CMSC 451: Lecture 18
Approximations: Vertex Cover and TSP

Coping with NP-completeness: With NP-completeness we have seen that there are many important optimization problems that are likely to be quite hard to solve exactly. Since these are important problems, we cannot simply give up at this point, since people do need solutions to these problems. How do we cope with NP-completeness:

Brute-force search: This is usually only a viable option for small input sizes (e.g., $n \leq 20$).

Heuristics: This is a strategy for producing a valid solution, but may be there no guarantee on how close it is to optimal.

General Search Algorithms: There are a number of very powerful techniques for solving general combinatorial optimization problems. These go under various names such as *branch-and-bound*, *Metropolis-Hastings*, *simulated annealing*, and *genetic algorithms*. The performance of these approaches varies considerably from one problem to problem and instance to instance. But in some cases they can perform quite well.

Approximation Algorithms: This is an algorithm that runs in polynomial time (ideally), and produces a solution that is guaranteed to be within some factor of the optimum solution.

Approximation Bounds: Most NP-complete problems have been stated as decision problems for theoretical reasons. However underlying most of these problems is a natural optimization problem. For example, find the vertex cover of *minimum* size, the independent set of *maximum* size, color a graph with the *minimum* number of colors, or find the traveling salesman tour with the *shortest* cost. An approximation algorithm is one that returns a valid (or feasible) answer, but not necessarily one of the optimal size/weight/cost.

How do we measure how good an approximation algorithm is? We define the *approximation ratio* of an approximation algorithm as the worst-case ratio between the answer produced by the approximation algorithm and the optimum solution. For minimization problems (where the approximate solution will usually be larger than the optimum) this is expressed as approx/opt, and for maximization problems (where the approximate solution will be smaller than the optimum) this is expressed as opt/approx. Thus, in either case, the ratio will be at least 1, and the smaller the better.

Although NP-complete problems are equivalent with respect to whether they can be solved exactly in polynomial time in the worst case, their approximability varies considerably. Here are some possibilities (from best to worst):

- Some NP-complete problems can be approximated *arbitrarily well*. In particular, the user provides a parameter $\varepsilon > 0$ and the algorithm achieves an approximation ratio of $(1 + \varepsilon)$. Of course, as ε approaches 0 the algorithm's running time gets worse. If such an algorithm runs in polynomial time for any fixed ε, it is called a *polynomial time approximation scheme*, or PTAS.
- Some NP-complete problems can be approximated and the approximation ratio is a *constant*. (We will see examples later in this lecture.)

- Some NP-complete problems can be approximated, but the approximation ratio is a *function of* n. (For example, the set cover problem, can be approximated to within a factor of $\ln n$ by the greedy heuristic. It is provably hard to do better than $O(\log n)$ unless P = NP.
- Some NP-complete problems are *inapproximable* in the sense no polynomial time algorithm achieves an approximation ratio smaller than ∞ unless P = NP. (For example, there is no bound on the approximability of either independent set or graph coloring, unless P = NP.)

Vertex Cover: We begin by showing that there is an approximation algorithm for vertex cover with an approximation ratio of 2, that is, this algorithm will be guaranteed to find a vertex cover whose size is at most twice that of the optimum. Here is the vertex cover problem.

Vertex Cover (optimization): Given a graph $G = (V, E)$, compute a subset $V' \subseteq V$ of minimum size such that every edge $e \in E$ is incident to some vertex in V'. (See Fig. 1.)

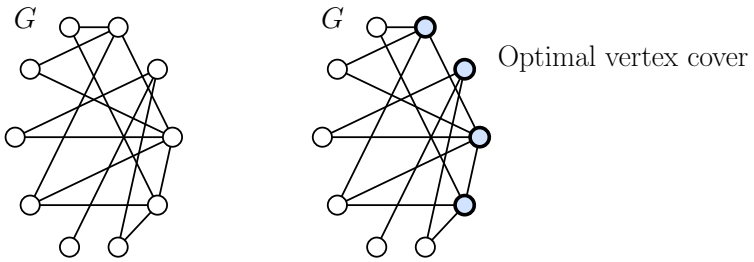

Fig. 1: Vertex cover (optimal solution).

How does one go about finding an approximation algorithm? A reasonably good strategy is the greedy heuristic, which involves repeatedly selecting the vertex of highest degree and add it to V'. Remove this vertex and its incident edges, and recompute degrees. This essentially simulates the greedy set-cover heuristic in the context of selecting vertices to cover edges. It can be shown that this leads to an approximation ratio of $\ln n$, where $n = |V|$.

The greedy heuristic is a good idea (and it probably works well in practice), but here's an approach that achieves an approximation ratio of 2. Recall that a *matching* in a graph $G = (V, E)$ is a subset of edges, $M \subseteq E$, such that each vertex is incident to at most one edge of M. A *maximum matching* has the largest number of edges. A weaker notion is a *maximal matching*, which has the property that it is not possible to add another edge.

While computing the maximum matching in a graph involves a fairly sophisticated algorithm (Edmond's blossom algorithm), there is a very simple algorithm for computing a maximal matching, which is all we need. The relevance of maximal matchings to vertex covers is given in the following claim.

Lemma: Given a graph $G = (V, E)$, let M be a maximal matching and let V^* be the optimum vertex cover in G. Then:

(i) $|V^*| \geq |M|$

(ii) $|V^*| \leq 2|M|$

Proof: To prove (i), observe that for any $(u, v) \in M$ *any* vertex cover needs to include either u or v, since otherwise this edge is not covered. Since this holds for any vertex cover, it holds for V^*.

To prove (ii), observe that since M is maximal, every edge (u, v) of G shares at least one endpoint with an edge of M, for otherwise we could add (u, v) to M, contradicting the fact that M is maximal. Thus, the union of all of M's endpoints, a set of size $2|M|$, is a vertex cover.

This lemma suggests a very simple algorithm. First, let's construct a maximal matching M in G. To do this, we repeatly, take any edge (u, v) of G, add it to M, then remove all the edges incident to either u or v. (If you do not want to actually change the graph, you can instead simply mark the edges, and repeatedly select any unmarked edge.) The result is clearly a matching, since whenever an edge is added, all conflicting edges are removed. It is also clearly maximal, since on termination, all edges have been removed.

To obtain a vertex cover, we exploit (ii) from the above lemma. Putting both endpoints of the edges of any maximal matching in G yields a vertex cover V' of size $2|M|$. By (i), $2|M| \leq 2|V^*|$, implying that V' is at most twice the size of the optimal vertex cover. The algorithm shown in the following code fragment, and it is illustrated in Fig. 2.

Approximation to vertex cover

```
approx-VC(G) {                          // returns an approximation to G's vertex cover
  V' = empty
  while (G has an edge (u,v)) {
    add both u and v to V'                          // add u and v to the cover
    delete from G all edges incident to u and to v  // remove all conflicting edges
  }
  return V' as the approximate vertex cover
}
```

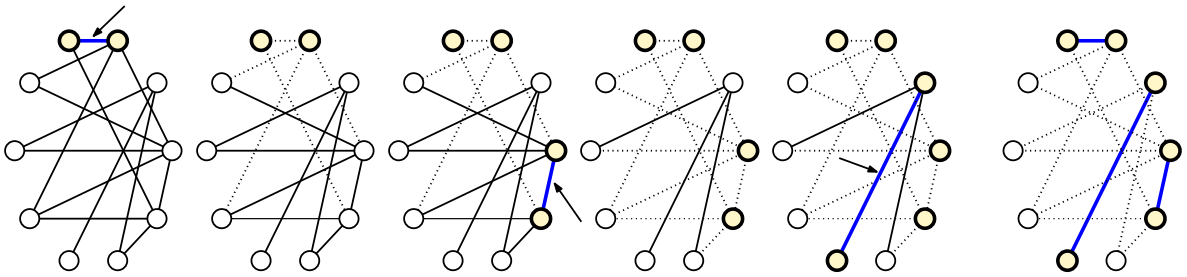

Fig. 2: The matching-based heuristic for vertex cover.

Summarizing the above analysis, we have:

Lemma: Algorithm `approx-VC` has an approximation ratio of 2.

This proof illustrates one of the main features of the analysis of any approximation algorithm. Namely, that we need some way of finding a bound on the optimal solution. (For minimization

problems we want a lower bound, for maximization problems an upper bound.) The bound should be related to something that we can compute in polynomial time. In this case, the bound is related to the set of edges M in the maximal matching.

Reductions and Approximations: Now that we have a factor-2 approximation for one NP-complete problem (vertex cover), you might be tempted to believe that we now have a factor-2 approximation for all NP-complete problems. Unfortunately, this is not true. The reason is that approximation factors are not generally preserved by transformations.

For example, in an earlier lecture, we proved that if V' is a vertex cover for G, then the complement vertex set, $V \setminus V'$, is an independent set for G. That is, G has a vertex cover of size k if and only if G has an independent set of size $n-k$, where $n = |V|$. Let k^* denote the size of the optimum vertex cover in G. By our heuristic, we can compute a vertex cover of size at most $2k^*$.

What if we try to apply this to independent set? We know that G's optimum independent set has size $n - k^*$. Our heuristic yields an independent set (through complementation) of $n - 2k^*$. Thus, we have achieve the following approximation ratio for independent set:

$$\rho(n,k) \;=\; \frac{n-k}{n-2k}.$$

How good is this? It could be terrible! For example, if $n = 1001$ and $k = 500$, then the ratio is $501/(1001 - 1000) = 501/1 = 501$. Yikes! (And if this is not bad enough, just make both n and k a thousand times larger!)

In summary, just because you have a good approximation algorithm one NP-complete problem, it does not follow that you have a good approximation algorithm for them all.

Metric TSP Approximation: Our next result involves a famous topic called the *Traveling Salesman Problem* (TSP). Before introducing it, let us recall the concept of a metric space (which we saw the earlier lecture on the k-center problem). A (discrete) *metric space* consists of a finite set P points and a *distance function* $\delta(u,v)$ defined on each pair $u, v \in P$. This distance satisfies the following properties:

Positivity: $\delta(u,v) \geq 0$ and $\delta(u,v) = 0$ if and only if $u = v$.

Symmetry: $\delta(u,v) = \delta(v,u)$

Triangle inequality: $\delta(u,w) \leq \delta(u,v) + \delta(v,w)$

Most natural distance functions are metrics. (E.g., Euclidean distance for vectors, the shortest-path distance in graphs, and edit distance for strings.) Discrete metrics can be represented as an $n \times n$ distance matrix (which is symmetric) or equivalently as a complete undirected graph with nonnegative edge weights, where the weight on edge (u,v) is $\delta(u,v)$. Define the *weight* (or *cost*) of a path to be the sum of edge weights on the path. Given any set of edges T, let $\text{wt}(T)$ denote the sum of edge weights in T.

(Metric) Traveling Salesman Problem: (TSP) Given a set P of n points in a metric space, compute a simple cycle that visits all vertices (a Hamiltonian cycle) of minimum total weight (see Fig. 3).

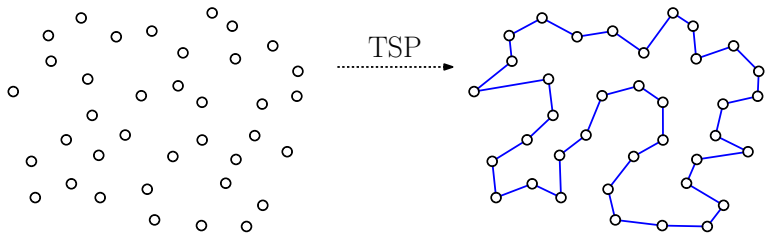

Fig. 3: The metric TSP Problem.

Factor-2 Approximation: We will present a simple MST-based algorithm that achieves a approximation ratio of 2. We will call this algorithm the *twice-around tour algorithm*.

A key insight to the algorithm is the observation if we remove any one edge from the TSP, we have a spanning tree. Of course, it is not necessarily (indeed, very unlikely to be) a minimum spanning tree. Therefore, the cost of the minimum TSP tour for P is at least as large as the cost of the MST, that is:

$$\text{wt}(\text{MST}(P)) \;\leq\; \text{wt}(\text{TSP}(P)).$$

This suggests the following idea for computing an approximation. We know we can compute the MST efficiently (e.g., by Kruskal's algorithm). Suppose we can compute a feasible TSP tour H that is larger than the MST by a constant factor c, then we will have

$$\text{wt}(H) \;\leq\; c \cdot \text{wt}(\text{MST}(P)) \;\leq\; c \cdot \text{wt}(\text{TSP}(P)),$$

implying that we have a factor-c approximation to TSP!

How to convert a spanning tree into a TSP tour? Given any free tree we can generate a tour, called a *twice-around tour*, as follows:

- Starting at any vertex of the tree, traverse the edges by walking around the tree. Thinking of the tree as a wall, place your hand against the wall, and walk along it until returning to the starting point (see Fig. 4).
- This cycle may visit vertices multiple times. Shortcut the cycle by skipping all repeated vertices. Note that, by the triangle inequality, whenever we apply short-cutting, we cannot increase the length of the cycle (see Fig. 4).

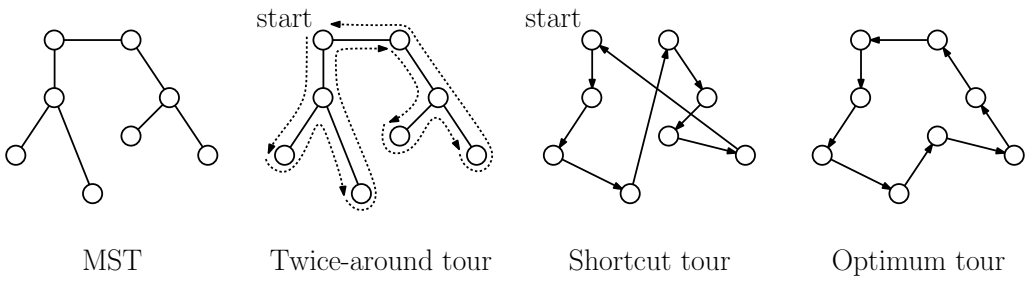

Fig. 4: The twice-around TSP Approximation.

Since the original graph is a complete graph, the MST can be computed in $O(n^2 \log n)$ time, and the twice-around tour can be computed by DFS in $O(n)$ time. Observe that the resulting cycle is a valid TSP tour. How close is it to the optimal TSP tour? (See Fig. 5 for a larger example).

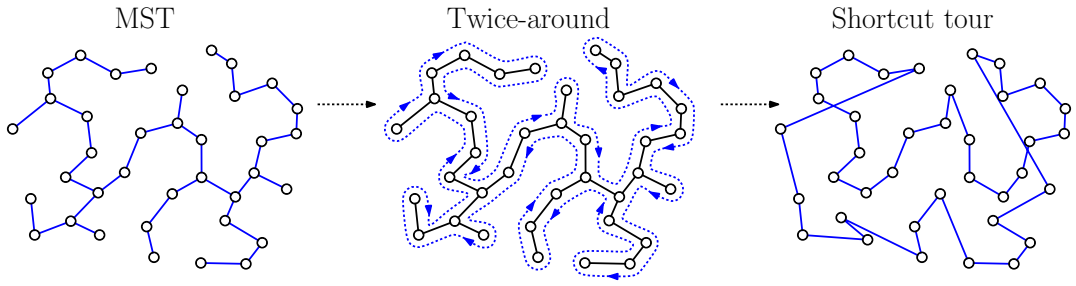

Fig. 5: A larger example of the execution of the twice-around TSP heuristic.

Lemma: The Twice-around TSP heuristic achieves an approximation ratio of 2.

Proof: Let H denote the tour produced by this algorithm, and let H^* be the optimum TSP tour. Let T be the edges of the minimum spanning tree. As observed earlier

$$\text{wt}(T) \;\leq\; \text{wt}(H^*).$$

Since every edge is traversed twice, the pure twice-around tour of T has weight exactly $2 \cdot \text{wt}(T)$. By the triangle inequality, whenever we shortcut an edge of T to form H we do not increase the cost of the tour, and so we have

$$\text{wt}(H) \;\leq\; 2 \cdot \text{wt}(T).$$

Combining these we have

$$\text{wt}(H) \;\leq\; 2 \cdot \text{wt}(T) \;\leq\; 2 \cdot \text{wt}(H^*) \quad\Longrightarrow\quad \frac{\text{wt}(H)}{\text{wt}(H^*)} \;\leq\; 2,$$

as desired.

Christofides Algorithm: We can improve on this 2-factor approximation to obtain a 1.5-factor approximation by a clever observation. The result is called *Christofides algorithm*, and it was discovered by Nicos Christofides in 1976 (and like many algorithms of the day, it was independently discovered in the former Soviet Union by Anatoliy Serdyukov).

The key observation that Christofides made is that a source of wastage in the twice-around tour involves the fact that the lower bound hits every edge twice. What if we only had to hit each edge once? Are there graphs that have a single cycle that hits all the edges exactly once? Such a cycle is called an *Eulerian circuit* (or *Eulerian cycle*). We will use the following fact, without proof.

Lemma: A graph $G = (V, E)$ has an Eulerian circuit if and only if every vertex has even degree (see Fig. 6). Given such a graph, an Eulerian cycle can be computed in time $O(n + m)$, where $n = |V|$ and $m = |E|$.

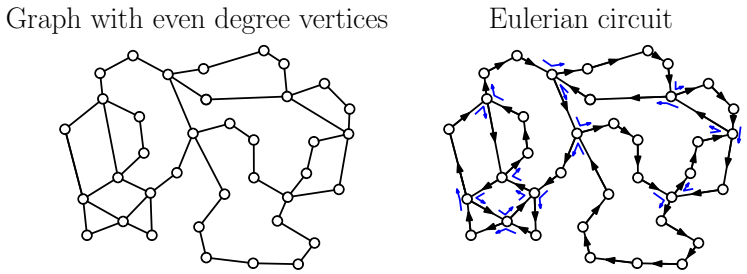

Fig. 6: Eulerian circuit in a graph where every vertex has even degree.

But how can we exploit this? The MST will certainly contain vertices of odd degree. A useful fact (called the *Handshaking Theorem*) states that every graph has an even number of vertices with odd degree. (This is an easy consequence of the observation that the sum of vertex degrees is twice the number of edges, which is always even.) Given the minimum spanning tree of P, let P_0 denote the vertices having even degree in the MST and let P_1 denote the vertices of odd degree. The handshaking theorem states that $|P_1|$ is even. Christofides algorithm works by adding a set of edges to to the vertices of P_1 so that each gets one additional edge. Combining these edges with the MST edges, all the vertices of P now have even degree, and hence by the above lemma, we can efficiently construct an Eulerian tour in this graph. The question is how to do this without increasing the weight very much.

This will be done with the concept of matchings. Recall that a *matching* in a graph is a set of edges M such that each vertex is incident to at most one edge of M. A matching is *perfect* if every vertex is incident to an edge of M. Consider the complete graph induced by a P of points of even cardinality in a metric space, where the edges are weighted by the distance between vertices. Because this graph has an even number of vertices and is complete, it has a perfect matching. Define the *minimum-weight matching*, denoted MWM(P) to be the perfect matching of minimum total weight on P (see Fig. 7). Such a matching can be computed in $O(|V|^2|E|)$ time by a modification of Edmond's blossom algorithm, which for us is $O(n^4)$, since our graph is complete.

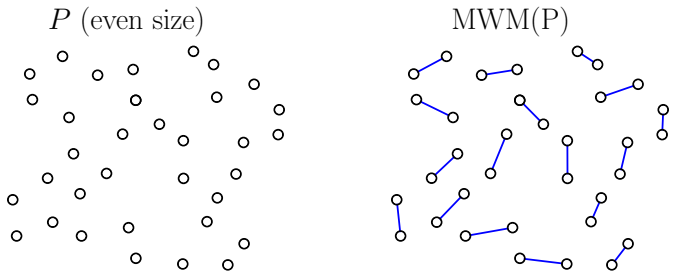

Fig. 7: A point set P of even cardinality and its minimum-weight matching MWM(P).

We can now describe Christofides algorithm. The input is a set of n points P in a metric space.

(a) Compute the edges of the minimum spanning tree of P, that is, $T \leftarrow \text{MST}(P)$ (see Fig. 8(a)).

(b) Partition P into sets P_0 and P_1 of points whose MST vertex has even or odd degree, respectively. (P_1 has an even number of points.)

(c) Compute the minimum-weight perfect matching of the points of P_1, that is, $M \leftarrow \text{MWM}(P_1)$.

(d) Form the graph $G = (P, T \cup M)$ on the point set P whose edge set is the union of T and M. (All the vertices in G have even degrees.)

(e) Let C' be any Eulerian circuit in G. Apply short-cutting to C' by removing repeated vertices. Let C denote the resulting tour. (By the triangle inequality, $\text{wt}(C) \leq \text{wt}(C')$.)

(f) Return C as the final approximation.

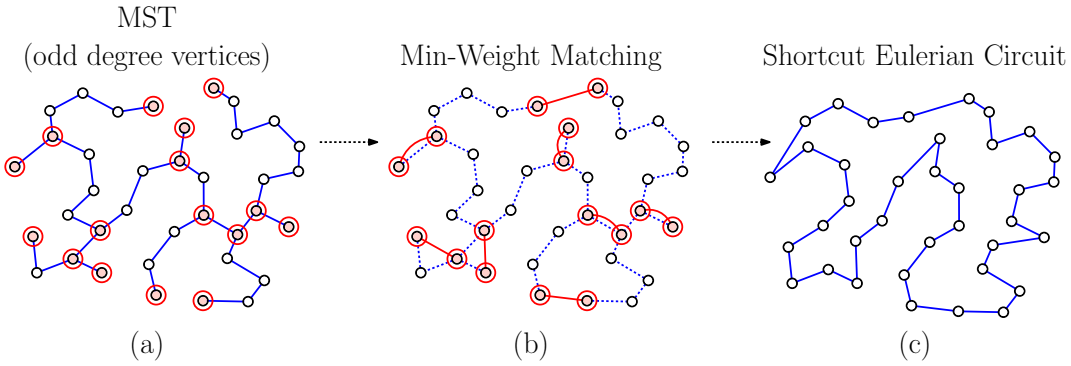

Fig. 8: Christofides algorithm: (a) MST(P) with odd-degree vertices highlighted, (b) MWM(P_1) shown with red edges, (c) the final tour after short-cutting.

A key to the efficiency of Christofides algorithm is that the added edges from the minimum-weight matching are small relative to the weight of the TSP tour. This is shown in the following lemma.

Lemma: Given a point set P, let P_1 be any subset of P of even cardinality. Then

$$\text{wt}(\text{MWM}(P_1)) \;\leq\; \frac{\text{wt}(\text{TSP}(P))}{2}.$$

Proof: First, observe that if we remove any points from P, the weight of the optimum TSP tour can only decrease, and hence $\text{wt}(\text{TSP}(P_1)) \leq \text{wt}(\text{TSP}(P))$. Consider the edges of TSP(P_1). Since P_1 has even cardinality, the tour has an even number of edges. Let M_0 and M_1 denote result of taking every other edge from TSP(P_1), with M_0 taking the even numbered edges and M_1 taking the odd numbered edges (see Fig. 9). Thus, we have

$$\text{wt}(M_0) + \text{wt}(M_1) \;=\; \text{wt}(\text{TSP}(P_1)).$$

Observe that both M_0 and M_1 are matchings over P_1. Their weight cannot be smaller than the weight of the minimum weight matching. Using this and the fact that for any numbers a and b, their minimum cannot be larger than their mean ($\min(a, b) \leq (a+b)/2$), we have

$$\text{wt}(\text{MWM}(P_1)) \;\leq\; \min(\text{wt}(M_0), \text{wt}(M_1)) \;\leq\; \frac{\text{wt}(\text{TSP}(P_1))}{2} \;\leq\; \frac{\text{wt}(\text{TSP}(P))}{2},$$

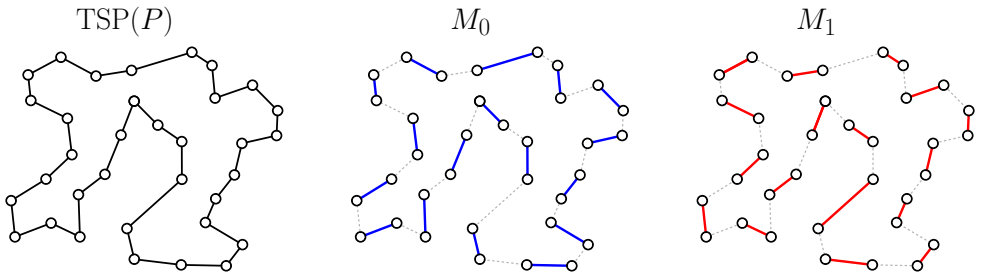

Fig. 9: Given a set P of even cardinality, the edges of TSP(P) can be partitioned into two matchings.

as desired.

We can now prove the approximation bound for Christofides algorithm.

Lemma: Christofides algorithm achieves an approximation ratio of 1.5.

Proof: Let C denote the tour produced by this algorithm and let H^* be the optimum TSP tour. Let T be the edges of the minimum spanning tree, and let M be the edges of the minimum-weight matching on the vertices P_1 of odd degree. As shown in the twice-around heuristic, we have

$$\text{wt}(T) \;\leq\; \text{wt}(H^*).$$

As proved in the previous lemma $\text{wt}(M) \leq \text{wt}(H^*)/2$. Therefore,

$$\text{wt}(T) + \text{wt}(M) \;\leq\; \frac{3}{2} \cdot \text{wt}(H^*).$$

Therefore, the length of the Eulerian circuit C' satisfies this same bound, since it visits each of these edges exactly once. By the triangle inequality, whenever we shortcut the edges of the Eulerian circuit we do not increase the cost of the tour, and so we have

$$\text{wt}(C) \;\leq\; \text{wt}(C') \;\leq\; \frac{3}{2} \cdot \text{wt}(H^*).$$

In summary, letting C denote the result of Christofides algorithm, we have

$$\frac{\text{wt}(C)}{\text{wt}(H^*)} \;\leq\; \frac{3}{2},$$

as desired.