum_{c=2}^{w_j} T_c$. This is because the jobs assigned to node j_c account for $\sum_i x_{ij} = 1$, and each have $p_{ij} \geq T_{c+1}$. Finally, observe that $T_1 \leq t$, as by definition we didn't allow any jobs to be placed on machines where their processing time exceeded t. So $M(\vec{x}) + t \geq \sum_c T_c$. Finally, observe that the maximum possible processing time of the unique job assigned to node j_c is T_c, so the total processing time of machine j is $\sum_c T_c \leq M(\vec{x}) + t$. □

This is a really influential rounding scheme that accomplishes much more than just what is proved here — see the original paper and follow-ups for details. We conclude by using this rounding scheme inside a full approximation algorithm.

Definition 6 (ST Approximation Algorithm)**.** *The ST Approximation Algorithm does the following:*

1. *Initialize* $M := \infty$.
2. *Initalize* $\vec{y} = \vec{0}$.
3. *For* $i = 1$ *to* n, *and* $j = 1$ *to* m:
   (a) *Solve* $LP(p_{ij})$, *and let* T *be its optimal value, and* $\vec{x}$ *be its fractional assignment.*
   (b) *If* $T + p_{ij} \leq M$:
      i. *Update* $M := T + p_{ij}$.
      ii. *Update* $\vec{y} := \vec{x}$.
4. *Round* $\vec{y}$ *to an integral solution* $\vec{y}^*$ *using ST Rounding Algorithm and output* $\vec{y}^*$.

Theorem 8. *ST Approximation Algorithm satisfies* $\sum_{i,j} y^*_{ij} p_{ij} \leq 2\textsc{Opt}$.

Proof. Let i', j' denote the maximum processing time that is used in the optimal integral schedule, and let i^*, j^* denote the round where M is set in ST Approximation Algorithm. Then we have:

$$\sum_{i,j} y^*_{ij} \cdot p_{ij} \leq M \leq \textsc{Opt} + p_{i'j'} \leq 2\textsc{Opt}.$$

The first inequality follows from Theorem 7. The second follows by definition of the for loop, and because the optimal integral schedule is one feasible schedule for $LP(p_{i'j'})$. The final inequality follows as $\textsc{Opt} \geq p_{i'j'}$, as machine i' takes at least time $p_{i'j'}$ to process. □

6 Christofides' Algorithm for Metric TSP

The next approximation algorithm we'll see actually doesn't come from LP rounding. But it's a beautiful approximation algorithm, and recent follow-up work based on this actually does use LP rounding (roughly, it replaces the MST below with an LP-rounded tree). Here are two examples I'm aware of, but there are probably many others: [**?**, **?**].

Metric Traveling Salesman Problem: Given as input a weighted, complete, undirected graph where weights form a *metric* (that is, the edge weight between u, v is the length of the shortest path from u to v), a *Hamiltonian Cycle* is a cycle of length n. The weight of a Hamiltonian cycle is the sum of edge weights along that cycle. Output the minimum-weight Hamiltonian cycle.

The Metric TSP is NP-hard, but we'll now see a really clever 3/2-approximation, due to Christofides.

Definition 7 (Christofides' Algorithm)**.** *Christofides' Algorithm is the following:*

1. *Let T be minimum spanning tree for G.*
2. *Let O be the set of vertices with odd degree in T. Note that $|O|$ must be even.*
3. *Let M be the min-cost matching between vertices in O.*
4. *Consider the graph with edges $T \cup M$. Let C be any Eulerian tour (that is, any tour which traverses all edges, possibly revisiting the same node multiple times — we will later prove quickly that such a tour exists).*
5. *Turn C into a Hamiltonian cycle by short-cutting. That is, sort nodes $v_1, \ldots, v_n$ in the order that C visits them, and output this as the proposed solution.*

Theorem 9 ([**?**])**.** *Christofides' algorithm produces a 3/2-approximation to Metric TSP.*

Proof. The proof will follow from a series of short lemmas. We first establish that every step of the algorithm is valid. Observe first that because $|O|$ is even and the original input is complete, there is indeed a matching between nodes in O.

Observation 4. *Every node has even degree in $T \cup M$.*

Proof. All nodes either have even degree in T and degree 0 in M (which sums to even), or odd degree in T and degree 1 in M (which also sums to even). □

It is also known that a graph admits an Eulerian tour if and only if all nodes have even degree (if you haven't seen this fact before, it is a good exercise to prove it). Therefore, all steps in the algorithm are well-defined. We now transition to analyzing the approximation ratio. We first begin by claiming that the weight of C is at most the weight of $T \cup M$.

Lemma 10. *The weight of C is at most the weight of $T \cup M$.*

Proof. We know that the Eulerian tour visits every edge in $T \cup M$ exactly once. Therefore, the Eulerian tour is a set of paths from v_i to v_{i+1} for all i (and also a path from v_n to v_1). Because the graph is metric, the length of the path from v_i to v_{i+1} is at least the length of the edge from v_i to v_{i+1}. As the weight of $T \cup M$ is exactly the sum of the weights of these paths, the weight of C can only be less. □

Now, our analysis reduces to analyzing the weight of T, and the weight of M.

Lemma 11. *The weight of T is at most the optimal Hamiltonian circuit.*

Proof. Every Hamiltonian circuit contains a tree plus one additional edge. So if we remove any edge from a Hamiltonian circuit, we get a tree with *less* weight. This tree is a candidate for the MST, and T is the MST. So T must have less weight than this tree, and therefore also the optimal Hamiltonian circuit. □

Lemma 12. *The weight of M is at most half the optimal Hamiltonian circuit.*

Proof. List the nodes of O in order that they are visited in the Hamiltonian circuit, $v_1, \dots, v_{|O|}$ (recall $|O|$ is even). Consider the following two matchings between nodes in O:

- Match v_i to v_{i+1}, for all odd i.
- Match v_i to v_{i+1}, for all even i, and also match v_1 to $v_{|O|}$.

We claim that the sum of these two matchings is at most the weight of the optimal circuit. To see this, observe that the optimal circuit has a path from v_i to v_{i+1} for all i (and $v_{|O|}$ to v_1). For all i, the weight of the edge between v_i and v_{i+1} is at most the weight of this path (because the graph is Metric). Therefore, the sum of the weights of both matchings is upper bounded by the weight of the Hamiltonian circuit. Therefore, one of the matchings must have weight at most half that of the Hamiltonian circuit. □

We now wrap up the proof by observing that our total weight is at most that of $T \cup M$, which by the lemmas is at most 3/2 that of the optimum. □

References