CS880: Approximation and Online Algorithms	**Scribe:** Evangelia Gergatsouli
Lecture 1: Approximation Algorithms	**Date:** 04 Sept 2019

1.1 Introduction

Computers nowadays are asked to solve all sorts of problems, and optimization problems are a large part of them. An optimization problem consists of a set of constraints and an objective function. We need to find a *feasible* solution, i.e. one solution that satisfies the constraints, and the value of this solution given by the objective function should be the minimum (or maximum) possible.

Many of the interesting optimization problems we are asked to solve are NP-Complete. This means that unless $P = NP$ we cannot find an algorithm that solves them efficiently i.e. in polynomial time. So, what do we do in this case? One way to confront this problem is by using approximation algorithms. This essentially means that we relax the optimality constraint in order to get a better running time, and hopefully we will not be far from the optimal solution. Such an algorithm will always return a feasible solution but not an optimal one.

In this set of notes, we start with some preliminaries and notation, and then we describe two NP-Hard optimization problems and we wil discuss two simple algorithms that solve them aproximately. The problems are Vertex Cover and Steiner Tree.

1.2 Preliminaries and Notation

We start with some preliminary notation on graphs, and some definitions for approximation algorithms.

1.2.1 Graph Notation

We denote by $G(V, E)$ a graph with vertex set V and edge set E. The graph G will be undirected unless otherwise specified. We now give some definitions regarding graphs, that will be used in the following sections.

Definition 1.2.1 (Spanning Subgraph) *A* spanning *subgraph of G is a connected subgraph obtained only by the deletion of edges.*

Definition 1.2.2 (Closed walk) *A* walk *in graph G is a sequence $W := v_0e_1v_1 \dots v_{\ell-1}e_\ell v_\ell$, whose terms are alternately vertices and edges of G (not necessarily distinct), such that v_{i-1} and v_i are the ends of e_i, $1 \leq i \leq \ell$.*

A walk is closed *if its initial and terminal vertices are the same.*

Definition 1.2.3 (Tour) *A* tour *of a connected graph G is a closed walk that traverses each edge of G at least once*

Note that a tour can visit some node multiple times. Next we define what is a matching and a

maximal matching in a graph G.

Definition 1.2.4 (Matching) *A* matching *of a graph G is a subset of edges such that no two edges share an endpoint. The size of the matching is the number of edges included in it, and is denoted by $|M|$.*

Definition 1.2.5 (Maximal Matching) *A matching is* maximal *if we cannot of a graph G is a subset of edges such that no two edges share an endpoint.*

Observe that a *maximal* matching is not necessarily a *maximum* matching, as seen in the example of figure 1.2.1.

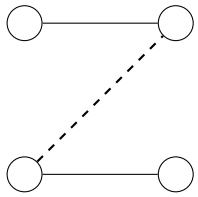

Figure 1.2.1: The bold edge is a maximal but not a maximum matching

1.2.2 Approximation algorithms

As we said in the introduction, one way to get a solution to an NP-Hard optimization problem is to relax the optimality constraint. For example, in a minimization problem, we may find a solution larger than the optimal, but not very far. This notion of "how far" we are from the optimal solution is quantified with the approximation ratio. Denote by F the set of feasible solutions i.e. the solutions that satisfy the given set of constraints and by Obj the objective function. From the definition of an optimal solution, it can be written as $\text{OPT}(F, Obj) = \min_{x \in F} Obj(x)$. Then an approximation algorithm may return $x' \in F$ such that $Obj(x') \neq \text{OPT}$. We denote by $ALG(F, Obj)$ the solution given by our algorithm when the feasible set is F and the objective function Obj. We define the approximation ratio to be the worst ratio between the value of the objective for the algorithm and the optimal. Formally:

Definition 1.2.6 (Approximation ratio) *We define the approximation ratio r of an algorithm as*

$$r = \max_{(F,Obj)} \frac{\text{ALG}(F, Obj)}{\text{OPT}(F, Obj)}$$

Therefore, the approximation ratio is essentially a guarantee of how "bad" at most the algorithm can perform. So how do we calculate this since we do not know OPT? One trick is to find a lower bound for the optimal value, sometimes exploiting the similarity of the NP-Hard problem to some easier one, and obtaining a lower bound this way. Now the ratio between the value of ALG and the lower bound is an upper bound for the approximation ratio. This relation is depicted more clearly in figure 1.2.2.

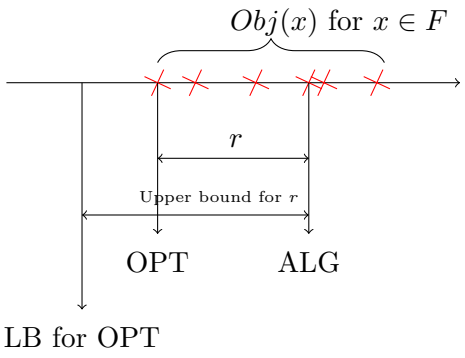

Figure 1.2.2: Relation between the approximation ratio r and lower bound for OPT

1.3 Vertex Cover

We start by defining the Vertex cover problem. We are given an undirected graph $G(V, E)$, and the objective is to find a set $S \subseteq V$ such that for every edge $(u, v) \in E$ either $v \in S$ or $u \in S$. This is one of Karp's 21-NP Complete problems [4] (mentioned as *Node Cover*). It is also shown by Dinur and Safra [1] to be NP-hard to approximate within 1.36. Additionally, under a stronger conjecture (the Unique Games Conjecture), Khot and Regev [6] showed that it is NP-hard to approximate within $2 - \epsilon$. The best approximation algorithm so far [5] gives a $2 - \Theta(1/\sqrt{\log n})$ approximation.

In this section we will show a simple 2-approximation algorithm, using the idea of finding another -similar- problem to be used as a lower bound for the optimal value of our problem. In this case this would be the matching problem.

Claim 1.3.1 *For any matching M in G and any vertex cover S it holds that $|S| \geq |M|$*

Proof: Observe that for every edge in a matching, we need to include at least one of its endpoints in the vertex cover, in order to cover this particular edge. Therefore $|S| \geq |M|$. ■

An immediate corollary of the previous claim is that: $\min_{S \text{ vertex cover}} |S| \geq \max_{M \text{ matching}} |M|$. Using this claim, we will lower bound the size of the optimal vertex cover by the size of any matching, as seen in figure 1.3.3. Using these observations we construct the following algorithm:

Algorithm 1: Vertex Cover Algorithm

Data: Graph $G(v, E)$
1 Find a maximal matching M in G
2 $S = \{u \in V : \exists (u, v) \in M\}$
3 **return** S

We will initially show that the set the algorithm returns is feasible; i.e. that S is a vertex cover.

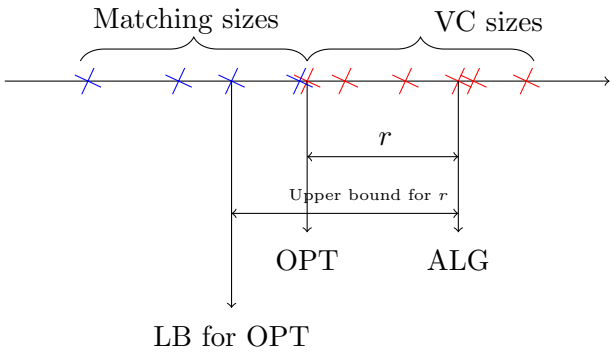

Figure 1.3.3: Relation between matching and vertex cover sizes

Claim 1.3.2 *S is a vertex cover*

Proof: Assume that S is not a vertex cover, then there should exist an edge $e = (u, v) \in E$ such that $v \notin S$ and $v \notin S$. This means that we could add this edge to M and get a matching that has larger size. This is impossible since M is a maximal matching. ■

Theorem 1.3.3 *Algorithm 1 gives a 2-approximation to the Vertex Cover problem*

Proof: Observe that the set of nodes the algorithm returns has size twice that of the maximal matching. But from Claim 1.3 we know that the size of any matching is a lower bound for OPT. Putting this together we get $\text{ALG} = |S| = 2|M| \leq 2\,\text{OPT}$. ■

1.4 Steiner Tree

In this problem we are given a graph $G(V, E)$, a set $T \subseteq V$ of terminal vertices and a function $\ell : E \to \mathbb{R}_+$ which assigns a length ℓ_e to every edge $e \in E$. The goal is to find the minimum cost tree spanning T. We denote by $\text{cost}(S) = \sum_{e \in S} l_e$ the total cost of the set S of edges.

This problem is also shown to be NP-Hard by Karp [4] and the best algorithm at this until now gives a 1.39 approximation [2]. Additionally, this problem was also shown in [3] to be hard to approximate within 96/95.

As a first attempt to solve this, we will drop all non terminal vertices from the graph and find the Minimum Spanning Tree. Consider the graph shown in figure 1.4.4 for example. After removing the middle node, we can indeed find the minimum cost spanning tree for the terminals.

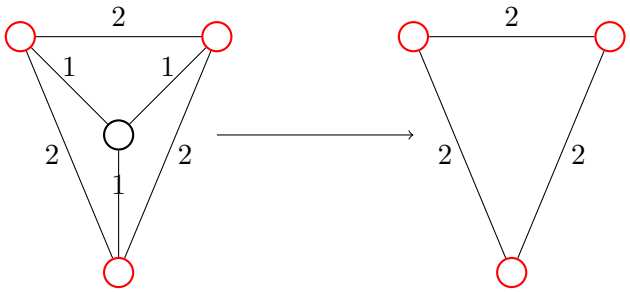

Figure 1.4.4: Graph after removing the non terminal nodes

The problem with this approach is that after removing the non terminals, we might end up with a graph that is not connected, as shown in figure 1.4.5 for example.

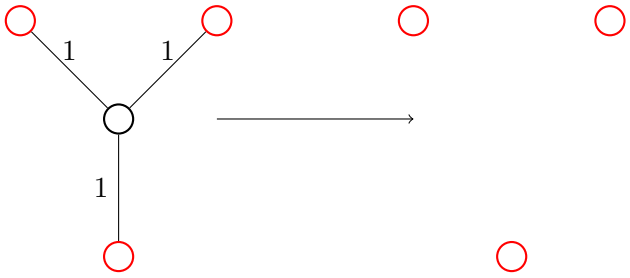

Figure 1.4.5: Bad case of a graph after removing the non terminal nodes

Now, we build on this initial idea while ensuring that we always get a feasible solution that is also close to the optimal one. To do this, we define a graph $G(T, T \times T)$ that contains just the terminals and the edges among them. Then, we modify the function ℓ to a function ℓ' such that for every $u, v \in T$ the new function $\ell'_{(u,v)}$ is the length of the shortest path from u to v in the initial graph G. The algorithm is shown below.

Algorithm 2: Steiner Tree Algorithm

Data: Graph $G(V, E)$, set of terminals T, length function ℓ

Create graph $G'(T, T \times T)$
Create new length function ℓ'
Find the MST S' of G'.
$S \leftarrow \emptyset$
foreach $(u, v) \in S'$ **do**
  add to S every edge $e \in G$ that belongs to the shortest path from u to v
end
Remove cycles from S
return S

Theorem 1.4.1 *Algorithm 2 gives a feasible solution to Steiner Tree problem of cost at most* $2\,\mathrm{OPT}$

Proof: The feasibility follows immediately from the algorithm, since we return a tree spanning all terminal nodes. The 2-approximation factor results immediately from the combination of Claim 1.4.2 and Claim 1.4.3. ■

Claim 1.4.2 $cost(S) \leq cost(S')$

Proof: Observe that in order to obtain S from S' we make two operations

- The edges between terminals that also exist in the initial graph G remain the same
- The edges (u, v) between terminals that do not exist in G are replaced by the set of edges that are part of the shortest path between the nodes u and v

Both these operations do not change the cost, as it follows from the way we constructed the graph G' and the new cost function ℓ'. After this step however, we will remove cycles from S, which may result in the cost being reduced. Therefore the claim follows. ■

Claim 1.4.3 $cost(S') \leq 2\,\mathrm{OPT}$

Proof: We will show that there exists a spanning tree of cost less than $2\,\mathrm{OPT}$ in G'. Then since S' is the minimum spanning tree in G', the claim follows. Imagine we traverse the optimal tree in some way (for example using DFS). This traversal gives us a cycle of length exactly $2\,\mathrm{OPT}$. Now, in this cycle, imagine we remove all non terminal nodes, and connect the consecutive terminals with the cost of the path between them, keeping only the first occurrence of every terminal node. This process will result in a tree connecting all terminal nodes, that has cost at most $2\,\mathrm{OPT}$ ■

1.5 Travelling Salesman Problem

In the Travelling Salesman Problem we are given a graph $G(V, E)$, and a function $\ell_e :\to \mathbb{R}_+$ of lengths on the edges. The goal is to find the shortest spanning tour. This is also called the *symmetric* TSP since the graph is undirected and the distance function is symmetric ($\ell(u, v) = \ell(v, u)$). More on the TSP in the next lecture.

References

[1] Dinur Irit, Safra Samuel. On the Hardness of Approximating Minimum Vertex Cover *Annals of Mathematics, vol 162, 2004*

[2] Byrka Jaroslaw, Grandoni Fabrizio, Rothvoß Thomas, Sanità Laura. An Improved LP-based Approximation for Steiner Tree, *Proceedings of the Forty-second ACM Symposium on Theory of Computing,2010*

[3] Chlebik Miroslav, Chlebikova Janka. The Steiner tree problem on graphs: Inapproximability results, *Theoretical Computer Science, vol 406, 2008*

[4] Karp Richard. Reducibility Among Combinatorial Problems, *Complexity of Computer Computations, vol 40, 1972*

[5] Karakostas George. A Better Approximation Ratio for the Vertex Cover Problem, *ACM Transactions on Algorithms - TALG, 2005*

[6] Khot Subhash, Regev Oded Vertex cover might be hard to approximate to within $2-\epsilon$, *Journal of Computer and System Sciences, Vol 74, Issue 3, Pages 335-349, 2008*