Approximation Algorithms

Cristina G. Fernandes
University of São Paulo, Brazil

Guanajuato, Nov 7th, 2022

LATIN 2022

15th Latin American Theoretical Informatics Symposium

Outline of the tutorial

Part 1:

- Approximation algorithms: an example and definitions
- Clustering problems: k-center and k-median
- Bottleneck problems: 2-approximation for k-center
- Local search: $(3 + \epsilon)$-approximation for k-median

Outline of the tutorial

Part 1:

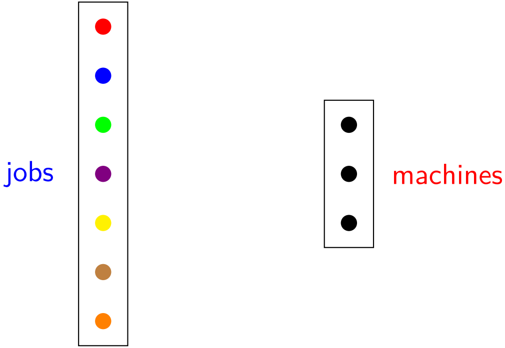
- Approximation algorithms: an example and definitions
- Clustering problems: k-center and k-median
- Bottleneck problems: 2-approximation for k-center
- Local search: $(3+\epsilon)$-approximation for k-median

Part 2:

- Probabilistic strategies: 0.5-approximation for MaxSAT
- Linear programming: 0.63-approximation for the MaxSAT
- Mixed strategies: 0.75-approximation for the MaxSAT
- Closing remarks

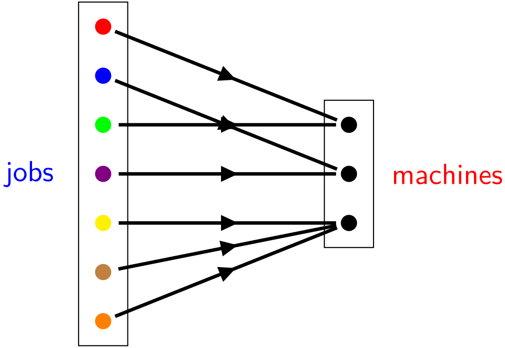
Scheduling in identical machines

Given: m machines
n jobs
processing time t_i of job i $(i = 1, \ldots, n)$



Scheduling in identical machines

Given: m machines
n jobs
processing time t_i of job i $\quad (i = 1, \ldots, n)$



a scheduling is a partition $\{M_1, \ldots, M_m\}$ of $\{1, \ldots, n\}$.

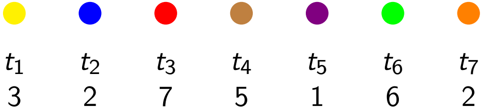
Example 1

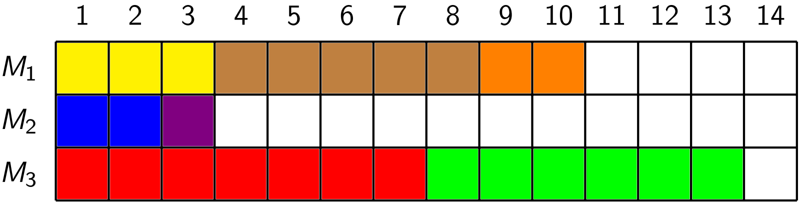
$m = 3$ and $n = 7$

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2

Example 1

$m = 3$ and $n = 7$

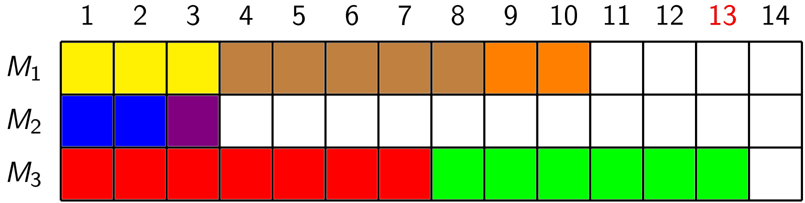




Example 1

$m = 3$ and $n = 7$

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2

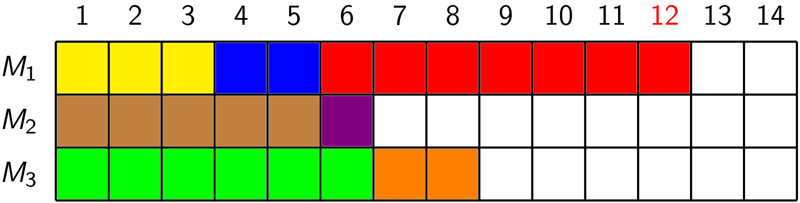


partition $\{\{1,4,7\},\{2,5\},\{3,6\}\} \Rightarrow$ makespan $= 13$

Example 2

$m = 3$ and $n = 7$

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2

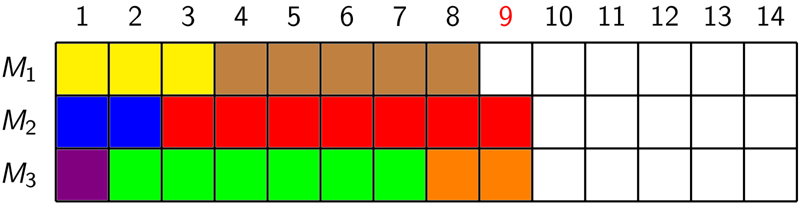


partition $\{\{1, 2, 3\}, \{4, 5\}, \{6, 7\}\} \Rightarrow$ makespan $= 12$

Problem

Find a scheduling with minimum makespan.

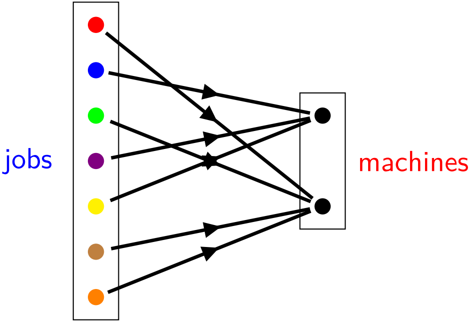
t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2



partition $\{\{1,4\},\{2,3\},\{5,6,7\}\} \Rightarrow$ makespan $= 9$

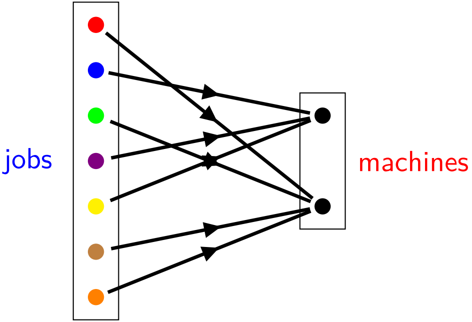
Hardness

Scheduling on two machines: given n and t,
find a scheduling for two machines with minimum makespan.



Hardness

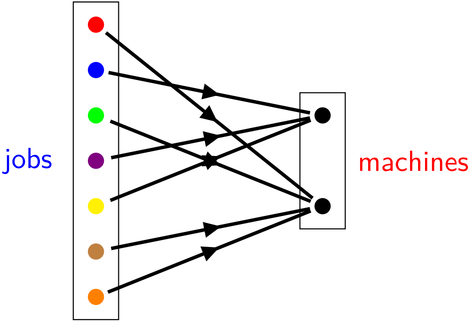
Scheduling on two machines: given n and t,
find a scheduling for two machines with minimum makespan.



Partition: Given a set S numbers,
decide if there is a subset $X \subseteq S$ such that $\sum_{s \in X} s = \sum_{s \in S \setminus X} s$.

Hardness

Scheduling on two machines: given n and t,
find a scheduling for two machines with minimum makespan.



Even this particular case is NP-hard, that is,
if there is a polynomial-time algorithm for this case, then $\mathrm{P} = \mathrm{NP}$.

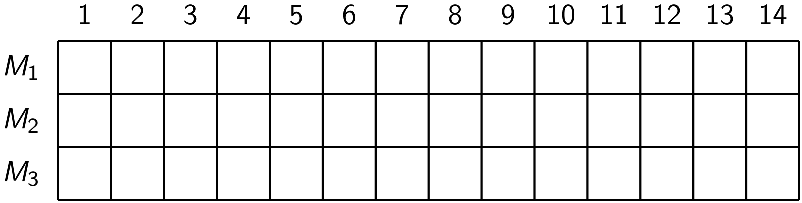
Graham's algorithm

Assign each job, one by one, to the first available machine.

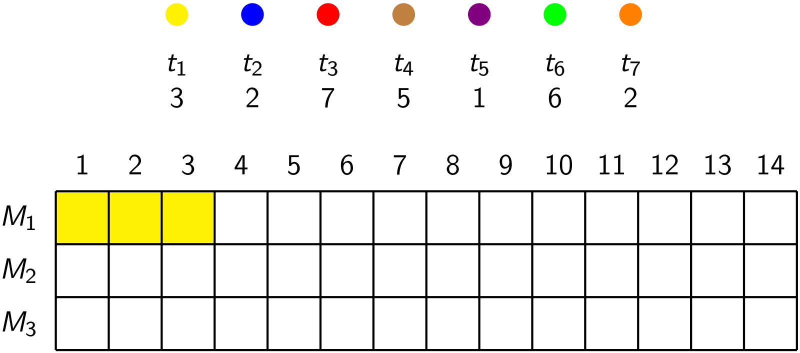
Graham's algorithm

Assign each job, one by one, to the first available machine.

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2



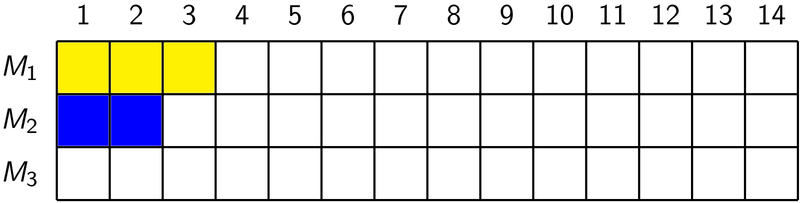
Assign each job, one by one, to the first available machine.



Graham's algorithm

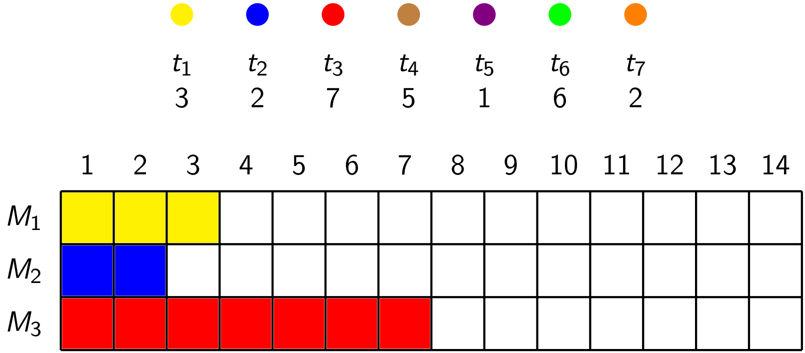
Assign each job, one by one, to the first available machine.

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2



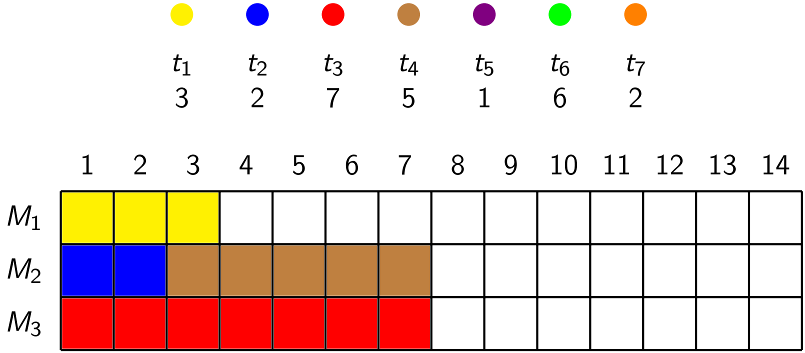
Assign each job, one by one, to the first available machine.

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2



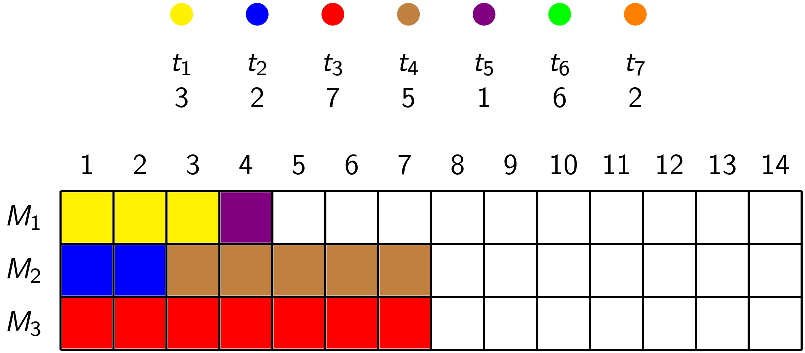
Graham's algorithm

Assign each job, one by one, to the first available machine.



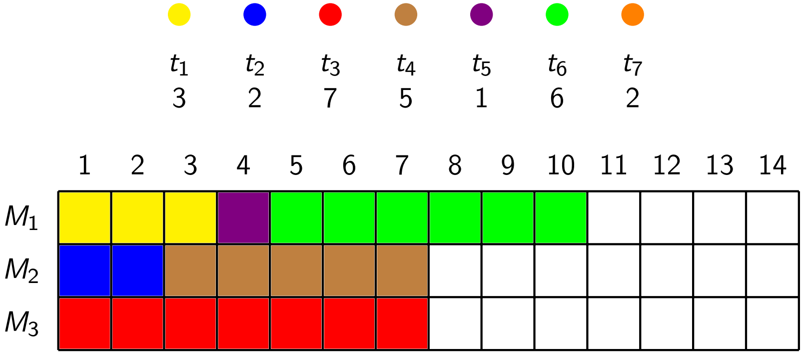
Graham's algorithm

Assign each job, one by one, to the first available machine.



Graham's algorithm

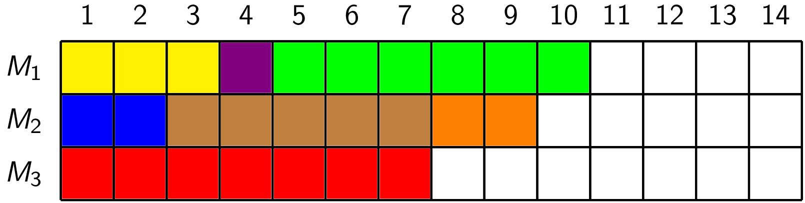
Assign each job, one by one, to the first available machine.



Graham's algorithm

Assign each job, one by one, to the first available machine.

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3	2	7	5	1	6	2



Graham's algorithm is polynomial.

How bad can the makespan be?

Bounds on OPT

OPT = minimum makespan

Bounds on OPT

OPT = minimum makespan

- Largest processing time of a job:

$$\mathrm{OPT} \geq \max\{t_1, t_2, \ldots, t_n\}$$

Bounds on OPT

OPT = minimum makespan

- Largest processing time of a job:

$$\mathrm{OPT} \geq \max\{t_1, t_2, \ldots, t_n\}$$

- Balanced distribution:

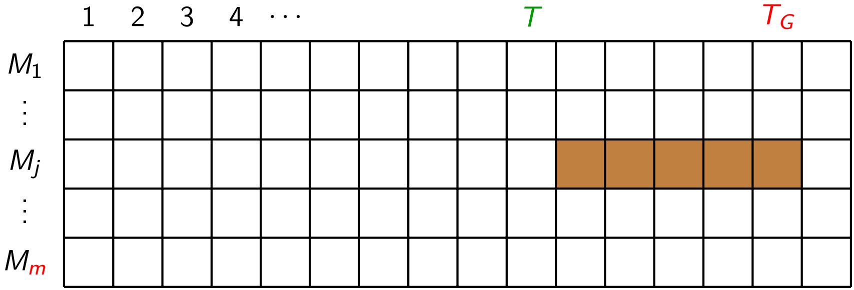
$$\mathrm{OPT} \geq \frac{t_1 + t_2 + \cdots + t_n}{m}$$

Makespan of Graham's scheduling

T_G: makespan of the algorithm

job i: job that finishes at time T_G

time T: time previous to the starting time of job i

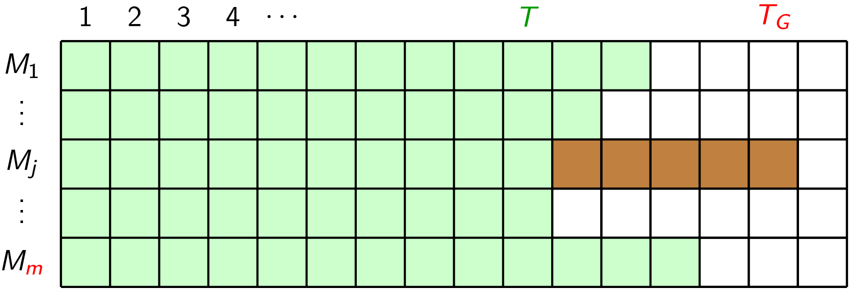


Makespan of Graham's scheduling

T_G: makespan of the algorithm

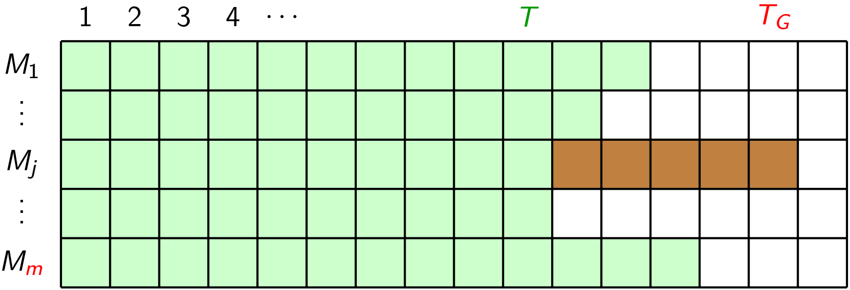
job i: job that finishes at time T_G

time T: time previous to the starting time of job i



$$T \cdot m < t_1 + \cdots + t_n \quad \Rightarrow \quad T < \frac{t_1 + \cdots + t_n}{m} \leq \mathrm{OPT}$$

Quality of Graham's scheduling



$$T_G = T + t_i$$
$$< \mathrm{OPT} + \max\{t_1, \ldots, t_n\}$$
$$\leq \mathrm{OPT} + \mathrm{OPT}$$
$$= 2\,\mathrm{OPT}$$

Approximation algorithm

Context:

- Π: optimization problem (minimization)
- $\mathrm{cost}(S, I)$: cost of the feasible solution S for instance I of Π
- $\mathrm{OPT}(I)$: minimum cost of a feasible solution for instance I of Π

Approximation algorithm

Context:

- Π: optimization problem (minimization)
- $\mathrm{cost}(S, I)$: cost of the feasible solution S for instance I of Π
- $\mathrm{OPT}(I)$: minimum cost of a feasible solution for instance I of Π

Algorithm for Π:

given any instance I for Π, produces a feasible solution for I.

$A(I)$: solution produced by algorithm A on instance I

Approximation algorithm

Context:

- Π: optimization problem (minimization)
- $\mathrm{cost}(S, I)$: cost of the feasible solution S for instance I of Π
- $\mathrm{OPT}(I)$: minimum cost of a feasible solution for instance I of Π

Algorithm for Π:

given any instance I for Π, produces a feasible solution for I.

$A(I)$: solution produced by algorithm A on instance I

Approximation algorithm

if A is polynomial and there exists a number $\alpha \geq 1$ such that

$$\mathrm{cost}(A(I), I) \leq \alpha \, \mathrm{OPT}(I) \text{ for every instance } I \text{ of } \Pi,$$

then A is an α-**approximation**.

Graham's algorithm

Input: positive integers m and n, and an array $t[1..n]$
Output: a scheduling of $\{1, \ldots, n\}$ in m machines.

Algorithm Graham (m, n, t)

1. for $j := 1$ to m do
2. $\quad M_j := \emptyset$
3. $\quad T_j := 0$ $\qquad \triangleright$ available instant for machine j
4. for $i := 1$ to n do
5. $\quad$ let k be such that T_k is minimum
6. $\quad M_k := M_k \cup \{i\} \qquad T_k := T_k + t_i$
7. return $\{M_1, \ldots, M_m\}$

Graham's algorithm is a 2-approximation.

Graham's algorithm

Input: positive integers m and n, and an array $t[1\,..\,n]$
Output: a scheduling of $\{1, \ldots, n\}$ in m machines.

Algorithm Graham (m, n, t)

1. for $j := 1$ to m do
2. $\quad M_j := \emptyset$
3. $\quad T_j := 0$ $\qquad \triangleright$ available instant for machine j
4. for $i := 1$ to n do
5. $\quad$ let k be such that T_k is minimum
6. $\quad M_k := M_k \cup \{i\} \qquad T_k := T_k + t_i$
7. return $\{M_1, \ldots, M_m\}$

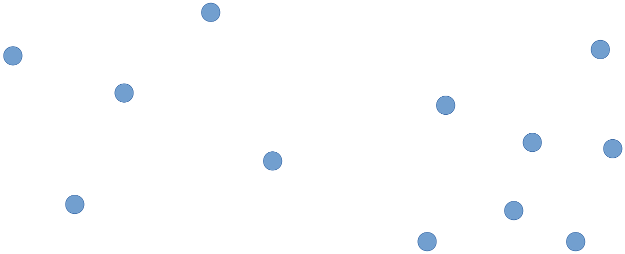
Exercise 1:
What if we schedule the jobs in decreasing order of the processing time?

Clustering problems

Classical k-center

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \to \mathbb{Q}^+$,



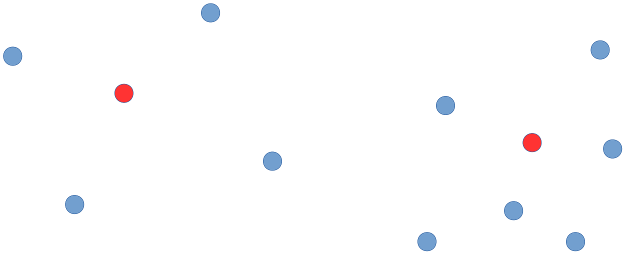
Clustering problems

Classical k-center

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \to \mathbb{Q}^+$,

find a set $S \subseteq V$ with $|S| = k$ that ...



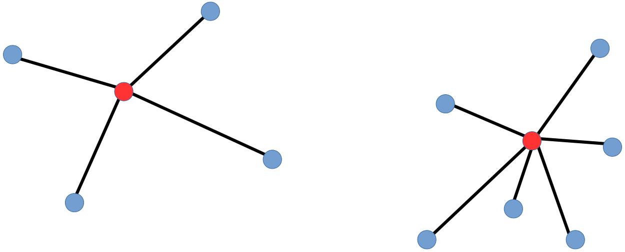
Clustering problems

Classical k-center

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \to \mathbb{Q}^+$,

find a set $S \subseteq V$ with $|S| = k$ that minimizes $\max_{u \in V} \min_{v \in S} d(u, v)$.



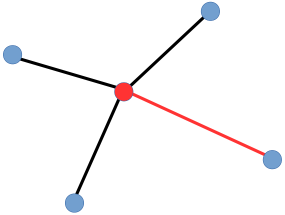
Clustering problems

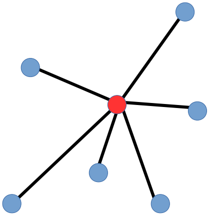
Classical k-center

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \rightarrow \mathbb{Q}^+$,

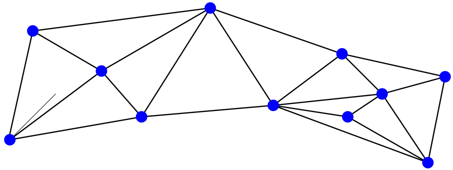
find a set $S \subseteq V$ with $|S| = k$ that minimizes $\max_{u \in V} \min_{v \in S} d(u, v)$.





Hardness

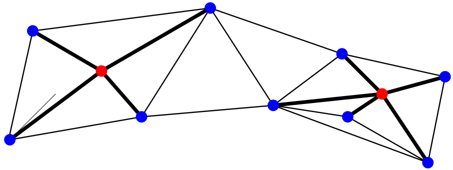
Graph G



Dominating set: set S of vertices of G such that each vertex of G is in S or has a neighbor in S.

Hardness

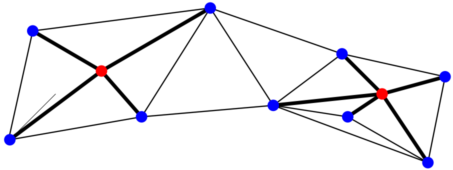
Graph G



Dominating set: set S of vertices of G such that each vertex of G is in S or has a neighbor in S.

Hardness

Graph G

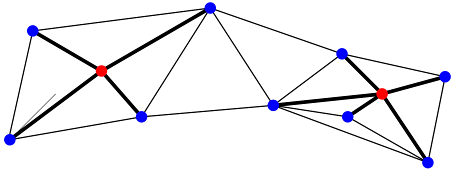


Dominating set: set S of vertices of G such that each vertex of G is in S or has a neighbor in S.

Reduction to a k-center instance: take the same k, $V = V(G)$ and $d(x, y) = 1$ if x and y are adjacent in G, and $d(x, y) = M$ otherwise.

Hardness

Graph G



Dominating set: set S of vertices of G such that each vertex of G is in S or has a neighbor in S.

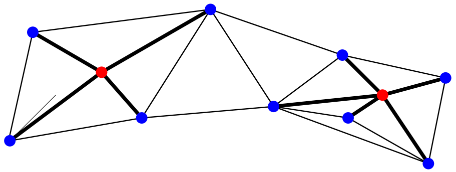
Reduction to a k-center instance: take the same k, $V = V(G)$ and $d(x, y) = 1$ if x and y are adjacent in G, and $d(x, y) = M$ otherwise.

Theorem

There is a dominating set of size k in G if and only if there is a k-center solution of radius 1 for the instance (k, V, d).

Inapproximability

Graph G



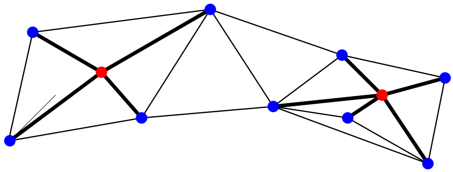
The **k-center instance** is $V = V(G)$ and
$d(x, y) = 1$ if x and y are adjacent in G, and $d(x, y) = M$ otherwise.

Hard even to approximate:

An α-approximation for k-center with $\alpha < M$ solves dominating set.

Inapproximability

Graph G



The ***k*-center instance** is $V = V(G)$ and
$d(x, y) = 1$ if x and y are adjacent in G, and $d(x, y) = M$ otherwise.

Hard even to approximate:

An α-approximation for k-center with $\alpha < M$ solves dominating set.

Theorem

There is no α-approximation for the k-center problem, unless $\mathrm{P} = \mathrm{NP}$.

Too hard to approximate?

What to do?

Restrict attention to specific classes of instances.

Too hard to approximate?

What to do?

Restrict attention to specific classes of instances.

A function $d : V \times V \to \mathbb{Q}^+$ is a metric if, for every $x, y, w \in V$,

- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, y) \leq d(x, w) + d(w, y)$ (triangle inequality)

Such a function d is called a distance function.

Metric instances

If d is a distance function, then the instance is metric.

Metric instances

Metric instances

If d is a distance function, then the instance is metric.

The **k-center instance built from the dominating set instance:**
$d(x, y) = 1$ if x and y are adjacent in G and $d(x, y) = M$ otherwise.

This k-center instance is metric only if $M \leq 2$.

Metric instances

Metric instances

If d is a distance function, then the instance is metric.

The **k-center instance built from the dominating set instance:**
$d(x, y) = 1$ if x and y are adjacent in G and $d(x, y) = M$ otherwise.

This k-center instance is metric only if $M \leq 2$.

Exercise 2:

How does the previous inapproximability result apply to the metric k-center?

Bottleneck problems

- Optimal value is the length of an edge

Bottleneck problems

- Optimal value is the length of an edge
- We can "guess" the optimal value

Bottleneck problems

- Optimal value is the length of an edge
- We can "guess" the optimal value
- Consider the corresponding threshold graph

Bottleneck problems

- Optimal value is the length of an edge
- We can “guess” the optimal value
- Consider the corresponding threshold graph
- Approximately solve the unweighted version of the problem, if possible

Bottleneck problems

- Optimal value is the length of an edge
- We can "guess" the optimal value
- Consider the corresponding threshold graph
- Approximately solve the unweighted version of the problem, if possible

Example of bottleneck problem: k-center

Instance: positive integer k, set V, and distance function d on V.

Bottleneck problems

- Optimal value is the length of an edge
- We can “guess” the optimal value
- Consider the corresponding threshold graph
- Approximately solve the unweighted version of the problem, if possible

Example of bottleneck problem: k-center

Instance: positive integer k, set V, and distance function d on V.

G: complete graph on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Bottleneck problems

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Bottleneck problems

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.

Bottleneck problems

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.
2. Threshold graph G_i : $G[E_i]$ where $E_i := \{e_1, \ldots, e_i\}$.

Bottleneck problems

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.
2. Threshold graph G_i : $G[E_i]$ where $E_i := \{e_1, \ldots, e_i\}$.
3. i^*: smallest i such that G_i has a dominating set of size k.

Bottleneck problems

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.
2. Threshold graph G_i : $G[E_i]$ where $E_i := \{e_1, \ldots, e_i\}$.
3. i^*: smallest i such that G_i has a dominating set of size k.
4. $\ell(e_{i^*})$: radius of optimal k-center solution.

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.
2. Threshold graph G_i : $G[E_i]$ where $E_i := \{e_1, \ldots, e_i\}$.
3. i^*: smallest i such that G_i has a dominating set of size k $\leftarrow$ HARD
4. $\ell(e_{i^*})$: radius of optimal k-center solution.

Bottleneck problems

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.
2. Threshold graph G_i : $G[E_i]$ where $E_i := \{e_1, \ldots, e_i\}$.
3. i^*: smallest i such that G_i has a dominating set of size k $\leftarrow$ HARD
4. $\ell(e_{i^*})$: radius of optimal k-center solution.

H^2: the square of H (add edges between vertices at distance 2 in H)

Bottleneck problems

Metric k-center: positive integer k and complete graph G on V with length $\ell(uv) = d(u,v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.
2. Threshold graph G_i : $G[E_i]$ where $E_i := \{e_1, \ldots, e_i\}$.
3. i^*: smallest i such that G_i has a dominating set of size k $\leftarrow$ HARD
4. $\ell(e_{i^*})$: radius of optimal k-center solution.

H^2: the square of H (add edges between vertices at distance 2 in H)

A maximal independent set in a graph is a dominating set.

Bottleneck problems

Metric k-center: positive integer k and
complete graph G on V with length $\ell(uv) = d(u, v)$ for each edge uv.

Idea for an algorithm:

1. Sort edges of G by length: $\ell(e_1) \leq \cdots \leq \ell(e_m)$.
2. Threshold graph G_i : $G[E_i]$ where $E_i := \{e_1, \ldots, e_i\}$.
3. i^*: smallest i such that G_i has a dominating set of size k $\leftarrow$ HARD
4. $\ell(e_{i^*})$: radius of optimal k-center solution.

H^2: the square of H (add edges between vertices at distance 2 in H)

A maximal independent set in a graph is a dominating set.

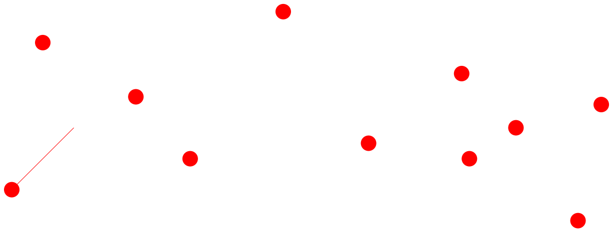
A maximal independent set in G_i^2 is a set of centers in G of radius $2\,\ell(e_i)$.

Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) $\triangleright$ Gonzalez '85, Hochbaum and Shmoys '85

$M_0 := V(G) \quad i := 0$
while $|M_i| > k$
    $i := i + 1$
    Let M_i be a maximal independent set on G_i^2
return M_i

$k = 2$

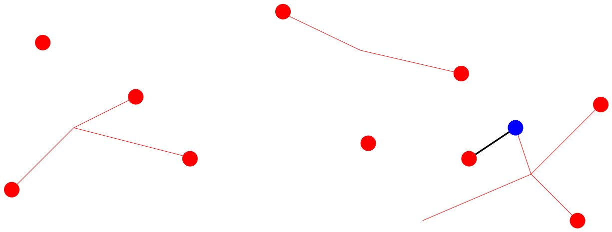


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) $\triangleright$ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$

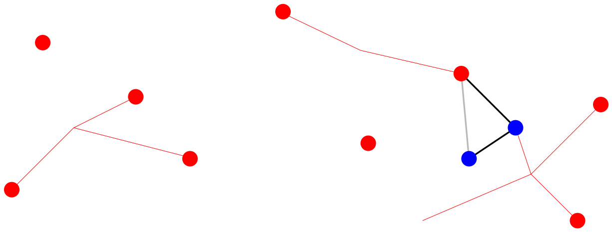


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) ▷ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$

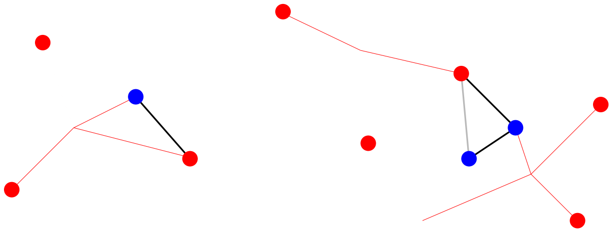


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) ▷ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$

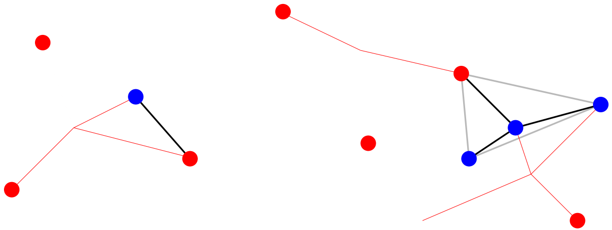


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) ▷ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$

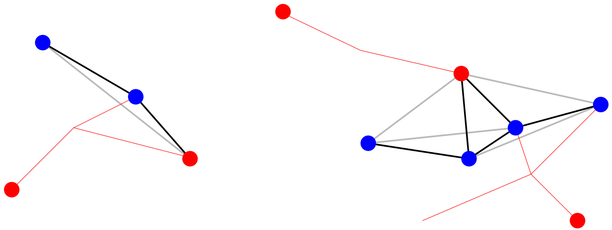


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) ▷ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$

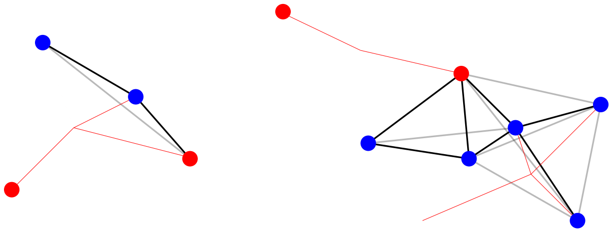


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) ▷ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$

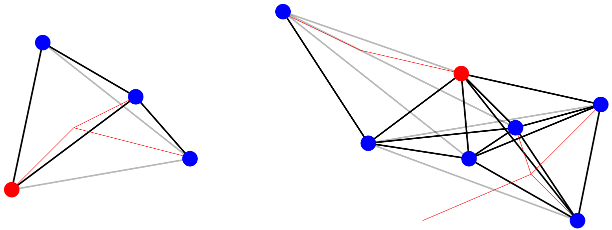


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) ▷ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$

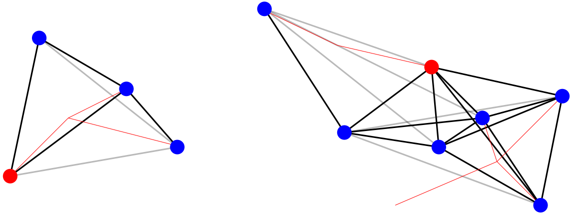


Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) $\triangleright$ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i

$k = 2$



The radius of M_i is at most $2\,\ell(e_i)$.

Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) $\triangleright$ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$ $\triangleright$ $i \leq i^*$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i $\triangleright$ gives a 2-approximation

The radius of M_i is at most $2\,\ell(e_i)$.

Because G_{i^*} has a dominating set of size k,
any maximal independent set in $G_{i^*}^2$ has size at most k.

Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) ▷ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$ ▷ $i \leq i^*$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i ▷ gives a 2-approximation

The radius of M_i is at most $2\,\ell(e_i)$.

Because G_{i^*} has a dominating set of size k,
any maximal independent set in $G_{i^*}^2$ has size at most k.

So certainly $|M_{i^*}| \leq k$, thus $i \leq i^*$.

Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) $\triangleright$ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$ $\triangleright$ $i \leq i^*$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i $\triangleright$ gives a 2-approximation

The radius of M_i is at most $2\,\ell(e_i)$.

Because G_{i^*} has a dominating set of size k,
any maximal independent set in $G_{i^*}^2$ has size at most k.

So certainly $|M_{i^*}| \leq k$, thus $i \leq i^*$.

Hence the radius of M_i is at most $2\,\ell(e_i) \leq 2\,\ell(e_{i^*}) = 2\,\mathrm{OPT}$.

Bottleneck problems: metric k-center

Algorithm GHS (k, G, ℓ) $\triangleright$ Gonzalez '85, Hochbaum and Shmoys '85

1. $M_0 := V(G) \quad i := 0$
2. while $|M_i| > k$ $\triangleright$ $i \leq i^*$
3. $\quad i := i + 1$
4. $\quad$ Let M_i be a maximal independent set on G_i^2
5. return M_i $\triangleright$ gives a 2-approximation

Exercise 3:

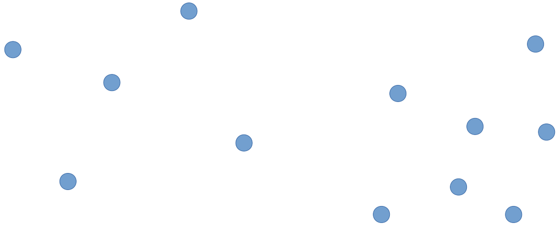
Is there an α-approximation with $\alpha < 2$ for the metric k-center?

Clustering problems

Classical k-median

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \to \mathbb{Q}^+$,



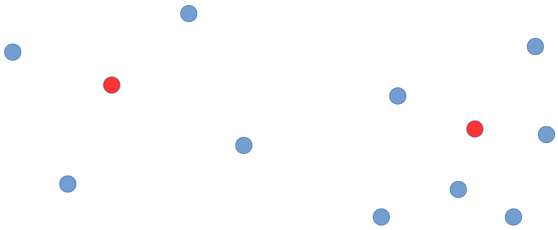
Clustering problems

Classical k-median

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \to \mathbb{Q}^+$,

find a set $S \subseteq V$ with $|S| = k$ that ...



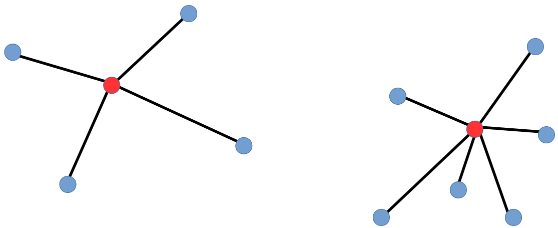
Clustering problems

Classical k-median

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \to \mathbb{Q}^+$,

find a set $S \subseteq V$ with $|S| = k$ that minimizes $\sum_{u \in V} \min_{v \in S} d(u, v)$.



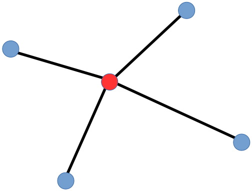
Clustering problems

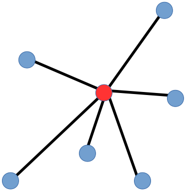
Classical k-median

Given:

- a positive integer k,
- a set V of elements, and
- a function $d : V \times V \to \mathbb{Q}^+$,

find a set $S \subseteq V$ with $|S| = k$ that minimizes $\sum_{u \in V} \min_{v \in S} d(u, v)$.





There is no α-approximation
for constant $\alpha > 1$ unless $\mathrm{P} = \mathrm{NP}$.

Local search: metric k-median

k-median instance: positive integer k, set V, and distance function d.

Local search: metric k-median

k-median instance: positive integer k, set V, and distance function d.

Let S be a subset of V of size k. Let $u \in S$ and $v \notin S$.

Local search: metric k-median

k-median instance: positive integer k, set V, and distance function d.

Let S be a subset of V of size k. Let $u \in S$ and $v \notin S$.

Pair (u, v) is an improving swap for S if $S' = S - u + v$ has better cost:

$$\sum_{u \in V} \min_{v \in S} d(u, v) > \sum_{u \in V} \min_{v \in S'} d(u, v).$$

Local search: metric k-median

k-median instance: positive integer k, set V, and distance function d.

Let S be a subset of V of size k. Let $u \in S$ and $v \notin S$.

Pair (u, v) is an improving swap for S if $S' = S - u + v$ has better cost:

$$\sum_{u \in V} \min_{v \in S} d(u, v) > \sum_{u \in V} \min_{v \in S'} d(u, v).$$

Algorithm AGKMMP (k, V, d) ▷ Arya et al. '01

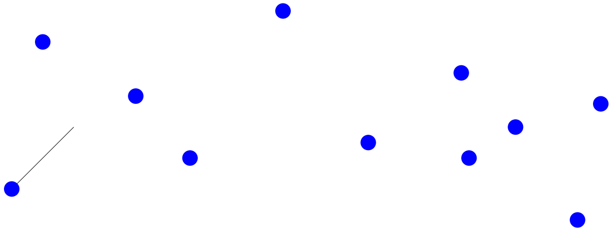
1. let S be an arbitrary set of k elements of V
2. while there is an improving swap (u, v) for S
3. $\quad S := S - u + v$
4. return S

Local search: metric k-median

Algorithm AGKMMP (k, V, d) ▷ Arya et al. '01

1. let S be an arbitrary set of k elements of V
2. while there is an improving swap (u, v) for S
3. $\quad S := S - u + v$
4. return S

$k = 2$

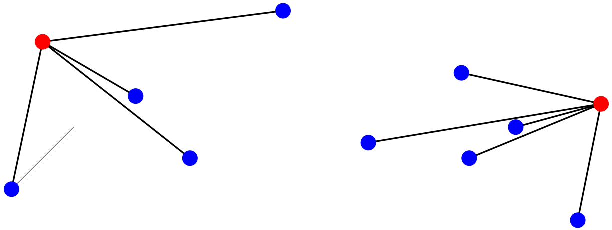


Local search: metric k-median

Algorithm AGKMMP (k, V, d) ▷ Arya et al. '01

1. let S be an arbitrary set of k elements of V
2. while there is an improving swap (u, v) for S
3. $\quad S := S - u + v$
4. return S

$k = 2$

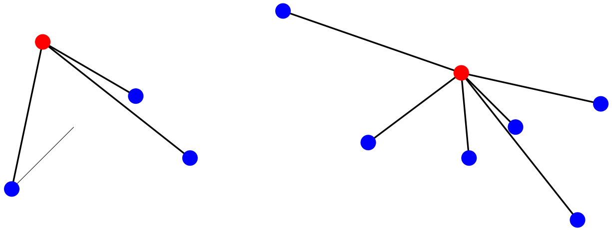


Local search: metric k-median

Algorithm AGKMMP (k, V, d) ▷ Arya et al. '01

1. let S be an arbitrary set of k elements of V
2. while there is an improving swap (u, v) for S
3. $\quad S := S - u + v$
4. return S

$k = 2$

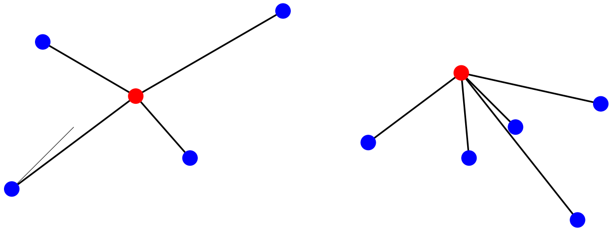


Local search: metric k-median

Algorithm AGKMMP (k, V, d) ▷ Arya et al. '01

1. let S be an arbitrary set of k elements of V
2. while there is an improving swap (u, v) for S
3. $\quad S := S - u + v$
4. return S

$k = 2$

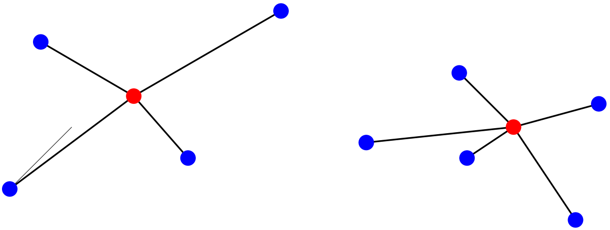


Local search: metric k-median

Algorithm AGKMMP (k, V, d) ▷ Arya et al. '01

1. let S be an arbitrary set of k elements of V
2. while there is an improving swap (u, v) for S
3. $\quad S := S - u + v$
4. return S

$k = 2$

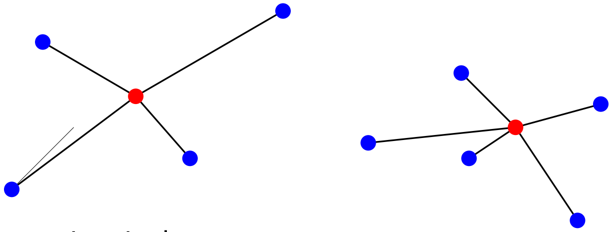


Local search: metric k-median

Algorithm AGKMMP (k, V, d) ▷ Arya et al. '01

1. let S be an arbitrary set of k elements of V
2. while there is an improving swap (u, v) for S
3. $\quad S := S - u + v$
4. return S

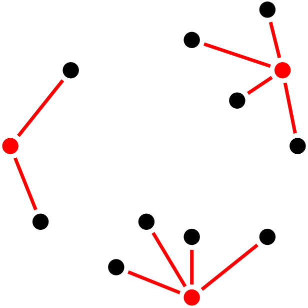
$k = 2$



This is a 5-approximation!

Sketch of the approximation analysis

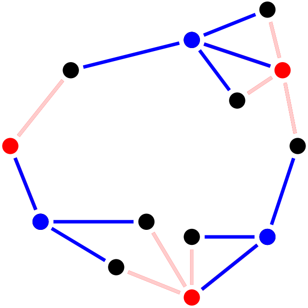
Let S^* be an optimal solution and $N^*(o)$ be the clients of o in S^*



Sketch of the approximation analysis

Let S^* be an optimal solution and $N^*(o)$ be the clients of o in S^*

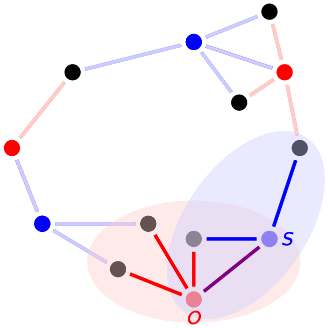
Let S be the output of the algorithm and $N(s)$ be the clients of s in S.



Sketch of the approximation analysis

Let S^* be an optimal solution and $N^*(o)$ be the clients of o in S^*

Let S be the output of the algorithm and $N(s)$ be the clients of s in S.

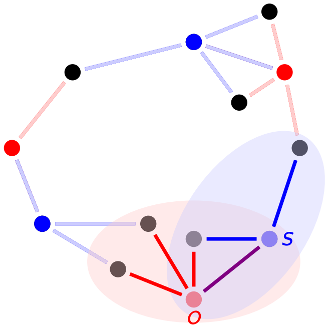


$s \in S$ captures $o \in S^*$ if $|N^*(o) \cap N(s)| > |N^*(o)|/2$.

Sketch of the approximation analysis

Let S^* be an optimal solution and $N^*(o)$ be the clients of o in S^*

Let S be the output of the algorithm and $N(s)$ be the clients of s in S.



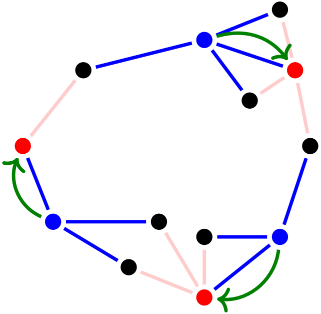
$s \in S$ captures $o \in S^*$ if $|N^*(o) \cap N(s)| > |N^*(o)|/2$.

Each $o \in S^*$ is captured by at most one element from S.

Sketch of the approximation analysis

$s \in S$ captures $o \in S^*$ if $|N^*(o) \cap N(s)| > |N^*(o)|/2$.

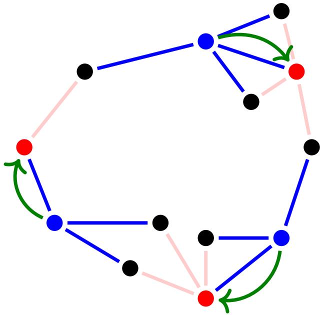
Assume that each $s \in S$ captures exactly one element o from S^*.



Sketch of the approximation analysis

$s \in S$ captures $o \in S^*$ if $|N^*(o) \cap N(s)| > |N^*(o)|/2$.

Assume that each $s \in S$ captures exactly one element o from S^*.

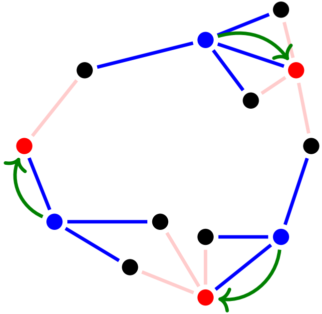


In this case, let us prove that $\mathrm{cost}(S) \leq 3\,\mathrm{cost}(S^*)$.

Sketch of the approximation analysis

$s \in S$ captures $o \in S^*$ if $|N^*(o) \cap N(s)| > |N^*(o)|/2$.

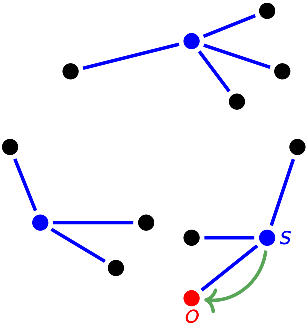
Assume that each $s \in S$ captures exactly one element o from S^*.



In this case, let us prove that $\mathrm{cost}(S) \leq 3\,\mathrm{cost}(S^*)$.

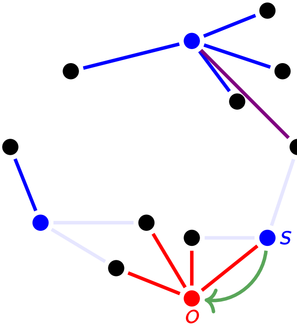
Because (s, o) is not an improving swap, $\mathrm{cost}(S - s + o) \geq \mathrm{cost}(S)$.

Sketch of the approximation analysis



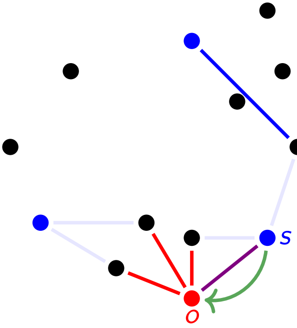
$\text{cost}(S - s + o) \ \leq$???

Sketch of the approximation analysis



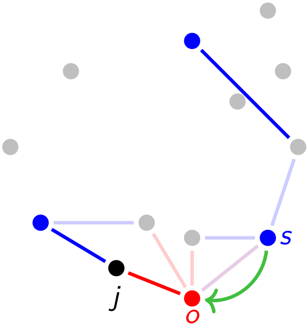
$\mathrm{cost}(S - s + o) \ \leq$???

Sketch of the approximation analysis



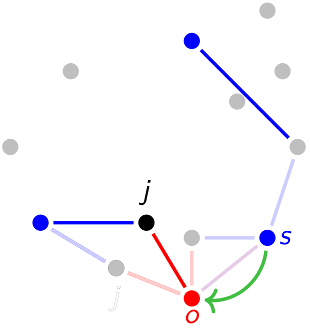
$\mathrm{cost}(S - s + o) \;\leq\; \mathrm{cost}(S) \quad + \quad ???$

Sketch of the approximation analysis



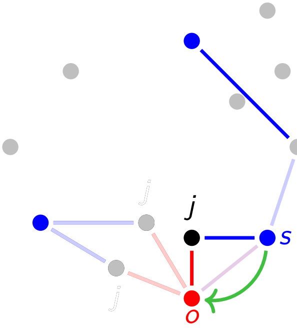
$$\mathrm{cost}(S - s + o) \leq \mathrm{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) +$$

Sketch of the approximation analysis



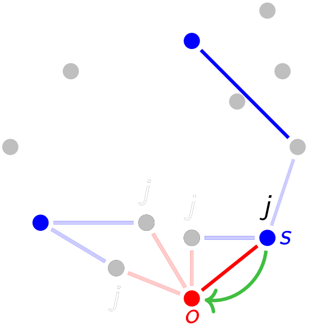
$$\mathrm{cost}(S - s + o) \leq \mathrm{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) +$$

Sketch of the approximation analysis



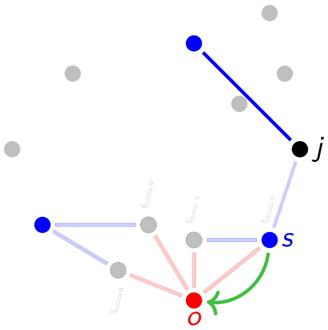
$$\text{cost}(S - s + o) \leq \text{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) +$$

Sketch of the approximation analysis



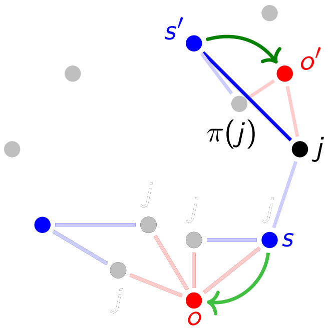
$$\mathrm{cost}(S - s + o) \leq \mathrm{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) +$$

Sketch of the approximation analysis



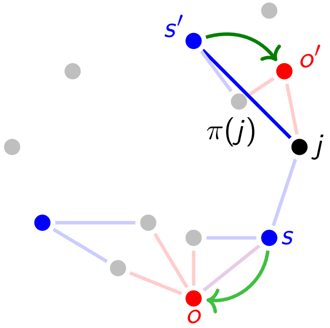
$$\text{cost}(S - s + o) \leq \text{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) +$$

Sketch of the approximation analysis



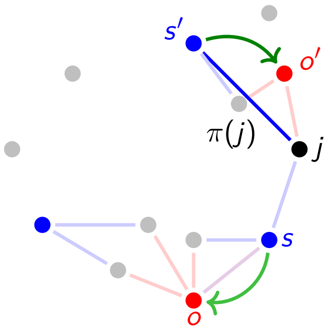
$$\text{cost}(S - s + o) \leq \text{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) +$$

Sketch of the approximation analysis



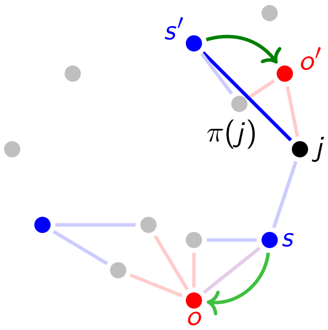
$$\mathrm{cost}(S - s + o) \leq \mathrm{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) + \sum_{j \in N(s) \setminus N^*(o)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j).$$

Sketch of the approximation analysis



$$\mathrm{cost}(S - s + o) \le \mathrm{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) \; + \; \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j).$$

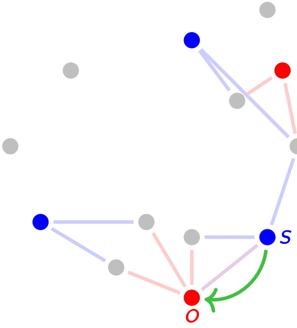
Sketch of the approximation analysis



$$\mathrm{cost}(S - s + o) \leq \mathrm{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) \; + \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j).$$

Permutation π is selected using that $|N^*(o') \cap N(s')| > |N^*(o)|/2$.

Sketch of the approximation analysis



$$\mathrm{cost}(S - s + o) \leq \mathrm{cost}(S) + \sum_{j \in N^*(o)} (o_j - s_j) \; + \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j).$$

But $\mathrm{cost}(S - s + o) \geq \mathrm{cost}(S)$, so

$$\sum_{j \in N^*(o)} (o_j - s_j) \; + \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0.$$

Sketch of the approximation analysis

$$\sum_{j \in N^*(o)} (o_j - s_j) \; + \; \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0$$

Thus, summing over all $s \in S$ and the corresponding $o \in S^*$, we get

$$(\mathrm{cost}(S^*) - \mathrm{cost}(S)) + (\mathrm{cost}(S^*) + \mathrm{cost}(S^*) + \mathrm{cost}(S) - \mathrm{cost}(S)) \quad \geq \quad 0$$

Sketch of the approximation analysis

$$\sum_{j \in N^*(o)} (o_j - s_j) \; + \; \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0$$

Thus, summing over all $s \in S$ and the corresponding $o \in S^*$, we get

$$(\mathrm{cost}(S^*) - \mathrm{cost}(S)) + (\mathrm{cost}(S^*) + \mathrm{cost}(S^*) + \mathrm{cost}(S) - \mathrm{cost}(S)) \quad \geq \quad 0$$

Therefore $\mathrm{cost}(S) \leq 3\,\mathrm{cost}(S^*)$.

Sketch of the approximation analysis

$$\sum_{j \in N^*(o)} (o_j - s_j) \; + \; \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0$$

Thus, summing over all $s \in S$ and the corresponding $o \in S^*$, we get

$$(\mathrm{cost}(S^*) - \mathrm{cost}(S)) + (\mathrm{cost}(S^*) + \mathrm{cost}(S^*) + \mathrm{cost}(S) - \mathrm{cost}(S)) \quad \geq \quad 0$$

Therefore $\mathrm{cost}(S) \leq 3\,\mathrm{cost}(S^*)$.

Without the assumption that
each $s \in S$ captures exactly one element o from S^*,
by a similar analysis, we can derive that $\mathrm{cost}(S) \leq 5\,\mathrm{cost}(S^*)$.

Sketch of the approximation analysis

$$\sum_{j \in N^*(o)} (o_j - s_j) \; + \; \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0$$

Thus, summing over all $s \in S$ and the corresponding $o \in S^*$, we get

$$(\mathrm{cost}(S^*) - \mathrm{cost}(S)) + (\mathrm{cost}(S^*) + \mathrm{cost}(S^*) + \mathrm{cost}(S) - \mathrm{cost}(S)) \quad \geq \quad 0$$

Therefore $\mathrm{cost}(S) \leq 3\,\mathrm{cost}(S^*)$.

Without the assumption that
each $s \in S$ captures exactly one element o from S^*,
by a similar analysis, we can derive that $\mathrm{cost}(S) \leq 5\,\mathrm{cost}(S^*)$.

Exercise 4:

Argue that each $s \in S$ captures at most two elements from S^* and derive that $\sum_{j \in N^*(o)} (o_j - s_j) + 2 \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0$.

Short break before Part 2

Exercise 1:

Sort the jobs in decreasing order of the processing time before running Graham's algorithm. Can you prove an approximation ratio better than 2 for this algorithm?

Exercise 2:

How does the inapproximability result for k-center apply to metric instances?

Exercise 3:

Is there an α-approximation with $\alpha < 2$ for the metric k-center?

Exercise 4:

Argue that each $s \in S$ captures at most two elements from S^* and derive that $\sum_{j \in N^*(o)} (o_j - s_j) + 2 \sum_{j \in N(s)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0$.

Satisfiability

Boolean formulas

v_i: boolean variable
$\bar{v}_i$: negation of the boolean variable v_i

literal: a variable or its negation

clause: disjunction (OR) of literals, as for instance $v_1 \vee \bar{v}_2 \vee v_3$

Satisfiability

Boolean formulas

v_i: boolean variable
$\bar{v}_i$: negation of the boolean variable v_i

literal: a variable or its negation

clause: disjunction (OR) of literals, as for instance $v_1 \vee \bar{v}_2 \vee v_3$
(literals in the same clause correspond to distinct variables)

Satisfiability

Boolean formulas

v_i: boolean variable
$\bar{v}_i$: negation of the boolean variable v_i

literal: a variable or its negation

clause: disjunction (OR) of literals, as for instance $v_1 \vee \bar{v}_2 \vee v_3$
(literals in the same clause correspond to distinct variables)

Boolean formula in conjunctive normal form (CNF):

$$\phi = (v_1 \vee \bar{v}_2 \vee v_3)(\bar{v}_1 \vee \bar{v}_3)(v_2 \vee v_3 \vee \bar{v}_4 \vee v_5)(\bar{v}_1 \vee v_4 \vee \bar{v}_5)$$

Satisfiability

Boolean formulas

v_i: boolean variable
$\bar{v}_i$: negation of the boolean variable v_i

literal: a variable or its negation

clause: disjunction (OR) of literals, as for instance $v_1 \vee \bar{v}_2 \vee v_3$
(literals in the same clause correspond to distinct variables)

Boolean formula in conjunctive normal form (CNF):

$$\phi = (v_1 \vee \bar{v}_2 \vee v_3)(\bar{v}_1 \vee \bar{v}_3)(v_2 \vee v_3 \vee \bar{v}_4 \vee v_5)(\bar{v}_1 \vee v_4 \vee \bar{v}_5)$$

assignment for ϕ: function that assigns **True** or **False** to each variable in ϕ

To decide whether there exists an assignment
that satisfies a CNF formula is NP-complete.

MAX SAT

MAX SAT Problem

Given a CNF formula ϕ,
find an assignment for ϕ that maximizes the number of satisfied clauses.

MAX SAT

MAX SAT Problem

Given a CNF formula ϕ,
find an assignment for ϕ that maximizes the number of satisfied clauses.

Probabilistic algorithm:

For each i, with probability $1/2$ each,
choose to set $v_i =$ **True** or to set $v_i =$ **False**

MAX SAT

MAX SAT Problem

Given a CNF formula ϕ,
find an assignment for ϕ that maximizes the number of satisfied clauses.

Probabilistic algorithm:

For each i, with probability $1/2$ each,
choose to set $v_i =$ **True** or to set $v_i =$ **False**

A *k-clause* is a clause with exactly k literals.

What is the probability that a k-clause C ends up satisfied?

MAX SAT

MAX SAT Problem

Given a CNF formula ϕ,
find an assignment for ϕ that maximizes the number of satisfied clauses.

Probabilistic algorithm:

For each i, with probability $1/2$ each,
choose to set $v_i =$ **True** or to set $v_i =$ **False**

A *k*-clause is a clause with exactly k literals.

What is the probability that a k-clause C ends up satisfied?

$$\Pr[C \text{ is satisfied}] = 1 - \frac{1}{2^k}$$

Probabilistic $\frac{1}{2}$-approximation for SAT

Algorithm Johnson (ϕ) $\triangleright$ Johnson '74

1. let V be the set of variables in ϕ
2. for each $v \in V$
3. $\quad x_v := \textsc{Rand}(1/2)$
4. return x $\triangleright$ gives a (probabilistic) 0.5-approximation

$\textsc{Rand}(p)$: returns 1 with probability p or 0 with probability $1 - p$.

Probabilistic $\frac{1}{2}$-approximation for SAT

Algorithm Johnson (ϕ) $\triangleright$ Johnson '74

1. let V be the set of variables in ϕ
2. for each $v \in V$
3. $\quad x_v := \text{Rand}(1/2)$
4. return x $\triangleright$ gives a (probabilistic) 0.5-approximation

$\text{Rand}(p)$: returns 1 with probability p or 0 with probability $1 - p$.

As each clause in ϕ has at least one literal,
each clause is satistied with probability at least $1/2$.

Probabilistic $\frac{1}{2}$-approximation for SAT

Algorithm Johnson (ϕ) ▷ Johnson '74

1. let V be the set of variables in ϕ
2. for each $v \in V$
3. $\quad x_v := \textsc{Rand}(1/2)$
4. return x ▷ gives a (probabilistic) 0.5-approximation

$\textsc{Rand}(p)$: returns 1 with probability p or 0 with probability $1 - p$.

As each clause in ϕ has at least one literal,
each clause is satistied with probability at least $1/2$.

Let m be the number of clauses in ϕ.

Then clearly $\mathrm{Exp}[\mathrm{cost}(x)] \geq m/2 \geq \mathrm{OPT}(\phi)/2$.

One more exercise

Proposed algorithm

1. let V be the set of variables in ϕ
2. $s := \textsc{Rand}(1/2)$ ▷ unique coin flip
3. for each $v \in V$
4. $\quad x_v := s$
5. return x

$\textsc{Rand}(p)$: returns 1 with probability p and 0 with probability $1 - p$.

Exercise 5:

Prove that the proposed algorithm is an α-approximation for some α, or argue that the algorithm is not an approximation algorithm.

Integer programming formulations

For a CNF formula ϕ,
V is its set of variables.

For a clause C of ϕ,
$C_0 := \{v_i : \bar{v}_i \in C\}$ and $C_1 := \{v_i : v_i \in C\}$.

C_0 are the negative variables in C and C_1 are the positive variables in C.

Integer programming formulations

For a CNF formula ϕ,
V is its set of variables.

For a clause C of ϕ,
$C_0 := \{v_i : \bar{v}_i \in C\}$ and $C_1 := \{v_i : v_i \in C\}$.

C_0 are the negative variables in C and C_1 are the positive variables in C.

Consider the following integer linear program (IP) built from ϕ:

maximize $\sum_{C \in \phi} z_C$
subject to

$$\begin{array}{rcll} \sum_{v \in C_0}(1 - x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\ z_C & \in & \{0,1\} & \text{for every } C \in \phi \\ x_v & \in & \{0,1\} & \text{for every } v \in V \end{array}$$

Integer programming formulations

For a CNF formula ϕ,
V is its set of variables.

For a clause C of ϕ,
$C_0 := \{v_i : \bar{v}_i \in C\}$ and $C_1 := \{v_i : v_i \in C\}$.

C_0 are the negative variables in C and C_1 are the positive variables in C.

Consider the following integer linear program (IP) built from ϕ:

maximize $\sum_{C \in \phi} z_C$
subject to

$$\begin{array}{rcll} \sum_{v \in C_0}(1 - x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\ z_C & \in & \{0,1\} & \text{for every } C \in \phi \\ x_v & \in & \{0,1\} & \text{for every } v \in V \end{array}$$

Solving this IP is equivalent to finding $\mathrm{OPT}(\phi)$.

Linear programming and rounding

The linear relaxation of the IP built from ϕ is:

maximize $\sum_{C \in \phi} z_C$
subject to

$$\begin{array}{rcll} \sum_{v \in C_0}(1 - x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\ 0 \leq z_C & \leq & 1 & \text{for every } C \in \phi \\ 0 \leq x_v & \leq & 1 & \text{for every } v \in V \end{array}$$

Linear programming and rounding

The linear relaxation of the IP built from ϕ is:

maximize $\sum_{C \in \phi} z_C$
subject to

$$\begin{array}{rcll} \sum_{v \in C_0}(1 - x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\ 0 \leq z_C & \leq & 1 & \text{for every } C \in \phi \\ 0 \leq x_v & \leq & 1 & \text{for every } v \in V \end{array}$$

There are polynomial-time algorithms that solve linear programs.

Linear programming and rounding

The linear relaxation of the IP built from ϕ is:

maximize $\sum_{C \in \phi} z_C$
subject to

$$\begin{array}{rcll} \sum_{v \in C_0}(1 - x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\ 0 \leq z_C & \leq & 1 & \text{for every } C \in \phi \\ 0 \leq x_v & \leq & 1 & \text{for every } v \in V \end{array}$$

There are polynomial-time algorithms that solve linear programs.

If z^* is the optimum value of this linear program (LP), then $\mathrm{OPT}(\phi) \leq z^*$.

Linear programming and rounding

The linear relaxation of the IP built from ϕ is:

maximize $\sum_{C \in \phi} z_C$
subject to

$$
\begin{array}{rcll}
\sum_{v \in C_0} (1 - x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\
0 \leq z_C & \leq & 1 & \text{for every } C \in \phi \\
0 \leq x_v & \leq & 1 & \text{for every } v \in V
\end{array}
$$

There are polynomial-time algorithms that solve linear programs.

If z^* is the optimum value of this linear program (LP), then $\mathrm{OPT}(\phi) \leq z^*$.

Idea

Use the value of $x_v \in [0, 1]$ to decide how to set v to **True** or **False**.

Probabilistic rounding

maximize $\sum_{C \in \phi} z_C$
subject to

$$\begin{array}{rcll} \sum_{v \in C_0}(1 - x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\ 0 \leq z_C & \leq & 1 & \text{for every } C \in \phi \\ 0 \leq x_v & \leq & 1 & \text{for every } v \in V \end{array}$$

Algorithm GW (ϕ) ▷ Goemans and Williamson '94

1. solve the LP above obtaining $\hat{z}$ and $\hat{x}$
2. for each $v \in V$
3. $\quad \dot{x}_v := \text{RAND}(\hat{x}_v)$
4. return $\dot{x}$ ▷ gives a (probabilistic) 0.63-approximation

Probabilistic rounding

maximize $\sum_{C \in \phi} z_C$
subject to

$$\begin{array}{rcll} \sum_{v \in C_0}(1-x_v) + \sum_{v \in C_1} x_v & \geq & z_C & \text{for every } C \in \phi \\ 0 \leq z_C & \leq & 1 & \text{for every } C \in \phi \\ 0 \leq x_v & \leq & 1 & \text{for every } v \in V \end{array}$$

Algorithm GW (ϕ) ▷ Goemans and Williamson '94

1. solve the LP above obtaining $\hat{z}$ and $\hat{x}$
2. for each $v \in V$
3. $\quad \dot{x}_v := \textsc{Rand}(\hat{x}_v)$
4. return $\dot{x}$ ▷ gives a (probabilistic) 0.63-approximation

Indeed, $\dot{x}$ satisfies at least $0.63 \sum_{C \in \phi} \hat{z}_C \geq 0.63\,\mathrm{OPT}(\phi)$ clauses.

Analysis

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

Analysis

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\mathrm{Exp}[Z_C] \;=\; \Pr[Z_C = 1] \;=\; 1 - \prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)$$

Analysis

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\mathrm{Exp}[Z_C] \;=\; \Pr[Z_C = 1] \;=\; 1 - \prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)$$

For non-negative numbers, geometric is smaller than aritmetic mean:

$$\Big(\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)\Big)^{1/t} \;\leq\; \frac{\sum_{v \in C_0} \hat{x}_v + \sum_{v \in C_1} (1 - \hat{x}_v)}{t}$$

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\mathrm{Exp}[Z_C] \;=\; \Pr[Z_C = 1] \;=\; 1 - \prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)$$

For non-negative numbers, geometric is smaller than aritmetic mean:

$$\Big(\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)\Big)^{1/t} \;\le\; \frac{\sum_{v \in C_0} \hat{x}_v + \sum_{v \in C_1} (1 - \hat{x}_v)}{t}$$

$$=\; \frac{(|C_0| - \sum_{v \in C_0} (1 - \hat{x}_v)) + (|C_1| - \sum_{v \in C_1} \hat{x}_v)}{t}$$

Analysis

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\mathrm{Exp}[Z_C] \;=\; \Pr[Z_C = 1] \;=\; 1 - \prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)$$

For non-negative numbers, geometric is smaller than aritmetic mean:

$$\begin{aligned}
\Big(\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)\Big)^{1/t} &\leq \frac{\sum_{v \in C_0} \hat{x}_v + \sum_{v \in C_1} (1 - \hat{x}_v)}{t} \\
&= \frac{(|C_0| - \sum_{v \in C_0} (1 - \hat{x}_v)) + (|C_1| - \sum_{v \in C_1} \hat{x}_v)}{t} \\
&= \frac{t - \sum_{v \in C_0} (1 - \hat{x}_v) - \sum_{v \in C_1} \hat{x}_v}{t}
\end{aligned}$$

Analysis

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\mathrm{Exp}[Z_C] \;=\; \Pr[Z_C = 1] \;=\; 1 - \prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)$$

For non-negative numbers, geometric is smaller than aritmetic mean:

$$\begin{aligned}
\Big(\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)\Big)^{1/t} &\leq \frac{\sum_{v \in C_0} \hat{x}_v + \sum_{v \in C_1} (1 - \hat{x}_v)}{t} \\
&= \frac{(|C_0| - \sum_{v \in C_0} (1 - \hat{x}_v)) + (|C_1| - \sum_{v \in C_1} \hat{x}_v)}{t} \\
&= \frac{t - \sum_{v \in C_0} (1 - \hat{x}_v) - \sum_{v \in C_1} \hat{x}_v}{t}
\end{aligned}$$

Analysis

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\mathrm{Exp}[Z_C] \;=\; \Pr[Z_C = 1] \;=\; 1 - \prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)$$

For non-negative numbers, geometric is smaller than aritmetic mean:

$$\begin{aligned}
\Big(\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)\Big)^{1/t} &\leq \frac{\sum_{v \in C_0} \hat{x}_v + \sum_{v \in C_1} (1 - \hat{x}_v)}{t} \\
&= \frac{t - \sum_{v \in C_0} (1 - \hat{x}_v) - \sum_{v \in C_1} \hat{x}_v}{t} \\
&\leq \frac{t - \hat{z}_C}{t}.
\end{aligned}$$

Analysis

For each clause C, let t be the number of literals in C.

Consider the binary random variable
Z_C that is 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\mathrm{Exp}[Z_C] \;=\; \Pr[Z_C = 1] \;=\; 1 - \prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)$$

For non-negative numbers,

$$\Big(\prod_{v \in C_0} \hat{x}_v \prod_{v \in C_1} (1 - \hat{x}_v)\Big)^{1/t} \leq \frac{t - \hat{z}_C}{t}.$$

Hence

$$\mathrm{Exp}[Z_C] \;\geq\; 1 - \Big(\frac{t - \hat{z}_C}{t}\Big)^t.$$

Analysis

For each clause C with t literals,
let Z_C be 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\begin{aligned} \mathrm{Exp}[Z_C] &\geq 1 - (\frac{t - \hat{z}_C}{t})^t \\ &= 1 - (1 - \frac{\hat{z}_C}{t})^t \end{aligned}$$

Analysis

For each clause C with t literals,
let Z_C be 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\begin{aligned} \mathrm{Exp}[Z_C] &\geq 1 - (\frac{t - \hat{z}_C}{t})^t \\ &= 1 - (1 - \frac{\hat{z}_C}{t})^t \\ &\geq (1 - (1 - \frac{1}{t})^t)\hat{z}_C \end{aligned}$$

because $f(z) = 1 - (1 - \frac{z}{t})^t$ is concave in the interval $[0, 1]$,
and $f(0) = 0$, so $f(z) \geq z\, f(1)$, which implies the last inequality.

Analysis

For each clause C with t literals,
let Z_C be 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

$$\begin{aligned}
\mathrm{Exp}[Z_C] &\geq 1-(\frac{t-\hat{z}_C}{t})^t \\
&= 1-(1-\frac{\hat{z}_C}{t})^t \\
&\geq (1-(1-\frac{1}{t})^t)\hat{z}_C \\
&> (1-\frac{1}{e})\hat{z}_C \\
&> 0.63\,\hat{z}_C
\end{aligned}$$

because $(1-\frac{1}{t})^t < \frac{1}{e}$ for every $t \geq 1$.

Euler's number $e = 2.71828$, the base of the natural logarithm.

Analysis

For each clause C with t literals,
let Z_C be 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

Note that $\sum_{C \in \phi} Z_C$ is the number of clauses satisfied by the assignment $\dot{x}$ produced by the GW algorithm.

Analysis

For each clause C with t literals,
let Z_C be 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

Note that $\sum_{C \in \phi} Z_C$ is the number of clauses satisfied by the assignment $\dot{x}$ produced by the GW algorithm.

As, for every $C \in \phi$,

$$\mathrm{Exp}[Z_C] \quad > \quad 0.63\, \hat{z}_C,$$

then we deduce that

$$\mathrm{Exp}[\sum_{C \in \phi} Z_C] \; > \; 0.63 \sum_{C \in \phi} \hat{z}_C \; \geq \; 0.63\, \mathrm{OPT}(\phi).$$

Analysis

For each clause C with t literals,
let Z_C be 1 if C is satisfied by $\dot{x}$ and 0 otherwise.

Note that $\sum_{C \in \phi} Z_C$ is the number of clauses satisfied by the assignment $\dot{x}$ produced by the GW algorithm.

As, for every $C \in \phi$,

$$\mathrm{Exp}[Z_C] \quad > \quad 0.63\, \hat{z}_C,$$

then we deduce that

$$\mathrm{Exp}[\sum_{C \in \phi} Z_C] \; > \; 0.63 \sum_{C \in \phi} \hat{z}_C \; \geq \; 0.63\, \mathrm{OPT}(\phi).$$

The GW algorithm is a 0.63-approximation for MAXSAT.

Joining ideas

If all clauses have k literals,
then Johnson's algorithm is a $(1 - \frac{1}{2^k})$-approximation,
which improves as k grows.

Joining ideas

If all clauses have k literals,
then Johnson's algorithm is a $(1 - \frac{1}{2^k})$-approximation,
which improves as k grows.

If all clauses have k literals,
then GW algorithm is a $(1 - (1 - \frac{1}{k})^k)$-approximation,
which gets worse as k grows.

Joining ideas

If all clauses have k literals,
then Johnson's algorithm is a $(1 - \frac{1}{2^k})$-approximation,
which improves as k grows.

If all clauses have k literals,
then GW algorithm is a $(1 - (1 - \frac{1}{k})^k)$-approximation,
which gets worse as k grows.

So one of the algorithms works better on formulas whose clauses are long,
and the other on formulas whose clauses are short.

Idea

Run both algorithms and output the best solution.

Joining ideas

Algorithm Combined (ϕ) ▷ Goemans and Williamson '94

1. $x_J := \text{JOHNSON}(\phi)$
2. $x_{GW} := \text{GW}(\phi)$
3. let s_J be the number of clauses of ϕ satisfied by x_J
4. let s_{GW} be the number of clauses of ϕ satisfied by x_{GW}
5. if $s_J \geq s_{GW}$ then return x_J
6. else return x_{GW} ▷ gives a 0.75-approximation

Joining ideas

Algorithm Combined (ϕ) ▷ Goemans and Williamson '94

1. $x_J := \textsc{Johnson}(\phi)$
2. $x_{GW} := \mathrm{GW}(\phi)$
3. let s_J be the number of clauses of ϕ satisfied by x_J
4. let s_{GW} be the number of clauses of ϕ satisfied by x_{GW}
5. if $s_J \geq s_{GW}$ then return x_J
6. else return x_{GW} ▷ gives a 0.75-approximation

X_J: number of clauses satisfied by Johnson's algorithm.

X_{GW}: number of clauses satisfied by GW algorithm.

Joining ideas

X_J: number of clauses satisfied by Johnson's algorithm.

X_{GW}: number of clauses satisfied by GW algorithm.

$\mathcal{C}_k$: clauses in ϕ with exactly k literals

$$\mathrm{Exp}[\max\{X_J, X_{GW}\}] \;\; \geq \;\; \mathrm{Exp}[\frac{X_J + X_{GW}}{2}]$$

Joining ideas

X_J: number of clauses satisfied by Johnson's algorithm.

X_{GW}: number of clauses satisfied by GW algorithm.

$\mathcal{C}_k$: clauses in ϕ with exactly k literals

$$\begin{aligned}
\mathrm{Exp}[\max\{X_J, X_{GW}\}] &\geq \mathrm{Exp}[\frac{X_J + X_{GW}}{2}] \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k}((1-\frac{1}{2^k}) + (1-(1-\frac{1}{k})^k)\hat{z}_C) \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k}(1-\frac{1}{2^k} + 1 - (1-\frac{1}{k})^k)\hat{z}_C
\end{aligned}$$

Joining ideas

X_J: number of clauses satisfied by Johnson's algorithm.

X_{GW}: number of clauses satisfied by GW algorithm.

$\mathcal{C}_k$: clauses in ϕ with exactly k literals

$$
\begin{aligned}
\mathrm{Exp}[\max\{X_J, X_{GW}\}] &\geq \mathrm{Exp}[\frac{X_J + X_{GW}}{2}] \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k} ((1-\frac{1}{2^k}) + (1-(1-\frac{1}{k})^k)\hat{z}_C) \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k} (1-\frac{1}{2^k} + 1-(1-\frac{1}{k})^k)\hat{z}_C \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k} \frac{3}{2}\,\hat{z}_C
\end{aligned}
$$

Joining ideas

X_J: number of clauses satisfied by Johnson's algorithm.

X_{GW}: number of clauses satisfied by GW algorithm.

$\mathcal{C}_k$: clauses in ϕ with exactly k literals

$$
\begin{aligned}
\mathrm{Exp}[\max\{X_J, X_{GW}\}] &\geq \mathrm{Exp}[\frac{X_J + X_{GW}}{2}] \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k} ((1 - \frac{1}{2^k}) + (1-(1-\frac{1}{k})^k)\hat{z}_C) \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k} (1 - \frac{1}{2^k} + 1-(1-\frac{1}{k})^k)\hat{z}_C \\
&\geq \frac{1}{2}\sum_k \sum_{C\in\mathcal{C}_k} \frac{3}{2}\,\hat{z}_C \\
&= \frac{3}{4}\,\mathrm{OPT}(\phi)
\end{aligned}
$$

Conclusions

If you like algorithms and
the use of smart ideas to design beautiful and efficient algorithms,
join the force to study approximation algorithms!

Two books on the subject

Approximation Algorithms, by Vazirani
The Design of Approximation Algorithms, by Williamson and Shmoys

THANK YOU!!!