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 form a solution step by step

Greedy and Local Search

- Both strategies attempt to make the best decision
- Greedy algorithms form 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.

- Suppose $d_j < 0$ to reduce complexity

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.

- Suppose $d_j < 0$ to reduce complexity
- How do we define earlieness/lateness?

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.

- Suppose $d_j < 0$ to reduce complexity
- How do we define earlieness/lateness?
- Suppose job j is finished at time C_j, the lateness is

$$L_j := C_j - d_j$$

The lateness of all jobs is

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

Scheduling with deadlines on a Single machine - Formal Definition

Input: n jobs with release time r_j, processing time p_j, and deadline $d_j < 0$.

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

What if we always choose the job with the earliest deadlines?

Earliest Deadline Rule

Algorithm - Analysis

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

$$c_1 L^*_{max} \geq C_j, c_2 L^*_{max} \geq -d_j$$

Algorithm - Analysis

We start with an observation. Let S be a subset 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

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

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

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

- We consider it in a way of calculus

Algorithm - Analysis

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

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

Algorithm - Analysis

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

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

Algorithm - Analysis

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

- We consider it in a way of calculus
- Find the earliest 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 $L^*_{max} \geq C_j$

- We consider it in a way of calculus
- Find the earliest 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 $M \neq M_1$ is assigned to another machine M'.

Local Search - Analysis

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

Suppose that a job j on machine $M \neq M_1$ is assigned to another machine M'.

$\implies$ machine M has the latest running time now

Local Search - Analysis

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

Suppose that a job j on machine $M \neq M_1$ is assigned to another machine M'.

$\implies$ machine M has the latest running time now

$\implies$ on M' it holds $t' < t$

Local Search - Analysis

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

Suppose that a job j on machine $M \neq M_1$ is assigned to another machine M'.

$\implies$ machine M has the latest running time now

$\implies$ on M' it holds $t' < t$

BUT: by our strategy t has to be minimal when j was assigned to M 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 $M \neq M_1$ is assigned to another machine M'.

$\implies$ machine M has the latest running time now

$\implies$ on M' it holds $t' < t$

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

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

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

Algorithm - Greedy and More

- This is a 2-approximation.
- 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 maximal distance between each point and its nearest center 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 $r = \max_{p \in S} d(p, C)$ 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 $r = \max_{p \in S} d(p, C)$ is minimized

We don't require constraints on distance, as we need to do to TSP

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

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

Algorithm - Description

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

Any arbitrary point would do!

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

$$r \leq 2r^*$$

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

$$r \leq 2r^*$$

Guarantees $\alpha = \dfrac{r}{r^*} = 2$

Approximation ratio: General

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

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

Approximation ratio: Improvement?

Can we have a better approximation?

Sadly, NO

Dominating Set

If there is $\alpha < 2 \implies$ Dominating Set is solvable in polytime.

Input: Graph $G = (V, E)$

Output: $D \subseteq V$ and integer k s.t. $|D| = k$ and each vertice is V is either in D or is adjacent to a vertice in D

Dominating Set - Reduction

We reduce the instance of dominating set to k-center.

- For each vertice $u, v \in V$
  1. If u, v are adjacent: $d(u, v) = 2$
  2. If not: $d(u, v) = 1$
- If there is $|D| = k$: each cluster in k-center has radius $r^* = 1$
- If $|D| > k$: there exists cluster with $r^* = 2$
- If we can approximate $k-$center with $\alpha < 2$

$$r \leq \alpha r^* = \alpha < 2$$

This solves dominating set directly.

Travelling 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. $\sum_{i=1}^{n-1} w(\pi(i), \pi(i+1))$ is minimized

TSP - Formal Definition

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

$\Longrightarrow \; w_G \leq 2w_{Opt^*}$

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

Matching: a set of edges that don't share vertices.

Perfect Matching: a matching covering all vertices

Algorithm - A bit help from Christofide

Matching: a set of edges that don't share vertices.

Perfect Matching: a matching covering all vertices

- Remove all vertices with even degrees and their edges in G
- Find a PM in the result graph O
- Combine this with the MST

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 $\mathbf{P} = \mathbf{NP}$