6.854J / 18.415J Advanced Algorithms
Fall 2008

18.415/6.854 Advanced Algorithms November 3, 2008

Lecture 16: Approximation Algorithms

Michel X. Goemans

1 Approximation Algorithms

Many optimizations problems arising in practice are NP hard. Under the widely accepted conjecture that $P \neq NP$, we cannot compute efficiently and exactly an optimal solution for all possible instances of these problems. Several approaches have been used to deal with this intractability. On one hand, *dynamic programming*, *branch and bound*, and *implicit enumeration* algorithms always find an optimal solution by navigating the space of feasible solutions in a more efficient way than an exhaustive search, but their running time is not guaranteed to be polynomial in the input's size. On the other hand, *heuristic* algorithms provide a sub-optimal solution to the problem, but their running time is polynomial in the size of the input problem. In this lecture we will focus on *approximation algorithms*, which are heuristics that always find a solution whose objective value is guaranteed to be within a certain factor of the optimum solution.

Definition 1 (Approximation Algorithm) *Let $\mathcal{P}$ be a minimization (resp. maximization) problem with instances $I \in \mathcal{I}$. An α-approximation factor for $\alpha \geq 1$ (resp. $\alpha \leq 1$) algorithm for $\mathcal{P}$ is an algorithm $\mathcal{A}$ whose running time is polynomial in the size of the given instance I, and outputs a feasible solution of cost c_A such that $c_A \leq \alpha \cdot OPT_I$ (resp. $c_A \geq \alpha \cdot OPT_I$), where OPT_I is the cost of the optimal solution for instance I.*

In this lecture, we will discuss three general techniques of designing approximation algorithms for NP-hard problems:

1. Using optimal value in the analysis without explicitly knowing it.
2. Linear programming relaxation and rounding.
3. Primal-dual technique.

2 A 3/2-Approximation Algorithm for the Metric TSP

The Traveling Salesman Problem is one of the most extensively studied problems in combinatorial optimization. In the metric version of the problem, an instance is a complete undirected graph $G = (V, E)$ and $c : E \to \mathbb{R}_+$, where c satisfies the metric property: $c(u, v) = c(v, u)$ for all $u, v \in V$, and the triangle inequality, $c(u, v) \leq c(u, w) + c(w, v)$, for all $u, v, w \in V$. The objective is to find tour, that is a cycle visiting every vertex exactly once (also called a *tour*) minimum cost .

A $\frac{3}{2}$ approximation algorithm for this problem by Christofides [1] is as follows.

1. Find a minimum spanning tree T of G.
2. Compute a minimum cost perfect matching M on the set of odd-degree vertices $V_{odd} \subseteq T$.
3. Find an Eulerian tour C' (a cycle visiting all the edges exactly once) in $M \cup T$.
4. Output the tour C that visits the vertices of G in the order of their first appearance in the C'.

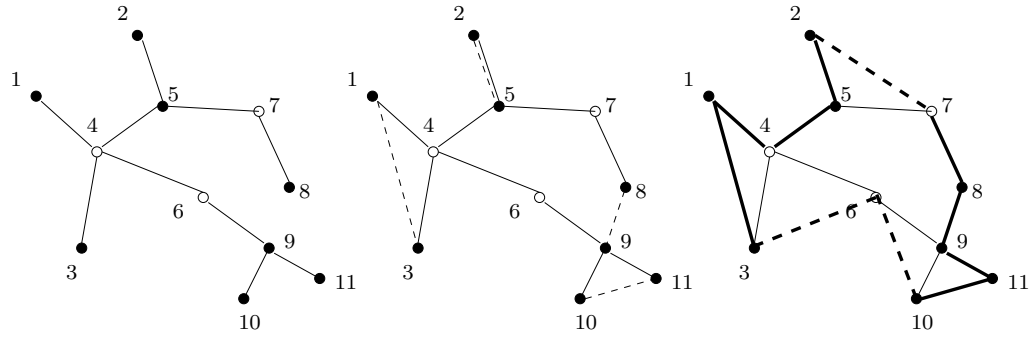

Figure 1: Execution of Christofides' algorithm on an instance. The first figure shows a minimum cost spanning tree. The second figure shows the addition of a minimum cost matching on odd degree vertices in the tree, and the third figure shows a cycle obtained after "shortcutting" an Eulerian tour in the previous graph, starting from vertex 1.

Theorem 1 *The above algorithm is a* $3/2$*-approximation algorithm for the metric TSP.*

Proof: It is clear that all steps in the algorithm can be implemented in polynomial time. The minimum spanning tree can be found using a greedy algorithm, and the minimum cost matching for V_{odd} can be found in polynomial time using the ellipsoid algorithm, as discussed in one of the previous lectures (or by a purely combinatorial algorithm also based on the linear program we discussed). Note that $c(T) \leq OPT$, because the optimal tour without an edge becomes a tree. Also, $c(M) \leq OPT/2$. To see this, consider any optimal tour, and then short-cut it to get a cycle visiting only vertices in V_{odd} with cost at most OPT. Since the cycle induces two matchings consisting of alternating edges, at least one of them will have cost at most $OPT/2$. From this, the total cost of the Eulerian cycle, an upper bound of the cost of the algorithm, is at most $OPT + OPT/2 = 3/2 \cdot OPT$. □

Note that in the analysis of the algorithm, we used the value of OPT even without explicitly computing it exactly, or getting a lower bound on it. Figure 1 shows an instance of the metric TSP, and the execution of the algorithm on this instance.

A few remarks:

- The above analysis for the algorithm is tight, i.e. $\forall \varepsilon > 0$ there is an instance I such that the algorithm returns a solution which is $3/2 - \varepsilon$ times the optimal solution.
- Currently, no algorithm with an approximation factor better than $3/2$ is known for metric TSP.
- TSP is known to be MAX-SNP hard [5] even for the case when distances are either 1 or 2. Also, Papadimitriou and Vempala [4] have proved that a 1.01 approximation algorithm for the metric TSP will imply $P = NP$.

3 Designing Approximation Algorithms via Relaxations

One of the most important paradigms in the design of approximation algorithms are relaxations. Consider the following (hard) minimization problem.

$$\begin{aligned} &\min f(x) \\ \text{s.t.} \quad &x \in S. \end{aligned}$$

One approach to solve this problem is to extend S to a bigger set $P \supseteq S$ where the same problem is easier to solve. Namely, we extend the function f to a function $g : P \to \mathbb{R}$ satisfying $g(x) = f(x), \forall x \in S$ (or $g(x) \leq f(x)$). If this condition holds, then

$$\min_{x \in S} f(x) \geq \min_{x \in P} g(x),$$

which gives a lower bound for the value of the optimal solution. Therefore, if an algorithm gives a solution $x^* \in S$ which satisfies $f(x^*) \leq \alpha \min_{x \in P} g(x)$, then this is an α- approximation algorithm for the problem.

For example, many combinatorial optimization problems can be expressed as

$$\begin{aligned} & \min c^T x \\ \text{s.t.} \quad & Ax = b, \\ & x \in \{0,1\}^n. \end{aligned}$$

A natural relaxation is to replace the integrality constraint $x_i \in \{0,1\}$ by the linear constraint $0 \leq x_i \leq 1$, we obtain the *LP relaxation* of the integer program above.

$$\begin{aligned} & \min c^T x \\ \text{s.t.} \quad & Ax = b, \\ & 0 \leq x_i \leq 1, \quad \forall i = 1, \ldots, n. \end{aligned}$$

In some cases, the polytope corresponding to the LP relaxation has all integral extreme points. In such cases, it is sufficient to solve the LP relaxation to solve the original problem exactly. But this is not true in general.

3.1 LP Relaxation for the Vertex Cover Problem

Given an undirected graph $G = (V, E)$, a *vertex cover* of G is a collection of vertices $C \subset V$ such that all edges $e = (u, v)$ in E satisfy $C \cap \{u, v\} \neq \emptyset$. The Vertex Cover problem on an instance $G = (V, E), c : E \to \mathbb{R}^+$ is to find a cover C of G of minimum cost $c(C) = \sum_{v \in C} c(v)$. This is known to be an NP-hard problem.

A natural formulation using integer variables and linear constraints is the following. We define a variable $x_u \in \{0, 1\}$ which takes value 1 if it is in the vertex cover, 0 otherwise. Then the following is an integer programming formulation for the vertex cover problem.

$$\min \sum_{v \in V} c_v x_v \tag{1a}$$

$$\text{s.t.} \quad x_u + x_v \geq 1, \quad \forall (u, v) \in E, \tag{1b}$$

$$x_u \in \{0, 1\}, \quad \forall u \in V. \tag{1c}$$

The LP relaxation for the vertex cover problem is

$$\min \sum_{v \in V} c_v x_v \tag{2a}$$

$$\text{s.t.} \quad x_u + x_v \geq 1, \quad \forall (u, v) \in E, \tag{2b}$$

$$x_u \geq 0, \quad \forall u \in V. \tag{2c}$$

Note that we removed the $x_u \leq 1$ constraints, since if $x_u > 1$ we can change it to $x_u = 1$ without increasing the cost, and still have a feasible solution.

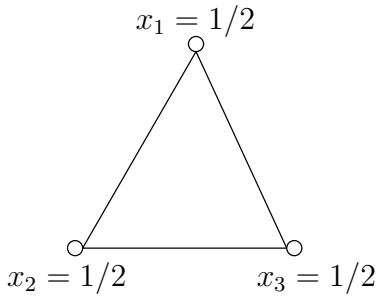

Figure 2: An example where the LP relaxation for the Vertex Cover does not have an integer optimal solution.

The LP relaxation does not necessarily have an optimal integral solution in general. For example, consider the graph given in Figure 3.1 with all costs equal to 1. The optimal solution for this instance has cost $OPT = 2$, but the optimal solution for the LP relaxation has cost $LP = 3/2$, as shown in the figure. What this example shows is not only that $LP < OPT$ in general, but also an interesting fact about the strength of this relaxation. Suppose that we are going to use LP as a lower bound on OPT in order to prove an approximation guarantee. As we will see in the next subsection, we will be able to find a cover C with cost at most $2LP$. Therefore, we can say

$$c(C) \leq 2LP \leq 2OPT$$

to prove an approximation guarantee of 2, However, the example proves that we will not be able to decrease this factor beyond 4/3. This follows from the fact that

$$OPT \leq c(C) \leq \alpha LP \leq \alpha OPT \Rightarrow OPT/LP \leq \alpha$$

then the best we can hope for is at most 4/3 by using this relaxation. This important property of the "bad examples" is captured in the concept of integrality gap.

Definition 2 (Integrality gap) *Given a relaxation $LP(\Pi)$ for an integer program $IP(\Pi)$ that formulates a combinatorial (minimization) optimization problem on a collection of instances $\{\Pi\}$, the* integrality gap of the linear program relaxation *is the largest ratio between the optimal solution of both formulations, namely:*

$$\textit{Integrality gap} = \sup_{\Pi} \frac{IP(\Pi)}{LP(\Pi)}$$

For the Vertex Cover LP relaxation, the integrality gap is exactly 2. To see that it is at least 2, consider the complete graph $G = K_n$, with unitary costs. The minimum vertex cover has cost $n - 1$, while the linear program relaxation can assign $1/2$ to all variables, which gives a total cost of $n/2$. Therefore, the integrality gap is at least $\frac{2(n-1)}{n} \to 2$. The upper bound follows from the 2-approximation algorithm we will see in the next subsection.

3.2 A 2-approximation Algorithm for Vertex Cover

A natural approach to get an integral solution from a fractional solution is to round the fractional values. A simple rounding scheme for the vertex cover is as follows.

1. Solve the linear programming relaxation given by (2a)-(2c), to get the fractional solution x^*.

2. Compute the vertex cover as $C = \{v \in V, x_v^* \geq 1/2\}$ (i.e., round each fractional variable to the nearest integer).

Theorem 2 *The above rounding scheme is a 2-approximation algorithm for the Vertex Cover problem.*

Proof: First, we need to check that C is indeed a vertex cover. For each $e = (u, v) \in E$, $x_u^* + x_v^* \geq 1$, so at least one of x_u^*, x_v^* has value at least $1/2$, and is in C. Next, the cost of this vertex cover satisfies

$$c(C) = \sum_{v: x_v^* \geq 1/2} c_v \leq 2 \sum_{v \in V} c_v x_v^* = 2LP \leq 2OPT,$$

hence the LP rounding is a 2-approximation algorithm for the vertex cover problem. □

This is a very basic (the simplest) example of rounding; more sophisticated rounding procedures have been used to design approximation algorithms; we'll see some in coming lectures.

4 The Primal Dual Technique

Yet another way of designing approximation algorithms for intractable problems is the primal dual method. The basic idea of the primal dual scheme is this: At every point of the algorithm, we keep a feasible dual solution, and a corresponding infeasible integer primal solution. The dual variables are then modified at every step and so is the infeasible primal solution, so as to achieve primal feasibility. At this point, the dual gives a lower bound (for minimization problems) on the optimal primal objective function value, which is used to derive the approximation factor for the algorithm. The interesting thing about this technique is that we do not need to explicitly solve the linear program (as is the case in rounding); the linear program is used only in the analysis of the algorithm.

We illustrate this method for the vertex cover problem. The linear program for the vertex cover problem is given by (2a)-(2c). The dual of this linear program is given by

$$\begin{aligned} \max & \sum_{e \in E} y_e \\ \text{s.t.} & \sum_{e \in \delta(v)} y_e \leq c_v \qquad \forall v \in V, \\ & y_e \geq 0 \qquad \forall e \in E. \end{aligned} \tag{3}$$

The primal dual algorithm for the vertex cover problem is as follows. In the algorithm, C corresponds to the set of vertices in the (supposed to be) vertex cover, and F is the set of edges in the graph not yet covered by C.

1. $y(v) \leftarrow 0 \quad \forall v \in V, \quad C \leftarrow \emptyset, \quad F \leftarrow E.$
2. While $F \neq \emptyset$
3. Let $e = (u, v)$ be any edge in F.
4. Increase y_e until the constraint (3) becomes tight for u or v.
5. Add that corresponding vertex (say it is v) to C.
6. $F \leftarrow F \setminus \delta(v)$.

Theorem 3 *The above algorithm achieves an approximation ratio of 2 for the vertex cover problem.*

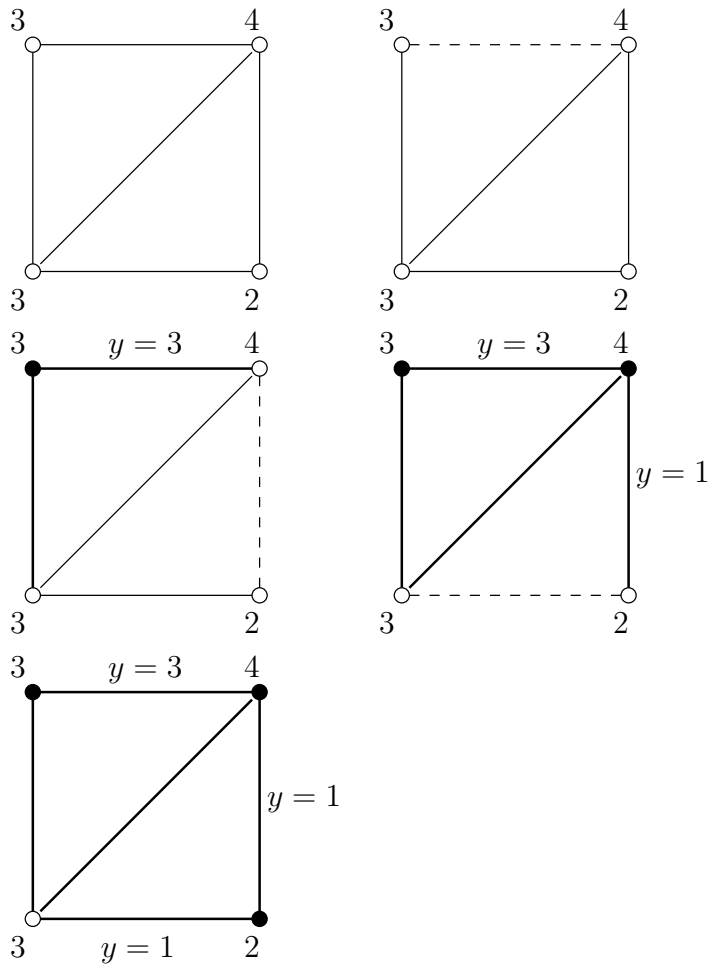

Figure 3: Illustration of the primal-dual algorithm for the vertex cover problem. The cost of the vertices are indicated next to each vertex. Dotted edge denotes the edge currently under consideration, thick edges denote those already covered by the current vertex cover. The vertices in the cover are shown as solid circles.

Proof: First of all, it is clear that the set C returned by the algorithm is a vertex cover. Let y be the dual solution returned. Observe that by construction, this solution is dual feasible (we maintain dual feasibility throughout the execution). Furthermore, for any $v \in C$, we have that $c_v = \sum_{e \in \delta(v)} y_e$. Let us now compute the cost of the vertex cover returned by the algorithm.

$$\begin{aligned} \sum_{v \in C} c_v = \sum_{v \in C} \Big(\sum_{e \in \delta(v)} y_e \Big) = \sum_{e \in E} \alpha_e y_e &\leq 2 \sum_{e \in E} y_e \\ &\leq 2LP \qquad (4a) \\ &\leq 2OPT, \qquad (4b) \end{aligned}$$

where $\alpha_e = 2$, for edge $e = (u, v)$ if both $u, v \in C$, 1 otherwise. The inequality (4a) follows from weak duality, and inequality (4b) follows from the fact that the primal LP is a relaxation of the vertex cover problem. □

Figure 3 illustrates the execution of the primal-dual algorithm on a graph. For this instance, the algorithm returns a vertex cover of cost 9, whereas the optimal solution in this instance has

cost 7 (corresponding to the two vertices on the diagonal edge). The lower bound given by the dual solution has value $3 + 1 + 1 = 5$.

A few final remarks:

- Dinur and Safra [2] have proved that it is NP-hard to approximate to the vertex cover with a factor better than 1.36.
- Currently, there is no algorithm for the vertex cover problem which achieves an approximation ratio better than 2. So the two (simple!) algorithms presented here are, in fact, the present best known approximation algorithms for this problem.
- Khot and Regev [3] have proved that it is UGC-hard to approximate vertex cover within a factor $2 - \varepsilon$, for any $\varepsilon > 0$.

References

[1] Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, CMU.

[2] Dinur, I. and S. Safra (2002). The importance of being biased. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pp. 33-42.

[3] Khot, S. and O. Regev (2008). Vertex cover might be hard to approximate to within $2 - \varepsilon$. *Journal of Computer and System Sciences*, 74:335-349.

[4] Papadimitriou, C.H. and S. Vempala (2000). On the approximability of the travelling salesman problem. In Proceedings of the 32nd ACM Symposium on Theory of Computing, pp. 126-133.

[5] Papadimitriou, C.H. and M. Yannakakis (1993). The travelling salesman problem with distances one and two. *Mathematics of Operations Research*, 18:1-11.