Advanced Algorithms (Fall 2023)

Greedy and Local Search

Lecturers: 尹一通，刘景铖，栗师

Nanjing University

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Maximum-Weight Spanning Tree Problem

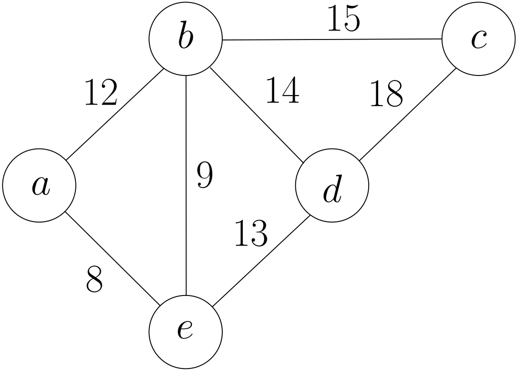
Input: Graph $G = (V, E)$ and edge weights $w \in \mathbb{Z}_{>0}^{E}$

Output: the spanning tree T of G with the maximum total weight

Maximum-Weight Spanning Tree Problem

Input: Graph $G = (V, E)$ and edge weights $w \in \mathbb{Z}^E_{>0}$

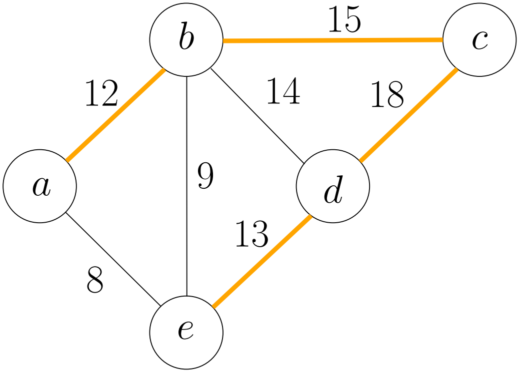
Output: the spanning tree T of G with the maximum total weight



Maximum-Weight Spanning Tree Problem

Input: Graph $G = (V, E)$ and edge weights $w \in \mathbb{Z}^E_{>0}$

Output: the spanning tree T of G with the maximum total weight

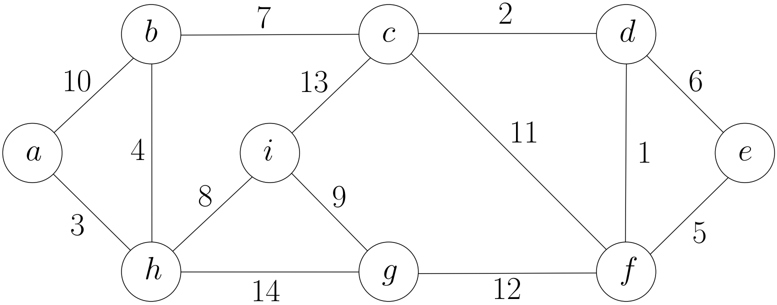


Kruskal's Algorithm for Maximum-Weight Spanning Tree

$F \leftarrow \emptyset$
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)

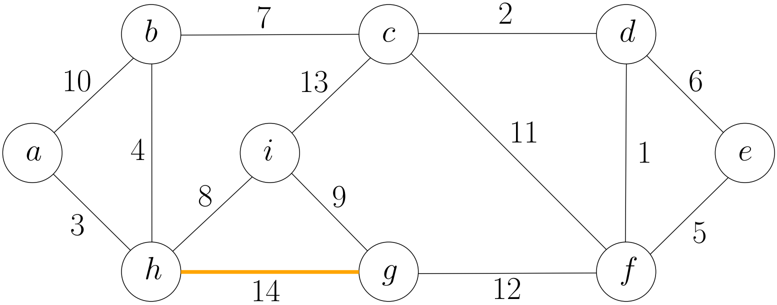
Kruskal's Algorithm for Maximum-Weight Spanning Tree

$F \leftarrow \emptyset$
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)



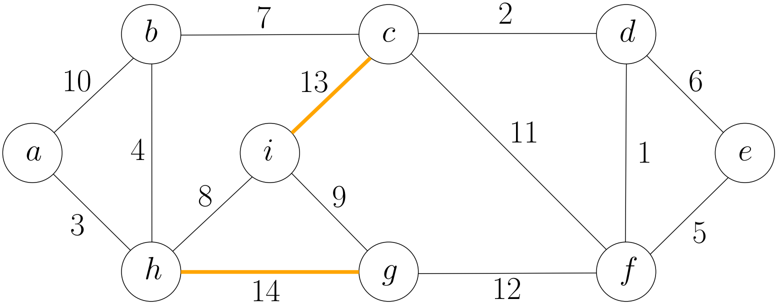
Kruskal's Algorithm for Maximum-Weight Spanning Tree

1: $F \leftarrow \emptyset$
2: sort edges in E in non-increasing order of weights w
3: **for** each edge (u, v) in the order **do**
4: **if** u and v are not connected by a path of edges in F **then**
5: $F \leftarrow F \cup \{(u, v)\}$
6: **return** (V, F)



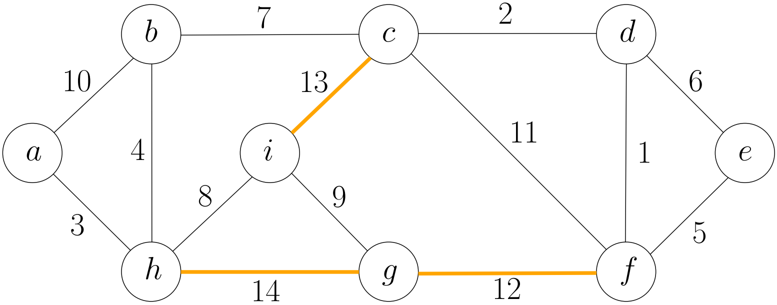
Kruskal's Algorithm for Maximum-Weight Spanning Tree

$F \leftarrow \emptyset$
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)



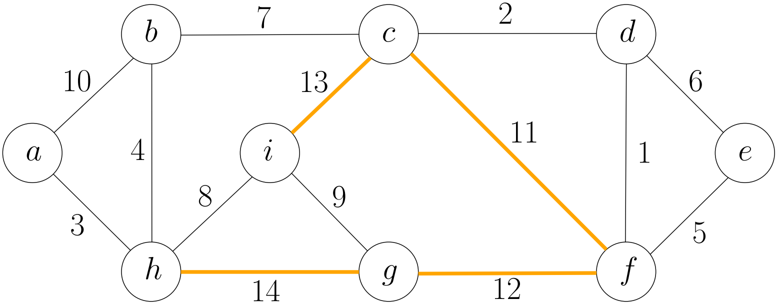
Kruskal's Algorithm for Maximum-Weight Spanning Tree

```
F ← ∅
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order do
    if u and v are not connected by a path of edges in F then
        F ← F ∪ {(u, v)}
return (V, F)
```



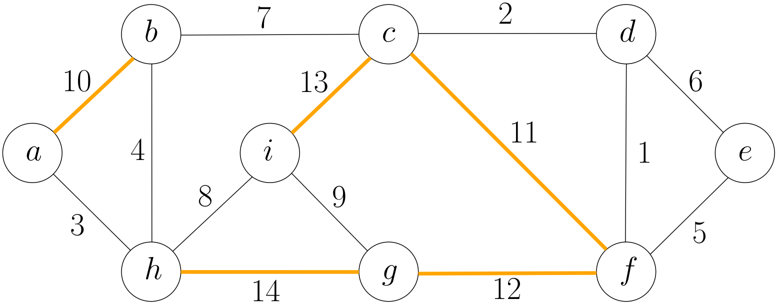
Kruskal's Algorithm for Maximum-Weight Spanning Tree

$F \leftarrow \emptyset$
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)



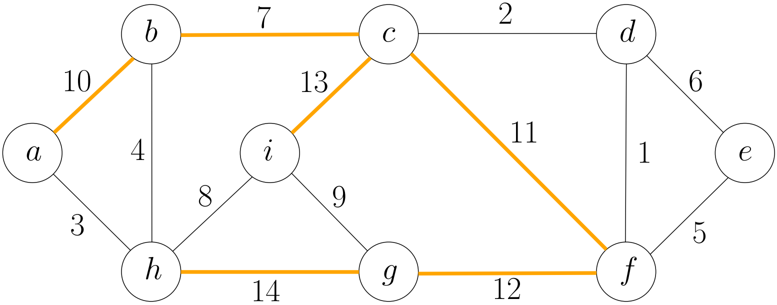
Kruskal's Algorithm for Maximum-Weight Spanning Tree

$F \leftarrow \emptyset$
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)



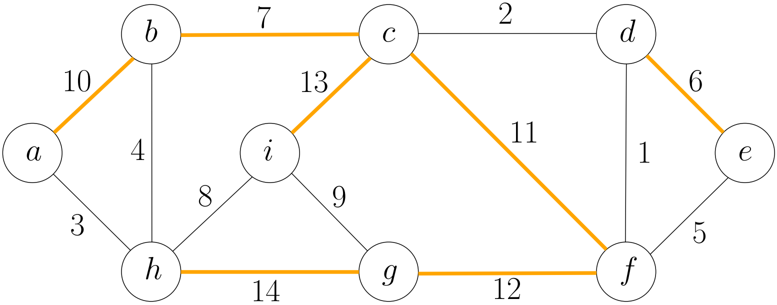
Kruskal's Algorithm for Maximum-Weight Spanning Tree

```
F ← ∅
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order do
    if u and v are not connected by a path of edges in F then
        F ← F ∪ {(u, v)}
return (V, F)
```



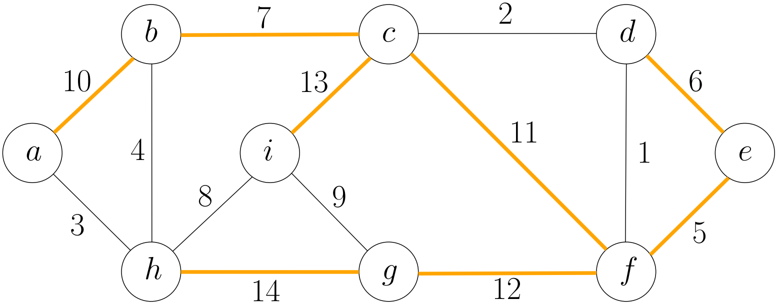
Kruskal's Algorithm for Maximum-Weight Spanning Tree

$F \leftarrow \emptyset$
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)



Kruskal's Algorithm for Maximum-Weight Spanning Tree

$F \leftarrow \emptyset$
sort edges in E in non-increasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)



Proof of Correctness of Kruskal's Algorithm

Maximum-Weight Spanning Tree (MST) with Pre-Selected Edges

Input: Graph $G = (V, E)$ and edge weights $w \in \mathbb{Z}_{>0}^E$

a set $F_0 \subseteq E$ of edges, that does not contain a cycle

Output: the maximum-weight spanning tree $T = (V, E_T)$ of G satisfying $F_0 \subseteq E_T$

Proof of Correctness of Kruskal's Algorithm

Maximum-Weight Spanning Tree (MST) with Pre-Selected Edges

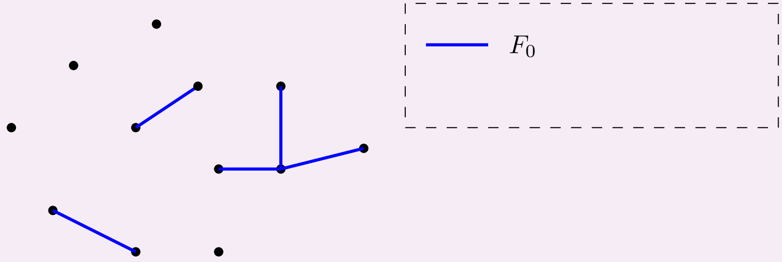
Input: Graph $G = (V, E)$ and edge weights $w \in \mathbb{Z}_{>0}^{E}$
a set $F_0 \subseteq E$ of edges, that does not contain a cycle

Output: the maximum-weight spanning tree $T = (V, E_T)$ of G satisfying $F_0 \subseteq E_T$

Lemma (Key Lemma) Given an instance $(G = (V, E), w, F_0)$ of the MST with pre-selected edges problem, let e^* be the maximum weight edge in $E \setminus F_0$ such that $F_0 \cup \{e^*\}$ does not contain a cycle. Then there is an optimum solution $T = (V, E_T)$ to the instance with $e^* \in E_T$.

Proof of Correctness of Kruskal's Algorithm

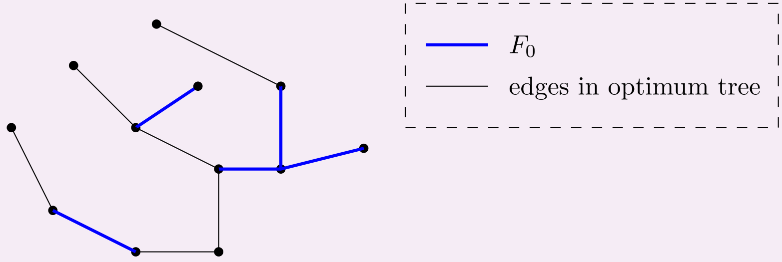
Proof of Key Lemma.



□

Proof of Correctness of Kruskal's Algorithm

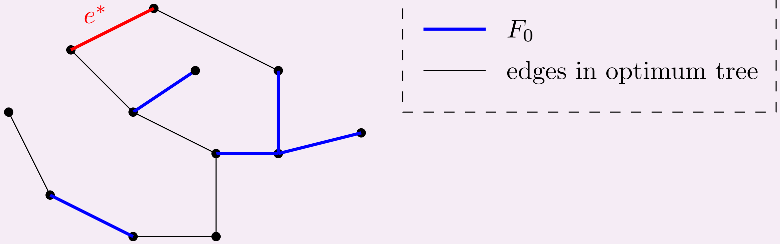
Proof of Key Lemma.



□

Proof of Correctness of Kruskal's Algorithm

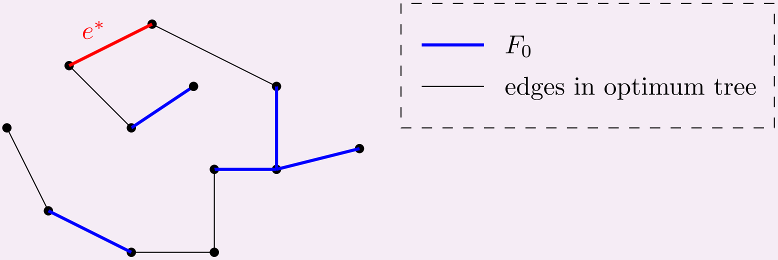
Proof of Key Lemma.



∎

Proof of Correctness of Kruskal's Algorithm

Proof of Key Lemma.



□

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Q: Does the greedy algorithm work for more general problems?

Q: Does the greedy algorithm work for more general problems?

A General Maximization Problem

Input: E: the ground set of elements

$w \in \mathbb{Z}_{>0}^E$: weight vector on elements

$\mathcal{S}$: an (implicitly given) family of subsets of E

- $\emptyset \in \mathcal{S}$
- $\mathcal{S}$ is downward closed: if $A \in \mathcal{S}, B \subsetneq A$, then $B \in \mathcal{S}$.

Output: $A \in \mathcal{S}$ that maximizes $\sum_{e \in A} w_e$

Q: Does the greedy algorithm work for more general problems?

A General Maximization Problem

Input: E: the ground set of elements

$w \in \mathbb{Z}_{>0}^{E}$: weight vector on elements

$\mathcal{S}$: an (implicitly given) family of subsets of E

- $\emptyset \in \mathcal{S}$
- $\mathcal{S}$ is downward closed: if $A \in \mathcal{S}, B \subsetneq A$, then $B \in \mathcal{S}$.

Output: $A \in \mathcal{S}$ that maximizes $\sum_{e \in A} w_e$

- maximum-weight spanning tree: $\mathcal{S} =$ family of forests

Greedy Algorithm

```
A ← ∅
sort elements in E in non-decreasing order of weights w
for each element e in the order do
    if A ∪ {e} ∈ S then A ← A ∪ {e}
return A
```

Greedy Algorithm

$A \leftarrow \emptyset$
sort elements in E in non-decreasing order of weights w
for each element e in the order **do**
    if $A \cup \{e\} \in \mathcal{S}$ **then** $A \leftarrow A \cup \{e\}$
return A

Examples where Greedy Algorithm is Not Optimum

- Knapsack Packing: given elements E, where every element has a value and a cost, and a cost budget C, the goal is to find a maximum value subset of items with cost at most C

Greedy Algorithm

1: $A \leftarrow \emptyset$
2: sort elements in E in non-decreasing order of weights w
3: **for** each element e in the order **do**
4: **if** $A \cup \{e\} \in \mathcal{S}$ **then** $A \leftarrow A \cup \{e\}$
5: **return** A

Examples where Greedy Algorithm is Not Optimum

- Knapsack Packing: given elements E, where every element has a value and a cost, and a cost budget C, the goal is to find a maximum value subset of items with cost at most C
- Maximum Weight Bipartite Graph Matching

Greedy Algorithm

$A \leftarrow \emptyset$
sort elements in E in non-decreasing order of weights w
for each element e in the order **do**
    if $A \cup \{e\} \in \mathcal{S}$ **then** $A \leftarrow A \cup \{e\}$
return A

Examples where Greedy Algorithm is Not Optimum

- Knapsack Packing: given elements E, where every element has a value and a cost, and a cost budget C, the goal is to find a maximum value subset of items with cost at most C
- Maximum Weight Bipartite Graph Matching

- Matroids: cases where greedy algorithm is optimum

Def. A (finite) matroid $\mathcal{M}$ is a pair $(E, \mathcal{I})$, where E is a finite set (called the ground set) and $\mathcal{I}$ is a family of subsets of E (called independent sets) with the following properties:

1. $\emptyset \in \mathcal{I}$.
2. (downward-closed property) If $B \subsetneq A \in \mathcal{I}$, then $B \in \mathcal{I}$.
3. (augmentation/exchange property) If $A, B \in \mathcal{I}$ and $|B| < |A|$, then there exists $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.

Def. A (finite) matroid $\mathcal{M}$ is a pair $(E, \mathcal{I})$, where E is a finite set (called the ground set) and $\mathcal{I}$ is a family of subsets of E (called independent sets) with the following properties:

1. $\emptyset \in \mathcal{I}$.
2. (downward-closed property) If $B \subsetneq A \in \mathcal{I}$, then $B \in \mathcal{I}$.
3. (augmentation/exchange property) If $A, B \in \mathcal{I}$ and $|B| < |A|$, then there exists $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.

Lemma Let $G = (V, E)$. $F \subseteq E$ is in $\mathcal{I}$ iff (V, F) is a forest. Then $(E, \mathcal{I})$ is a matroid, and it is called a graphic matroid.

Def. A (finite) matroid $\mathcal{M}$ is a pair $(E, \mathcal{I})$, where E is a finite set (called the ground set) and $\mathcal{I}$ is a family of subsets of E (called independent sets) with the following properties:

1. $\emptyset \in \mathcal{I}$.
2. (downward-closed property) If $B \subsetneq A \in \mathcal{I}$, then $B \in \mathcal{I}$.
3. (augmentation/exchange property) If $A, B \in \mathcal{I}$ and $|B| < |A|$, then there exists $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.

Lemma Let $G = (V, E)$. $F \subseteq E$ is in $\mathcal{I}$ iff (V, F) is a forest. Then $(E, \mathcal{I})$ is a matroid, and it is called a graphic matroid.

Proof of Exchange Property.

- $|B| < |A| \Rightarrow (V, B)$ has more CC than (V, A).
- Some edge in A connects two different CC of (V, B). □

Feasible Family for Knapsack Packing Does Not Satisfy Augmentation Property

- $c_1 = c_2 = 10, c_3 = 20, C = 20$.
- $\{1,2\}, \{3\} \in \mathcal{I}$, but $\{1,3\}, \{2,3\} \notin \mathcal{I}$.

Feasible Family for Knapsack Packing Does Not Satisfy Augmentation Property

- $c_1 = c_2 = 10, c_3 = 20, C = 20$.
- $\{1, 2\}, \{3\} \in \mathcal{I}$, but $\{1, 3\}, \{2, 3\} \notin \mathcal{I}$.

Feasible Family for Bipartite Matching Does Not Satisfy Augmentation Property

- Complete bipartite graph between $\{a_1, a_2\}$ and $\{b_1, b_2\}$.
- $\{(a_1, b_1), (a_2, b_2)\}, \{(a_1, b_2)\} \in \mathcal{I}$.

Feasible Family for Knapsack Packing Does Not Satisfy Augmentation Property

- $c_1 = c_2 = 10, c_3 = 20, C = 20$.
- $\{1, 2\}, \{3\} \in \mathcal{I}$, but $\{1, 3\}, \{2, 3\} \notin \mathcal{I}$.

Feasible Family for Bipartite Matching Does Not Satisfy Augmentation Property

- Complete bipartite graph between $\{a_1, a_2\}$ and $\{b_1, b_2\}$.
- $\{(a_1, b_1), (a_2, b_2)\}, \{(a_1, b_2)\} \in \mathcal{I}$.

Theorem The greedy algorithm gives optimum solution for the maximum-weight independent set problem in a matroid.

Lemma (Key Lemma)

- given: matroid $\mathcal{M} = (E, \mathcal{I})$, weights $w \in \mathbb{Z}_{>0}^{E}$, $A \in \mathcal{I}$,
- goal: find a maximum weight independent set containing A
- $e^* = \arg\max_{e \in E \setminus A: A \cup \{e\} \in \mathcal{I}} w_e$, assuming e^* exists

Lemma (Key Lemma)

- given: matroid $\mathcal{M} = (E, \mathcal{I})$, weights $w \in \mathbb{Z}_{>0}^{E}$, $A \in \mathcal{I}$,
- goal: find a maximum weight independent set containing A
- $e^* = \arg\max_{e \in E \setminus A: A \cup \{e\} \in \mathcal{I}} w_e$, assuming e^* exists
- Then, some optimum solution contains e^*

Lemma (Key Lemma)

- given: matroid $\mathcal{M} = (E, \mathcal{I})$, weights $w \in \mathbb{Z}_{>0}^E$, $A \in \mathcal{I}$,
- goal: find a maximum weight independent set containing A
- $e^* = \arg\max_{e \in E \setminus A: A \cup \{e\} \in \mathcal{I}} w_e$, assuming e^* exists
- Then, some optimum solution contains e^*

Proof.

- let $S \supseteq A, S \in \mathcal{I}$ be an optimum solution, $e^* \notin S$

Lemma (Key Lemma)

- given: matroid $\mathcal{M} = (E, \mathcal{I})$, weights $w \in \mathbb{Z}_{>0}^{E}$, $A \in \mathcal{I}$,
- goal: find a maximum weight independent set containing A
- $e^* = \arg\max_{e \in E \setminus A: A \cup \{e\} \in \mathcal{I}} w_e$, assuming e^* exists
- Then, some optimum solution contains e^*

Proof.

- let $S \supseteq A, S \in \mathcal{I}$ be an optimum solution, $e^* \notin S$

$S' \leftarrow A \cup \{e^*\}$
while $|S'| < |S|$ **do**
    let e be any element in $S \setminus S'$ with $S' \cup \{e\} \in \mathcal{I}$
    $\triangleright$ e exists due to exchange property
    $S' \leftarrow S \cup \{e\}$

Lemma (Key Lemma)

- given: matroid $\mathcal{M} = (E, \mathcal{I})$, weights $w \in \mathbb{Z}_{>0}^E$, $A \in \mathcal{I}$,
- goal: find a maximum weight independent set containing A
- $e^* = \arg\max_{e \in E \setminus A : A \cup \{e\} \in \mathcal{I}} w_e$, assuming e^* exists
- Then, some optimum solution contains e^*

Proof.

- let $S \supseteq A, S \in \mathcal{I}$ be an optimum solution, $e^* \notin S$

  $S' \leftarrow A \cup \{e^*\}$
  while $|S'| < |S|$ **do**
      let e be any element in $S \setminus S'$ with $S' \cup \{e\} \in \mathcal{I}$
          $\triangleright$ e exists due to exchange property
      $S' \leftarrow S \cup \{e\}$

- S' and S differ by exactly one element

Lemma (Key Lemma)

- given: matroid $\mathcal{M} = (E, \mathcal{I})$, weights $w \in \mathbb{Z}_{>0}^E$, $A \in \mathcal{I}$,
- goal: find a maximum weight independent set containing A
- $e^* = \arg\max_{e \in E \setminus A : A \cup \{e\} \in \mathcal{I}} w_e$, assuming e^* exists
- Then, some optimum solution contains e^*

Proof.

- let $S \supseteq A, S \in \mathcal{I}$ be an optimum solution, $e^* \notin S$

```
S' ← A ∪ {e*}
while |S'| < |S| do
    let e be any element in S \ S' with S' ∪ {e} ∈ I
                                ▷ e exists due to exchange property
    S' ← S ∪ {e}
```

- S' and S differ by exactly one element
- $w(S') := \sum_{e \in S'} w_e \geq w(S) \implies S'$ is also optimum □

Examples of Matroids

- E: the ground set $\mathcal{I}$: the family of independent sets

Examples of Matroids

- E: the ground set $\qquad$ $\mathcal{I}$: the family of independent sets
- Uniform Matroid: $k \in \mathbb{Z}_{>0}$.

  $$\mathcal{I} = \{A \subseteq E : |A| \leq k\}.$$

Examples of Matroids

- E: the ground set $\qquad$ $\mathcal{I}$: the family of independent sets
- Uniform Matroid: $k \in \mathbb{Z}_{>0}$.

  $$\mathcal{I} = \{A \subseteq E : |A| \leq k\}.$$
- Partition Matroid: partition $(E_1, E_2, \cdots, E_t)$ of E, positive integers $k_1, k_2, \cdots, k_t$

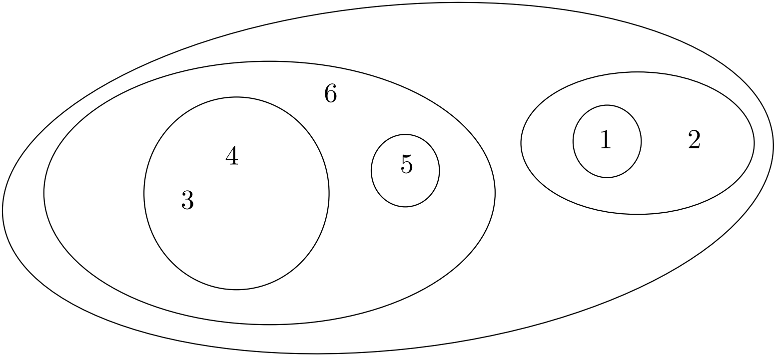
  $$\mathcal{I} = \{A \subseteq E : |A \cap E_i| \leq k_i, \forall i \in [t]\}.$$

Examples of Matroids

- E: the ground set $\qquad$ $\mathcal{I}$: the family of independent sets
- Uniform Matroid: $k \in \mathbb{Z}_{>0}$.
  $$\mathcal{I} = \{A \subseteq E : |A| \leq k\}.$$
- Partition Matroid: partition $(E_1, E_2, \cdots, E_t)$ of E, positive integers $k_1, k_2, \cdots, k_t$
  $$\mathcal{I} = \{A \subseteq E : |A \cap E_i| \leq k_i, \forall i \in [t]\}.$$
- Laminar Matroid: laminar family of subsets of E $\{E_1, E_2, \cdots, E_t\}$, positive integers $k_1, k_2, \cdots, k_t$
  $$\mathcal{I} = \{A \subseteq E : |A \cap E_i| \leq k_i, \forall i \in [t]\}.$$

Def. A family $\{E_1, E_2, \cdots, E_t\}$ of subsets of E is said to be laminar if for every two distinct subsets E_i, E_j in the family, we have $E_i \cap E_j = \emptyset$ or $E_i \subsetneq E_j$ or $E_j \subsetneq E_i$.

- $\Big\{\{1\},\{1,2\},\{3,4\},\{5\},\{3,4,5,6\},\{1,2,3,4,5,6\}\Big\}$ is a laminar family.



Examples of Matroids

- E: the ground set $\qquad$ $\mathcal{I}$: the family of independent sets
- Graphic Matroid: graph $G = (V, E)$

  $\mathcal{I} = \{A \subseteq E : (V, A) \text{ is a forest}\}$

Examples of Matroids

- E: the ground set $\qquad\qquad\qquad \mathcal{I}$: the family of independent sets
- Graphic Matroid: graph $G = (V, E)$

  $\mathcal{I} = \{A \subseteq E : (V, A) \text{ is a forest}\}$
- Transversal Matroid: a bipartite graph $G = (E \uplus B, \mathcal{E})$

  $\mathcal{I} = \{A \subseteq E : \text{there is a matching in } G \text{ covering } A\}$

Examples of Matroids

- E: the ground set $\qquad\qquad \mathcal{I}$: the family of independent sets
- Graphic Matroid: graph $G = (V, E)$

  $$\mathcal{I} = \{A \subseteq E : (V, A) \text{ is a forest}\}$$
- Transversal Matroid: a bipartite graph $G = (E \uplus B, \mathcal{E})$

  $$\mathcal{I} = \{A \subseteq E : \text{there is a matching in } G \text{ covering } A\}$$
- Linear Matroid: a vector $\vec{v}_e \in \mathbb{R}^d$ for every $e \in E$

  $$\mathcal{I} = \{A \subseteq E : \text{vectors } \{\vec{v}_e\}_{e \in A} \text{ are linearly independent}\}$$

Examples of Matroids

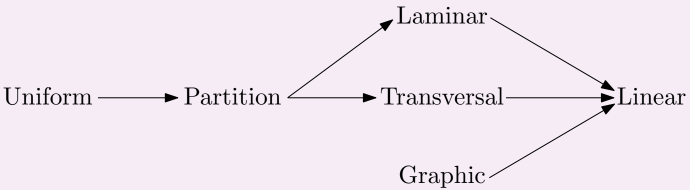
- E: the ground set $\qquad$ $\mathcal{I}$: the family of independent sets
- Graphic Matroid: graph $G = (V, E)$

  $\mathcal{I} = \{A \subseteq E : (V, A) \text{ is a forest}\}$
- Transversal Matroid: a bipartite graph $G = (E \uplus B, \mathcal{E})$

  $\mathcal{I} = \{A \subseteq E : \text{there is a matching in } G \text{ covering } A\}$
- Linear Matroid: a vector $\vec{v}_e \in \mathbb{R}^d$ for every $e \in E$

  $\mathcal{I} = \{A \subseteq E : \text{vectors } \{\vec{v}_e\}_{e \in A} \text{ are linearly independent}\}$

Relationship between matroids



Other Terminologies Related To a Matroid $\mathcal{M} = (E, \mathcal{I})$

- A subset of E that is not independent is dependent.
- A maximal indepent set is called a basis (plural: bases)
- A minimal dependent set is called a circuit

Other Terminologies Related To a Matroid $\mathcal{M} = (E, \mathcal{I})$

- A subset of E that is not independent is dependent.
- A maximal indepent set is called a basis (plural: bases)
- A minimal dependent set is called a circuit

Lemma All bases of a matroid have the same size.

Proof.

By exchange property. □

Other Terminologies Related To a Matroid $\mathcal{M} = (E, \mathcal{I})$

- A subset of E that is not independent is dependent.
- A maximal indepent set is called a basis (plural: bases)
- A minimal dependent set is called a circuit

Lemma All bases of a matroid have the same size.

Proof.

By exchange property. □

Def. Given a matroid $\mathcal{M} = (E, \mathcal{I})$, the rank of a subset A of E, denoted as $r_{\mathcal{M}}(A)$, is defined as the size of the maximum independent subset of A. $r_{\mathcal{M}} : 2^E \to \mathbb{Z}_{\geq 0}$ is called the rank function of $\mathcal{M}$.

Outline

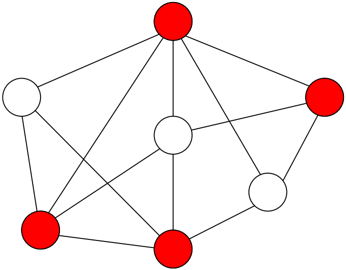
1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $(1 - \frac{1}{e})$-Approximation for Maximum Coverage
   - $(1 - \frac{1}{e})$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

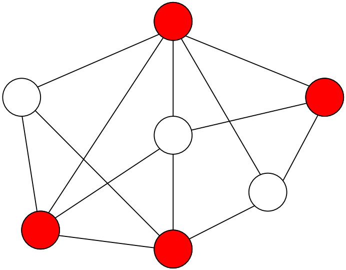
Vertex Cover Problem

Def. Given a graph $G = (V, E)$, a vertex cover of G is a subset $C \subseteq V$ such that for every $(u, v) \in E$ then $u \in C$ or $v \in C$.



Vertex Cover Problem

Def. Given a graph $G = (V, E)$, a vertex cover of G is a subset $C \subseteq V$ such that for every $(u, v) \in E$ then $u \in C$ or $v \in C$.



Vertex-Cover Problem

Input: $G = (V, E)$

Output: a vertex cover C with minimum $|C|$

First Try: A "Natural" Greedy Algorithm

Natural Greedy Algorithm for Vertex-Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let v be the vertex of the maximum degree in (V, E')
    $C \leftarrow C \cup \{v\}$,
    remove all edges incident to v from E'
return C

First Try: A "Natural" Greedy Algorithm

Natural Greedy Algorithm for Vertex-Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let v be the vertex of the maximum degree in (V, E')
    $C \leftarrow C \cup \{v\}$,
    remove all edges incident to v from E'
return C

Theorem Greedy algorithm is an $(\ln n + 1)$-approximation for vertex-cover.

First Try: A "Natural" Greedy Algorithm

Natural Greedy Algorithm for Vertex-Cover

1: $E' \leftarrow E, C \leftarrow \emptyset$
2: **while** $E' \neq \emptyset$ **do**
3: let v be the vertex of the maximum degree in (V, E')
4: $C \leftarrow C \cup \{v\}$,
5: remove all edges incident to v from E'
6: **return** C

Theorem Greedy algorithm is an $(\ln n + 1)$-approximation for vertex-cover.

- We prove it for the more general set cover problem

First Try: A "Natural" Greedy Algorithm

Natural Greedy Algorithm for Vertex-Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let v be the vertex of the maximum degree in (V, E')
    $C \leftarrow C \cup \{v\}$,
    remove all edges incident to v from E'
return C

Theorem Greedy algorithm is an $(\ln n + 1)$-approximation for vertex-cover.

- We prove it for the more general set cover problem
- The logarithmic factor is tight for this algorithm

2-Approximation Algorithm for Vertex Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let (u, v) be any edge in E'
    $C \leftarrow C \cup \{u, v\}$
    remove all edges incident to u and v from E'
return C

2-Approximation Algorithm for Vertex Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let (u, v) be any edge in E'
    $C \leftarrow C \cup \{u, v\}$
    remove all edges incident to u and v from E'
return C

- counter-intuitive: adding both u and v to C seems wasteful

2-Approximation Algorithm for Vertex Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let (u, v) be any edge in E'
    $C \leftarrow C \cup \{u, v\}$
    remove all edges incident to u and v from E'
return C

- counter-intuitive: adding both u and v to C seems wasteful
- intuition for the 2-approximation ratio:

2-Approximation Algorithm for Vertex Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let (u, v) be any edge in E'
    $C \leftarrow C \cup \{u, v\}$
    remove all edges incident to u and v from E'
return C

- counter-intuitive: adding both u and v to C seems wasteful
- intuition for the 2-approximation ratio:
  - optimum solution C^* must cover edge (u, v), using either u or v

2-Approximation Algorithm for Vertex Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let (u, v) be any edge in E'
    $C \leftarrow C \cup \{u, v\}$
    remove all edges incident to u and v from E'
return C

- counter-intuitive: adding both u and v to C seems wasteful
- intuition for the 2-approximation ratio:
  - optimum solution C^* must cover edge (u, v), using either u or v
  - we select both, so we are always ahead of the optimum solution

2-Approximation Algorithm for Vertex Cover

```
E' ← E, C ← ∅
while E' ≠ ∅ do
    let (u, v) be any edge in E'
    C ← C ∪ {u, v}
    remove all edges incident to u and v from E'
return C
```

- counter-intuitive: adding both u and v to C seems wasteful
- intuition for the 2-approximation ratio:
  - optimum solution C^* must cover edge (u, v), using either u or v
  - we select both, so we are always ahead of the optimum solution
  - we use at most 2 times more vertices than C^* does

2-Approximation Algorithm for Vertex Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let (u, v) be any edge in E'
    $C \leftarrow C \cup \{u, v\}$
    remove all edges incident to u and v from E'
return C

2-Approximation Algorithm for Vertex Cover

$E' \leftarrow E, C \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
    let (u, v) be any edge in E'
    $C \leftarrow C \cup \{u, v\}$
    remove all edges incident to u and v from E'
return C

Theorem The algorithm is a 2-approximation algorithm for vertex-cover.

2-Approximation Algorithm for Vertex Cover

1: $E' \leftarrow E, C \leftarrow \emptyset$
2: **while** $E' \neq \emptyset$ **do**
3: let (u, v) be any edge in E'
4: $C \leftarrow C \cup \{u, v\}$
5: remove all edges incident to u and v from E'
6: **return** C

Theorem The algorithm is a 2-approximation algorithm for vertex-cover.

Proof.

- Let E' be the set of edges (u, v) considered in Step 3

2-Approximation Algorithm for Vertex Cover

1: $E' \leftarrow E, C \leftarrow \emptyset$
2: **while** $E' \neq \emptyset$ **do**
3:     let (u, v) be any edge in E'
4:     $C \leftarrow C \cup \{u, v\}$
5:     remove all edges incident to u and v from E'
6: **return** C

Theorem The algorithm is a 2-approximation algorithm for vertex-cover.

Proof.

- Let E' be the set of edges (u, v) considered in Step 3
- Observation: E' is a matching and $|C| = 2|E'|$

2-Approximation Algorithm for Vertex Cover

```
1: E' ← E, C ← ∅
2: while E' ≠ ∅ do
3:     let (u, v) be any edge in E'
4:     C ← C ∪ {u, v}
5:     remove all edges incident to u and v from E'
6: return C
```

Theorem The algorithm is a 2-approximation algorithm for vertex-cover.

Proof.

- Let E' be the set of edges (u, v) considered in Step 3
- Observation: E' is a matching and $|C| = 2|E'|$
- To cover E', the optimum solution needs $|E'|$ vertices □

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Set Cover

Input: $U, |U| = n$: ground set

$S_1, S_2, \cdots, S_m \subseteq U$

Output: minimum size set $C \subseteq [m]$ such that $\bigcup_{i \in C} S_i = U$

Set Cover with Bounded Frequency f

Input: $U, |U| = n$: ground set

$S_1, S_2, \cdots, S_m \subseteq U$

every $j \in U$ appears in at most f subsets in $\{S_1, S_2, \cdots, S_n\}$

Output: minimum size set $C \subseteq [m]$ such that $\bigcup_{i \in C} S_i = U$

Set Cover with Bounded Frequency f

Input: $U, |U| = n$: ground set

$S_1, S_2, \cdots, S_m \subseteq U$

every $j \in U$ appears in at most f subsets in $\{S_1, S_2, \cdots, S_n\}$

Output: minimum size set $C \subseteq [m]$ such that $\bigcup_{i \in C} S_i = U$

Vertex Cover = Set Cover with Frequency 2

- edges $\Leftrightarrow$ elements
- vertices $\Leftrightarrow$ sets
- every edge (element) can be covered by 2 vertices (sets)

f-Approximation Algorithm for Set Cover with Frequency f

```
C ← ∅
while ⋃_{i∈C} S_i ≠ U do
    let e be any element in U \ ⋃_{i∈C} S_i
    C ← C ∪ {i ∈ [m] : e ∈ S_i}
return C
```

f-Approximation Algorithm for Set Cover with Frequency f

```
C ← ∅
while ⋃_{i∈C} S_i ≠ U do
    let e be any element in U \ ⋃_{i∈C} S_i
    C ← C ∪ {i ∈ [m] : e ∈ S_i}
return C
```

Theorem The algorithm is a f-approximation algorithm.

f-Approximation Algorithm for Set Cover with Frequency f

1: $C \leftarrow \emptyset$
2: **while** $\bigcup_{i \in C} S_i \neq U$ **do**
3: let e be any element in $U \setminus \bigcup_{i \in C} S_i$
4: $C \leftarrow C \cup \{i \in [m] : e \in S_i\}$
5: **return** C

Theorem The algorithm is a f-approximation algorithm.

Proof.

- Let U' be the set of all elements e considered in Step 3
- Observation: no set S_i contains two elements in U'
- To cover U', the optimum solution needs $|U'|$ sets
- $C \leq f \cdot |U'|$ □

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $(1 - \frac{1}{e})$-Approximation for Maximum Coverage
   - $(1 - \frac{1}{e})$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Set Cover

Input: $U, |U| = n$: ground set

$S_1, S_2, \cdots, S_m \subseteq U$

Output: minimum size set $C \subseteq [m]$ such that $\bigcup_{i \in C} S_i = U$

Set Cover

Input: $U, |U| = n$: ground set

$S_1, S_2, \cdots, S_m \subseteq U$

Output: minimum size set $C \subseteq [m]$ such that $\bigcup_{i \in C} S_i = U$

Greedy Algorithm for Set Cover

1: $C \leftarrow \emptyset, U' \leftarrow U$
2: **while** $U' \neq \emptyset$ **do**
3: choose the i that maximizes $|U' \cap S_i|$
4: $C \leftarrow C \cup \{i\}, U' \leftarrow U' \setminus S_i$
5: **return** C

- g: minimum number of sets needed to cover U

Lemma Let $u_t, t \in \mathbb{Z}_{\geq 0}$ be the number of uncovered elements after t steps. Then for every $t \geq 1$, we have

$$u_t \leq \left(1 - \frac{1}{g}\right) \cdot u_{t-1}.$$

- g: minimum number of sets needed to cover U

Lemma Let $u_t, t \in \mathbb{Z}_{\geq 0}$ be the number of uncovered elements after t steps. Then for every $t \geq 1$, we have

$$u_t \leq \left(1 - \frac{1}{g}\right) \cdot u_{t-1}.$$

Proof.

- Consider the g sets $S_1^*, S_2^*, \cdots, S_g^*$ in optimum solution
- $S_1^* \cup S_2^* \cup \cdots \cup S_g^* = U$

- g: minimum number of sets needed to cover U

Lemma Let $u_t, t \in \mathbb{Z}_{\geq 0}$ be the number of uncovered elements after t steps. Then for every $t \geq 1$, we have

$$u_t \leq \left(1 - \frac{1}{g}\right) \cdot u_{t-1}.$$

Proof.

- Consider the g sets $S_1^*, S_2^*, \cdots, S_g^*$ in optimum solution
- $S_1^* \cup S_2^* \cup \cdots \cup S_g^* = U$
- at beginning of step t, some set in $S_1^*, S_2^*, \cdots, S_g^*$ must contain $\geq \frac{u_{t-1}}{g}$ uncovered elements
- $u_t \leq u_{t-1} - \frac{u_{t-1}}{g} = \left(1 - \frac{1}{g}\right) u_{t-1}$. □

Proof of $(\ln n + 1)$-approximation.

- Let $t = \lceil g \cdot \ln n \rceil$. $u_0 = n$. Then
$$u_t \leq \left(1 - \frac{1}{g}\right)^{g \cdot \ln n} \cdot n < e^{-\ln n} \cdot n = n \cdot \frac{1}{n} = 1.$$
- So $u_t = 0$, approximation ratio $\leq \frac{\lceil g \cdot \ln n \rceil}{g} \leq \ln n + 1$. □

Proof of $(\ln n + 1)$-approximation.

- Let $t = \lceil g \cdot \ln n \rceil$. $u_0 = n$. Then

$$u_t \leq \left(1 - \frac{1}{g}\right)^{g \cdot \ln n} \cdot n < e^{-\ln n} \cdot n = n \cdot \frac{1}{n} = 1.$$

- So $u_t = 0$, approximation ratio $\leq \frac{\lceil g \cdot \ln n \rceil}{g} \leq \ln n + 1$. □

- A more careful analysis gives a H_n-approximation, where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the n-th harmonic number.
- $\ln(n+1) < H_n < \ln n + 1$.

Proof of $(\ln n + 1)$-approximation.

- Let $t = \lceil g \cdot \ln n \rceil$. $u_0 = n$. Then
$$u_t \le \left(1 - \frac{1}{g}\right)^{g \cdot \ln n} \cdot n < e^{-\ln n} \cdot n = n \cdot \frac{1}{n} = 1.$$
- So $u_t = 0$, approximation ratio $\le \frac{\lceil g \cdot \ln n \rceil}{g} \le \ln n + 1$. □

- A more careful analysis gives a H_n-approximation, where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the n-th harmonic number.
- $\ln(n+1) < H_n < \ln n + 1$.

$(1 - c) \ln n$-hardness for any $c = \Omega(1)$

Let $c > 0$ be any constant. There is no polynomial-time $(1 - c) \ln n$-approximation algorithm for set-cover, unless

- NP $\subseteq$ quasi-poly-time, [Lund, Yannakakis 1994; Feige 1998]
- P = NP. [Dinur, Steuer 2014]

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $(1 - \frac{1}{e})$-Approximation for Maximum Coverage
   - $(1 - \frac{1}{e})$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

- set cover: use smallest number of sets to cover all elements.
- maximum coverage: use k sets to cover maximum number of elements

- set cover: use smallest number of sets to cover all elements.
- maximum coverage: use k sets to cover maximum number of elements

Maximum Coverage

Input: $U, |U| = n$: ground set,

$S_1, S_2, \cdots, S_m \subseteq U$, $\qquad k \in [m]$

Output: $C \subseteq [m], |C| = k$ with the maximum $\bigcup_{i \in C} S_i$

- set cover: use smallest number of sets to cover all elements.
- maximum coverage: use k sets to cover maximum number of elements

Maximum Coverage

Input: $U, |U| = n$: ground set,
$S_1, S_2, \cdots, S_m \subseteq U$, $\quad k \in [m]$

Output: $C \subseteq [m], |C| = k$ with the maximum $\bigcup_{i \in C} S_i$

Greedy Algorithm for Maximum Coverage

1: $C \leftarrow \emptyset, U' \leftarrow U$
2: **for** $t \leftarrow 1$ to k **do**
3: choose the i that maximizes $|U' \cap S_i|$
4: $C \leftarrow C \cup \{i\}, U' \leftarrow U' \setminus S_i$
5: **return** C

Theorem Greedy algorithm gives $(1 - \frac{1}{e})$-approximation for maximum coverage.

Theorem Greedy algorithm gives $(1 - \frac{1}{e})$-approximation for maximum coverage.

Proof.

- o: max. number of elements that can be covered by k sets.
- p_t: $\#$(covered elements) by greedy algorithm after step t

Theorem Greedy algorithm gives $(1 - \frac{1}{e})$-approximation for maximum coverage.

Proof.

- o: max. number of elements that can be covered by k sets.
- p_t: #(covered elements) by greedy algorithm after step t
- $p_t \geq p_{t-1} + \dfrac{o - p_{t-1}}{k}$

Theorem Greedy algorithm gives $(1 - \frac{1}{e})$-approximation for maximum coverage.

Proof.

- o: max. number of elements that can be covered by k sets.
- p_t: #(covered elements) by greedy algorithm after step t
- $p_t \geq p_{t-1} + \dfrac{o - p_{t-1}}{k}$
- $o - p_t \leq o - p_{t-1} - \frac{o - p_{t-1}}{k} = \left(1 - \frac{1}{k}\right)(o - p_{t-1})$

Theorem Greedy algorithm gives $(1 - \frac{1}{e})$-approximation for maximum coverage.

Proof.

- o: max. number of elements that can be covered by k sets.
- p_t: #(covered elements) by greedy algorithm after step t
- $p_t \geq p_{t-1} + \dfrac{o - p_{t-1}}{k}$
- $o - p_t \leq o - p_{t-1} - \frac{o - p_{t-1}}{k} = \left(1 - \frac{1}{k}\right)(o - p_{t-1})$
- $o - p_k \leq \left(1 - \frac{1}{k}\right)^k (o - p_0) \leq \frac{1}{e} \cdot o$

Theorem Greedy algorithm gives $(1 - \frac{1}{e})$-approximation for maximum coverage.

Proof.

- o: max. number of elements that can be covered by k sets.
- p_t: #(covered elements) by greedy algorithm after step t
- $p_t \geq p_{t-1} + \dfrac{o - p_{t-1}}{k}$
- $o - p_t \leq o - p_{t-1} - \frac{o - p_{t-1}}{k} = \left(1 - \frac{1}{k}\right)(o - p_{t-1})$
- $o - p_k \leq \left(1 - \frac{1}{k}\right)^k (o - p_0) \leq \frac{1}{e} \cdot o$
- $p_k \geq \left(1 - \frac{1}{e}\right) \cdot o$ □

Theorem Greedy algorithm gives $(1-\frac{1}{e})$-approximation for maximum coverage.

Proof.

- o: max. number of elements that can be covered by k sets.
- p_t: #(covered elements) by greedy algorithm after step t
- $p_t \geq p_{t-1} + \dfrac{o - p_{t-1}}{k}$
- $o - p_t \leq o - p_{t-1} - \frac{o-p_{t-1}}{k} = \left(1 - \frac{1}{k}\right)(o - p_{t-1})$
- $o - p_k \leq \left(1 - \frac{1}{k}\right)^k (o - p_0) \leq \frac{1}{e} \cdot o$
- $p_k \geq \left(1 - \frac{1}{e}\right) \cdot o$ $\square$

- The $(1-\frac{1}{e})$-approximation extends to a more general problem.

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $(1 - \frac{1}{e})$-Approximation for Maximum Coverage
   - $(1 - \frac{1}{e})$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Def. Let $n \in \mathbb{Z}_{>0}$. A set function $f : 2^{[n]} \to \mathbb{R}$ is called submodular if it satisfies one of the following three equivalent conditions:

(1) $\forall A, B \subseteq [n]$:

$$f(A \cup B) + f(A \cap B) \leq f(A) + f(B).$$

(2) $\forall A \subseteq B \subsetneq [n], i \in [n] \setminus B$:

$$f(B \cup \{i\}) - f(B) \leq f(A \cup \{i\}) - f(A).$$

(3) $\forall A \subseteq [n], i, j \in [n] \setminus A, i \neq j$:

$$f(A \cup \{i, j\}) + f(A) \leq f(A \cup \{i\}) + f(A \cup \{j\}).$$

Def. Let $n \in \mathbb{Z}_{>0}$. A set function $f : 2^{[n]} \to \mathbb{R}$ is called submodular if it satisfies one of the following three equivalent conditions:

(1) $\forall A, B \subseteq [n]$:

$$f(A \cup B) + f(A \cap B) \leq f(A) + f(B).$$

(2) $\forall A \subseteq B \subsetneq [n], i \in [n] \setminus B$:

$$f(B \cup \{i\}) - f(B) \leq f(A \cup \{i\}) - f(A).$$

(3) $\forall A \subseteq [n], i, j \in [n] \setminus A, i \neq j$:

$$f(A \cup \{i, j\}) + f(A) \leq f(A \cup \{i\}) + f(A \cup \{j\}).$$

- (2): diminishing marginal values: the marginal value by getting i when I have B is at most that when I have $A \subseteq B$.

Def. Let $n \in \mathbb{Z}_{>0}$. A set function $f : 2^{[n]} \to \mathbb{R}$ is called submodular if it satisfies one of the following three equivalent conditions:

(1) $\forall A, B \subseteq [n]$:
$$f(A \cup B) + f(A \cap B) \leq f(A) + f(B).$$
(2) $\forall A \subseteq B \subsetneq [n], i \in [n] \setminus B$:
$$f(B \cup \{i\}) - f(B) \leq f(A \cup \{i\}) - f(A).$$
(3) $\forall A \subseteq [n], i, j \in [n] \setminus A, i \neq j$:
$$f(A \cup \{i, j\}) + f(A) \leq f(A \cup \{i\}) + f(A \cup \{j\}).$$

- (2): diminishing marginal values: the marginal value by getting i when I have B is at most that when I have $A \subseteq B$.
- $(1) \Rightarrow (2) \Rightarrow (3)$, $\qquad (3) \Rightarrow (2) \Rightarrow (1)$

Examples of Sumodular Functions

- linear function: $f(S) = \sum_{i \in S} w_i, \forall S \subseteq [n]$

Examples of Sumodular Functions

- linear function: $f(S) = \sum_{i \in S} w_i, \forall S \subseteq [n]$
- budget-additive function: $f(S) = \min \left\{ \sum_{i \in S} w_i, B \right\}, \forall S \subseteq [n]$

Examples of Sumodular Functions

- linear function: $f(S) = \sum_{i \in S} w_i, \forall S \subseteq [n]$
- budget-additive function: $f(S) = \min \left\{ \sum_{i \in S} w_i, B \right\}, \forall S \subseteq [n]$
- coverage function: given sets $S_1, S_2, \cdots, S_n \subseteq \Omega$,

$$f(C) := \left| \bigcup_{i \in C} S_i \right|, \forall C \subseteq [n]$$

Examples of Sumodular Functions

- linear function: $f(S) = \sum_{i \in S} w_i, \forall S \subseteq [n]$
- budget-additive function: $f(S) = \min \left\{ \sum_{i \in S} w_i, B \right\}, \forall S \subseteq [n]$
- coverage function: given sets $S_1, S_2, \cdots, S_n \subseteq \Omega$,

$$f(C) := \left| \bigcup_{i \in C} S_i \right|, \forall C \subseteq [n]$$

- matroid rank function: given a matroid $\mathcal{M} = ([n], \mathcal{I})$

$$r_{\mathcal{M}}(A) = \max\{|A'| : A' \subseteq A, A' \in \mathcal{I}\}, \forall A \subseteq [n]$$

Examples of Sumodular Functions

- linear function: $f(S) = \sum_{i \in S} w_i, \forall S \subseteq [n]$
- budget-additive function: $f(S) = \min \left\{ \sum_{i \in S} w_i, B \right\}, \forall S \subseteq [n]$
- coverage function: given sets $S_1, S_2, \cdots, S_n \subseteq \Omega$,

$$f(C) := \left| \bigcup_{i \in C} S_i \right|, \forall C \subseteq [n]$$

- matroid rank function: given a matroid $\mathcal{M} = ([n], \mathcal{I})$

$$r_{\mathcal{M}}(A) = \max\{|A'| : A' \subseteq A, A' \in \mathcal{I}\}, \forall A \subseteq [n]$$

- cut function: given graph $G = ([n], E)$

$$f(A) = \big|E(A, [n] \setminus A)\big|, \forall A \subseteq [n]$$

Examples of Sumodular Functions

- linear function, budget-additive function, coverage function,

Examples of Sumodular Functions

- linear function, budget-additive function, coverage function,
- matroid rank function, cut function

Examples of Sumodular Functions

- linear function, budget-additive function, coverage function,
- matroid rank function, cut function
- entropy function: given random variables $X_1, X_2, \cdots, X_n$

$$f(S) := H(X_i : i \in S), \forall S \subseteq [n]$$

Examples of Sumodular Functions

- linear function, budget-additive function, coverage function,
- matroid rank function, cut function
- entropy function: given random variables $X_1, X_2, \cdots, X_n$

$$f(S) := H(X_i : i \in S), \forall S \subseteq [n]$$

Def. A submodular function $f : 2^{[n]} \to \mathbb{R}$ is said to be monotone if $f(A) \leq f(B)$ for every $A \subseteq B \subseteq [n]$.

Examples of Sumodular Functions

- linear function, budget-additive function, coverage function,
- matroid rank function, cut function
- entropy function: given random variables $X_1, X_2, \cdots, X_n$

$$f(S) := H(X_i : i \in S), \forall S \subseteq [n]$$

Def. A submodular function $f : 2^{[n]} \to \mathbb{R}$ is said to be monotone if $f(A) \leq f(B)$ for every $A \subseteq B \subseteq [n]$.

Def. A submodular function $f : 2^{[n]} \to \mathbb{R}$ is said to be symmetric if $f(A) = f([n] \setminus A)$ for every $A \subseteq [n]$.

Examples of Sumodular Functions

- linear function, budget-additive function, coverage function,
- matroid rank function, cut function
- entropy function: given random variables $X_1, X_2, \cdots, X_n$

$$f(S) := H(X_i : i \in S), \forall S \subseteq [n]$$

Def. A submodular function $f : 2^{[n]} \to \mathbb{R}$ is said to be monotone if $f(A) \leq f(B)$ for every $A \subseteq B \subseteq [n]$.

Def. A submodular function $f : 2^{[n]} \to \mathbb{R}$ is said to be symmetric if $f(A) = f([n] \setminus A)$ for every $A \subseteq [n]$.

- coverage, matroid rank and entropy functions are monotone

Examples of Sumodular Functions

- linear function, budget-additive function, coverage function,
- matroid rank function, cut function
- entropy function: given random variables $X_1, X_2, \cdots, X_n$

$$f(S) := H(X_i : i \in S), \forall S \subseteq [n]$$

Def. A submodular function $f : 2^{[n]} \to \mathbb{R}$ is said to be monotone if $f(A) \leq f(B)$ for every $A \subseteq B \subseteq [n]$.

Def. A submodular function $f : 2^{[n]} \to \mathbb{R}$ is said to be symmetric if $f(A) = f([n] \setminus A)$ for every $A \subseteq [n]$.

- coverage, matroid rank and entropy functions are monotone
- cut function is symmetric

$(1-\frac{1}{e})$-Approximation for Submodular Maximization with Cardinality Constraint

Submodular Maximization under a Cardinality Constraint

Input: An oracle to a non-negative monotone submodular function $f: 2^{[n]} \to \mathbb{R}_{\geq 0}$, $k \in [n]$

Output: A subset $S \subseteq [n]$ with $|S| = k$, so as to maximize $f(S)$

$(1 - \frac{1}{e})$-Approximation for Submodular Maximization with Cardinality Constraint

Submodular Maximization under a Cardinality Constraint

Input: An oracle to a non-negative monotone submodular function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$, $k \in [n]$

Output: A subset $S \subseteq [n]$ with $|S| = k$, so as to maximize $f(S)$

- We can assume $f(\emptyset) = 0$

$(1-\frac{1}{e})$-Approximation for Submodular Maximization with Cardinality Constraint

Submodular Maximization under a Cardinality Constraint

Input: An oracle to a non-negative monotone submodular function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$, $k \in [n]$

Output: A subset $S \subseteq [n]$ with $|S| = k$, so as to maximize $f(S)$

- We can assume $f(\emptyset) = 0$

Greedy Algorithm for the Problem

1: $S \leftarrow \emptyset$
2: **for** $t \leftarrow 1$ to k **do**
3: choose the i that maximizes $f(S \cup \{i\})$
4: $S \leftarrow S \cup \{i\}$
5: **return** S

Theorem Greedy algorithm gives $(1 - \frac{1}{e})$-approximation for submodular-maximization under a cardinality constraint.

Theorem Greedy algorithm gives $(1-\frac{1}{e})$-approximation for submodular-maximization under a cardinality constraint.

Proof.

- o: optimum value
- p_t: value obtained by greedy algorithm after step t
- need to prove: $p_t \geq p_{t-1} + \dfrac{o - p_{t-1}}{k}$
- $o - p_t \leq o - p_{t-1} - \frac{o-p_{t-1}}{k} = \left(1 - \frac{1}{k}\right)(o - p_{t-1})$
- $o - p_k \leq \left(1 - \frac{1}{k}\right)^k (o - p_0) \leq \frac{1}{e} \cdot o$
- $p_k \geq \left(1 - \frac{1}{e}\right) \cdot o$ □

Def. A set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is sub-additive if for every two sets $A, B \subseteq [n]$, we have $f(A \cup B) \leq f(A) + f(B)$.

Def. A set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is sub-additive if for every two sets $A, B \subseteq [n]$, we have $f(A \cup B) \leq f(A) + f(B)$.

Lemma A non-negative submodular set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is sub-additive.

Def. A set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is sub-additive if for every two sets $A, B \subseteq [n]$, we have $f(A \cup B) \leq f(A) + f(B)$.

Lemma A non-negative submodular set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is sub-additive.

Proof.

For $A, B \subseteq [n]$, we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$.
So, $f(A \cup B) \leq f(A) + f(B)$ as $f(A \cap B) \geq 0$. □

Lemma Let $f : 2^{[n]} \to \mathbb{R}$ be submodular. Let $S \subseteq [n]$, and $f_S(A) = f(S \cup A) - f(S)$ for every $A \subseteq [n]$. (f_S is the marginal value function for set S.) Then f_S is also submodular.

Lemma Let $f : 2^{[n]} \to \mathbb{R}$ be submodular. Let $S \subseteq [n]$, and $f_S(A) = f(S \cup A) - f(S)$ for every $A \subseteq [n]$. (f_S is the marginal value function for set S.) Then f_S is also submodular.

Proof.

- Let $A, B \subseteq [n] \setminus S$; it suffices to consider ground set $[n] \setminus S$.

$$\begin{aligned} & f_S(A \cup B) + f_S(A \cap B) - f_S(A) + f_S(B) \\ &= f(S \cup A \cup B) - f(S) + f(S \cup (A \cap B)) - f(S) \\ &\quad - \Big(f(S \cup A) - f(S) + f(S \cup B) - f(S)\Big) \\ &= f(S \cup A \cup B) + f(S \cup (A \cap B)) - f(S \cup A) - f(S \cup B) \\ &\leq 0 \end{aligned}$$

- The last inequality is by $S \cup A \cup B = (S \cup A) \cup (S \cup B)$, $S \cup (A \cap B) = (S \cup A) \cap (S \cup B)$ and submodularity of f. □

Proof of $p_t \geq p_{t-1} + \frac{o - p_{t-1}}{k}$.

- $S^* \subseteq [n]$: optimum set, $|S^*| = k$, $o = f(S^*)$
- S: set chosen by the algorithm at beginning of time step t

  $|S| = t - 1$, $p_{t-1} = f(S)$

Proof of $p_t \geq p_{t-1} + \frac{o - p_{t-1}}{k}$.

- $S^* \subseteq [n]$: optimum set, $|S^*| = k$, $o = f(S^*)$
- S: set chosen by the algorithm at beginning of time step t
  $|S| = t - 1$, $p_{t-1} = f(S)$
- f_S is submodular and thus sub-additive

$$f_S(S^*) \leq \sum_{i \in S^*} f_S(i) \quad \Rightarrow \quad \exists i \in S^*, f_S(i) \geq \frac{1}{k} f_S(S^*)$$

Proof of $p_t \geq p_{t-1} + \frac{o - p_{t-1}}{k}$.

- $S^* \subseteq [n]$: optimum set, $|S^*| = k$, $o = f(S^*)$
- S: set chosen by the algorithm at beginning of time step t
  $|S| = t - 1$, $p_{t-1} = f(S)$
- f_S is submodular and thus sub-additive

$$f_S(S^*) \leq \sum_{i \in S^*} f_S(i) \quad \Rightarrow \quad \exists i \in S^*, f_S(i) \geq \frac{1}{k} f_S(S^*)$$

- for the i, we have

$$f(S \cup \{i\}) - f(S) \geq \frac{1}{k}(f(S^*) - f(S))$$

$$p_t \geq f(S \cup \{i\}) \geq p_{t-1} + \frac{1}{k}(o - p_{t-1}) \qquad \square$$

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. **Local Search**
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $(1 - \frac{1}{e})$-Approximation for Maximum Coverage
   - $(1 - \frac{1}{e})$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

Local Search for Maximum-Cut

Maximum-Cut

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

Output: partition of V into $(S, T = V \setminus S)$ so as to maximize $|E(S,T)|$, $E(S,T) = \{uv \in E : u \in S \wedge v \in T\}$.

Local Search for Maximum-Cut

Maximum-Cut

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

Output: partition of V into $(S, T = V \setminus S)$ so as to maximize $|E(S,T)|$, $E(S,T) = \{uv \in E : u \in S \wedge v \in T\}$.

Def. A solution (S, T) is a local-optimum if moving any vertex to its opposite side can not increase the cut value.

Local Search for Maximum-Cut

Maximum-Cut

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

Output: partition of V into $(S, T = V \setminus S)$ so as to maximize $|E(S,T)|$, $E(S,T) = \{uv \in E : u \in S \wedge v \in T\}$.

Def. A solution (S, T) is a local-optimum if moving any vertex to its opposite side can not increase the cut value.

Local-Search for Maximum-Cut

```
1: (S, T) ← any cut
2: while ∃v ∈ V, changing side of v increases cut value do
3:     switch v to the other side in (S, T)
4: return (S, T)
```

Lemma Local search gives a 2-approximation for maximum-cut.

Lemma Local search gives a 2-approximation for maximum-cut.

- d_v: degree of v

Proof.

- $\forall v \in S : E(v, S) \leq E(v, T) \Rightarrow |E(v, S)| \geq \frac{1}{2} d_v$
- $\forall v \in T : E(v, T) \leq E(v, S) \Rightarrow |E(v, T)| \geq \frac{1}{2} d_v$

Lemma Local search gives a 2-approximation for maximum-cut.

- d_v: degree of v

Proof.

- $\forall v \in S : E(v, S) \leq E(v, T) \Rightarrow |E(v, S)| \geq \frac{1}{2} d_v$
- $\forall v \in T : E(v, T) \leq E(v, S) \Rightarrow |E(v, T)| \geq \frac{1}{2} d_v$
- adding all inequalities:

$$2|E(S, T)| \geq \frac{1}{2} \sum_{v \in V} d_v = |E|.$$

- So $|E(S, T)| \geq \frac{1}{2}|E| \geq \frac{1}{2}$(value of optimum cut). □

- The following algorithm also gives a 2-approximation

Greedy Algorithm for Maximum-Cut

$S \leftarrow \emptyset, T \leftarrow \emptyset$
for every $v \in V$, in arbitrary order **do**
    adding v to S or T so as to maximize $|E(S,T)|$
return (S,T)

- The following algorithm also gives a 2-approximation

Greedy Algorithm for Maximum-Cut

1: $S \leftarrow \emptyset, T \leftarrow \emptyset$
2: **for** every $v \in V$, in arbitrary order **do**
3: adding v to S or T so as to maximize $|E(S,T)|$
4: **return** (S,T)

- [Goemans-Williamson] 0.878-approximation via Semi-definite programming (SDP)

- The following algorithm also gives a 2-approximation

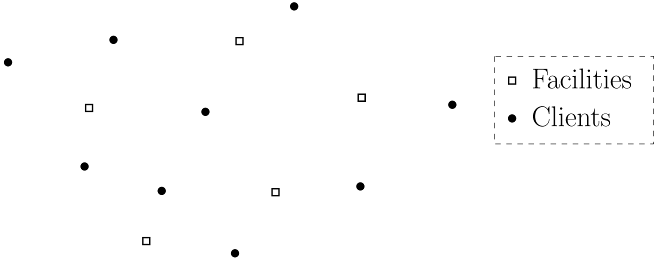
Greedy Algorithm for Maximum-Cut

1: $S \leftarrow \emptyset, T \leftarrow \emptyset$
2: **for** every $v \in V$, in arbitrary order **do**
3: adding v to S or T so as to maximize $|E(S, T)|$
4: **return** (S, T)

- [Goemans-Williamson] 0.878-approximation via Semi-definite programming (SDP)
- Under Unique-Game-Conjecture (UGC), the ratio is best possible

Outline

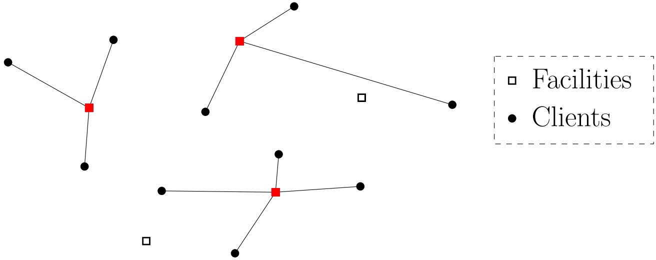
1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $(1 - \frac{1}{e})$-Approximation for Maximum Coverage
   - $(1 - \frac{1}{e})$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost



Uncapacitated Facility Location

Input: F: Facilities C: Clients

d: metric over $F \cup C$ $(f_i)_{i \in F}$: facility costs



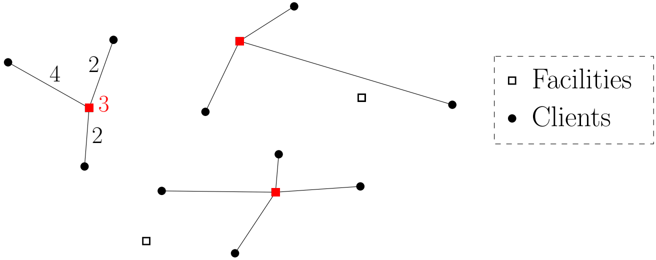
Uncapacitated Facility Location

Input: F: Facilities C: Clients

d: metric over $F \cup C$ $(f_i)_{i \in F}$: facility costs

Output: $S \subseteq F$, so as to minimize $\sum_{i \in S} f_i + \sum_{j \in C} d(j, S)$

$d(j, S)$: smallest distance between j and a facility in S



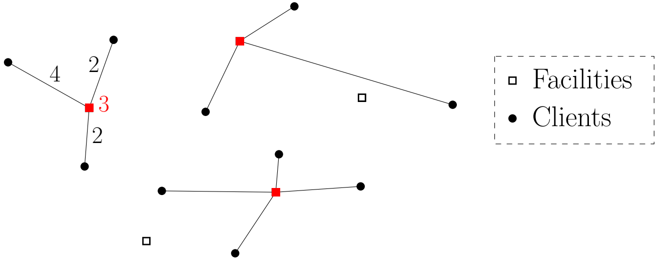
Uncapacitated Facility Location

Input: F: Facilities $\quad$ C: Clients

d: metric over $F \cup C$ $\quad$ $(f_i)_{i \in F}$: facility costs

Output: $S \subseteq F$, so as to minimize $\sum_{i \in S} f_i + \sum_{j \in C} d(j, S)$

$d(j, S)$: smallest distance between j and a facility in S



Uncapacitated Facility Location

Input: F: Facilities C: Clients

d: metric over $F \cup C$ $(f_i)_{i \in F}$: facility costs

Output: $S \subseteq F$, so as to minimize $\sum_{i \in S} f_i + \sum_{j \in C} d(j, S)$

$d(j, S)$: smallest distance between j and a facility in S

- Best-approximation ratio: 1.488-Approximation [Li, 2011]
- 1.463-hardness, $1.463 \approx$ root of $x = 1 + 2e^{-x}$

- $\text{cost}(S) := \sum_{i \in S} f_i + \sum_{j \in C} d(j, S), \forall S \subseteq F$

Local Search Algorithm for Uncapacitated Facility Location

$S \leftarrow$ arbitrary set of facilities
while exists $S' \subseteq F$ with $|S \setminus S'| \leq 1$, $|S' \setminus S| \leq 1$ and $\text{cost}(S') < \text{cost}(S)$ **do**
    $S' \leftarrow S$
return S

- The algorithm runs in pseodu-polynomial time, but we ignore the issue for now.

- $\text{cost}(S) := \sum_{i \in S} f_i + \sum_{j \in C} d(j, S), \forall S \subseteq F$

Local Search Algorithm for Uncapacitated Facility Location

1: $S \leftarrow$ arbitrary set of facilities
2: **while** exists $S' \subseteq F$ with $|S \setminus S'| \leq 1$, $|S' \setminus S| \leq 1$ and $\text{cost}(S') < \text{cost}(S)$ **do**
3: $\quad S' \leftarrow S$
4: **return** S

- The algorithm runs in pseodu-polynomial time, but we ignore the issue for now.

S is a local optimum, under the following local operations

- add$(i), i \notin S$: $S \leftarrow S \cup \{i\}$
- delete$(i), i \in S$: $S \leftarrow S \setminus \{i\}$
- swap$(i, i'), i \in S, i' \notin S$: $S \leftarrow S \setminus \{i\} \cup \{i'\}$

- S: the local optimum returned by the algorithm
- S^*: the (unknown) optimum solution

$$F := \sum_{i \in S} f_i \qquad\qquad C := \sum_{j \in C} d(j, S)$$

$$F^* := \sum_{i \in S^*} f_i \qquad\qquad C^* := \sum_{j \in C} d(j, S^*)$$

- S: the local optimum returned by the algorithm
- S^*: the (unknown) optimum solution

$$F := \sum_{i \in S} f_i \qquad\qquad C := \sum_{j \in C} d(j, S)$$

$$F^* := \sum_{i \in S^*} f_i \qquad\qquad C^* := \sum_{j \in C} d(j, S^*)$$

Lemma (analysis for connection cost) $C \leq F^* + C^*$

Lemma (analysis for facility cost) $F \leq F^* + 2C^*$

- S: the local optimum returned by the algorithm
- S^*: the (unknown) optimum solution

$$F := \sum_{i \in S} f_i \qquad C := \sum_{j \in C} d(j, S)$$

$$F^* := \sum_{i \in S^*} f_i \qquad C^* := \sum_{j \in C} d(j, S^*)$$

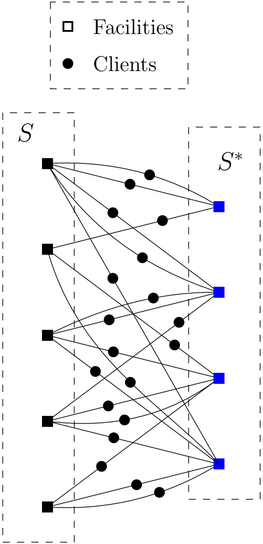
Lemma (analysis for connection cost) $C \leq F^* + C^*$

Lemma (analysis for facility cost) $F \leq F^* + 2C^*$

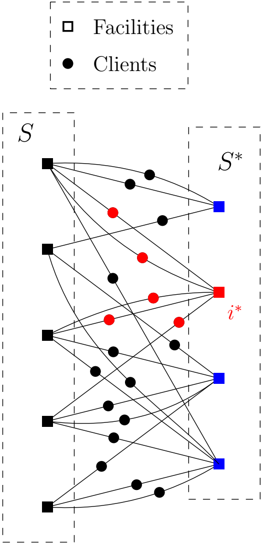
So, $F + C \leq 2F^* + 3C^* \leq 3(F^* + C^*)$

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

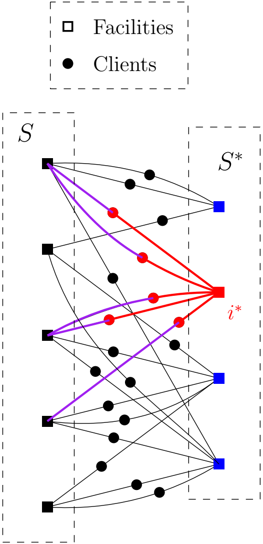


Analysis of C



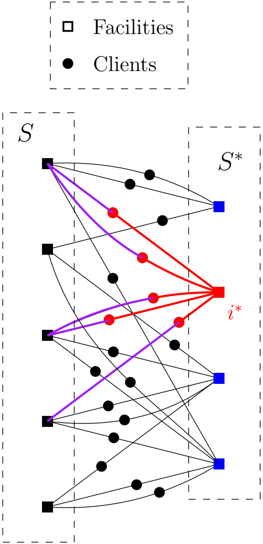
Analysis of C

- adding i^* does not increase the cost:



Analysis of C

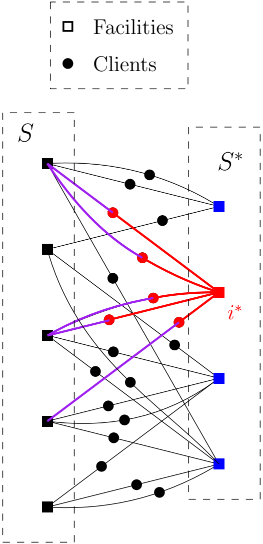
- adding i^* does not increase the cost:



Analysis of C

- adding i^* does not increase the cost:

$$\sum_{j \in \sigma^{*-1}(i^*)} c_{\sigma(j)j} \leq f_{i^*} + \sum_{j \in \sigma^{*-1}(i^*)} c_{i^*j}$$



Analysis of C

- adding i^* does not increase the cost:

$$\sum_{j \in \sigma^{*-1}(i^*)} c_{\sigma(j)j} \le f_{i^*} + \sum_{j \in \sigma^{*-1}(i^*)} c_{i^*j}$$

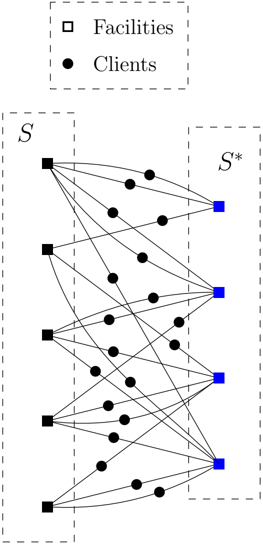
- summing up over all $i^* \in F^*$, we get

$$\sum_{j \in J} d(\sigma(j), j) \le \sum_{i^* \in F^*} f_{i^*} + \sum_{j \in J} d(\sigma^*(j), j)$$

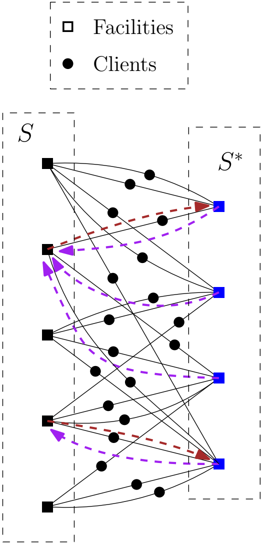
$$C \le F^* + C^*$$

Outline

1. Greedy Algorithms: Maximum-Weight Independent Set in Matroids
   - Recap: Maximum-Weight Spanning Tree Problem
   - Matroids and Maximum-Weight Independent Set in Matroids
2. Greedy Algorithms: Set Cover and Related Problems
   - 2-Approximation Algorithm for Vertex Cover
   - f-Approximation for Set-Cover with Frequency f
   - $(\ln n + 1)$-Approximation for Set-Cover
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Maximum Coverage
   - $\left(1 - \frac{1}{e}\right)$-Approximation for Submodular Maximization under a Cardinality Constraint
3. Local Search
   - Warmup Problem: 2-Approximation for Maximum-Cut
   - Local Search for Uncapacitated Facility Location Problem
   - Local Search for UFL: Analysis for Connection Cost
   - Local Search for UFL: Analysis for Facility Cost

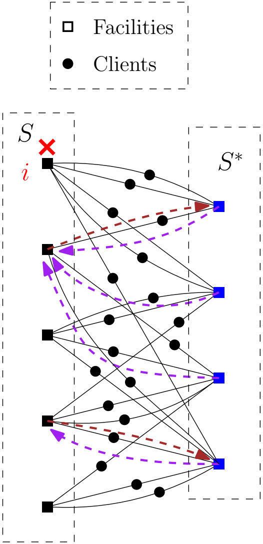


Analysis of F



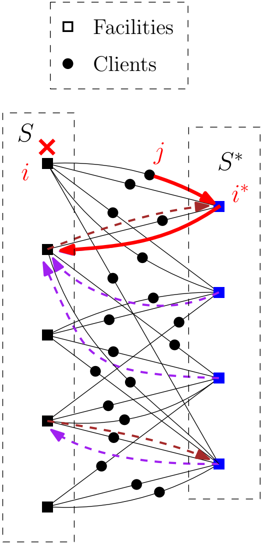
Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i



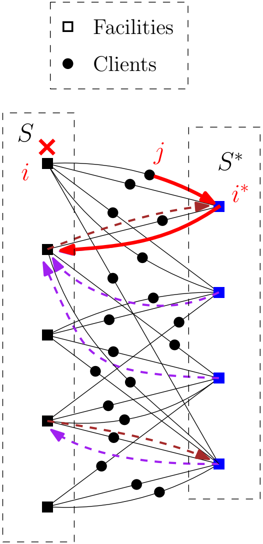
Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $i \in S, \phi^{-1}(i) = \emptyset$: consider delete$(i)$



Analysis of F

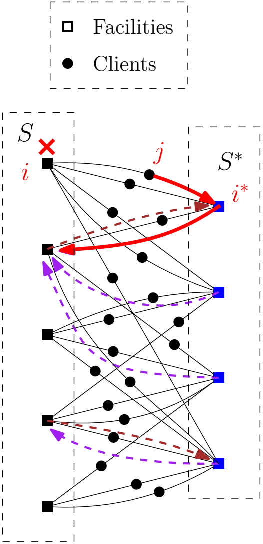
- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $i \in S, \phi^{-1}(i) = \emptyset$: consider delete$(i)$
  - $j \in \sigma^{-1}(i)$ reconnected to $i^* := \phi(\sigma^*(j))$



Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $i \in S, \phi^{-1}(i) = \emptyset$: consider delete$(i)$
  - $j \in \sigma^{-1}(i)$ reconnected to $i^* := \phi(\sigma^*(j))$
  - reconnection distance is at most

$$c_{i^*j} + c_{i^*\phi(i^*)} \leq c_{i^*j} + c_{i^*i}$$
$$\leq c_{i^*j} + c_{i^*j} + c_{ij} = 2c_{i^*j} + c_{ij}$$

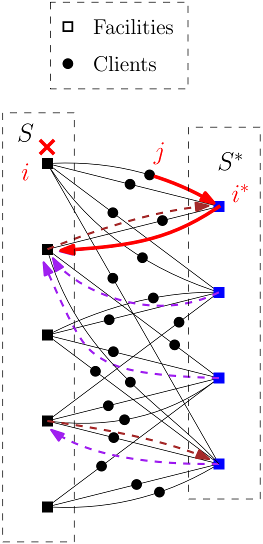


Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $i \in S, \phi^{-1}(i) = \emptyset$: consider delete$(i)$
  - $j \in \sigma^{-1}(i)$ reconnected to $i^* := \phi(\sigma^*(j))$
  - reconnection distance is at most

$$c_{i^*j} + c_{i^*\phi(i^*)} \leq c_{i^*j} + c_{i^*i}$$
$$\leq c_{i^*j} + c_{i^*j} + c_{ij} = 2c_{i^*j} + c_{ij}$$

  - distance increment is at most $2c_{i^*j}$



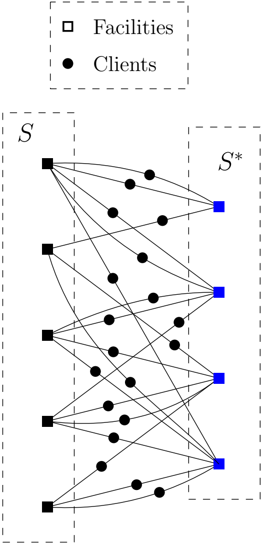
Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $i \in S, \phi^{-1}(i) = \emptyset$: consider delete$(i)$
  - $j \in \sigma^{-1}(i)$ reconnected to $i^* := \phi(\sigma^*(j))$
  - reconnection distance is at most

$$c_{i^*j} + c_{i^*\phi(i^*)} \leq c_{i^*j} + c_{i^*i}$$
$$\leq c_{i^*j} + c_{i^*j} + c_{ij} = 2c_{i^*j} + c_{ij}$$

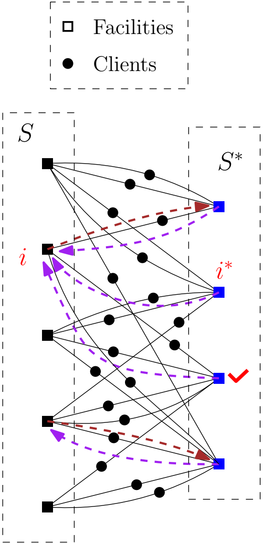
  - distance increment is at most $2c_{i^*j}$
  - by local optimality:

$$f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$$



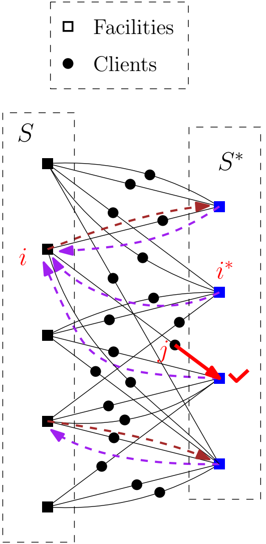
Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i



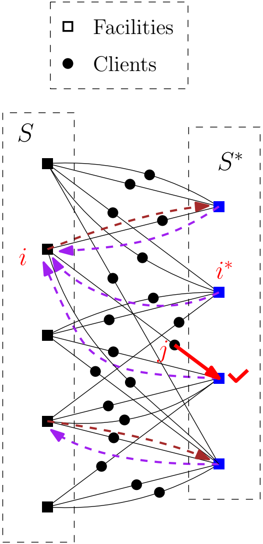
Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $\phi(i^*) = i, \psi(i) \neq i^*$: consider add$(i^*)$



Analysis of F

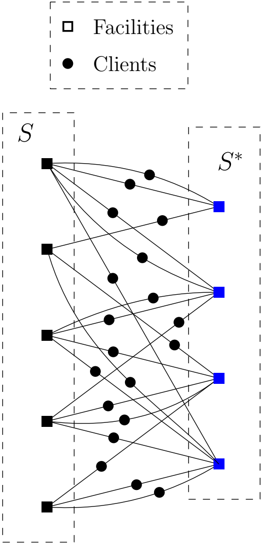
- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $\phi(i^*) = i, \psi(i) \neq i^*$: consider add$(i^*)$
  - $\sigma(j) = i, \sigma^*(j) = i^*$: reconnect j to i^*



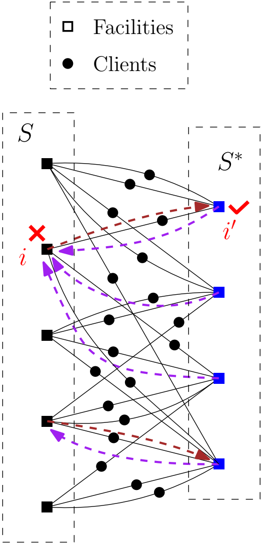
Analysis of F

- $\phi(i^*), i^* \in S^*$: closest facility in S to i^*
- $\psi(i), i \in S$: closest facility in $\phi^{-1}(i)$ to i
- $\phi(i^*) = i, \psi(i) \neq i^*$: consider add$(i^*)$
  - $\sigma(j) = i, \sigma^*(j) = i^*$: reconnect j to i^*
  - by local optimality:

$$0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} (c_{i^*j} - c_{\sigma(j)j})$$

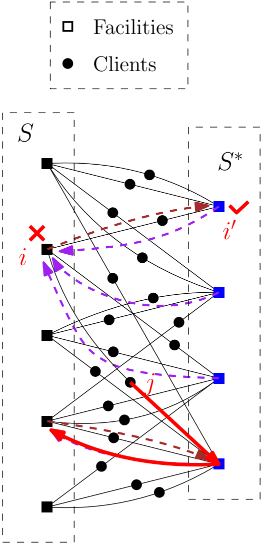


Analysis of F



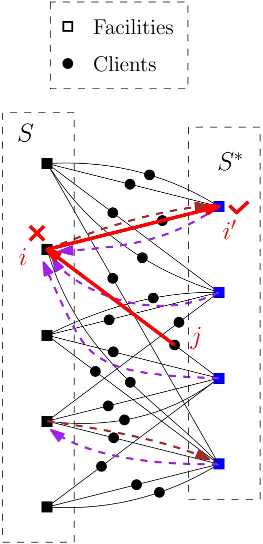
Analysis of F

- $i \in S, \phi^{-1}(i) \neq \emptyset, \phi(i') = i, \psi(i) = i'$: consider swap$(i, i')$



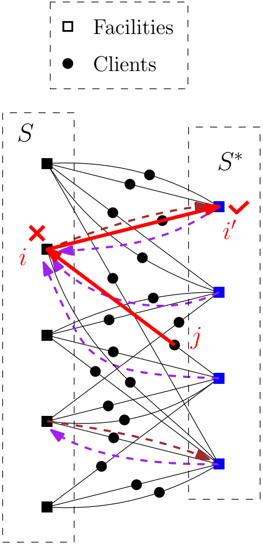
Analysis of F

- $i \in S, \phi^{-1}(i) \neq \emptyset, \phi(i') = i, \psi(i) = i'$: consider swap$(i, i')$
  - $\sigma(j) = i, \phi(\sigma^*(j)) \neq i$: reconnect j to it distance increment is at most $2c_{\sigma^*(j)j}$



Analysis of F

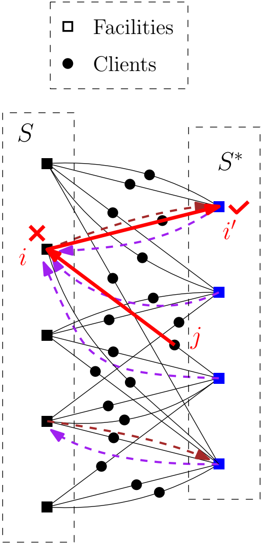
- $i \in S, \phi^{-1}(i) \neq \emptyset, \phi(i') = i, \psi(i) = i'$: consider swap$(i, i')$
  - $\sigma(j) = i, \phi(\sigma^*(j)) \neq i$: reconnect j to it distance increment is at most $2c_{\sigma^*(j)j}$
  - $\sigma(j) = i, \phi(\sigma^*(j)) = i$: reconnect j to i'



Analysis of F

- $i \in S, \phi^{-1}(i) \neq \emptyset, \phi(i') = i, \psi(i) = i'$: consider swap$(i, i')$
  - $\sigma(j) = i, \phi(\sigma^*(j)) \neq i$: reconnect j to it distance increment is at most $2c_{\sigma^*(j)j}$
  - $\sigma(j) = i, \phi(\sigma^*(j)) = i$: reconnect j to i' distance increment is at most

    $$c_{ij} + c_{ii'} - c_{ij} = c_{ii'} \leq c_{i\sigma^*(j)} \leq c_{ij} + c_{\sigma^*(j)j}$$



Analysis of F

- $i \in S, \phi^{-1}(i) \neq \emptyset, \phi(i') = i, \psi(i) = i'$: consider swap$(i, i')$
  - $\sigma(j) = i, \phi(\sigma^*(j)) \neq i$: reconnect j to it distance increment is at most $2c_{\sigma^*(j)j}$
  - $\sigma(j) = i, \phi(\sigma^*(j)) = i$: reconnect j to i' distance increment is at most
  $$c_{ij} + c_{ii'} - c_{ij} = c_{ii'} \leq c_{i\sigma^*(j)} \leq c_{ij} + c_{\sigma^*(j)j}$$
  - $$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) = i} (c_{ij} + c_{\sigma^*(j)j})$$

- $i \in S$ is not paired: $f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} \big(c_{i^*j} - c_{\sigma(j)j}\big)$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) = i} \big(c_{ij} + c_{\sigma^*(j)j}\big)$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \leq \sum_{i^* \in S^*} f_{i^*}$$

- $i \in S$ is not paired: $f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} \left(c_{i^*j} - c_{\sigma(j)j}\right)$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) = i} \left(c_{ij} + c_{\sigma^*(j)j}\right)$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \leq \sum_{i^* \in S^*} f_{i^*}$$

- $i \in S$ is not paired: $f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} (c_{i^*j} - c_{\sigma(j)j})$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i):\phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i):\phi(\sigma^*(j)) = i} (c_{ij} + c_{\sigma^*(j)j})$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \leq \sum_{i^* \in S^*} f_{i^*} + 2 \sum_{j \in D:\phi(\sigma^*(j)) \neq \sigma(j)} c_{\sigma^*(j)j}$$

- $i \in S$ is not paired: $f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} (c_{i^*j} - c_{\sigma(j)j})$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) = i} (c_{ij} + c_{\sigma^*(j)j})$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \leq \sum_{i^* \in S^*} f_{i^*} + 2 \sum_{j \in D : \phi(\sigma^*(j)) \neq \sigma(j)} c_{\sigma^*(j)j}$$

- $i \in S$ is not paired: $f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} \left(c_{i^*j} - c_{\sigma(j)j}\right)$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) = i} \left(c_{ij} + c_{\sigma^*(j)j}\right)$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \leq \sum_{i^* \in S^*} f_{i^*} + 2 \sum_{j \in D : \phi(\sigma^*(j)) \neq \sigma(j)} c_{\sigma^*(j)j} + \sum_{j \in D : \phi(\sigma^*(j)) = \sigma(j)} \left(c_{\sigma^*(j)j} - c_{\sigma(j)j} + c_{\sigma(j)j} + c_{\sigma^*(j)j}\right)$$

- $i \in S$ is not paired: $f_i \le 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \le f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} \big(c_{i^*j} - c_{\sigma(j)j}\big)$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \le f_{i'} + 2 \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) \ne i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) = i} \big(c_{ij} + c_{\sigma^*(j)j}\big)$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \le \sum_{i^* \in S^*} f_{i^*} + 2 \sum_{j \in D: \phi(\sigma^*(j)) \ne \sigma(j)} c_{\sigma^*(j)j} + 2 \sum_{j \in D: \phi(\sigma^*(j)) = \sigma(j)} c_{\sigma^*(j)j}$$

- $i \in S$ is not paired: $f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} \left(c_{i^*j} - c_{\sigma(j)j}\right)$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i): \phi(\sigma^*(j)) = i} \left(c_{ij} + c_{\sigma^*(j)j}\right)$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \leq \sum_{i^* \in S^*} f_{i^*} + 2 \sum_{j \in D} c_{\sigma^*(j)j}$$

- $i \in S$ is not paired: $f_i \leq 2 \sum_{j \in \sigma^{-1}(i)} c_{\sigma^*(j)j}$
- $i^* \in S^*$ is not paired: $0 \leq f_{i^*} + \sum_{j \in \sigma^{-1}(\phi(i^*)) \cap \sigma^{*-1}(i^*)} \left(c_{i^*j} - c_{\sigma(j)j}\right)$
- $i \in S$ and $i' \in S^*$ are paired:

$$f_i \leq f_{i'} + 2 \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) \neq i} c_{\sigma^*(j)j} + \sum_{j \in \sigma^{-1}(i) : \phi(\sigma^*(j)) = i} \left(c_{ij} + c_{\sigma^*(j)j}\right)$$

- summing all the inequalities:

$$\sum_{i \in S} f_i \leq \sum_{i^* \in S^*} f_{i^*} + 2 \sum_{j \in D} c_{\sigma^*(j)j}$$

$$F \leq F^* + 2C^*$$

$$C \leq F^* + C^*, \qquad F \leq F^* + 2C^*$$

$$\Rightarrow \qquad F + C \leq 2F^* + 3C^* \leq 3(F^* + C^*)$$

$$C \le F^* + C^*, \qquad F \le F^* + 2C^*$$
$$\Rightarrow \qquad F + C \le 2F^* + 3C^* \le 3(F^* + C^*)$$

Exercise: scaling facility costs by some $\lambda > 1$ can give a $(1 + \sqrt{2})$-approximation.

$$C \le F^* + C^*, \qquad F \le F^* + 2C^*$$
$$\Rightarrow \qquad F + C \le 2F^* + 3C^* \le 3(F^* + C^*)$$

Exercise: scaling facility costs by some $\lambda > 1$ can give a $(1+\sqrt{2})$-approximation.

- Handling pseudo-polynomial running time issue:

Local Search Algorithm for Uncapacitated Facility Location

```
S ← arbitrary set of facilities, δ ← ε/(4|F|)
while exists S' ⊆ F with |S \ S'| ≤ 1, |S' \ S| ≤ 1 and
    cost(S') < (1 − δ)cost(S) do
        S' ← S
return S
```