

Lecture 23: December 3

Lecturer: Micah Adler

Scribe: Sherief Abdallah

23.1 Approximation and Reduction of NP-complete Problems

Since all NP problems can be reduced, in polynomial time, to any **NP-complete** problem, then if we can approximate one **NP-complete** problem, can we approximate every **NP-complete** Problem? The answer is **NO!** Something that people might try is to reduce an **NP-complete** problem to the approximated **NP-complete** problem (say the approximation solution of Vertex Cover), use the approximation algorithm to get an approximated solution, and then reduce it back to the original **NP** problem that we want to solve. Unfortunately, this will **NOT** work!

Let's look at an example: solving the **Independent-Set** problem:

Input: An undirected graph $G = (V, E)$.

Output: A set $U \subseteq V$ of maximum size such that no two vertices in U are connected by an edge.

Claim 23.1 *Independent-Set \leq_p Vertex Cover.*

Proof: We know that if U is an Independent-Set then $V - U$ is a Vertex Cover. We can do the following reduction:

1. $G \rightarrow G$;
2. $k \rightarrow |V| - k$;

This reduction can be completed in polynomial time. We can show that this reduction is correct:

$$\begin{aligned}
 U \text{ is Independent-set} &\leftrightarrow \text{No edge has both endpoints in the set } U \\
 &\leftrightarrow \text{every edge has at least one end point in set } V - U \\
 &\leftrightarrow V - U \text{ is a Vertex Cover}
 \end{aligned}$$

Therefore, we can reduce Independent-Set to Vertex Cover in polynomial time. ■

“Approximation” for the Independent-Set Problem:

1. Find S : the approximation set for Vertex Cover; using the approximation algorithm shown in previous lecture.
2. Return $V - S$.

This “*approximation*” algorithm is not efficient for solving Independent-Set problem. We can see this from Figure 23.1. For the input graph of Figure 23.1(a), we have the optimal Independent-Set as shown in

Figure 23.1(b). But by running the above approximation algorithm, we will get the very bad solution shown in Figure 23.1(c).

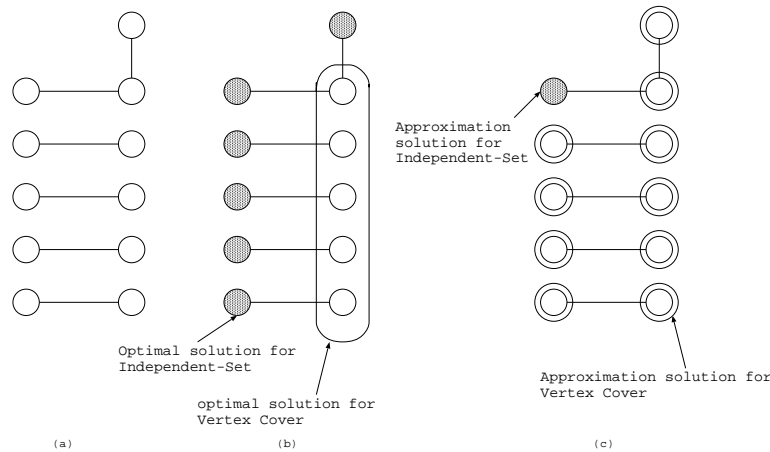


Figure 23.1: Independent-Set Problem

The reason that this scheme does not work well is that when we do the reduction, we only preserve the decision version of the problem, which has nothing to do with the quantity solution and the performance ratio. Therefore, a good performance ratio of an approximation solution can not be guaranteed if reduction is used (e.g. in the worst case above, the performance ratio for the independent set = $\frac{C_{opt}(G)}{C_{alg}(G)} = \frac{\lfloor n/2 \rfloor}{1}$).

23.2 Approximation for the Max Cut Problem

Recall the Max Cut problem:

Input: An undirected graph, $G = (V, E)$.

Output: A partition of V such that the number of edges between the partitions is as large as possible.

We know this problem is **NP-complete**, but we can construct a simple algorithm to approximate the solution:

1. $S = \emptyset$ defines the vertices on one side of the cut (i.e., $\bar{S} = V$).
2. **while** $\exists v$ such that moving v from S to \bar{S} , or vice versa, improves the cut:
3. switch v .
4. **return** the resulting cut, S, \bar{S} .

We would like to show that the running time is polynomial. Switching the vertex (step 3) is easy. The issue arises when we consider that a vertex may switch several times in the course of the algorithm (step 2). However, the size of the cut is increasing and cannot be greater than $|E|$. Therefore, the algorithm runs in polynomial time because we have at most $|E|$ iterations.

Theorem 23.2 *The algorithm is a 2-approximation.*

Proof: We want to show that the max cut is at most a factor of 2 larger than the cut returned by the algorithm (no matter what the input graph and the order of switching outlined above).

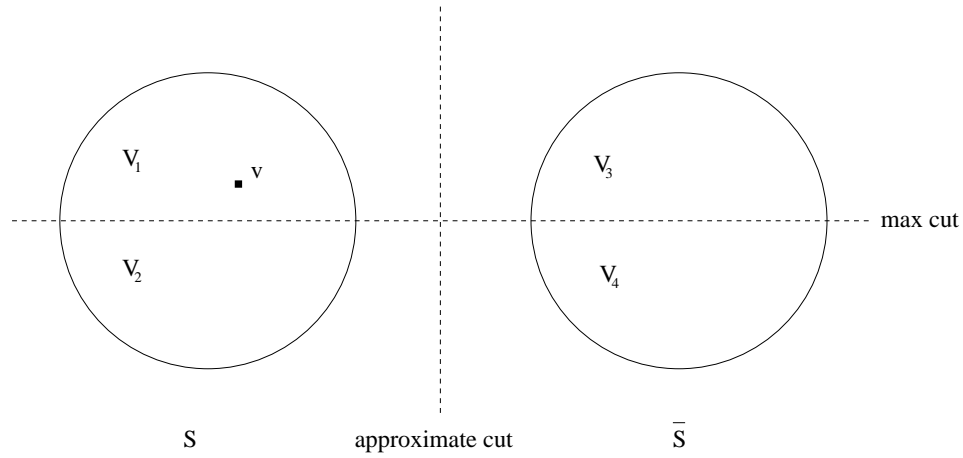


Figure 23.2: Minimum and Approximate Cuts.

The approximation is going to return S and \bar{S} (see Figure 23.2). The optimal cut is going to split S and \bar{S} into two sets each. Consequently, four sets of vertices remain, $V_1, V_2, V_3,$ and V_4 . Consider $v \in V_1$. We know the number of edges from v into V_1 and V_2 is smaller than the number of edges into V_3 and V_4 (if it were not, we could switch the vertex and increase the size of the cut).

$$|\text{edges from } v \text{ to } (V_1 \cup V_2)| \leq |\text{edges from } v \text{ to } (V_3 \cup V_4)|$$

This must be true of all vertices in V_1 . So,

$$\sum_{v \in V_1} |\text{edges from } v \text{ to } (V_1 \cup V_2)| \leq \sum_{v \in V_1} |\text{edges from } v \text{ to } (V_3 \cup V_4)|$$

but

$$\sum_{v \in V_1} |\text{edges from } v \text{ to } (V_2)| \leq \sum_{v \in V_1} |\text{edges from } v \text{ to } (V_1 \cup V_2)|$$

hence:

$$\sum_{v \in V_1} |\text{edges from } v \text{ to } (V_2)| \leq \sum_{v \in V_1} |\text{edges from } v \text{ to } (V_3 \cup V_4)|$$

We are interested in the number of edges between any of the quadrants of the graph. Define e such that $\forall i, j \in \{1, 2, 3, 4\}, e_{ij} = |\text{edges between } V_i \text{ and } V_j|$. We can rewrite the inequality:

$$e_{12} \leq e_{13} + e_{14}$$

We can do the same for $V_2, V_3,$ and V_4 , respectively.

$$\begin{aligned} e_{12} &\leq e_{23} + e_{24} && \text{because } e_{ij} = e_{ji} \\ e_{34} &\leq e_{23} + e_{13} \\ e_{34} &\leq e_{24} + e_{14} \end{aligned}$$

If these inequalities are summed,

$$\begin{array}{rcl}
 e_{12} & \leq & e_{13} + e_{14} \\
 +e_{12} & \leq & e_{23} + e_{24} \\
 +e_{34} & \leq & e_{23} + e_{13} \\
 +e_{34} & \leq & e_{24} + e_{14} \\
 \hline
 2(e_{12} + e_{34}) & \leq & 2(e_{13} + e_{14} + e_{23} + e_{24}) \\
 e_{12} + e_{34} & \leq & e_{13} + e_{14} + e_{23} + e_{24}
 \end{array}$$

Notice the right-hand side of the inequality is equal to the size of the returned cut. The left-hand side represents only part of the optimal cut. We can construct an inequality to handle the remainder of the optimal cut:

$$e_{14} + e_{23} \leq e_{14} + e_{23} + e_{13} + e_{24}$$

Adding this inequality to the sum above gives us a relation between the cuts.

$$\begin{array}{rcl}
 e_{12} + e_{34} & \leq & e_{13} + e_{14} + e_{23} + e_{24} \\
 +e_{14} + e_{23} & \leq & e_{13} + e_{14} + e_{23} + e_{24} \\
 \hline
 e_{12} + e_{14} + e_{34} + e_{23} & \leq & 2(e_{13} + e_{14} + e_{23} + e_{24})
 \end{array}$$

The inequality proves that the max cut is at most twice the size of the returned cut, i.e. the performance ratio = 2. There is a more complicated approximation algorithm that achieves 1.1383 performance ratio. ■

23.3 Approximation of the Metric Traveling Salesman Problem (MTSP)

The Metric Traveling Salesman Problem (MTSP) is:

Input: n cities and a distance function, d , that obeys the triangle inequality ($\forall 1 \leq i, j, k \leq n, d_{ij} \leq d_{ik} + d_{kj}$).

Output: A tour of minimum length that passes through every city except the start (and end) city exactly once.

We know that the Traveling Salesman Problem is **NP-complete**. Let us verify that the Metric Traveling Salesman Problem (MTSP) is also **NP-complete**.

Theorem 23.3 *MTSP is NP-complete.*

Proof: We'll prove this by showing that the Hamiltonian Cycle problem \leq_p MTSP.

Let $G = (V, E)$ be the input to the Hamiltonian Cycle problem. We want to transform this to be an input to MTSP.

Let the n MTSP input cities be defined by V , and the MTSP distances, $d_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 2 & \text{otherwise} \end{cases}$

The largest d_{ij} is 2 and the smallest possible indirect path is $1 + 1 = 2$ so the triangle inequality holds.

If there is a Hamiltonian cycle, a TSP tour has length $|V|$. The tour would consist of the the solution to Hamiltonian cycle. Since each edge in this tour would have distance 1, the length of the tour is $|V|$.

Relatedly, we can show that if there is a TSP tour of length $|V|$, the tour must use only edges with a distance of 1. (A TSP tour uses exactly $|V|$ edges, each of which has distance at least 1. If the length of the tour is $|V|$ then each edge must have distance 1.) Such a tour would define a Hamiltonian cycle. ■

23.3.1 First Algorithm for MTSP (a 2-Approximation)

1. Cast the MTSP as a graph where the cities are vertices in the graph, all edges are present, and the weights are defined by the distance function.
2. Find the Minimum Spanning Tree, T .

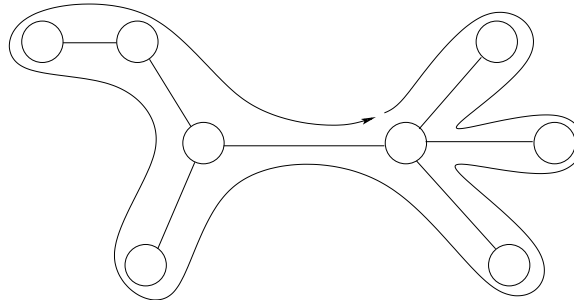


Figure 23.3: MTSP Pseudo-tour.

3. A *pseudo-tour* starts and ends at the same vertex and visits every vertex at least once using edges in a spanning tree. Construct a pseudo-tour using every edge of T twice (accomplished by walking along the perimeter T as in Figure 23.3). We are guaranteed to visit every city because there are no cycles and the tree touches every vertex.
4. Derive the tour by short-cutting repeated vertices (tour with shortcuts shown in Figure 23.4; shortcuts are represented by dotted lines).

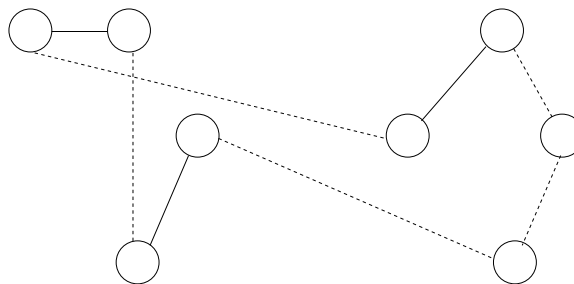


Figure 23.4: MTSP Approximation.

Claim 23.4 *The algorithm is a 2-approximation.*

Proof: The cost of the pseudo-tour is twice the cost of the tree (every edge is visited twice). Because of the triangle inequality, the length of our tour is at most twice the cost of T . That is, the shortcuts are at most the sum of the length of the edges along the long route.

Further, because a tour is a spanning tree (with an extra edge), the weight of T is at most the cost of the optimal tour. The spanning tree may cost less. Hence,

$$\text{length of our tour} \leq 2 \times \text{cost}(T) \leq 2 \times \text{optimal tour length}$$

Thus achieving performance ratio of 2. ■

23.3.2 Second Algorithm for MTSP (a $\frac{3}{2}$ -Approximation)

Definition 23.5 *A Eulerian tour is a path that traverses every edge of the graph exactly once and returns back to the initial vertex (it may visit vertices multiple times).*

We know that G has a Eulerian tour if and only if G is connected and every vertex has even degree. We also know that if an Eulerian tour exists we can find it in polynomial time.

Using Eulerian tours, we can define a better approximation for MTSP. This approximation is due to Christofides [C76].

1. Cast the MTSP as a graph where the cities are vertices in the graph, all edges are present, and the weights are defined by the distance function.

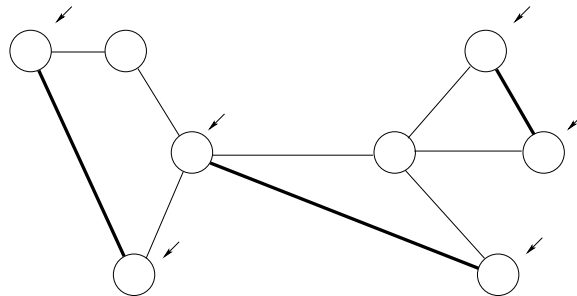


Figure 23.5: MTSP Minimum spanning tree plus matchings.

2. Find the Minimum Spanning Tree, T (represented by the thin lines in Figure 23.5).
3. Add edges to T so that every vertex has even degree as follows:
 - (a) Find the set D of vertices of T whose degree is odd, (these are denoted by the arrows in Figure 23.5). $|D|$ will always be even for every graph. This is because every edge contributes two incidences. The sum of the degree of all vertices is an even number (twice the number of edges). Therefore the sum of the degrees of the odd degree vertices is even (the only way to get an even number from sum of n odd numbers is if n is even). This will allow the next step.
 - (b) Add to T a minimum weight perfect matching on the general graph D , (these edges are shown by thick lines in Figure 23.5).

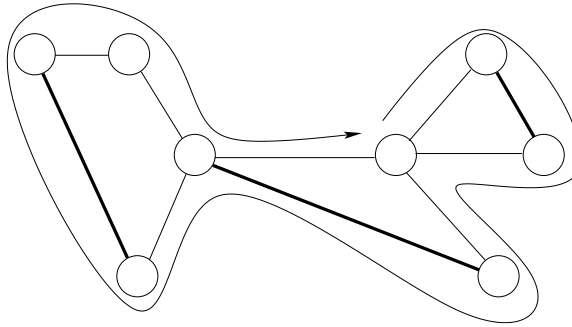


Figure 23.6: An Eulerian tour.

4. Find an Eulerian tour on the resulting graph. We know that every vertex must be in the tour (because we started with a MST) and that each vertex is visited at least once (because this is an Eulerian tour). The tour is shown in Figure 23.6.

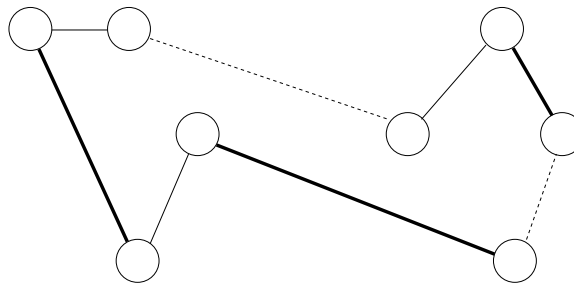


Figure 23.7: The Final Approximation to the MTSP.

5. Solve using shortcuts as in the first approximation algorithm. The tour with shortcuts is shown in Figure 23.7, where the shortcuts are represented by dotted lines.

The analysis of the algorithm will be covered in next lecture.

References

- C76 N. CHRISTOFIDES, *Worst-case analysis of a new heuristic for the traveling salesman problem*, TR CS-93-13, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1976.
- CLR90 T. CORMEN and C. LEISERSON and R. RIVEST, *Introduction to Algorithms*, MIT Press, 1990
- PJ00 P. JI, *Scribe Notes, Lecture 20, Advanced Algorithms Fall 2000*
- FD00 F. DIAZ, *Scribe Notes, Lecture 21, Advanced Algorithms Fall 2000*