Advanced Algorithms

南京大学

尹一通
Set Cover

**Instance:** A number of sets $S_1, S_2, ..., S_m \subseteq U$.
Find the smallest $C \subseteq \{1, 2, ..., m\}$ that $\bigcup_{i \in C} S_i = U$. 

![Diagram showing set cover example](image-url)
**Hitting Set**

**Instance:** A number of sets $S_1, S_2, ..., S_n \subseteq U$. Find the smallest $H \subseteq U$ that $\forall i, S_i \cap H \neq \emptyset$. 

```
S_1  S_2  S_3  S_4  S_5
```

```
\begin{array}{c}
x_1 \\
x_4 \\
U \\
\end{array}
```
Set Cover

**Instance:** A number of sets $S_1, S_2, ..., S_m \subseteq U$.

Find the smallest $C \subseteq \{1, 2, ..., m\}$ that $\bigcup_{i \in C} S_i = U$.

- **NP-hard**
- one of Karp’s 21 **NP**-complete problems
- **frequency:** # of sets an element is in

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

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

Find the smallest $C \subseteq V$ that every edge has at least one endpoint in $C$.

![Incidence graph of a vertex cover instance](image)
**Vertex Cover**

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

Find the smallest $C \subseteq V$ that every edge has at least one endpoint in $C$.

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

VC is NP-hard $\Rightarrow$ SC is NP-hard
**Set Cover**

**Instance:** A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.
Find the smallest $C \subseteq \{1, 2, \ldots, m\}$ that $\bigcup_{i \in C} S_i = U$.

**GreedyCover**

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 number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

### GreedyCover

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

**OPT**(I): value of minimum set cover of instance $I$

**SOL**(I): value of the set cover returned by the **GreedyCover** algorithm on instance $I$

**GreedyCover** has *approximation ratio* $\alpha$ if

$$\forall \text{ instance } I, \quad \frac{\text{SOL}(I)}{\text{OPT}(I)} \leq \alpha$$
**Instance:** A number of sets $S_1, S_2, ..., S_m \subseteq U$.

**GreedyCover**

Initially $C = \emptyset$;

while $U \neq \emptyset$ do:

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

$U = U \setminus S_i$;

$\forall x \in S_i$, price$(x) = 1/|S_i \cap U|$

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

enumerate $x_1, x_2, ..., x_n$ in the order in which they are covered

elements can be *matched* to the sets in OPT cover

$$\exists S_i, |S_i| \geq \frac{|U|}{OPT} \quad \Rightarrow \quad \text{price}(x_1) \leq \frac{OPT}{|U|}$$
**Instance:** A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

| price=1/3 | $x_1$ | $S_1$ |
| price=1/3 | $x_2$ | $S_2$ |
| price=1   | $x_3$ | $S_3$ |
| price=1/3 | $x_4$ | $S_4$ |
| price=1   | $x_5$ |         |

**GreedyCover**

Initially $C=\emptyset$;

while $U \neq \emptyset$ do:

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

$U = U \setminus S_i$;  $\forall x \in S_i$, price($x$) = $1/|S_i \cap U|$

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

enumerate $x_1, x_2, \ldots x_n$ in the order in which they are covered

consider $U_t$ in iteration $t$ where $x_k$ is covered:

$$|U_t| \geq n-k+1$$

all $S_i \cap U_t$ form a set cover instance: $\leq \text{OPT}$

$p_{\text{price}(x)}(k) \leq \frac{\text{OPT}}{n-|U_t|+1}$
**Instance:** A number of sets $S_1, S_2, ..., S_m \subseteq U$.

**GreedyCover**

Initially $C=\emptyset$;

while $U \neq \emptyset$ do:

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

$U = U \setminus S_i$; \quad \forall x \in S_i$, price$(x) = 1/|S_i \cap U|$

$|C| = \sum_{x \in U} \text{price}(x) \leq \sum_{k=1}^{n} \frac{\text{OPT}}{n-k+1} = H_n \cdot \text{OPT}$

enumerate $x_1, x_2, ..., x_n$ in the order in which they are covered

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

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 approximation ratio $H_n \approx \ln n + O(1)$.

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

• [Ras, Safra 1997] For some $c$ 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 $\textbf{NP} = \textbf{P}$.
**Instance:** A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

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

$$\text{OPT}_{\text{primal}} = \min |C|$$

**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

$$\forall M: |M| \leq \text{OPT}_{\text{primal}}$$
**Instance:** A number of sets $S_1, S_2, ..., S_m \subseteq U$.

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

$$\text{OPT}_{\text{primal}} = \min |C|$$

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

$M$ is *maximal* $\Rightarrow$ $C$ must be a cover

$$|C| \leq f \cdot |M| \leq f \cdot \text{OPT}_{\text{primal}}$$

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

$$\forall M: |M| \leq \text{OPT}_{\text{primal}}$$
**Instance:** A number of sets $S_1, S_2, ..., S_m \subseteq U$.

Find a maximal $M \subseteq U$ that $\forall i$, $|S_i \cap M| \leq 1$; return $C = \{i : S_i \cap M \neq \emptyset\}$;

**Frequency assumption:**
\[ \forall x \in U, \ |\{i : x \in S_i\}| \leq f \]

For vertex cover: This gives a 2-approximation algorithm.
Vertex Cover

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

Find the smallest $C \subseteq V$ that every edge has at least one endpoint in $C$.

a 2-approximation algorithm:

Find a *maximal matching*; return the *matched* vertices;

- There is no poly-time $<1.36$-approximation algorithm unless $\text{NP} = \text{P}$.
- Assuming the unique game conjecture, there is no poly-time $(2-\varepsilon)$-approximation algorithm.
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:

\[ C_{\text{max}} = \max_i C_i \]
Instance: \( n \) jobs \( j=1, 2, \ldots, n \)
each with processing time \( p_j \in \mathbb{Z}^+ \).

Solution: A schedule of \( n \) jobs to \( m \) machines
that minimizes the makespan \( C_{\text{max}} \).

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

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

- \( r_j \): each job has a release time \( r_j \);
- \( d_j \): each job has a deadline \( d_j \);
- pmtn: preemption is allowed;

\( \gamma \): objective

- \( C_{\text{max}} \): makespan; \( \Sigma_j C_j \): total completion time; \( L_{\text{max}} \): maximum lateness;
**Instance:** $n$ jobs $j=1, 2, ..., n$ each with processing time $p_j \in \mathbb{Z}^+$. 

**Solution:** A schedule of $n$ jobs to $m$ machines that minimizes the *makespan* $C_{\text{max}}$.

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

when $m=2$, the problem can solve the **partition** problem:

**Input:** $n$ numbers $x_1, x_2, ..., x_n \in \mathbb{Z}^+$.

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

the **partition** problem is among Karp’s 21 **NPC** problems
Graham’s List Algorithm

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

\[
OPT \geq \max_j p_j
\]

\[
OPT \geq \frac{1}{m} \sum_j p_j
\]
**List Algorithm** (Graham 1966)

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

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

---

$n$ jobs: $p_1, p_2, \ldots, p_n$; $m$ machines

\[ OPT \geq \max_j p_j \quad OPT \geq \frac{1}{m} \sum_j p_j \]

for the schedule returned by the list algorithm:

makespan $C_{\text{max}} = C_i \leq 2 \cdot OPT$

the last job assigned to machine $i$ is job $l$

before job $l$ was assigned, machine $i$ is the least heavily loaded

\[ C_i - p_l \leq \frac{1}{m} \sum_j p_j \leq OPT \]

\[ p_l \leq \max_j p_j \leq OPT \]
**List Algorithm** (Graham 1966)

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

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

returns a schedule with makespan $C_{\text{max}} \leq (2 \cdot \Theta \frac{p}{m}) \cdot OPT$

Tight in the worst-case!
Local Search

start with a solution:

\[\text{locally modify the solution to make improvement until no improvement can be made (local optimum)}\]

Start with an arbitrary schedule;
repeat until no job is reassigned (a local optimum is encountered):
let \(l\) be a job that finished last;
if \(\exists\) machine \(i\) s.t. \(l\) will finish earlier after reassigned to machine \(i\)
transfer job \(l\) to machine \(i\);
Start with an arbitrary schedule; repeat until no job is reassigned (a local optimum is encountered):

let $l$ be a job that fished last;

if $\exists$ machine $i$ s.t. job $l$ will finish earlier after reassigned to machine $i$

transfer job $l$ to machine $i$;

$$OPT \geq \max_j p_j$$

$$OPT \geq \frac{1}{m} \sum_j p_j$$

in a local optimum: suppose makespan $C_{\text{max}} = C_i$

for the job $l$ that finished last

local optimum $\Rightarrow$ $C_i - p_l$ must be the least heavily loaded

$$C_i - p_l \leq \frac{1}{m} \sum_{j \neq l} p_j$$

$$C_i \leq \frac{1}{m} \sum_j p_j + \left(1 - \frac{1}{m}\right) p_l \leq \left(2 - \frac{1}{m}\right) \cdot OPT$$
Start with an arbitrary schedule;
repeat until no job is reassigned (a local optimum is encountered):
  let \( l \) be a job that fished last;
  if \( \exists \) machine \( i \) s.t. job \( l \) will finish earlier after reassigned to machine \( i \)
    transfer job \( l \) to machine \( i \);

finds a schedule with makespan \( C_{\text{max}} \leq (2 - \frac{1}{m}) \cdot OPT \)

\textbf{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 optimum

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

\[ \text{OPT} \geq \frac{1}{m} \sum_j p_j \]

for the schedule returned by the LPT algorithm:

makespan \( C_{\text{max}} = C_i \leq \frac{3}{2} \cdot \text{OPT} \)

the last job assigned to machine \( i \) is job \( l \)

WLOG:

\[ C_i > \max_j p_j \]

\[ p_\ell \leq p_{m+1} \]

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

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

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

- With a more careful analysis, the LPT is a 4/3-approximation algorithm.

- The problem of minimum makespan on identical machines has a \textbf{PTAS} (\textbf{P}olynomial \textbf{T}ime \textbf{A}pproximation \textbf{S}cheme).

  $$\forall \varepsilon > 0, \exists \text{ poly-time } (1-\varepsilon)\text{-algorithm for the problem}$$
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)

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

**List Algorithm** (Graham 1966)

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

the **competitive ratio** of the online algorithm is $\alpha$ if:

$\forall$ input sequence $I$:

- solution returned by the online algorithm on $I$
  - $\leq \alpha$

- solution returned by the optimal offline algorithm on $I$

the list algorithm is 2-competitive