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

$\nu_i$ 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 \( E(S, T) \);

\( G(V, E) \)

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

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

\( \forall v_i, \quad \geq \frac{1}{2} \text{ of } |E(S_i, v_i)| + |E(T_i, v_i)| \)

\( \text{contributes to } SOL_G \)

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

\( E(S, T) = \{uv \in E \mid u \in S, v \in T\} \)
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:**
in an arbitrary cut; repeat until nothing changed:

- if $\exists v$ switching side increases cut
  - $v$ switches to the other side;

in a local optima:

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

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

$OPT \leq |E| = |E(S, S)| + |E(T, T)| + |E(S, T)| \leq 2|E(S, T)|$

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

$m$ machines

$n$ jobs

processing time $p_j$
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:  
$$C_{\text{max}} = \max_{1 \leq i \leq} C_i$$
**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”: \( P \mid \mid C_{\text{max}} \)

Graham’s “\( \alpha \mid \beta \mid \gamma \)” 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; \( \text{pmtm} \): 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 $m$ identical machines that minimizes the makespan $C_{\text{max}}$

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

$$\forall \text{ instance } I, \quad \frac{SOL_I}{OPT_I} \leq \alpha$$

- $SOL_I$: solution returned by the algorithm on instance $I$
- $OPT_I$: optimal solution of instance $I$
Graham’s List Algorithm

$m$ machines

$n$ jobs

List algorithm (Graham 1966):

For $j = 1, 2, \ldots, n$:

assign job $j$ to the current least heavily loaded machine;

\[
OPT \geq \max_{1 \leq j \leq n} p_j
\]

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

- 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

\[
\Rightarrow \quad C_{i^*} - p_\ell \leq \frac{1}{m} \sum_{j \neq \ell} p_j \left\{ \begin{array}{l}
p_\ell \leq \max_{1 \leq j \leq n} p_j
\end{array} \right\}
\]
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:
  $$ C_{\text{max}} \leq \left( 2 - \frac{1}{m} \right) \text{OPT} $$
  - This is tight in the worst case.
Local Search

locaely 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: $OPT \geq \max_{1 \leq j \leq n} p_j$  
  $OPT \geq \frac{1}{m} \sum_{1 \leq j \leq n} p_j$

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

- local optima $\implies C_{i^*} - p_\ell$ is the least heavy load

$$C_{i^*} - p_\ell \leq \frac{1}{m} \sum_{j \neq \ell} p_j \quad \{ \quad p_\ell \leq \max_{1 \leq j \leq n} p_j \}$$
**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: \[ C_{max} \leq \left( 2 - \frac{1}{m} \right) OPT \]

**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

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

\( m \) machines

\[ \text{List algorithm (Graham 1966):} \]

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

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

\( n \) jobs
**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:**
  \[ \text{OPT} \geq \frac{1}{m} \sum_{1 \leq j \leq n} p_j \]

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

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

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

**WLOG:** \( C_{i^*} > \max_{1 \leq j \leq n} p_j \implies p_\ell \leq p_{m+1} \)

\[ \text{OPT} \geq p_m + p_{m+1} \geq 2p_{m+1} \]

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

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

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. $\bigcup_{i \in C} S_i = U$. 

![Diagram](image)
Hitting Set

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

![Diagram showing the hitting set concept with subsets $S_1, S_2, S_3, S_4, S_5$ and elements $x_1, x_4$ hitting the subsets, with $U$ as the universal set.]
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

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

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

- **Vertex Cover** graph
  - $v_1$, $v_2$, $v_3$, $v_4$
  - Edges: $e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_6$

- **Incidence graph**
  - Edges $e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_6$

- **Set cover instance** with frequency $=2$
  - $v_1$, $v_2$, $v_3$, $v_4$
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

$VC$ is **NP-hard** $\implies$ $SC$ 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 $S_1, \ldots, S_m \subseteq U$.

- **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

  $price(x_1) = \frac{1}{\max_i |S_i|}$  \implies  $price(x_1) \leq \frac{OPT}{|U|}$

Greedy Cover:

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

- $|C| = \sum_{x \in U} price(x)$
Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.

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

\[ \forall 1 \leq k \leq \ell : \quad \text{price}(x_k) = \text{price}(x_1) = \frac{1}{\max_i |S_i|} \]

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

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

\[ |C| = \sum_{x \in U} \text{price}(x) \]
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 \( S_1, \ldots, S_m \subseteq U \).

\[
\text{price} = \frac{1}{3}
\]

- \( x_1 \): 1st element covered by the \textbf{GreedyCover} algorithm
- \( x_2 \): 2nd element covered by the \textbf{GreedyCover} algorithm
- \( x_3 \): 3rd element covered by the \textbf{GreedyCover} algorithm
- \( x_4 \): 4th element covered by the \textbf{GreedyCover} algorithm
- \( x_5 \): 5th element covered by the \textbf{GreedyCover} algorithm

**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 \textbf{GreedyCover} algorithm

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

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

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 $\text{NP} \subseteq \text{quasi-poly-time}$.

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

- [Dinur, Steuer 2014] There is no poly-time $(1 - o(1))\ln n$-approximation algorithm unless $\text{NP} = \text{P}$.
**Instance**: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.

**Primal**: $C \subseteq \{1, 2, \ldots, m\}$ that $\bigcup_{i \in C} S_i = U$.

$$OPT_{\text{primal}} = \min |C|$$

**Dual**: $M \subseteq U$ that $\forall i, |S_i \cap M| \leq 1$.

$$\forall C, \forall M : |M| \leq |C|$$

every $x \in M$ must consume a set to cover

$$\forall M : |M| \leq OPT_{\text{primal}}$$
**Instance:** A sequence of subsets $S_1, \ldots, S_m \subseteq U$.

**Primal:** $C \subseteq \{1, 2, \ldots, m\}$ that $\bigcup_{i \in C} S_i = U$.

**OPT\text{\textsubscript{primal}}** $= \min |C|$  

- **Find a maximal $M$;**  
- **return** $C = \{i \mid S_i \cap M \neq \emptyset\}$;

**S3** $M$ is maximal $\Rightarrow$ $C$ is a cover

**S4** $|C| \leq f \cdot |M| \leq f \cdot OPT\text{\textsubscript{primal}}$

**Dual:** $M \subseteq U$ that $\forall i, |S_i \cap M| \leq 1$.

$\forall M : |M| \leq OPT\text{\textsubscript{primal}}$
2-Approximation of Vertex Cover

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

Find a **maximal** $M \subseteq U$ s.t. $\forall i : |S_i \cap M| \leq 1$; return $C = \{i \mid S_i \cap M \neq \emptyset\}$.

**Frequency assumption:**
$\forall x \in U : \left| \{i \mid x \in S_i\} \right| \leq f$

$|C| \leq f \cdot OPT$

2-approx. for vertex cover
Vertex Cover

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

- 2-approximation algorithm:
  - Find a *maximal matching*; return the *matched* vertices;

- [Dinur, Safra 2005] There is no poly-time $<1.36$-approximation algorithm unless $\text{NP} = \text{P}$.

- [Khot, Regev 2008] Assuming the unique games conjecture, there is no poly-time $(2 - \epsilon)$-approximation algorithm.
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 \( |C| \leq k \) 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

\[
\Sigma_{\ell} = \sum_{j=1}^{\ell} \Delta_j
\]

\[
\Delta_{\ell} \geq \frac{1}{k} (OPT - \Sigma_{\ell-1})
\]
Greedy Cover:
initially $C = \emptyset$;
while $|C| \leq k$ do:
    add $i$ with largest $|S_i \cap U|$ to $C$;
    $U = U \setminus S_i$;

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

- # 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 in the $\ell$th iteration.
Greedy Cover:
initially $C = \emptyset$;
while $|C| \leq k$ do:
    add $i$ with largest $|S_i \cap U|$ to $C$;
    $U = U \setminus S_i$;

$\Delta_\ell \geq \frac{1}{k}(OPT - \Sigma_{\ell-1})$ \quad \Rightarrow \quad \Sigma_\ell \geq \left(1 - \left(1 - \frac{1}{k}\right)^\ell\right)OPT$

• Basis: $\Sigma_1 = \Delta_1 \geq \frac{1}{k}OPT$

• Induction step:
  $\Sigma_\ell = \Sigma_{\ell-1} + \Delta_\ell \geq \Sigma_{\ell-1} + \frac{1}{k}(OPT - \Sigma_{\ell-1})$

  $= (1 - \frac{1}{k})\Sigma_{\ell-1} + \frac{1}{k}OPT \geq \left[(1 - \frac{1}{k})\left(1 - \left(1 - \frac{1}{k}\right)^{\ell-1}\right) + \frac{1}{k}\right]OPT$

  (I.H.)

  $= \left(1 - \left(1 - \frac{1}{k}\right)^\ell\right)OPT$
Greedy Cover:
initially \( C = \emptyset \);
while \( |C| \leq k \) do:
    add \( i \) with largest \( |S_i \cap U| \) to \( C \);
    \( U = U \setminus S_i \);

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

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

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

- \( (1 - 1/e) \)-approximation:
  \[ SOL = \Sigma_k \geq \left( 1 - \left( 1 - \frac{1}{k} \right)^k \right) OPT \geq \left( 1 - \frac{1}{e} \right) OPT \]

- [Feige 1998] There is no poly-time \( (1 - 1/e + \epsilon) \)-approximation algorithm unless \( \text{NP} = \text{P} \)
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
Examples of Submodular Functions

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

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

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

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

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

- **Matroid rank**: $f(S) = \text{rank}(A_{m \times n})$ 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 $f : 2^{[n]} \to \mathbb{R}$, 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:

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

A set function $f : 2^{[n]} \to \mathbb{R}$ 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 \( f : 2^{[n]} \rightarrow \mathbb{R} \).

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

**Greedy Submodular Maximization:**
initially \( S = \emptyset \);
while \( |S| \leq k \) do:
    add \( i \not\in S \) with largest \( f_S(i) \) into \( S \);

**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:
\[
\forall S \subseteq T, \forall i \not\in T : \quad f_S(i) \geq f_T(i)
\]
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 $|S| \leq k$ do:
  add $i \not\in S$ with largest $f_S(i)$ into $S$;

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

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

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

initially $S = \emptyset$;

while $|S| \leq k$ do:

- add $i \notin S$ with largest $f_{S}(i)$ into $S$;

$S \ni i$:

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

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

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

Submodular:

$\forall S \subseteq T, \forall i \notin T : f_{S}(i) \geq f_{T}(i)$

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

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

\[ f_{S}(S^{*}) \triangleq f(S^{*} \cup S) - f(S) \leq \sum_{j \in S^{*}} f_{S}(j) \leq k \cdot f_{S}(i) \]

(monotone + submodular $\implies$ subadditivity of $f_{S}(\cdot)$)
\[ f : 2^{[n]} \to \mathbb{R} \]

\[ \forall S, T \subseteq [n] : \quad f_S(T) \triangleq f(S \cup T) - f(S) \]

A function \( g : 2^{[n]} \to \mathbb{R} \) is subadditive if

\[ \forall A, B \subseteq [n] : \quad g(A \cup B) \leq g(A) + g(B) \]

\( f \) is monotone and submodular \( \implies f_S(\cdot) \) is subadditive

\[ f_S(A \cup B) = f(A \cup B \cup S) - f(S) \]

(submodularity) \[ \leq f(A \cup S) + f(B \cup S) - f((A \cup S) \cap (B \cup S)) - f(S) \]

(monotonicity) \[ \leq f(A \cup S) + f(B \cup S) - 2f(S) \]

\[ f_S(A) = f(A \cup S) - f(S) \]

\[ f_S(B) = f(B \cup S) - f(S) \]
Greedy Submodular Maximization:

\[
\begin{align*}
\text{initially } S &= \emptyset; \\
\text{while } |S| \leq k \text{ do:} & \quad \text{add } i \notin S \text{ with largest } f_S(i) \text{ into } S; \\
\end{align*}
\]

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

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

- Let \( S^* \) be the optimal solution that achieves \( OPT = f(S^*) \).
  \[
f_S(S^*) \triangleq f(S^* \cup S) - f(S) \leq \sum_{j \in S^*} f_S(j) \leq k \cdot f_S(i)
\]

(monotone + submodular \( \implies \) subadditivity of \( f_S(\cdot) \))
Greedy Submodular Maximization:

Initially $S = \emptyset$;

while $|S| \leq k$ do:
   add $i \notin S$ with largest $f_S(i)$ into $S$;

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

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

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

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

By the same induction.
Greedy Submodular Maximization:

- **Basis**: \( f(S^{(1)}) = f_\emptyset(i_1) \geq \frac{1}{k}OPT \)

- \( f(S^{(\ell)}) = f(S^{(\ell-1)}) + f_{S^{(\ell-1)}}(i_\ell) \geq f(S^{(\ell-1)}) + \frac{1}{k}(OPT - f(S^{(\ell-1)})) \)

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

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

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

\(i_\ell : \) the \( i \) selected in the \( \ell \)th iteration
**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 $|S| \leq k$ do:

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

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

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

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

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

- Submodularity + monotonicity:

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

\[
1 - \frac{1}{e}
\]

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