Approximation: Greedy and Local Search

Zixuan Fan

Technische Universität München

July 2024

Agenda

1. Introduction
2. Scheduling Problems
   - Scheduling with deadlines on a Single machine
   - Scheduling on Multiple Machines
3. Graph Problems
   - K-center Problem
   - Travelling Salesman Problem
4. Conclusion

Purpose of Approximation

- Some problems not efficiently solvable: NP-hardness

Purpose of Approximation

- Some problems not efficiently solvable: NP-hardness
- Can we find a flawed solution?

Purpose of Approximation

- Some problems not efficiently solvable: NP-hardness
- Can we find a flawed solution?
- How flawed/good is this solution?

Purpose of Approximation

- Some problems not efficiently solvable: NP-hardness
- Can we find a flawed solution?
- How flawed/good is this solution?
- What is the limit of this flawed solution?

Quick Recap: Approximation Ratio

Opt^*: the optimal solution
Opt: the suboptimal solution we compute

For minimization problems, we have

$$\alpha = \frac{|Opt|}{|Opt^*|} > 1$$

And for maximization problems

$$\alpha = \frac{|Opt|}{|Opt^*|} < 1$$

Techniques in Approximation

1. Randomization: MAXSAT, MAXCUT

Techniques in Approximation

1. Randomization: MAXSAT, MAXCUT
2. Dynamic Programming: Knapsack, Bin Packing

Techniques in Approximation

1. Randomization: MAXSAT, MAXCUT
2. Dynamic Programming: Knapsack, Bin Packing
3. Linear & Integer Programming: Primal-Dual, Semidefinite Program

Techniques in Approximation

1. Randomization: MAXSAT, MAXCUT
2. Dynamic Programming: Knapsack, Bin Packing
3. Linear & Integer Programming: Primal-Dual, Semidefinite Program
4. Greedy & Local Search

Greedy and Local Search

- Both strategies attempt to make the best decision

Greedy and Local Search

- Both strategies attempt to make the best decision
- Greedy algorithms forms a solution step by step

Greedy and Local Search

- Both strategies attempt to make the best decision
- Greedy algorithms forms a solution step by step
- Local Search starts search from an arbitrary solution

Scheduling with deadlines on a Single machine

Problem Statement: Given n jobs to be processed on a single machine. How can we schedule them such that they will finish as early as possible.

- What if there is not deadline?
- How do we define earlieness/lateness?

Scheduling with deadlines on a Single machine - Formal Definition

Suppose job j is finished at time C_j, the lateness is

$$L_j := C_j - d_j$$

The lateness of all jobs is

$$L_{max} = \max_{i \in [n]} L_i$$

Input: n jobs with release time r_j, processing time p_j, and deadline d_j.

Output: a schedule such that L_{max} is minimized.

Algorithm - Intuition

If the job j is finished before the deadline we don't get penalty for it, so ...

1. What if we always choose the job with the earliest deadlines

Algorithm - Intuition

If the job j is finished before the deadline we don't get penalty for it, so . . .

1. What if we always choose the job with the earliest deadlines
2. But, we may have deal with negative values.

Algorithm - Intuition

If the job j is finished before the deadline we don't get penalty for it, so . . .

1. What if we always choose the job with the earliest deadlines
2. But, we may have deal with negative values.
3. So all deadlines are negative (by assumption).

Algorithm - Analysis

We start with an observation. Let S be a set of jobs,

- $r(S) := \min_{j \in S} r_j$
- $p(S) := \sum_{j \in S} p_j$
- $d(S) := \max_{j \in S} d_j$

We claim that

$$L^*_{max} \geq r(S) + p(S) - d(S)$$

This is proven by considering the optimal schedule.

Algorithm - Analysis

$$L_{max}^* \geq r(S) + p(S) - d(S)$$

This leads directly to

$$L_{max}^* \geq r(j) + p(j) - d(j) \geq -d_j$$

where j is the job that leads to the maximal lateness.

Algorithm - Analysis

$$L^*_{max} \geq -d_j$$

Recall $L_{max} = C_j - d_j$, it suffices to show

$$L^*_{max} \geq C_j$$

Algorithm - Analysis

We start with an ideal scenario: all jobs are released at time $t = 0$. We have

- $r(S) = 0$
- $p(S) = C_j$
- $d(S) < 0$, by assumption

From lemma it follows that

$$L^*_{max} \geq r(S) + p(S) - d(S) \geq p(S) = C_j$$

Algorithm - Analysis

We start with an ideal scenario: all jobs are released at time $t = 0$.
We have

- $r(S) = 0$
- $p(S) = C_j$
- $d(S) < 0$, by assumption

From lemma it follows that

$$L^*_{max} \geq r(S) + p(S) - d(S) \geq p(S) = C_j$$

How about the general cases?

Algorithm - Analysis

How about the general cases?

- We consider it in a way of calculus

Algorithm - Analysis

How about the general cases?

- We consider it in a way of calculus
- Find the time t such that $[t, C_j]$ has no idle time

Algorithm - Analysis

How about the general cases?

- We consider it in a way of calculus
- Find the time t such that $[t, C_j]$ has no idle time
- Denote jobs processed in this interval with S.

Algorithm - Analysis

How about the general cases?

- We consider it in a way of calculus
- Find the time t such that $[t, C_j]$ has no idle time
- Denote jobs processed in this interval with S.
- $r(S) = t$, $p(S) = C_j - t$

Algorithm - Analysis

How about the general cases?

- We consider it in a way of calculus
- Find the time t such that $[t, C_j]$ has no idle time
- Denote jobs processed in this interval with S.
- $r(S) = t$, $p(S) = C_j - t$
- Thus $C_j = r(S) + p(S) \leq L^*_{max}$

Algorithm - Analysis

Summarizing two results $L^*_{max} \geq -d_j$ and $L^*_{max} \geq C_j$, we obtain

$$L_{max} = C_j - d_j \leq 2L^*_{max}$$

Scheduling on Identical Paralle Machines

Problem Statement: Given n jobs to be processed on k machines, How can we schedule them such that they will finish as early as possible.

- What if there is not deadline?
- How do we define earlieness/lateness?

Scheduling on Identical Paralle Machines - Formal Definition

More machines $\implies$ more complexity

So the problem is **NP-hard** even if

we don't consider release time and deadlines

Scheduling on Identical Paralle Machines - Formal Definition

Input: n jobs with processing time p_j and k machines

Output: A schedule that minimizes

$$\max_{j \in [n]} C_j$$

where C_j is the completion time of job j.

Algorithm - Local Search

Recap: Local search

- starts with a valid solution

Algorithm - Local Search

Recap: Local search

- starts with a valid solution
- searches greedily until

Algorithm - Local Search

Recap: Local search

- starts with a valid solution
- searches greedily until
- no better strategy can be made

Algorithm - Local Search

Recap: Local search

- How do we find a valid solution?
- How to make greedy strategy?

Algorithm - Local Search

Recap: Local search

- How do we find a valid solution?

- How to make greedy strategy?

Algorithm - Local Search

Recap: Local search

- How do we find a valid solution?
- Finish all jobs in one machine.
- How to make greedy strategy?

Algorithm - Local Search

Recap: Local search

- How do we find a valid solution?
- Finish all jobs in one machine.
- How to make greedy strategy?
- Separate those jobs to other machines.

Local Search - Seperation of Jobs

- Pick the last job j at the machine M with the latest running time
- Let t be the processing time of all other jobs on M
- Find another machine M' such that $t' < t$ is minimized
- Add j to M'

Local Search - Seperation of Jobs

- Pick the last job j at the machine M with the latest running time
- Let t be the processing time of all other jobs on M
- Find another machine M' such that $t' < t$ is minimized
- Add j to M'

But, can we go into LOOPs ?

Local Search - Analysis

We show that a job is only assigned **once** in our local search algorithm.

Suppose that a job j on machine M is assigned for a **second** time.

Local Search - Analysis

We show that a job is only assigned **once** in our local search algorithm.

Suppose that a job j on machine M is assigned for a **second** time.

$\implies$ machine M has the latest running time

Local Search - Analysis

We show that a job is only assigned **once** in our local search algorithm.

Suppose that a job j on machine M is assigned for a **second** time.

$\implies$ machine M has the latest running time

$\implies$ there is another machine M' such that $t' < t$

Local Search - Analysis

We show that a job is only assigned **once** in our local search algorithm.

Suppose that a job j on machine M is assigned for a **second** time.

$\implies$ machine M has the latest running time

$\implies$ there is another machine M' such that $t' < t$

BUT: by our strategy t has to be minimal when j was assigned at an earlier step.

Local Search - Analysis

We show that a job is only assigned **once** in our local search algorithm.

Suppose that a job j on machine M is assigned for a **second** time.

$\implies$ machine M has the latest running time

$\implies$ there is another machine M' such that $t' < t$

BUT: by our strategy t has to be minimal when j was assigned at an earlier step.

We denote completion time on M' at this step as t'_0, it holds

$$t' \geq t'_0 \geq t \; \lightning$$

Local Search - Analysis

What about the approximation ratio?

Local Search - Analysis

What about the approximation ratio?

- Completion time on each machine should be close to mean:

$$\frac{\sum_{j\in[n]} p_j}{n}$$

- It's also the case for the optimal solution, and

$$|Opt^*| \geq \frac{\sum_{j\in[n]} p_j}{k}$$

- We show

$$|Opt| \leq \frac{2\sum_{j\in[n]} p_j}{k} \leq 2|Opt^*|$$

Local Search - Analysis

We show

$$|Opt| \leq \frac{2\sum_{j\in[n]} p_j}{k}$$

- Consider the machine M with the longest completion time
- Job j that completes last with processing time p_j
- All other jobs completes at t.

We have

$$|Opt| = t + p_j$$

and we may witness

$$t \leq \frac{\sum_{j\in[n]} p_j}{k}$$

$$p_j \leq \frac{\sum_{j\in[n]} p_j}{k}$$

Algorithm - Greedy and More

- There exists greedy algorithm that solves problem with $\alpha = 2, \frac{4}{3}$
- With rounding and DP, the problem can be approximated with $\alpha = 1 + \epsilon$ for any $\epsilon > 0$

K-Center Problem

Problem Statement: Given n points in a space, choose k centers from them such that the sum of distances between each point and its nearest neighbor is minimized.

K-Center Problem - Formal Definition

$d(p, q)$: distance between p, q
$d(p, S) := \min_{q \in S} d(p, q)$
Input: S, k
Output: $C \subseteq S$ s.t. $|C| = k$ and $\sum_{p \in S} d(p, S)$ is minimized

K-Center Problem - Formal Definition

$d(p, q)$: distance between p, q
$d(p, S) := \min_{q \in S} d(p, q)$
Input: S, k
Output: $C \subseteq S$ s.t. $|C| = k$ and $\sum_{p \in S} d(p, S)$ is minimized

What is the distance function?
Or what property does the distance function have?

Algorithm - Intuition

How do we search **greedily**?

If we have a temporary center $|C| < k$, how to build $|C'| = |C| + 1$?

Algorithm - Description

Find $v \in S \backslash C$ s.t. $d(v, C)$ is maximized.

$C' = C \cup \{v\}$

Algorithm - Description

Find $v \in S \setminus C$ s.t. $d(v, C)$ is maximized.

$C' = C \cup \{v\}$

What is the cost per iteration?

Algorithm - Description

Find $v \in S \backslash C$ s.t. $d(v, C)$ is maximized.

$C' = C \cup \{v\}$

What is the cost per iteration? $O(|S| \cdot |S \backslash V|) = O(n^2)$

Algorithm - Description

But, what is C at the beginning of the iteration?

Algorithm - Description

But, what is C at the beginning of the iteration?

1. The center of all points? $\implies$ Needs another $O(n^2)$ cost
2. Any arbitrary point $\implies$ Constant cost

Algorithm - Description

But, what is C at the beginning of the iteration?

1. The center of all points? $\implies$ Needs another $O(n^2)$ cost
2. Any arbitrary point $\implies$ Constant cost

Is the $O(n^2)$ cost worth it? We will discuss it later.

Analysis

Quadratic cost

Analysis

Quadratic cost

Approximation ratio?

Approximation ratio: Base

Suppose each optimal cluster contains exactly one center by our algorithm.

Approximation ratio: Base

Suppose each optimal cluster contains exactly one center by our algorithm.

If all points in this cluster is covered by the cluster of this new center

$$r \leq 2r^*$$

If they are not covered $\implies$ a shorter radius is ok.

Approximation ratio: Base

Suppose each optimal cluster contains exactly one center by our algorithm.

If all points in this cluster is covered by the cluster of this new center

$$r \leq 2r^*$$

If they are not covered $\implies$ a shorter radius is ok.

But guarantees $\alpha = \dfrac{r}{r^*} = 2$

Approximation ratio: General

What if one optimal cluster contains two or even more centers?

This is solve by our greediness. **HOW?**

Approximation ratio: Improvement?

Can we have a better approximation?

Sadly, NO

Traverlling Salesman Problem(TSP)

Problem Statement: Given a complete weighted graph of n vertices, find the round tour with the minimal cost.

TSP - Formal Definition

Input: $G = (V, d)$
Output: Path π s.t. $\sum_{i=1}^{n-1} w(\pi(1), \pi(2))$ is minimized

TSP - Formal Definition

Input: $G = (V, d)$
Output: Path π s.t. $\sum_{i=1}^{n-1} w(\pi(1), \pi(2))$ is minimized

What is the distance function?
Or what property does the distance function have?

TSP-Limit of Approximation

1. No existent approximation for TSP in general

TSP-Limit of Approximation

1. No existent approximation for TSP in general
2. If there was $\implies$ Hamilton-Cycle solvable in P!

Suppose for each vertice $u, v, w \in V$, triangular inequality holds

$$d(u, v) + d(v, w) \geq d(u, w)$$

TSP-Limit of Approximation

1. No existent approximation for TSP in general
2. If there was $\implies$ Hamilton-Cycle solvable in P!
3. But: metric distance can help us!

Suppose for each vertice $u, v, w \in V$, triangular inequality holds

$$d(u, v) + d(v, w) \geq d(u, w)$$

Algorithm - A bit help from Euler

- Minimum Spanning Tree(MST). The tree that connects all vertices with minimal edge cost.

Algorithm - A bit help from Euler

- Minimum Spanning Tree(MST). The tree that connects all vertices with minimal edge cost.
- Eulerian Graph: connected graph where all vertices have even degrees.

Algorithm - A bit help from Euler

- What if we make a copy of MST?

Algorithm - A bit help from Euler

- What if we make a copy of MST?
- This is make a Eulerian Graph $\implies$ Not a round tour

Algorithm - A bit help from Euler

- What if we make a copy of MST?
- This is make a Eulerian Graph $\implies$ Not a round tour
- What if we remove all but the first occurrences?

Algorithm - A bit help from Euler

- What if we make a copy of MST?
- This is make a Eulerian Graph $\implies$ Not a round tour
- What if we remove all but the first occurrences?
- This is a round tour!

Algorithm - A bit help from Euler

For the size of the tour:

1. $w_{MST} \leq w_{Opt^*}$: Round tour contains trees!
2. $w_G \leq 2wMST$: triangular inequality

$\implies w_G \leq 2w_{Opt^*}$

Algorithm - A bit help from Christofide

- We can do the same trick to all Eulerian Graphs

Algorithm - A bit help from Christofide

- We can do the same trick to all Eulerian Graphs
- As long as it connects all vertices

Algorithm - A bit help from Christofide

- We can do the same trick to all Eulerian Graphs
- As long as it connects all vertices
- How to make MST Eulerian?

Algorithm - A bit help from Christofide

- We can do the same trick to all Eulerian Graphs
- As long as it connects all vertices
- How to make MST Eulerian?
- Make all vertices have **even** degrees

Algorithm - A bit help from Christofide

HOW?

1. Connect all vertices with odd degrees

Algorithm - A bit help from Christofide

HOW?

1. Connect all vertices with odd degrees
2. How many vertices with odd degrees?

Algorithm - A bit help from Christofide

HOW?

1. Connect all vertices with odd degrees
2. How many vertices with odd degrees?
3. Finding a perfect matching

Algorithm - A bit help from Christofide

HOW?

1. Connect all vertices with odd degrees
2. How many vertices with odd degrees?
3. Finding a perfect matching
4. What is the cost? What is the size of the PM?

Algorithm - A bit help from Christofide

For the size of the tour:

1. $w_{MST} \leq w_{Opt^*}$: see 1st part
2. $w_{PM} \leq \frac{1}{2} w_{Opt^*}$:
3. $w_G \leq w_{MST} + wPM$: traingular inequality

$\implies w_G \leq \frac{3}{2} w_{Opt^*}$

Limit of Approximation(Again)

Even if it is possible to do the approximation for metric TSP, we cannot do better than $\alpha < \dfrac{220}{219}$ unless **P** $=$ **NP**