Advanced Algorithms
Greedy and Local Search
Max-Cut

Instance: An undirected graph $G(V, E)$.
Solution: A bipartition of $V$ into $S$ and $T$ that maximizes the cut $E(S, T) = \{ \{u, v\} \in E \mid u \in S \land v \in T \}$.

- **NP-hard.**
- One of Karp’s 21 **NP-complete** problems (reduction from the *Partition* problem).
- A typical **Max-CSP** (Constraint Satisfaction Problem).
- *Greedy* is $1/2$-approximate.
Greedy Algorithm

**Instance:** An undirected graph $G(V, E)$.

**Solution:** A bipartition of $V$ into $S$ and $T$ that maximizes the cut $E(S, T) = \{\{u, v\} \in E \mid u \in S \land v \in T\}$.

**Greedy Cut:**

initially, $S = T = \emptyset$;

for $i = 1, 2, \ldots, n$:

$v_i$ joins one of $S, T$

to maximize current $E(S, T)$;
Approximation Ratio

Algorithm $\mathcal{A}$:

Greedy Cut:
initially, $S = T = \emptyset$;
for $i = 1, 2, \ldots, n$:
    $v_i$ joins one of $S, T$
to maximize current $E(S, T)$;

$OPT_G$: value of max-cut in $G$
$SOL_G$: value of the cut returned by $\mathcal{A}$ on $G$

Algorithm $\mathcal{A}$ has approximation ratio $\alpha$ if

$$\forall \text{ instance } G, \quad \frac{SOL_G}{OPT_G} \geq \alpha$$
Approximation Algorithm

**Greedy Cut:**

initially, \( S = T = \emptyset \);

for \( i = 1,2,\ldots, n \):

\( v_i \) joins one of \( S, T \)

to maximize **current** \( E(S, T) \);

\( (S_i, T_i) \):

current \( (S, T) \) in the beginning of \( i \)-th iteration

\[ E(S, T) = \{ uv \in E \mid u \in S, v \in T \} \]

\[ |E| = \sum_{i=1}^{n} (|E(S_i, v_i)| + |E(T_i, v_i)|) \]

\[ \frac{SOL_G}{OPT_G} \geq \frac{SOL_G}{|E|} \geq \frac{1}{2} \]

\( \forall v_i, \geq 1/2 \text{ of } |E(S_i, v_i)| + |E(T_i, v_i)| \) contributes to \( SOL_G \)
Local Search

**Instance:** An undirected graph $G(V, E)$.

**Solution:** A bipartition of $V$ into $S$ and $T$ that maximizes the cut $E(S, T) = \{ \{u, v\} \in E \mid u \in S \land v \in T \}$.

**Local Search:**

initially, $(S, T)$ is an arbitrary cut;
repeat until nothing changed:
if $\exists v$ switching side increases cut $v$ switches to the other side;

*locally* improve the solution until no improvement can be made 
(local optima, fixpoint)
Local Search

**Local Search:**
initially, \((S, T)\) is an arbitrary cut;
repeat until nothing changed:
  if \(\exists v\) switching side increases cut
  \(v\) switches to the other side;

in a **local optimas**:

\[
\forall v \in S: |E(v, S)| \leq |E(v, T)| \quad \implies \quad 2|E(S, S)| \leq |E(S, T)|
\]
\[
\forall v \in T: |E(v, T)| \leq |E(v, S)| \quad \implies \quad 2|E(T, T)| \leq |E(S, T)|
\]

\[
|E(S, S)| + |E(T, T)| \leq |E(S, T)|
\]

\[
OPT \leq |E| = |E(S, S)| + |E(T, T)| + |E(S, T)| \leq 2|E(S, T)|
\]
\[
\implies \quad |E(S, T)| \geq \frac{1}{2}OPT
\]
Scheduling
Scheduling

$m$ machines

$n$ jobs

processing time $p_j$

3
1
4
2
6
3
5
2
4
3
Scheduling

$m$ machines

$n$ jobs with processing time $p_j$

Completion time:

\[ C_i = \sum_{j: \text{jobs assigned to machine } i} p_j \]

Makespan:

\[ C_{\max} = \max_{1 \leq i \leq \cdot} C_i \]
Instance: \( n \) jobs \( j = 1, \ldots, n \) with processing times \( p_j \in \mathbb{R}^+ \)
Solution: An assignment of \( n \) jobs to \( m \) identical machines that minimizes the makespan \( C_{\text{max}} \)

“minimum makespan on identical machines”: \( P | | C_{\text{max}} \)

Graham’s “\( \alpha \ | \beta \ | \gamma \)” notation for scheduling

- \( \alpha \): machine environment
  - 1: a single machine;
  - P: \( m \) identical machines;
  - Q: \( m \) machines with different speed \( s_i \), the length of job \( j \) on machine \( i \) is \( p_j/s_i \);
  - R: \( m \) unrelated machines, the length of job \( j \) on machine \( i \) is \( p_{ij} \);
- \( \beta \): job characteristics
  - \( r_j \): release times; \( d_j \): deadlines; \( \text{pmtn} \): preemption;
- \( \gamma \): objective
  - \( C_{\text{max}} \): makespan; \( \sum_i C_i \): total completion time; \( L_{\text{max}} \): maximum lateness;
**Instance:** \( n \) jobs \( j = 1, \ldots, n \) with processing times \( p_j \in \mathbb{R}^+ \)

**Solution:** An assignment of \( n \) jobs to \( m \) identical machines that minimizes the *makespan* \( C_{\text{max}} \)

“minimum *makespan* on *identical* machines”: \( P | | C_{\text{max}} \)

- Reducible from the *partition* problem:

**Instance:** \( n \) numbers \( x_1, \ldots, x_n \in \mathbb{Z}^+ \)

Determine whether \( \exists \) a partition of \( \{1, 2, \ldots, n\} \) into \( A \) and \( B \) such that \( \sum_{i \in A} x_i = \sum_{i \in B} x_i \).

- One of Karp’s 21 *NPC* problems
Approximation Ratio

**Instance:** $n$ jobs $j = 1, \ldots, n$ with processing times $p_j \in \mathbb{R}^+$

**Solution:** An assignment of $n$ jobs to $m \text{ identical}$ machines that minimizes the *makespan* $C_{\text{max}}$

An algorithm $\mathcal{A}$ for a minimization problem has **approximation ratio** $\alpha$ if

$$\forall \text{ instance } I, \quad \frac{\text{SOL}_I}{\text{OPT}_I} \leq \alpha$$

- $\text{SOL}_I$: solution returned by the algorithm on instance $I$
- $\text{OPT}_I$: optimal solution of instance $I$
Graham’s *List Algorithm*:

For $j = 1, 2, \ldots, n$:
- assign job $j$ to the current least heavily loaded machine;

$$OPT \geq \frac{1}{m} \sum_{j=1}^{n} p_j$$

$$OPT \geq \max_{1 \leq j \leq n} p_j$$
List algorithm (Graham 1966):

For \( j = 1, 2, \ldots, n \):

assign job \( j \) to the current
least heavily loaded machine;

- \( n \) jobs with processing times \( p_1, \ldots, p_n \) assigned to \( m \) machines:

- Optimal makespan:  \( OPT \geq \max_{1 \leq j \leq n} p_j \)  \( OPT \geq \frac{1}{m} \sum_{j=1}^{n} p_j \)

- Solution returned by the List algorithm:
  - suppose \( C_{\text{max}} = C_{i^*} \leq 2 \cdot OPT \)
  - and the last job assigned to machine \( i^* \) is \( \ell \)

- Before job \( \ell \) is assigned, machine \( i^* \) is the least heavily loaded

\[ C_{i^*} - p_\ell \leq \frac{1}{m} \sum_{1 \leq j \leq n} p_j \leq OPT \]

\[ p_\ell \leq \max_{1 \leq j \leq n} p_j \leq OPT \]
List algorithm (Graham 1966):

For \( j = 1, 2, \ldots, n \):

assign job \( j \) to the current
least heavily loaded machine;

- \( n \) jobs with processing times \( p_1, \ldots, p_n \) assigned to \( m \) machines:

- Optimal makespan: \( \text{OPT} \geq \max_{1 \leq j \leq n} p_j \) \( \text{OPT} \geq \frac{1}{m} \sum_{j=1}^{n} p_j \)

- Solution returned by the List algorithm:

  - suppose \( C_{\text{max}} = C_{i^*} \leq \left(1 - \frac{1}{m}\right) p_{\ell} + \frac{1}{m} \sum_{1 \leq j \leq n} p_j \leq \left(2 - \frac{1}{m}\right) \text{OPT} \)

  - and the last job assigned to machine \( i^* \) is \( \ell \)

- Before job \( \ell \) is assigned, machine \( i^* \) is the least heavily loaded

\[ C_{i^*} - p_\ell \leq \frac{1}{m} \sum_{j \neq \ell} p_j \]

\[ p_\ell \leq \max_{1 \leq j \leq n} p_j \]
Graham’s *List* Algorithm

**List algorithm** (Graham 1966):

For \( j = 1, 2, \ldots, n \):

assign job \( j \) to the current least heavily loaded machine;

- \( n \) jobs are assigned to \( m \) machines
- The *List* algorithm returns a schedule with makespan:

\[
C_{\text{max}} \leq \left( 2 - \frac{1}{m} \right) OPT
\]

- This is tight in the worst case.
Local Search

*locally* improve the solution until no improvement can be made (local optima, fixpoint)

Local search:
Start from an arbitrary schedule; repeat until no job is reassigned (a local optima):
- if the last finished job $\ell$ can finish earlier by moving to machine $i$:
  - transfer job $\ell$ to machine $i$;
Local search:

Start from an arbitrary schedule; repeat until no job is reassigned (a local optima):

if the last finished job $\ell$ can finish earlier by moving to machine $i$
transfer job $\ell$ to machine $i$;

- **Optimal makespan:**
  \[
  \text{OPT} \geq \max_{1 \leq j \leq n} p_j
  \]
  \[
  \text{OPT} \geq \frac{1}{m} \sum_{1 \leq j \leq n} p_j
  \]

- **In a local optima:**
  - suppose $C_{\text{max}} = C_{i^*} \leq \left(1 - \frac{1}{m}\right) p_{\ell} + \frac{1}{m} \sum_{1 \leq j \leq n} p_j \leq \left(2 - \frac{1}{m}\right) \text{OPT}$
  - and job $\ell$ finishes the last

- **local optima** $\implies C_{i^*} - p_{\ell}$ is the least heavy load
  \[
  C_{i^*} - p_{\ell} \leq \frac{1}{m} \sum_{j \neq \ell} p_j
  \]
  \[
  p_{\ell} \leq \max_{1 \leq j \leq n} p_j
  \]
Local search:
Start from an arbitrary schedule; repeat until no job is reassigned (a local optima):
if the last finished job \( \ell \) can finish earlier by moving to machine \( i \) transfer job \( \ell \) to machine \( i \);

For a local optima:
\[
C_{\text{max}} \leq \left( 2 - \frac{1}{m} \right) \text{OPT}
\]

List algorithm (Graham 1966):
For \( j = 1, 2, \ldots, n \):
assign job \( j \) to the current least heavily loaded machine;

• the schedule returned by the List algorithm must be a local optima

\[
C_{\text{max}} \leq \left( 2 - \frac{1}{m} \right) \text{OPT}
\]
Longest Processing Time (LPT)

\( m \) machines

\( n \) jobs

**List algorithm** (Graham 1966):

For \( j = 1, 2, \ldots, n \):

assign job \( j \) to the current least heavily loaded machine;
**Longest Processing Time (LPT)**

\[ p_1 \geq p_2 \geq \cdots \geq p_n; \]

For \( j = 1, 2, \ldots, n \):

- assign job \( j \) to the current least heavily loaded machine;

- **Optimal makespan:**
  \[ OPT \geq \frac{1}{m} \sum_{1 \leq j \leq n} p_j \]

- Solution returned by the \( LPT \) algorithm:
  - suppose \( C_{\text{max}} = C_{i*} \leq \frac{3}{2} \cdot OPT \)
  - and the last job assigned to machine \( i* \) is \( \ell \)

- Before job \( \ell \) is assigned, machine \( i* \) is the least heavily loaded

\[ \implies C_{i*} - p_\ell \leq \frac{1}{m} \sum_{1 \leq j \leq n} p_j \leq OPT \]

**WLOG:** \( \ell > m \implies p_\ell \leq p_{m+1} \)

**Pigeonhole:** \( OPT \geq p_m + p_{m+1} \geq 2p_{m+1} \)

\[ \implies p_\ell \leq \frac{1}{2}OPT \]
Solution returned by the LPT algorithm:

- makespan $C_{\text{max}} \leq \frac{3}{2} \cdot \text{OPT}$

Can be improved to $4/3$-approx. with a more careful analysis.

The problem of minimum makespan on identical machines has a PTAS (Polynomial-Time Approximation Scheme):

$$\forall \epsilon > 0, \text{ a } (1 + \epsilon)\text{-approx. solution can be returned in time } f(\epsilon) \cdot \text{poly}(n)$$
Online Scheduling

$m$ machines  
$n$ jobs arrive one-by-one

schedule decision must be made when a job arrives without seeing jobs in the future

*List algorithm* (Graham 1966):

Upon receiving a job:

assign the job to the current least heavily loaded machine;
Competitive Analysis

List algorithm (Graham 1966):
Upon receiving a job:
assign the job to the current
least heavily loaded machine;

the list algorithm is \((2 - 1/m)\)-competitive

An online algorithm \( \mathcal{A} \) for a minimization problem has competitive ratio \( \alpha \) if
\[
\forall \text{ instance } I, \quad \frac{SOL_I}{OPT_I} \leq \alpha
\]

- \( SOL_I \): solution returned by the online algorithm on instance \( I \)
- \( OPT_I \): solution returned by an optimal offline algorithm on \( I \)
Set Cover
Set Cover

Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$. Find the smallest $C \subseteq \{1, \ldots, m\}$ s.t. $\bigcup_{i \in C} S_i = U$. 
Hitting Set

Instance: A sequence of subsets \( S_1, \ldots, S_n \subseteq U \). Find the smallest \( H \subseteq U \) s.t. \( \forall i : S_i \cap H \neq \emptyset \).
Set Cover

Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.
Find the smallest $C \subseteq \{1, \ldots, m\}$ s.t. $\bigcup_{i \in C} S_i = U$.

- **NP-hard**
- one of Karp’s 21 **NP-complete** problems
- frequency of an element

$$\text{frequency}(x) = \left| \left\{ i \mid x \in S_i \right\} \right|$$
Vertex Cover

**Instance:** An undirected graph $G(V, E)$. Find the smallest $C \subseteq V$ that intersects all edges.
Vertex Cover

**Instance:** An undirected graph $G(V, E)$. Find the smallest $C \subseteq V$ that intersects all edges.

- **NP-hard**
- one of Karp’s 21 **NP-complete problems**

$\text{VC is NP-hard } \implies \text{SC is NP-hard}$
Greedy Set Cover

**Instance:** A sequence of subsets \( S_1, \ldots, S_m \subseteq U \).
Find the smallest \( C \subseteq \{1, \ldots, m\} \) s.t. \( \bigcup_{i \in C} S_i = U \).

**Greedy Cover:**
initially \( C = \emptyset \);
while \( U \neq \emptyset \) do:
    add \( i \) with largest \( |S_i \cap U| \) to \( C \);
    \( U = U \setminus S_i \);
### Instance

A sequence of subsets $S_1, \ldots, S_m \subseteq U$.

### Greedy Cover

- Initially $C = \emptyset$;
- While $U \neq \emptyset$ do:
  - Add $i$ with largest $|S_i \cap U|$ to $C$;
  - $U = U \setminus S_i$;

$$|C| = \sum_{x \in U} \text{price}(x)$$

### Averaging Principle

- Require $\geq \frac{|U|}{\max_i |S_i|}$ sets to cover $U$
- $OPT \geq \frac{|U|}{\max_i |S_i|}$

- $x_1$: first element covered by the **GreedyCover** algorithm

$$\text{price}(x_1) = \frac{1}{\max_i |S_i|} \implies \text{price}(x_1) \leq \frac{OPT}{|U|}$$
**Instance:** A sequence of subsets $S_1, \ldots, S_m \subseteq U$.

- $x_1, \ldots, x_\ell$ : covered in the 1st iteration in GreedyCover

**Greedy Cover:**

initially $C = \emptyset$;

while $U \neq \emptyset$ do:

add $i$ with largest $|S_i \cap U|$ to $C$;

$U = U \setminus S_i$;

$$|C| = \sum_{x \in U} \text{price}(x)$$

**price**

- $x_1$ : price $= \frac{1}{3}$
- $x_2$ : price $= \frac{1}{3}$
- $x_3$ : price $= 1$
- $x_4$ : price $= \frac{1}{3}$
- $x_5$ : price $= 1$

∀ $1 \leq k \leq \ell$:

$$\text{price}(x_k) = \text{price}(x_1) = \frac{1}{\max_i |S_i|}$$

∀ $1 \leq k \leq \ell$:

$$\text{price}(x_k) \leq \frac{OPT}{|U|} \leq \frac{OPT}{|U| - k + 1}$$
**Instance:** A sequence of subsets $S_1, \ldots, S_m \subseteq U$.

- $x_1, \ldots, x_\ell$ : covered in the 1st iteration in **GreedyCover**
- $x_{\ell+1}$ : 1st element covered by **GreedyCover** on a new instance $I'$ with $|U'| = |U| - \ell$ and $OPT' \leq OPT$

For $k = \ell + 1$:

$$\text{price}(x_k) \leq \frac{OPT'}{|U'|} \leq \frac{OPT}{|U| - k + 1}$$
**Instance:** A sequence of subsets $S_1, \ldots, S_m \subseteq U$.

**Greedy Cover:**

- Initially $C = \emptyset$;
- While $U \neq \emptyset$ do:
  - Add $i$ with largest $|S_i \cap U|$ to $C$;
  - $U = U \setminus S_i$;

\[ |C| = \sum_{x \in U} \text{price}(x) \]

- $x_k$: $k$th element covered by the **GreedyCover** algorithm

\[ \text{price}(x_k) \leq \frac{OPT}{|U| - k + 1} \]

\[ \text{SOL} = \sum_{k=1}^{n=|U|} \text{price}(x_k) \leq \sum_{k=1}^{n} \frac{OPT}{n - k + 1} = H_n \cdot OPT \]

Harmonic number
Approximation of Set Cover

**Greedy Cover:**

Initially $C = \emptyset$;

while $U \neq \emptyset$ do:

add $i$ with largest $|S_i \cap U|$ to $C$;

$U = U \setminus S_i$;

- **GreedyCover** has approx. ratio $H_n = (1 + o(1))\ln n$.

- [Lund, Yannakakis 1994; Feige 1998] There is no poly-time $(1 - o(1))\ln n$-approx. algorithm unless $\textbf{NP} \subseteq$ quasi-poly-time.

- [Ras, Safra 1997] For some constant $c$ there is no poly-time $c \ln n$-approximation algorithm unless $\textbf{NP} = \textbf{P}$.

- [Dinur, Steuer 2014] There is no poly-time $(1 - o(1))\ln n$-approximation algorithm unless $\textbf{NP} = \textbf{P}$. 
Submodular Optimization
**Set Cover with Budget**

**Instance:** A sequence of subsets \( S_1, \ldots, S_n \subseteq U \).

*(Minimum set cover)*

Find the smallest \( C \subseteq \{1, \ldots, n\} \) s.t. \( \bigcup_{i \in C} S_i = U \).

*(Maximum \( k \)-cover)*

Find \( C \subseteq \{1, \ldots, n\} \) with \(|C| \leq k\) to maximize \( \bigcup_{i \in C} S_i \).

- Objective and constraint are switched.
- Max-\( k \)-cover can solve minimum set cover
- Max-\( k \)-cover is **NP**-hard
Instance: A sequence of subsets $S_1, \ldots, S_n \subseteq U$. Find $C \subseteq \{1, \ldots, n\}$ with $|C| \leq k$ to maximize $\bigcup_{i \in C} S_i$.

Greedy Cover:

initially $C = \emptyset$;
while $|C| < k$ do:
    add $i$ with largest $|S_i \cap U|$ to $C$;
    $U = U \setminus S_i$;

• $\Delta_\ell$: # of elements covered additionally in the $\ell$th iteration
• $\Sigma_\ell$: # of elements covered within the first $\ell$ iterations

$$\Sigma_\ell = \Sigma_{\ell-1} + \Delta_\ell \quad \quad \Sigma_\ell = \sum_{j=1}^{\ell} \Delta_j \quad \quad SOL = \Sigma_k$$
Greedy Cover:

initially \( C = \emptyset \);
while \(| C | < k\) do:
    add \( i \) with largest \(| S_i \cap U |\) to \( C \);
    \( U = U \setminus S_i \);

\( \Delta_{\ell} \geq \frac{1}{k} (OPT - \Sigma_{\ell-1}) \)

• \( \Delta_{\ell} \): # of elements covered additionally in the \( \ell \)th iteration
• \( \Sigma_{\ell} \): # of elements covered within the first \( \ell \) iterations

• # of elements covered in OPT but not in the first \( \ell - 1 \) iterations are \( \geq OPT - \Sigma_{\ell-1} \)
• There are at most \( k \) sets in OPT.
• There is a set in OPT that can cover (in addition to the \( \Sigma_{\ell-1} \) elements covered in the first \( \ell - 1 \) iterations) \( \geq \frac{1}{k} (OPT - \Sigma_{\ell-1}) \) elements.
• \textit{GreedyCover} will select that set (or a better set) in the \( \ell \)th iteration.
Greedy Cover:

Initially $C = \emptyset$;

While $|C| < k$ do:

1. Add $i$ with largest $|S_i \cap U|$ to $C$;
2. $U = U \setminus S_i$;

$\Delta_\ell \geq \frac{1}{k} (OPT - \Sigma_{\ell-1}) \implies OPT - \Sigma_\ell \leq \left(1 - \frac{1}{k}\right) (OPT - \Sigma_{\ell-1})$

$\Sigma_\ell - \Sigma_{\ell-1} \geq \frac{1}{k} (OPT - \Sigma_{\ell-1})$

- $\Delta_\ell$: # of elements covered additionally in the $\ell$th iteration
- $\Sigma_\ell$: # of elements covered within the first $\ell$ iterations
Greedy Cover:

initially $C = \emptyset$;
while $|C| < k$ do:
    add $i$ with largest $|S_i \cap U|$ to $C$;
    $U = U \setminus S_i$;

$\Delta_\ell \geq \frac{1}{k}(OPT - \Sigma_{\ell-1})$ \implies OPT - \Sigma_{\ell} \leq \left(1 - \frac{1}{k}\right)(OPT - \Sigma_{\ell-1})$

$\implies OPT - \Sigma_k \leq \left(1 - \frac{1}{k}\right)^k OPT \leq \frac{1}{e}OPT$

$\implies SOL = \Sigma_k \geq \left(1 - \frac{1}{e}\right)OPT \quad (1 - 1/e)$-approx

- $\Delta_\ell$: # of elements covered additionally in the $\ell$th iteration
- $\Sigma_\ell$: # of elements covered within the first $\ell$ iterations

[Feige 1998] There is no poly-time $(1 - 1/e + \epsilon)$-approximation algorithm unless $\textbf{NP} = \textbf{P}$
Submodular Function

Submodular function:
A set function \( f : 2^{[n]} \rightarrow \mathbb{R} \) is submodular if
\[
\forall S, T \subseteq [n] : f(S \cup T) \leq f(S) + f(T) - f(S \cap T)
\]

Proposition: For set function \( f : 2^{[n]} \rightarrow \mathbb{R} \), define:
\[
\forall S \subseteq [n], \forall i \in [n] : \quad f_S(i) \equiv f(S \cup \{i\}) - f(S)
\]
A set function \( f : 2^{[n]} \rightarrow \mathbb{R} \) is submodular iff:
\[
\forall S \subseteq T, \forall i \notin T : \quad f_S(i) \geq f_T(i)
\]

- Submodular function captures the law of diminishing marginal productivity (diminishing returns) in many natural applications
Examples of Submodular Functions

- **Coverage**: given sets $S_1, \ldots, S_n \subseteq \Omega$

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

- **Cut**: graph $G([n], E)$, $\forall S \subseteq [n] : \quad f(S) = \left| E(S, V \setminus S) \right|$

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

- **Entropy**: $f(S) = H(X_i : i \in S)$ for random variables $X_1, \ldots, X_n$

- **Matroid rank**: $f(S) = \text{rank}(A_{[m] \times S})$ for $m \times n$ matrix $A$

- **Facility location, social welfare, influence in a social network, ...**
Submodular Function

Submodular function:
A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular if
\[
\forall S, T \subseteq [n] : f(S \cup T) \leq f(S) + f(T) - f(S \cap T)
\]

Proposition: For set function $f : 2^{[n]} \to \mathbb{R}$, define:
\[
\forall S \subseteq [n], \forall i \in [n] : f_S(i) \triangleq f(S \cup \{i\}) - f(S)
\]
A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular iff:
\[
\forall S \subseteq T, \forall i \notin T : f_S(i) \geq f_T(i)
\]

- Submodular function captures the law of diminishing marginal productivity (diminishing returns) in many natural applications
Submodularity of Coverage

**Proposition:** For set function \( f : 2^{[n]} \rightarrow \mathbb{R} \), define:
\[
\forall S \subseteq [n], \forall i \in [n] : \quad f_S(i) \triangleq f(S \cup \{i\}) - f(S)
\]
A set function \( f : 2^{[n]} \rightarrow \mathbb{R} \) is **submodular** iff:
\[
\forall S \subseteq T, \forall i \notin T : \quad f_S(i) \geq f_T(i)
\]

A set function \( f : 2^{[n]} \rightarrow \mathbb{R} \) is **monotone** if
\[
\forall S \subseteq T : \quad f(S) \leq f(T)
\]

**Instance:** A sequence of subsets \( S_1, \ldots, S_n \subseteq U \).
Find \( C \subseteq \{1, \ldots, n\} \) with \( |C| \leq k \) to maximize \( \bigcup_{i \in C} S_i \).
\[
\forall C \subseteq \{1, \ldots, n\} : \quad f(C) = \left| \bigcup_{i \in C} S_i \right|
\]
**Submodular Maximization**

**Instance:** A *monotone submodular* set function $f : 2^{[n]} \to \mathbb{R}$.

Maximize $f(S)$ subject to $|S| \leq k$. *(cardinality constraint)*

**Greedy Submodular Maximization:**
- Initially $S = \emptyset$;
- While $|S| < k$ do:
  - Add $i \notin S$ with largest $f_S(i)$ into $S$;

**Proposition:** For set function $f : 2^{[n]} \to \mathbb{R}$, define:

$$\forall S \subseteq [n], \forall i \in [n] : \quad f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular iff:

$$\forall S \subseteq T, \forall i \notin T : \quad f_S(i) \geq f_T(i)$$
Submodular Maximization

**Instance:** A monotone submodular set function $f : 2^{[n]} \to \mathbb{R}$.

Maximize $f(S)$ subject to $|S| \leq k$. (cardinality constraint)

**Greedy Submodular Maximization:**

initially $S = \emptyset$;

while $|S| < k$ do:

add $i \not\in S$ with largest $f_S(i)$ into $S$;

**Theorem (Nemhauser, Wolsey, Fisher 1978):**

For monotone submodular set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$, the greedy algorithm gives a $(1 - 1/e)$-approximation of

$$OPT = \max \ \{ f(S) \mid |S| \leq k \}$$
Greedy Submodular Maximization:

initially $S = \emptyset$;

while $|S| < k$ do:

add $i \notin S$ with largest $f_S(i)$ into $S$;

$\begin{aligned}
f : 2^n &\to \mathbb{R} \\
f_S(i) &\equiv f(S \cup \{i\}) - f(S)
\end{aligned}$

Submodular:

$\forall S \subseteq T, \forall i \notin T : f_S(i) \geq f_T(i)$

- $S$: current $S$ in an iteration
- $i$: the $i$ added into $S$ in that iteration

$\begin{aligned}
f_S(i) &\geq \frac{1}{k} \left( \text{OPT} - f(S) \right)
\end{aligned}$

- Let $S^*$ be the optimal solution that achieves $\text{OPT} = f(S^*)$.

$\begin{aligned}
\text{OPT} - f(S) &\leq f_S(S^*) \equiv f(S^* \cup S) - f(S) \\
&\leq \sum_{j \in S^*} f_S(j) \leq k \cdot f_S(i)
\end{aligned}$

(monotone) \hspace{1cm} (submodular) \hspace{1cm} (greedy)
Greedy Submodular Maximization:

\[ f: 2^{[n]} \rightarrow \mathbb{R} \]
\[ f_S(i) \triangleq f(S \cup \{i\}) - f(S) \]

Submodular:
\[ \forall S \subseteq T, \forall i \notin T : f_S(i) \geq f_T(i) \]

- \( S \): current \( S \) in an iteration
- \( i \): the \( i \) added into \( S \) in that iteration

\[ f_S(i) \geq \frac{1}{k} \left( \text{OPT} - f(S) \right) \]

- \( S^{(\ell)} \): the \( S \) constructed after \( \ell \) iterations

\[ f(S^{(\ell)}) - f(S^{(\ell-1)}) \geq \frac{1}{k} \left( \text{OPT} - f(S^{(\ell-1)}) \right) \]

\[ \Rightarrow \text{OPT} - f(S^{(\ell)}) \leq \left( 1 - \frac{1}{k} \right) \left( \text{OPT} - f(S^{(\ell-1)}) \right) \]
Greedy Submodular Maximization:

- Initially $S = \emptyset$;
- while $|S| < k$ do:
  - add $i \not\in S$ with largest $f_S(i)$ into $S$;

$S^{(\ell)}$ : the $S$ constructed after $\ell$ iterations

Submodular:

- $\forall S \subseteq T, \forall i \not\in T : f_S(i) \geq f_T(i)$

$$f : 2^{[n]} \rightarrow \mathbb{R}$$

$$f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

$$OPT - f(S^{(\ell)}) \leq \left(1 - \frac{1}{k}\right) \left(1 - \frac{1}{\text{e}}\right)OPT$$

$$\implies OPT - f(S^{(k)}) \leq \left(1 - \frac{1}{k}\right)^k \left(1 - \frac{1}{\text{e}}\right)OPT \leq \frac{1}{\text{e}}OPT$$

$$\implies SOL = f(S^{(k)}) \geq \left(1 - \frac{1}{\text{e}}\right)OPT$$
Greedy Submodular Maximization:

- Submodularity + monotonicity:

\[
S^{(\ell)} \leftarrow S^{(\ell-1)} \cup \{i_\ell\} \text{ with } i_\ell \text{ maximizing } f(S^{(\ell-1)} \cup \{i_\ell\}) - f(S^{(\ell-1)})
\]

\[
f(S^{(\ell-1)} \cup \{i_\ell\}) - f(S^{(\ell-1)}) \geq \frac{1}{k} \left( OPT - f(S^{(\ell-1)}) \right)
\]

\[
OPT - f(S^{(\ell)}) \leq \left(1 - \frac{1}{k}\right) \left( OPT - f(S^{(\ell-1)}) \right)
\]

\[
\implies OPT - f(S^{(k)}) \leq \left(1 - \frac{1}{k}\right)^k \text{ OPT} \leq \frac{1}{e} \text{ OPT}
\]
Submodular Maximization

**Instance:** A monotone submodular set function $f : 2^{[n]} \rightarrow \mathbb{R}$. Maximize $f(S)$ subject to $|S| \leq k$. (cardinality constraint)

**Greedy Submodular Maximization:**
initially $S = \emptyset$;
while $|S| < k$ do:
    add $i \notin S$ with largest $f_S(i)$ into $S$;

**Theorem (Nemhauser, Wolsey, Fisher 1978):**
For monotone submodular set function $f : 2^{[n]} \rightarrow \mathbb{R}_{\geq 0}$, the greedy algorithm gives a $(1 - 1/e)$-approximation of

$$OPT = \max \left\{ f(S) \mid |S| \leq k \right\}$$
# Submodular Maximization

## MONOTONE MAXIMIZATION

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Approximation</th>
<th>Hardness</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S</td>
<td>\leq k$</td>
<td>$1 - 1/e$</td>
</tr>
<tr>
<td>matroid</td>
<td>$1 - 1/e$</td>
<td>$1 - 1/e$</td>
<td>multilinear ext.</td>
</tr>
<tr>
<td>$O(1)$ knapsacks</td>
<td>$1 - 1/e$</td>
<td>$1 - 1/e$</td>
<td>multilinear ext.</td>
</tr>
<tr>
<td>$k$ matroids</td>
<td>$k + \epsilon$</td>
<td>$k / \log k$</td>
<td>local search</td>
</tr>
<tr>
<td>$k$ matroids &amp; $O(1)$ knapsacks</td>
<td>$O(k)$</td>
<td>$k / \log k$</td>
<td>multilinear ext.</td>
</tr>
</tbody>
</table>

## NON-MONOTONE MAXIMIZATION

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Approximation</th>
<th>Hardness</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconstrained</td>
<td>$1/2$</td>
<td>$1/2$</td>
<td>combinatorial</td>
</tr>
<tr>
<td>matroid</td>
<td>$1/e$</td>
<td>0.48</td>
<td>multilinear ext.</td>
</tr>
<tr>
<td>$O(1)$ knapsacks</td>
<td>$1/e$</td>
<td>0.49</td>
<td>multilinear ext.</td>
</tr>
<tr>
<td>$k$ matroids</td>
<td>$k + O(1)$</td>
<td>$k / \log k$</td>
<td>local search</td>
</tr>
<tr>
<td>$k$ matroids &amp; $O(1)$ knapsacks</td>
<td>$O(k)$</td>
<td>$k / \log k$</td>
<td>multilinear ext.</td>
</tr>
</tbody>
</table>

From Prof. Jan Vondrák’s slides “Optimization of Submodular Functions”
## Submodular Minimization

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Approximation</th>
<th>Hardness</th>
<th>alg. technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconstrained</td>
<td>1</td>
<td>1</td>
<td>combinatorial</td>
</tr>
<tr>
<td>Parity families</td>
<td>1</td>
<td>1</td>
<td>combinatorial</td>
</tr>
<tr>
<td>Vertex cover</td>
<td>2</td>
<td>2</td>
<td>Lovász ext.</td>
</tr>
<tr>
<td>$k$-unif. hitting set</td>
<td>$k$</td>
<td>$k$</td>
<td>Lovász ext.</td>
</tr>
<tr>
<td>Multiway $k$-partition</td>
<td>$2 - 2/k$</td>
<td>$2 - 2/k$</td>
<td>Lovász ext.</td>
</tr>
<tr>
<td>Facility location</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>combinatorial</td>
</tr>
<tr>
<td>Set cover</td>
<td>$n$</td>
<td>$n/\log^2 n$</td>
<td>trivial</td>
</tr>
<tr>
<td>$</td>
<td>S</td>
<td>\geq k$</td>
<td>$\tilde{O}(\sqrt{n})$</td>
</tr>
<tr>
<td>Shortest path</td>
<td>$O(n^{2/3})$</td>
<td>$\Omega(n^{2/3})$</td>
<td>combinatorial</td>
</tr>
<tr>
<td>Spanning tree</td>
<td>$O(n)$</td>
<td>$\Omega(n)$</td>
<td>combinatorial</td>
</tr>
</tbody>
</table>

From Prof. Jan Vondrák’s slides “Optimization of Submodular Functions”