APPROXIMATION ALGORITHMS

Lecture 9

Notes by Roman Vakhrushev
For CS-GY 6043 Design and Analysis of Algorithms II
11/17/2023 (Updated: 12/6/2023)

Contents

Motivation: As we have seen in the last few lectures, there are certain problems that we cannot afford to solve exactly. In order to address this issue, we use **approximation algorithms** that can provide us with some approximate solutions.

I Metric Space

Def: A metric space is an ordered pair (X, d), where X is the set of points, and d is a metric defined as $d : X^2 \to \mathbb{R}_0^+$ satisfying the following:

For all $a, b, c \in X$:

- $d(a, b) \geq 0$ and $d(a, b) = 0$ iff $a = b$.
- $d(a, b) = d(b, a)$ (symmetry)
- $d(a, c) \leq d(a, b) + d(b, c)$ (triangle inequality)

We now take a look into some problems that can be approximately solved. The concept of metric spaces is useful in the design and analysis of such approximation algorithms.

II Diameter Problem

II.1 Problem Formulation

Given a set of points S, we want to find the diameter of S, $diam(S) = \max_{a,b,\in S} d(a, b)$.

II.2 Naïve solution

Consider a naïve solution to this problem. Suppose $S = |n|$, then we can check all points in this set and in $\Theta(n^2)$ can compute $diam(S)$ (This is a brute force solution). This running time can be improved by the use of approximation algorithms.

II.3 Approximation Algorithm

Approximation Algorithm:

1. choose any point $a \in S$
2. compute and return $\delta = \max_{x \in S} d(a, x)$

II.4 Analysis of the Approximation Algorithm

Claim: Δ is an approximation to $d = diam(S)$.

Pf:

1. Note that $\Delta \leq d$ by definition of $diam(S)$.

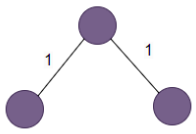
2. Suppose $d = d(x, y)$. Let point a be the point picked by our algorithm. Then:

$$\begin{aligned} d &= d(x, y) \\ &\leq d(x, a) + d(a, y) && \text{(By triangle inequality)} \\ &= d(a, x) + d(a, y) && \text{(By symmetry)} \\ &\leq 2\Delta && \text{(By the way we define } \Delta\text{)} \end{aligned}$$

Combining parts 1 and 2 of the proof, we have $\frac{d}{2} \leq \Delta \leq d$. So, our approximation is bounded above and below by the value of the real diameter times some constant (For the upper bound this constant is 1, for the lower bound this constant is 1/2). This is called **multiplicative error approximation**.

Note: This algorithm has a running time of $\Theta(n)$.

Additional Note: Consider the example below. We can deduce that the analysis we gave above (approximation factor is 2) is worst-case correct because of this example.



III Formal Definition of α-approximation

Suppose we have some optimization problem (to find min or max).

Let:

- X be an instance of a problem
- A be an approximation algorithm
- $OPT(X)$ be the quality of the optimum solution (in the case of the Diameter Problem, it would be a real diameter)
- $A(X)$ be the quality of solution by A (in the case of the Diameter problem, it would be the output of our approximation algorithm)

We say that A is an $\alpha(n)$-approximation if for all possible inputs:

$\frac{OPT(X)}{A(X)} \leq \alpha(n)$ and $\frac{A(X)}{OPT(X)} \leq \alpha(n)$

This type of approximation is called **multiplicative factor-$\alpha(n)$ approximation**.

IV Load Balancing Problem

IV.1 Problem Formulation

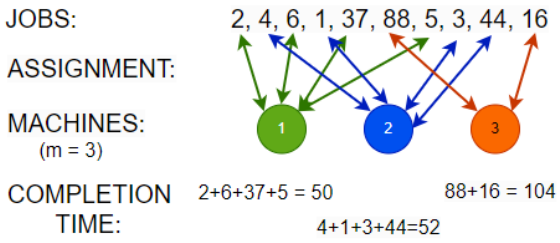
Suppose we have m machines which all have equal capabilities. Then we can specify the load balancing problem in the input-output format.

Input: $T[1], T[2], \ldots, T[n]$ job durations,

Output: An assignment (for each job, which machine does the job) minimizing **makespan** (maximum completion time for any machine, where completion time is just the sum of job durations for this machine).

Note this problem is NP-hard since the Partition problem reduces to 2-Machine-Load Balancing.

Ex: The example below shows a possible job assignment as arrows. The completion time is computed for this particular assignment.



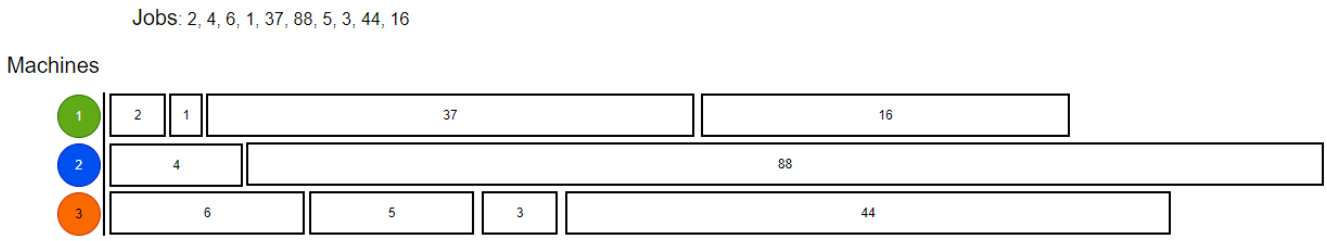
Also note that in this case, the **makespan** is computed as $\max(50, 52, 104) = 104$.

IV.2 Greedy LB Algorithm

Greedy LB Algorithm:

1. Consider jobs in some order
2. Repeatedly assign a current job to the currently least busy machine

Ex: The result of running the algorithm on the same data is shown below:



IV.3 Analysis of Greedy LB Algorithm

Claim: Our solution cannot be worse than 2 · optimum solution, meaning Greedy LB is a 2-approximation.

Pf:

We start by observing the following:

1. $OPT \geq \max_j T[j]$ (Obvious observation)
2. $OPT \geq \frac{1}{m} \sum_j T[j]$ (OPT is never smaller than the average completion time per machine)

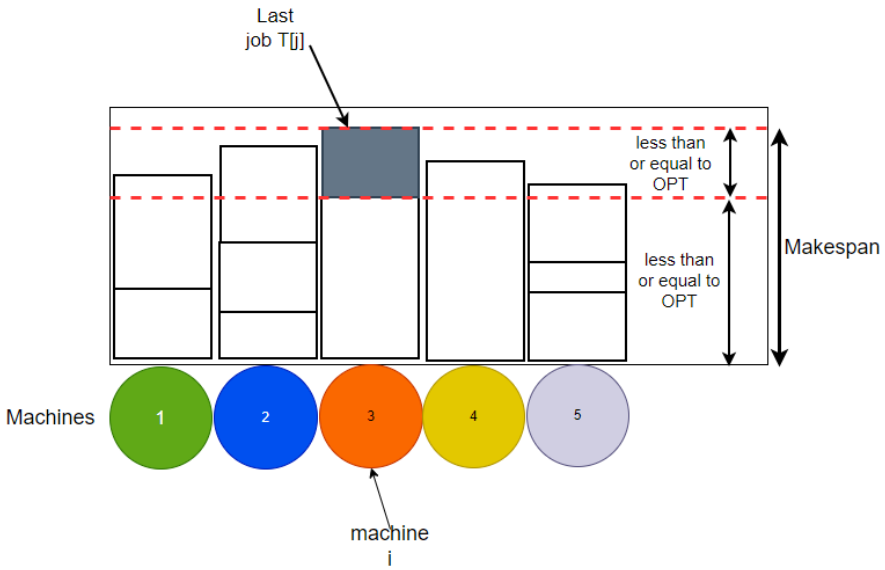
Now consider some machine i with the largest total running time. Consider the last job j assigned to this machine. By the observation 1, $T[j] \leq OPT$.

Denote the sum of jobs assigned to any machine k (after all jobs are processed) by $Total[k]$. Now consider the value of $Total[i] - T[j]$. By design of our algorithm, job j was assigned to machine i because this machine at that time had the smallest total running time among all the machines. That is $Total[i] - T[j] \leq Total[k]$ for any machine k at the time job j was assigned. Note that it is possible that later more jobs would be assigned to machine k, but this would only make $Total[k]$ larger.

Therefore, combining this result with observation 2, we have $Total[i] - T[j] \leq \frac{1}{m} \sum_{i=1}^{m} Total[i] = \frac{1}{m} \sum_k T[k] \leq OPT$

Since $T[j] \leq OPT$ and $Total[i] - T[j] \leq OPT$, $Total[i] - T[j] + T[j] = Total[i] \leq 2OPT$.

The illustration of this proof is shown below:



IV.4 Greedy Sorted LB Algorithm

Although the standard Greedy LB Algorithm is a 2-approximation, we can still improve the algorithm. Instead of taking the list of jobs as given, sort it first (in decreasing order). We then do everything the same as in the standard Greedy LB Algorithm, this algorithm is sometimes called **Greedy Sorted LB Algorithm**. It can be shown that it is a $\frac{3}{2}$-approximation.

V Vertex Cover

V.1 Problem Formulation

Given a graph $G = (V, E)$. We want to find the largest vertex cover C (a set of vertices that "touches" all edges).

There are at least 2 simple approximation algorithms that can solve this problem.

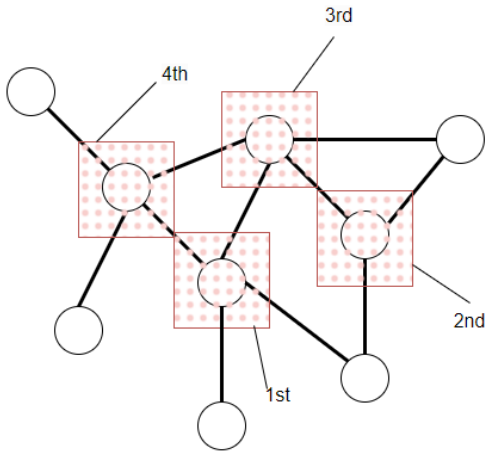
V.2 Greedy Vertex Cover Algorithm

Greedy Vertex Cover Algorithm:

- Repeatedly add a vertex such that it touches the maximum number of uncovered edges

Ex :

The result of running the Greedy Vertex Cover Algorithm algorithm on some graph is shown below. For each selected vertex, we show when this vertex was selected.



Note : The Greedy Vertex Cover algorithm is just a special case of a Greedy Set Cover algorithm. Therefore, it can be shown that The Greedy Vertex Cover is a $O(\log n)$-approximation.

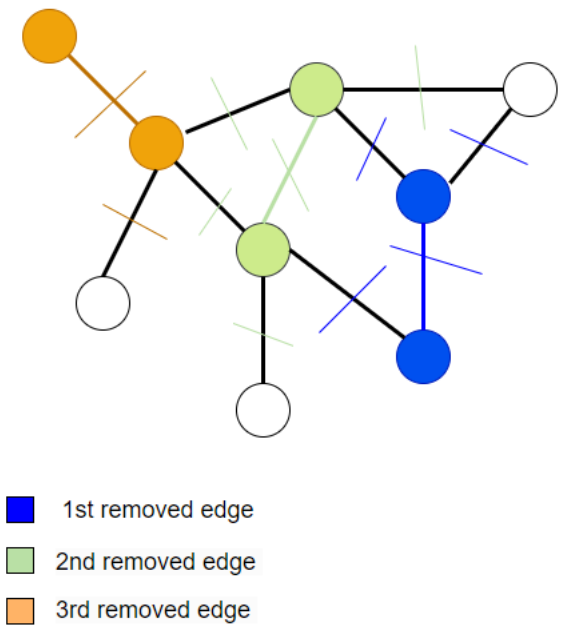
V.3 Dumb Vertex Cover Algorithm

Dumb Vertex Cover Algorithm:

- While G has still any edge left: (u, v)
  - Remove u, v from the vertex set
  - Remove all incident edges from the edge set
  - Add u, v to answer
- Return answer

Ex :

The result of running the Dumb Vertex Cover Algorithm algorithm on the same graph is shown below:



Note that Dumb Vertex Cover Algorithm is always a 2-approximation. Final answer has at least one vertex from each edge. Therefore, $OPT \leq C \leq 2OPT$.

VI Travelling Salesman Problem (TSP)

VI.1 Problem Statement

Given a weighted graph, find the Hamiltonian cycle with the smallest sum of weights.

We consider an undirected version of the TSP problem. We also impose an additional assumption that triangle inequality holds for all weights on edges (that is for all i, j, k we always have $w[i, j] \leq w[i, k] + w[k, j]$).

Note 1: It turns out that without the triangle inequality assumption, this problem cannot be approximated to any reasonable factor in polynomial time (unless $P = NP$). Equivalently, we can state that without this assumption, TSP is not only NP-hard to solve exactly, but even to approximate to any reasonable factor.

Note 2: TSP with weights 1,2, satisfies the triangle inequality constraint and it is already NP-hard. This is because the Hamiltonian Path Problem (HAM) reduces to it.

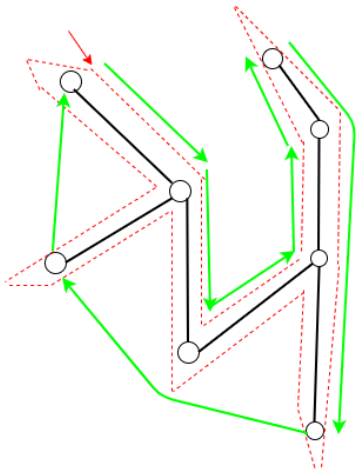
VI.2 Approximation Algorithm №1

Approximation Algorithm №1:

1. Given a graph G, find Minimal Spanning Tree of G. (MST)
2. Double edges of MST, construct Euler Circuit on it (EC)
3. Using EC, construct Shortcut (SC)

Note: Shortcut is constructed from EC by skipping vertices that we already visited.

Ex: The result of running this approximation algorithm is shown below. The original graph G is not shown. TSP is shown in black, EC is shown in red dash line, SC is shown in green.



VI.3 Analysis of Approximation Algorithm №1

Claim: $SC \leq 2OPT$.

Pf:

It is easy to observe that $OPT \geq MST$. This is because MST is the spanning tree, whose sum of weights is as small as possible. Since OPT needs to be a Hamiltonian cycle, it has to have sum of weights to be at least that of MST.

Then the proof goes as follows:

- $EC = 2MST$ (by construction, we repeat each edge of MST twice in EC)
- $SC \leq EC$ (by triangle inequality)
- $SC \leq EC = 2MST \leq 2OPT$

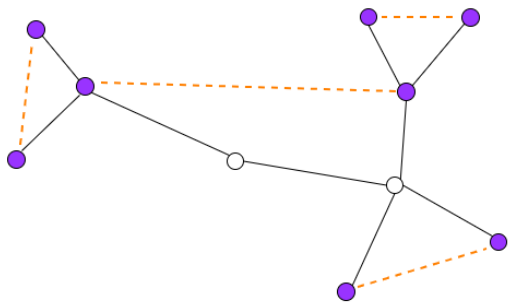
Thus, we have $SC \leq 2OPT$. We also note that obviously, $SC \geq OPT$. So, we have $OPT \leq SC \leq 2OPT$. Thus, this algorithm is a 2-approximation.

VI.4 Approximation Algorithm №2 (Christofides' TSP)

Christofides' TSP:

1. Given a graph G compute Minimum Spanning Tree (MST)
2. Let X be a set of odd-degree vertices in MST (X) (**Note:** $|X|$ is even)
3. Compute Min cost matching on X (MCM)
4. Combine $MST + MCM$ (**Note:** this "graph" is multigraph - some edges can be repeated). ($MST + MCM$)
5. This multigraph has all vertices with even degrees, so we can find an Euler Circuit, with cost of $MST + MCM$. (EC)
6. Compute Shortcut on EC (SC)

Ex: We show the result of applying the first 4 steps of the algorithm on some graph (original graph not shown). MST is shown in black, X is shown in purple, MCM is shown in orange dash line, everything together is $MST + MCM$. We would still have to do SC to finish the algorithm.



,

VI.5 Analysis of Christofedes' Approximation TSP:

Claim 1 $MCM \leq \frac{1}{2}OPT$

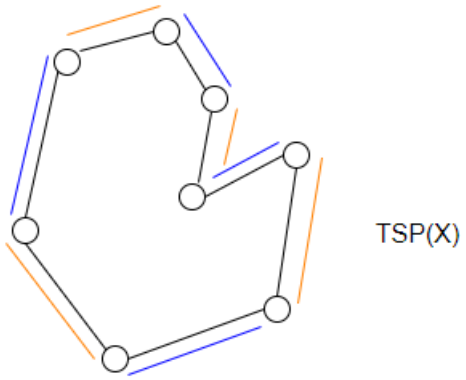
Pf:

Let X be any set of vertices such that $|X|$ is even.

Let

$MCM(X)$ - cost of min-cost perfect matching in X.

$TSP(X)$ - min cost TSP on X.



Consider $TSP(X)$ (shown above). Notice that it has two different disjoint matchings (shown in blue and orange). Each of these matchings is at least $MCM(X)$. But together, they form $TSP(X)$. Thus, $2MCM(X) \leq TSP(X)$. By the triangle inequality, $TSP(X) \leq TSP(V) = OPT$ (V is the vertex set of the original graph). So, $2MCM \leq OPT$ and thus $MCM \leq \frac{1}{2}OPT$.

Claim 2: $SC \leq \frac{3}{2}OPT$

Pf:

First, we can deduce $SC \leq EC \leq MCM + MST$. (By construction).

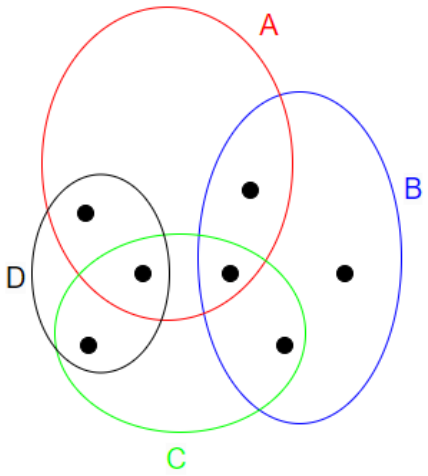
The claim $MST \leq OPT$ was shown in the analysis of the previous algorithm (Section $VI.3$). Using this result, Claim 1, and inequality $SC \leq EC \leq MCM + MST$ we have $SC \leq MCM + MST \leq OPT + \frac{1}{2}OPT = \frac{3}{2}OPT$.

VII Set Cover

VII.1 Problem Formulation

Given sets $S_1, ..., S_m$, suppose $X = \bigcup_{i=1}^{m} S_i$. Let $C \subset \{S_1, ..., S_m\}$. C is set cover if $\bigcup_{S_i \in C} S_i = X$. The problem is to find a set cover of maximum size.

Ex:



Consider the image above. Black points together form X. Four different sets A, B, C, D are shown as closed curves. Some of the covers include:

1. $\{A, B, C, D\}$
2. $\{A, B, C\}$
3. $\{D, B\}$

In contrast, $\{A, D\}$ or $\{B, C\}$ are **not** covers.

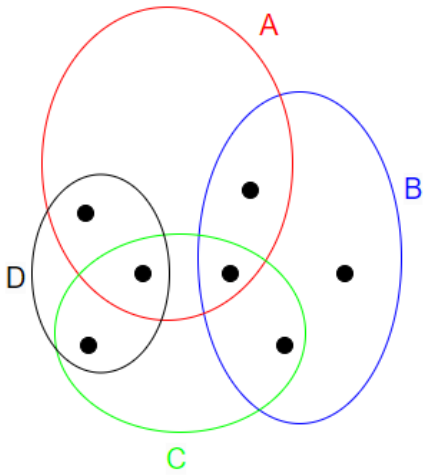
Side note: Everything beyond this point was covered in lecture 10 (not in lecture 9), but was added here for completeness.

VII.2 Greedy Set Cover Algorithm

Greedy Set Cover Algorithm

- Repeat until we do not form a cover:
  - Pick a set that covers max number of still uncovered elements.

Ex: For the same problem as above:



The algorithm may produce all of the possible outcomes:

1. $\{B, D\}$
2. $\{A, C, B\}$
3. $\{A, B, D\}$
4. $\{A, B, C\}$
5. $\{C, A, B\}$
6. $\{C, B, A\}$
7. $\{C, B, D\}$

There are many different possible outcomes. This is due to the algorithm being **underspecified** (i.e. ties are broken arbitrarily).

Note: Not all covers computed in this example are min size. In general, this heuristic does not guarantee optimal cover.

VII.3 Analysis of the Greedy Set Cover Algorithm

Let $n = |X|$ and $k = OPT = \#$ of sets in min set cover.

Let $n_i = \#$ of elements left uncovered after picking set i.

$n_0 = n$

At step $i + 1$, we have n_i elements left OPT has to cover later. This implies there exists a set that covers $\frac{n_i}{k}$ elements (by pigeonhole principle).

Therefore, $n_{i+1} \leq n_i - \frac{n_i}{k} = n_i(1 - \frac{1}{k})$.

Applying this recurrence relation to $n_0 = n$, we have $n_i \leq n(1 - \frac{1}{k})^i$.

Note: For all x, $1 - x \leq e^{-x}$ and $1 - x = e^{-x}$ iff $x = 0$.

Using this fact, $n_i \leq n[e^{-\frac{1}{k}}]^i = ne^{-\frac{i}{k}}$.

We want $n_i < 1$ (this would imply $n_i = 0$ since n_i is an integer). So, we solve the following equation $ne^{-\frac{i}{k}} = 1$ for i, which has the solution $i = k \ln n$.

This means, that after $i = k \ln n$ rounds, the algorithm stops (since there are no more elements to process).

During each round, we select one of the sets to cover some of the points. Since there are at most $k \ln n$ rounds, we use $k \ln n$ sets. Optimal solution (OPT) uses exactly k sets. Therefore, Greedy Set Cover is a factor($\ln n$)-approximation.