Discrete Optimization

Instructor: Yuan Zhou

Notes Taker: Zejin Lin

TSINGHUA UNIVERSITY.

linzj23@mails.tsinghua.edu.cn

lzjmaths.github.io

April 22, 2025

Contents

1 Introduction

Discrete (combinatorial) optimization is a subfield of mathematical optimization that consists of finding an optimal object from a finite set of objects, where the set of feasible solution is discrete or can be reduced to a discrete set.

However, usually this feasible solution set is very large (due to combinatorial explosion) and it is computationally infeasible to go through all feasible solutions and find the one with opimal objective function value.

Example 1.1 (Task Assignment)**.** There are n tasks and n workers. Each task has an importance score a_i and each worker has a skill level b_i. We need to assign each task to a worker such that the sum of $\sum_{i=1}^{n} a_i b_{\sigma(i)}$ is maximized.

1.1 Models of Computation: Turing Machines

Definition 1.2 (A Deterministic Turing Machine(DTM))**.** It consists of an infinitely-long tape (memory) and a deterministic finite automata that controls the head to move along the tape and read/write symbols from/to the tape cells.

Definition 1.3 (Complexity measure)**.** Running time is the number of steps of Turing machine.

Memory is the number of tape cells used.

Definition 1.4 (Caveat)**.** No random access of memory

- Single-tape DTM requires $\geqslant n^2$ steps to detect n bit palindromes.
- EAsy to detect palindromes within c_n steps on a real computer.

1.2 Models of Computation: word RAM

Definition 1.5. Each memory location and input/output cell stores a w-bit integer (assume $w \geqslant \log_2 \omega$).

Primitive Operations:

1.3 Polynomial Running Time

Definition 1.6. We say that an algorithm is **efficient** if its running time is polynomial of input size n.

Example 1.7 (Task machine)**.** Polynomial-time algorithm: selection sort/inserting sort/quick sort/merge sort.

Non-polynomial-time algorithm: try all possible matching and output the one with the highest score.

- Definition is relatively insensitive to model of computation.
- The poly-times algorithm that people develop have both small constants and small exponents
- Breaking through the exponential barrier is a major challenge.

1.4 Notation

Definition 1.8. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geqslant 1$ such that $0 \leqslant f(n) \leqslant c \cdot g(n)$ for all $n \geqslant n_0$.

$f(n)$ is $\Omega(g(n))$ is $g(n) \in O(f(n))$.

$f(n)$ is $\Theta(g(n))$ is both $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$.

1.5 Tentative Syllabus

We will introduce three exact discrete optimization algorithms(6 weeks):

- Greedy algorithms
- Dynamic programming
- Network flows

And some approximation algorithms for intractable discrete optimization problems(9 weeks)

- Definition of approximation algorithms
  - Algorithm techniques: greedy, linear programming relaxation, semidefinite programming relaxation.
- Hardness of approximation
  - Techniques: hardness reductions, Fourier analysis of Boolean functions.
- Problems studied: Set-Cover, facility location, K-center, Multi-Cut, Max-Cut, $\cdots$

2 Greedy Algorithms

2.1 Interval Scheduling

Example 2.1 (Interval Scheduling)**.** Input: n jobs, $\{(s_i, f_i)\}_{i=1}^n$. Goal: How to choose jobs with maximized number such that each pair of intervals do not intersect.

Greedy Framework Consider jobs in order $\pi(1), \pi(2), \cdots, \pi(n)$. For each $\pi(i)$, $i = 1, 2, \cdots, n$, if $\pi(i)$ compatible with all selected jobs, then select $\pi(i)$.

The choice of π: Earliest-start-time-first, Earliest-finish-time-first, Longest-job-first, Shortest-job-first, etc.

Theorem 2.2. *Earliest-finish-time-first greedy returns an optimal solution.*

Proof. Suppose algorithm selects $i_1, i_2, \cdots, i_k$, opt selects $k' > k$ jobs.

Choose an optimal solution agrees with algorithm in first r jobs so that r maxmized, $j_1, j_2, \cdots, j_{k'}$.

Obviously, $r < k$. Then $f_{i_{r+1}} < f_{j_{r+1}}$. Therefore, we can replace i_{r+1} with j_{r+1} to get another optimal solution, which contradicts to the fact that r maxmized. $\square$

2.2 Interval Partitioning

Example 2.3 (Interval Partitioning)**.** Input: n lectures, $\{(s_i, f_i)\}_{i=1}^n$.

Goal: Position lectures into minimum number of classrooms so that in each classroom lectures are compatible.

Greedy Framework Lectures in order $\pi(1), \cdots, \pi(n)$, the number of opening classrooms is zero in the beginning. For each $\pi(i)$,

If $\exists$ opening classroom j *s.t.* lecture $\pi(i)$ compatible with lectures in j, then $\pi(i) \rightarrow$ classroom j.

Else, open a new classroom for $\pi(i)$.

Proof. Introduce a concept **Depth**: $d(t) =$ Number of lectures active at time t, and $d = \max_t\{d(t)\}$.

Claim 2.2.1. OPT $\geqslant d$.

Lemma 2.4. *Alg* $\leqslant d$

Proof. Assume for contradiction.

At some point, Alg opens $d+1$ classroom.

Denote the lecture being considered by i. Then it is not compatible with other d lectures. Hence, there should be a time when $d+1$ lectures are active, which causes contradiction. $\square$

$\square$

2.3 Single-Source Shortest Path

Example 2.5 (Single-Source Shortest Path(SSSP))**.** Input: Graph $G = (V, E, w)$, V is the set of point and E is the set of edge with direction and $\omega : E \to \mathbb{R}_{\geqslant 0}$.

We want to find a path from s to t with minimum total cost.

Dijkstra's Algorithm Choose s as a source. $d[s] = 0, d[u] = \begin{cases} \omega(s,u) & \text{if } (s,u) \in E \\ +\infty & \text{otherwise} \end{cases}$, $S = \{s\}$ first. To record the path, we can use $\mathrm{Pred}[u] \leftarrow s$.

Algorithm 1 Dijkstra's Algorithm

while $S \neq V$ **do**
  Choose $u \in \arg\min_{\notin S}\{d[x]\}$.
  Update $S \leftarrow S \cup \{u\}$.
  for each $x \in V - S$,$(u,x) \in E$ **do**
    $d[x] \leftarrow \min\{d[x], d[u] + \omega(u,x)\}$.
    if $d[u] + \omega(u,x) < d[x]$ **then**
      $d[x] \leftarrow d[u] + \omega(u,x)$
      $\mathrm{Pred}[x] \leftarrow u$
    end if
  end for
end while

Theorem 2.6 (Invariant)**.** $\forall u \in S$, $d[u]$ *is the shortest path distance* $s \rightsquigarrow u$

Proof. Induction on $|S|$.

For $|S| = 1$ true.

Induction Step: Every time executing 2 in Algorithm 1, we need to prove $d[u]$ is the shortest distance $s \rightsquigarrow u$.

If $v = \text{Pred}[u] \in S$, then $d[u] = d[v] + \omega(v, u)$.

For any path from s to u, there exists $(\alpha, \beta) \in E$ such that $\alpha \in S, \beta \notin S$. Then

$$\begin{aligned}
\text{length}(P) &\geqslant \text{length}(P[s \to \beta]) \\
&= \text{length}(P[s \to \alpha]) + \omega(\alpha, \beta) \\
&\geqslant d[\alpha] + \omega(\alpha, \beta) \\
&\geqslant d[\beta] \geqslant d[u]
\end{aligned}$$

□

Remark 2.7. The straightforward implementation of Dijkstra's Algorithm is of $O(|v|^2)$.

If we use priority queue: Q with priority $Q.\pi()$. It has some methods:

- ExtractMin: Return $\arg\min_{x \in Q}\{Q.\pi(x)\}$ and remove x from Q.
- DecreaseKey: Update $Q.\pi(v)$ with newkey.

The time complexity is $|V| \times$ ExtractMin + $|E| \times$ DecreaseKey

Runtime	ExtractMin	DecreaseKey	Dijkstra
Simple Array	$O(\lvert V\rvert)$	$O(1)$	$O(\lvert v\rvert^2)$
Binary Heap	$O(\log \lvert V\rvert)$	$O(\log \lvert V\rvert)$	$O(\lvert E\rvert \cdot \log \lvert V\rvert)$
Fibonacci Heap	$O(\log \lvert V\rvert)$	$O(1)$ (amorized)	$O(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$

2.4 Minimum Spanning Tree

Example 2.8 (Minimum Spanning Tree (MST))**.** Input: Connected, undirected graph $G = (V, E, \omega)$.

Definition 2.9 (Spanning Tree)**.** $T \subset E$ is a **spanning tree** if $|T| = |V| - 1$, $G' = (V, T)$ is connected.

Goal of MST Find spanning tree T so that $\omega(T) = \sum_{e \in T} \omega(e)$ minimized.

Theorem 2.10 (Cayley Theorem)**.** *The number of spanning trees of n-vertex complete graph is n^{n-2}*

A **cut** $(S, V - S)$ has a **cutset** of $S = \{e = (u, v) : u \in S, v \notin S\}$.

Claim 2.4.1. Any cycle C and cutset D has intersection $|C \cap D|$ even.

Fundamental Cycle: Given G and spanning tree $T \subset E$, for each $e \in E \backslash T$, the unique cycle in $T \cup \{e\}$ is called **Fundamental cycle**.

Claim 2.4.2. For a fundamental cycle C related with e, $\forall f \in C \cap T$, $(T \cup \{e\}) \backslash \{f\}$ is also a spanning tree.

If T is MST, then $\omega(e) \geqslant \omega(f)$.

Fundamental Cut: Spanning tree $T \subset E$. For each $f \in T$, $T \backslash \{f\}$ has two connected components, whose cutset is called **fundamental cut**.

Claim 2.4.3. $\forall e \in D \backslash T$, $(T \cup \{e\}) \backslash \{f\}$ is a spanning tree.

If T is MST, then $\omega(e) \geqslant \omega(f)$.

MST Algorithm There are some rules. **Red rule**:Let C a cycle without red edges. Select an uncolored edge in C with max weight and color it red.

Blue rule: Let D be a cutset without blue edges. Select an un colored edge in D with min weight and color it blue.

Greedy Algorithm:Apply red or blue rules in any order iteratively until all edges colored.

Theorem 2.11. *Greedy algorithm terminates and blue edges from MST.*

Proof. Observed that during the algorithm, blue edges always from a forest. □

Invariant $\exists$ MST T^* *s.t.* T^* contains all blue edges and no red edges.

Proof. Proof by induction. If there is a MST T^* contains all blue edges no red edges now. If we apply blue rule, with cutset D and $f \in D$ but $f \notin T^*$, then for fundamental cycle C of f, $\forall e \in C \cap T, \omega(e) \geqslant \omega(f)$. Since C has even edges in the cutset by the claim, $\exists e \in C \cap T \quad s.t. \ e \in D$, which contradicts the fact that f is the edge in cutset D with min weight.

The case that we apply red rule is similar. □

Algorithm 2 Prim's Algorithm

1: Initialize $S \leftarrow \{s\}$.
2: **while** $n - 1$ times **do**
3: Choose e be the min weight edge in the cutset $(S, V \backslash S)$
4: add e to T, another endpoint of e to S.
5: **end while**

Remark 2.12. It is compatible with the simple idea: Each time chooses the min weight edge. However, it is more powerful since we only need to do this process in the cutset.

It is similar to Dijkstra's Algorithm. So its time complexity is $O(|E| + |V| \log |V|)$

Remark 2.13. The first step need time complexity $O(|E| \log |E|)$.

The second step need time complexity $O(|E| \cdot \alpha(|V|))$ using **Union-Find** data structure.

Algorithm 3 Kruskal's Algorithm

Consider edges in weight increasing order.
Add each edge to T if not introducing a cycle.

WLOG we can assume edge weights are distinct.

Algorithm 4 Boruvka's Algorithm

while $< (n-1)$ blue edges **do**
  Simultaneously apply blue rule to each blue compunent.
end while

Claim 2.4.4. WHILE loop iterates $\leqslant O(\log |V|)$.

So time complexity is $O(|E| \log |V|)$.

Remark 2.14. There is a "contraction View". For each step, we can view each component as a single point with edges to other components.

If the graph is **Planar Graph**, then $|E| \leqslant O(V)$. At the i-th WHILE iteration, $|V_i| \leqslant \frac{|V|}{2^{i-1}}$, $|E_i| \leqslant O(|V_i|)$.

So the time complexity is $\sum_i O(|E_i|) \leqslant \sum_i O\left(\frac{|V|}{2^{i-1}}\right) \leqslant O(|V|)$ which is linear!

Using the contraction view, we can get another algorithm:

Prim+Boruvka

- Run Boruvka for k iterates.
- Run Prim on the contracted graph.

Remark 2.15.

For step 1, time complexity is $k \cdot |E|$.

For step 2, time complexity is $|E| + \frac{|V|}{2^k} \cdot \log \frac{|V|}{2^k}$.

So the total time complexity is $k|E| + \frac{|V|}{2^k} \cdot \log \frac{|V|}{2^k}$.

Choose $k = \log_2 \log_2 |V|$, it comes to $(\log\log|V|) \cdot |E| + \frac{|V|}{\log_2 |V|} \cdot \log_2 |V| \leqslant O(|E|\log\log|V| + |V|)$.

2.5 Minimum Arborescence

Example 2.16 (Minimum Arborescence)**.** Input: Directed $G = (V, E)$, source $s \in V$ and weight $\omega : E \to \mathbb{R}$.

We want to find an **arborescence** $T = (V, E)$ with root r of minimum total weight.

Definition 2.17. Given directed $G = (V, E)$ and $r \in V$, $n = |V|, m = |E|$, $F \subset E$ is an **arborescence** if

- F is a spanning tree if ignoring directions
- $\forall v \in V$, $\exists$ unique path $r \to v$ in F.

Or equivalently, F has no directed cycles and every node $v \neg r$ has a unique incoming edge.

For this problem, WLOG we can assuem that the root r has no in-degree and assume $\omega \geqslant 0$.

For each $n \neq r$, let

$$\text{cheap}(v) = \text{argmin}_{e=(u,v)\in E}\{\omega(e)\}$$

Claim 2.5.1. Let $F = \{\text{cheap}(v) | v \neq r\}$. F is arborescense $\Rightarrow$ F is min-cost.

Define $\omega_r(u, v) = \omega(u, v) - \omega(\text{cheap}(v))$. Suffices to find the min-cost arborescence under ω_r.

If F is not an arborescense, then $\exists$ a directed cycle C with all edges of weight 0.

Using the contraction view, if we contract "0-cycle" and keep this process recursively. By taking degrees carefully we can easily confirm the legallity of the contraction view. Then suffices to prove it is indeed the min-cost arborescence when we expand after.

Theorem 2.18. *The min-cost arborescence $\widetilde{F}$ when we apply contraction to $0-$cycle is exactly the min-cost arborescence in the original graph after expanding.*

Lemma 2.19. *$\exists$ min-cost F^* s.t. only 1 edge in F^* entering C.*

Proof. Our goal is to prove $\omega_r(F) \leqslant \omega_r(F^*)$.

Let $F^*_C = F^* \cap (C \times C)$. Then $|F^*_C| = |C| - 1$.

Apply C-contraction to $F^* \backslash F^*_C$ we obtain an arborescence of $\widetilde{G}$. (Easy to check) So

$$\sum_{e \in F^* \backslash F^*_C} \omega_r(e) \geqslant \sum_{e \in \widetilde{F}} \omega_r(e)$$

So $\omega_r(F^*) \geqslant \omega_r(F)$ □

Proof of Lemma. Choose any $v \in C$.

Let $(x, y) \in r \to v$ be the first edge entering C.

Delete the edge entering $C \backslash \{y\}$ and add the edge of circle except the edge entering y.

Then it is an arborescence of less cost. □

3 Dynamic Programming

3.1 Weighted Interval Scheduling

Example 3.1 (Weighted Interval Scheduling)**.** Input: n jobs, $\{(s_i, f_i), \omega_i\}_{i=1}^n$. Want to find $\sum \omega_{i_k}$ maximum.

To make the structure simpler, we WLOG assume $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n$. We may

Algorithm 5 $\text{Search}(i)$

1: $j \leftarrow \min \text{id} > i, s_j \geqslant f_i$.
2: Return $\max\{\text{Search}(j) + \omega_i, \text{Search}(i+1)\}$.

find that there is a lot of repetitive computation. We can record each $\text{Search}(i)$

Algorithm 6 $\text{Search} - \text{Memorization}(i)$

1: If $i > n$, RETURN 0
2: If $i \neq$ bottom, RETURN $F[i]$.
3: $j(i) \leftarrow \min\{j | s_j \geqslant f_i\}$.
4: $F[i] \leftarrow \max\{\text{Search} - \text{M}(j(i)) + \omega_i, \text{Search} - \text{M}(i+1)\}$
5: RETURN $F[i]$

It can be written as

$$\begin{cases} F[i] = \max\{F[j(i)] + \omega_i, F[i+1]\} \\ F[n+1] = 0 \end{cases}$$

Such an equation is called **Bellman Equation**. So Dynamic Programming is a method to solve the problem by finding the optimal solution of each subproblem. We sometimes need to record the optimal solution of each subproblem to avoid repetition.

3.2 Segmented Least Square

Example 3.2 (Least Square)**.** We have n points $\{(x_i, y_i)\}_{i=1}^n$. We want to find a line $y = ax + b$ to minimize

$$\text{SSE} = \sum_{i=1}^{n} [y_i - (ax_i + b)]^2 \tag{3.1}$$

Actually,

$$\begin{cases} a = \dfrac{n \sum_i x_i y_i - \left(\sum_i x_i\right)\left(\sum_i y_i\right)}{n \sum_i x_i^2 - \left(\sum_i x_i\right)^2} \\ \\ b = \dfrac{\sum_i y_i - a \sum_i x_i}{n} \end{cases}$$

Example 3.3 (Segmented Least Square)**.** Input: $\{(x_i, y_i)\}_{i=1}^n$, $c > 0$.

Goal: Minimize $l = E + cL$ for piecewise line, where c is the **hyperparameter**, L is the number of the segments.

WLOG, assume $x_1 < x_2 < \cdots < x_n$.

We can define its subproblem as

$$\text{OPT}[i] : \text{min loss}$$

when in put is $(x_1, y_1), \cdots, (x_i, y_i)$.

Find solution $\text{OPT}[n]$. The boundary condition is $\text{OPT}[1] = \text{OPT}[2] = c$ and the **Bellman Equation** is

$$\text{OPT}[i] = \min_{1 \leqslant j \geqslant i} \{\text{OPT}[j-1] + l_{ji} + c\}$$

3.3 Knapsack Problem

Example 3.4 (Knapsack Problem)**.** Input: n items, w_i, v_i for its weight and value. The capacity of knapsack is w.

If assume integral weight, then denote $\mathrm{OPT}[i,w]$ as the optimal total value when in put is first knapsack capacity is w.

The **Bellman Equation** is

$$\mathrm{OPT}[i,w] = \begin{cases} \mathrm{OPT}[i-1,w] & w < w_i \\ \max\{\mathrm{OPT}[i-1,w], v_i + \mathrm{OPT}[i-1,w-w_i]\}, w \geqslant w_i \end{cases}$$

It has time complexity $O(nw)$, which is not a polynomial algorithm.

We can find another Value-Based DP: (Also assume integral values)

$$\mathrm{OPT}[i,v]\text{: choose min weight items.}$$

from item $1, 2, \cdots, i$ so that total value $\geqslant v$.

The final solution for maxmial $v \quad s.t.\ \mathrm{OPT}[n,v] \leqslant w$.

$$OPT[i,v] = \min \begin{cases} \mathrm{OPT}[i-1,v] \\ w_i + \mathrm{OPT}[i-1,(v-v_i)^+] \end{cases}$$

$\mathrm{OPT}[0,v] = \begin{cases} 0 & v = 0 \\ +\infty & v > 0 \end{cases}$ The time complexity is $O(n^2 v)$.

Now we consider a α**-approximation algorithm** that $\mathrm{ALG} \geqslant \alpha \cdot \mathrm{OPT}$ for $\alpha \in (0,1]$.

Let $\varepsilon = 1 - \alpha$.

Algorithm 7 Knapsack Problem

1: Assume WLOG $w_i \leqslant W$ so that $V \geqslant \mathrm{OPT}$.
2: Set $K = \dfrac{\varepsilon V}{n}$. Let $v_i' = \left[\dfrac{v_i}{K}\right]$
3: Run value-based DP to find optimal solution T for I'
4: Return T as a solution to I.

It is a feasible solution and

$$\begin{aligned}
\sum_{i\in T} v_i' &= \mathrm{OPT}(I') \\
&\geqslant v(S; I'),\ \forall \text{feasible} S \\
&\geqslant v(T^*, I') \\
&= \sum_{i\in T*} v_i' \\
&= \sum_{i\in T*} \left\lceil \frac{v_i}{K} \right\rceil \\
&\geqslant \sum_{i\in T*} \left(\frac{v_i}{K} - 1 \right) \\
&\geqslant \frac{1}{k} \sum_{i\in T*} \sum_{i\in T*} v_i - n \\
&= \frac{1}{K} \mathrm{OPT}(I) - n
\end{aligned}$$

So $\mathrm{ALG} \geqslant \sum_{i\in T} K \cdot v_i' \geqslant \mathrm{OPT}(I) - nK \geqslant (1-\varepsilon)\mathrm{OPT}(I)$.

The time complextity is $O(n^2 V') = O(n^2 \frac{V}{K}) = O(n^3 \varepsilon^{-1})$.

Remark 3.5. The time complextity depends on the accuracy ε instead of the maximum value V since the accuracy is based on scale.

In other words, ε^{-1} in time complexity represents not only accuracy but also the "size" of scale.

Fully Polynomial-Time Approximation Scheme(FPTAS) $\forall \varepsilon$, $\exists$ $(1-\varepsilon)$-approximation algorithm with time complexity $f(n,\varepsilon) = \mathrm{poly}(n, \frac{1}{\varepsilon})$.

PTAS : $\forall \varepsilon$, $\exists$ $(1-\varepsilon)$-approximation in time $f_\varepsilon(n) = \mathrm{poly}(n)$. For this algorithm, it is $(n \cdot 2^{\frac{1}{\varepsilon}}, n^{\frac{1}{\varepsilon}})$.

3.4 RNA Secondary Structure

Example 3.6 (RNA Secondary Structure)**.** RNA is a string $b_1 b_2 \cdots b_n$ where $b_i \in \{A, C, G, U\}$.

The secondary structure is what fold to form "base pairs" including:

$$U \cdots A, A \cdots U, C \cdots G, G \cdots C$$

Mathematically, second structure represented by set of base pairs $S = \{(i, j)\}$,

*) $\forall (i, x) \in S, (b_i, b_j) \in \{U \cdots A, A \cdots U, C \cdots G, G \cdots C\}$

*) no sharp turns: $\forall (i, j) \in S$, $i < j - 4$,

*) non-crossing: $\forall (i, j), (k, l) \in S$, cannot have $i < k < j < l$.

Goal: Maximize $|S|$.

A direct idea is to construct those subproblems:

$$\mathrm{OPT}[i, j] = \max_{i \leqslant k < j-4} \begin{cases} \mathrm{OPT}[i, j-1] & b_j \text{ not matched} \\ 1 + \mathrm{OPT}[i, k-1] + \mathrm{OPT}[k+1, j-1] & b_j \text{ matched with } b_k \end{cases}$$

$$\mathrm{OPT}[i, j] = 0 \text{ when } i \leqslant j < i + 4$$

3.5 Sequence Alignment(Edit Distance)

Example 3.7. For a wrong-spelled word, what cost do we need to make it right, using the gap and mismatch.

Or what is its edit distance to the correct word.

Mathematically, for string $(a_1 \cdots a_n), (b_1 \cdots b_m)$, a matching $M = \{(i, j)\}$ such that there is no $(i_1, j_1), (i_2, j_2) \in M \quad s.t.\ i_1 < i_2$ but $j_2 < j_1$. Define its cost

$$\mathrm{cost}(M) = \sum_{(i,j)\in M} \alpha_{a_i b_j} + \sum_{i\in[n], i \text{ not in } M} + \sum_{\substack{j\in[m] \\ j \text{ not in } M}} \delta$$

$\sum\limits_{(i,j)\in M} \alpha_{a_i b_j}$ is the mismatch cost and $\sum_{i\in[n], i \text{ not in } M} + \sum_{j\in[m], j \text{ not in } M} \delta$ is the gap cost

Define $\mathrm{OPT}[i, j]$ is the edit distance between $a_1 a_2 \cdots a_i$ and $b_1 b_2 \cdots b_j$.

$$\mathrm{OPT}[i, j] = \min_{1\leqslant k\leqslant j} = \begin{cases} \delta + \mathrm{OPT}[i-1, j] & a_i \text{ not matched} \\ \alpha_{a_i b_k} + \delta \cdot (j-k) + \mathrm{OPT}[i-1, k-1] & a_i \text{matched with } b_k \end{cases}$$

However, for each case it can be divided into three cases:

$$\mathrm{OPT}[i, j] = \min \begin{cases} \mathrm{OPT}[i-1, j-1] + \alpha_{a_i b_j} \\ \mathrm{OPT}[i-1, j] + \delta \\ \mathrm{OPT}[i, j-1] + \delta \end{cases}$$

The question is, if we need to trace the matching process, the space complexity is $O(nm)$, too large.

Here we use binary search.

Algorithm 8 Binary Search

1: Compute $A[j] = d[(0,0) \to (\frac{n}{2}, j)]$ and $B[j] = d[(\frac{n}{2}, j) \to (n, m)]$,
2: find $j^* = \mathrm{argmin}_j\, A[j] + B[j]$.
3: Run the sub-process $(0,0) \to (\frac{n}{2}, j^*)$ and $(\frac{n}{2}, j^*) \to (n, m)$

The complexity is still $O(nm) + \frac{1}{2}O(nm) + \cdots + \frac{1}{2^k}O(nm) = O(nm)$.

3.6 Matrix Multiplication

Example 3.8 (Matrix Multiplication)**.** Consider $M_1 \cdot M_2 \cdots M_k$ where M_i is a $n_{i-1} \times n_i$ matrix.

We want to find the optimal multiplicative order such that the time cost is minimal.

Denote $\mathrm{OPT}[i,j]$ is the min from M_i to M_j.

Using the binary tree, consider the last multiplication

$$\mathrm{OPT}[i,j] = \min_{i \leqslant l < j} \{\mathrm{OPT}[i,l] + \mathrm{OPT}[l+1,j] + n_{i-1}n_l n_j\}$$

4 Flow Network

4.1 Definition

Example 4.1. For directed graph $G = (V, E, s, t, c)$ where s is the source and t is the sink. $c : E \to \mathbb{R}_{\geqslant 0}$ is the capacity function.

The **st-flow** is $f : E \to \mathbb{R}_{\geqslant 0}$ *s.t.*

1) $\forall e \in E$, $f(e) \leqslant c(e)$.

2) $\forall v \in V \backslash \{s,t\}$, $\sum\limits_{(u,v)\in E} f(u,v) = \sum\limits_{(v,u)\in E} f(v,u)$, *i.e.* flow conservation.

$$\mathrm{val}(f) = \sum_{(s,u)\in E} f(s,u) - \sum_{(u,s)\in E} f(u,s)$$

Our goal is to maximize $\mathrm{val}(f)$

An **st-cut** is a partition (A,B) of V such that $s \in A, t \in B$, the capacity

$$c(A,B) = \sum_{\substack{(u,v)\in E \\ u\in A, v\in B}} c(u,v)$$

Claim 4.1.1. $\forall$ feasible flow f and st-cut (A, B),

$$\text{val}(f) \leqslant c(A, B)$$

Residual Network Given flow network G, feasible flow f, the residual network $G_f(v, E_f, s, t, c_f)$ is for each $e \in E$

$$c_f(e) = c(e) - f(e) + f(e^{\text{reverse}})$$

where $u \to v$ is on the flow.

Claim 4.1.2 (Weak Duality). f' is a feasible flow in G_f if and only if $f \oplus f'$ is feasible in G, where

$$(f \oplus f')(e) = f(e) + f'(e) - f'(e^{\text{reverse}})$$

An **augmenting path** P is an unsaturated $s \to t$ path in G_f.

Algorithm 9 Augment (f, P)

Let $\delta = \min_{e \in P} c_f(e)$.
for $e = (u, v) \in P$ **do**
    if $e \in E$ **then**
        $f(e) \leftarrow f(e) + \delta$
    else
        $f(v, u) \leftarrow f(v, u) - \delta$
    end if
end for

Now we give the Ford-Fulkerson Algorithm.

Theorem 4.2. *If F-F algorithm terminates, it finds a max flow.*

Algorithm 10 Ford-Fulkerson Algorithm

```
f ← 0
while ∃ augmenting path P in G_f do
    Augment(f, P)
end while
return f
```

Claim 4.1.3. $\forall$ st-cut (A, B), st-flow f, we have

$$\mathrm{val}(f) = \sum_{\substack{u \in A, v \in B \\ (u,v) \in E}} f(u, v) - \sum_{\substack{u \in E, v \in A \\ (u,v) \in E}} f(u, v)$$

It proves the previous claim weak duality.

Proof.

$$\begin{aligned} \mathrm{val}(f) = & \sum_{(s,v) \in E} f(s, v) - \sum_{(u,s) \in E} f(u, s) \\ & + \sum_{\omega \in A - \{s\}} \left(\sum_{(u,w) \in W} f(u, w) - \sum_{(w,v) \in E} f(w, v) \right) \end{aligned}$$

$\square$

Proof of the Theorem 4.2. Consider the residue graph G.

Denote A to be the set of nodes reachable from s. $B = V \backslash A$. $t \in B$ since there is no path from s to t.

Then st-cut (A, B) has capacity $c_f(A, B) = 0$. So for $u \in B, v \in A$, since $f(u, v) \neq 0 \Rightarrow c_f(v, u) > 0$, we have $c_f(v, u) = 0 \Rightarrow f(u, v) = 0$.

$$\mathrm{val}(f) = \sum_{\substack{u \in A, v \in B \\ (u,v) \in E}} f(u, v) - \sum_{\substack{u \in B, v \in A \\ (u,v) \in E}} f(u, v)$$

$$= \sum_{\substack{u \in A, v \in B \\ (u,v) \in E}} c(u,v) - 0$$

$$= c(A,B)$$

$\square$

Now suffices to proof that the algorithm terminates.

Lemma 4.3. *If capacities are integral and less than c, then F-F terminates in $O(nmC)$ time and returns an integral max flow.*

The lemma implies we should choose some proper path so that it will terminate fast.

Assume the integral capacities $\leqslant C$ and $G_f(\Delta)$ denoted as G_f with edges of capacites $\geqslant \Delta$.

Algorithm 11 Capacity-Scaling Algotihm

1: Initiate $f \equiv 0$, $\Delta \leftarrow$ largest $2^k \leqslant c$.
2: **while** $\Delta \geqslant 1$ **do**
3:     **while** $\exists$ augmenting path P in $G_f(\Delta)$ **do**
4:         Augment(f, P)
5:     **end while**
6:     $\Delta \leftarrow \Delta/2$
7: **end while**

Theorem 4.4. *The C-S runs in time $O(m^2 \log c)$ since the step 2 runs for $O(m)$ iterations.*

Lemma 4.5. *Every time inner WHILE terminates, max-flow value is less than* $\mathrm{val}(f) + m\Delta$.

Corollary 4.6. *Each inner WHILE iterates $\leqslant 2m$. The times complexity is $O(m^2 \log C)$*

Proof of Lemma. We let A be all nodes reachable from s and $B = S\backslash A$.

$$
\begin{aligned}
\mathrm{val}(f) &= \sum_{e\in E \text{ from } A \text{ to } B} f(e) - \sum_{e\in E \text{ from } B \text{ to } A} f(e) \\
&> \sum_{e\in E \text{ from } A \text{ to } B} (c(e) - \Delta) - \sum_{e\in E \text{ from } B \text{ to } A} (\Delta) \\
&= c(A,B) - \sum_{e\in E \text{ between } A, B} \Delta \\
&\geqslant c(A,B) - m\Delta \\
&\geqslant \mathrm{MaxFlow} - m\Delta
\end{aligned}
$$

□

Algorithm 12 Shortest Augment Path

```
Initiate f ← 0.
while ∃s → t path in G_f do
    Find P : s → t in G_f using least number of edges.
    Augment (f, P).
end while
```

Lemma 4.7. *Length of the shortest augmenting path never decreases.*

Lemma 4.8. *After $\leqslant m$ iterations, length of the shortest augmenting path strictly increases. Time complexity is $O(nm^2)$*

Proof. Assume $f \xrightarrow{\text{augment(f,P)}} f'$ Denote $l(u), l'(u)$ as the length of the shortest $s \to u$ path in $G_f, G_{f'}$ respectively.

Our goal is to prove $l(u) \leqslant l'(u)$.

$l(u)$ determines "distance" to s.

Define the level graph as the set of all $(u,v) \in E(G_f)$ such that $l(u) + 1 = l(v)$.

Call edges not belong to level graph as *back edge*.

Observation Consider any $e \in E(G_{f'}) \backslash E(G_f)$, e must be a back edge in G_f.

Choose u such that $l'(u) < l(u)$ and $l'(u)$ minimized.

If (v, u) is the edge in the shortest path of $G_{f'}$

$$l(v) \leqslant l'(v) = l'(u) - 1 < l(u) - 1 \leqslant l(u) - 2$$

so (u, v) is not a back edge in G_f, hence $(v, u) \notin E(G_{f'})$, which causes contradiction. □

Lemma 4.9. *After $\leqslant m$ augmentation, $\exists u$, $l(u)$ strictly increases. It goes on no more n^2 times, so the time complexity is $O(n^2 m^2)$.*

Proof. This lemma is much easier than the previous lemma.

Noticed that each augmentation adds back edges and removes at least one edges in level graph. □

Proof of Lemma 4.8.

Claim 4.1.4. During the period when $l(t)$ doesn't increase, the added edges in residual graph does not appear in shortest augmenting path.

Suppose for contradiction: $\exists j < i$, $l_j(t) = l_i(t)$, $\exists$ (v, u) appears in the shortest augmenting path P in G_{f_i} and $l_j(v) = l_j(u) + 1$.

Choose the edge (v, u) with smallest i and then with largest $l_i(u)$.

Then $l_i(u) \geqslant l_j(u) + 2$. So

$$\begin{aligned} l_j(t) &\leqslant l_j(u) + |P[u \to t]| \\ &\leqslant l_i(u) + |P[u \to t]| - 2 \\ &\leqslant l_i(t) - 2 \end{aligned}$$

$\square$

Recent work [Chen et at al. '2022] we can do in $O(m^{1+o(1)})$.

4.2 Appllication

4.2.1 Bipartite Matching

Example 4.10 (Bipartite Matching)**.** For Bipartite graph $G = (U, V, E)$, a matching $M \subset E$, we want to find M to maximize $|M|$.

We can construct two virtual nodes s, t such that $s \to$ all nodes in U and $t \to$ all nodes in V, with capacity 1.

Then the maximum capacity of flow in the augmented graph is what we need.

So the meaning of capacity can be generalized as the number of one node who can accommonded

Now consider so-called "perfect matching", *i.e.* $|M| = |U| = |V|$.

Note that $\exists$ perfect matching *s.t.* $\forall S \subset U$, $|\Gamma(S)| \geqslant |S|$.

Theorem 4.11 (Hall's Theorem)**.** *The inverse still holds. i.e. If* $\forall S \subset U$, $|T(S)| \geqslant |S|$*, then* $\exists M$ *is perfect matching if* $|M| = n$.

Proof. It suffices to prove max-flow$= n$. Or equivalently, to prove min-cut$\geqslant n$.

i.e. $\forall$ s-t cut (A, B), $c(A, B) \geqslant n$.

$c(A, B) < +\infty \overset{1}{\Rightarrow}$ if $u \in A$ then $\Gamma(u) \in A \Rightarrow \Gamma(A \cap U) \subset A \cap V$.

$$
\begin{aligned}
c(A, B) &\geqslant |B \cap U| + |A \cap V| \\
&\geqslant n - |A \cap U| + |\Gamma(A \cap U)| \\
&\geqslant n
\end{aligned}
$$

$\square$

Remark 4.12. Here we mark the weight between U and V to be ∞ such that the fact 1 holds.

The duality of max-flow and min-cut is very useful in this problem.

4.2.2 Network Connectivity

Example 4.13 (Network Connectivity)**.** Directed $G = (V, E)$, source s, sink t. Then Max-flow = the maximum numbered of edge-disjoint $s \to t$ path. Two paths are called *edge-disjoint* if they have no edge in common.

Connectivity of the graph is defined as the $\min\limits_{E' \subset E} |E'|$ such that $s \to t$ disconnected in $(V, E \backslash E')$

Theorem 4.14 (Menger′s Theorem)**.** *Connectivity=min-cut=max-flow=maximum number of edge-disjoint $s \to t$ path.*

4.2.3 Circulation

Example 4.15. Directed graph $G = (V, E)$ with capacity $c : E \to \mathbb{R}_{\geqslant 0}$ and node demand $d : V \to \mathbb{R}$. ($d(u) < 0$ means the supply node)

We have the flow conservation

$$\sum_{e \text{ into } u} f(e)0 - \sum_{e \text{ out of } u} f(e) = d(u), \ \forall u$$

Our task is to decide whether there exists a feasible flow f satisfies the flow conservation.

Indeed, we can consrtuct two vitual nodes s, t such that s to all nodes with demand $d < 0$, equipped with capacity d and t has edges from all nodes with demand $d > 0$, equipped with capacity d.

Then the task is equivalent to check whether the max-flow saturates all edges out of s and in of t.

Moreover, we can use the cut to discuss.

We have:

$$\nexists \text{ feasible circulation} \Leftrightarrow \exists \text{ cut } (A,B) \quad s.t.\ c(A,B) < \sum_{v\in B} \mathrm{d}(v).$$

Remark 4.16. This crieterion, similar as Hall's Theorem 4.11, is so-called "*polynomial proof*", under the meaning that for a specific case, we can give a proof in polynomial time to check.

Flow Lower Bounds If we have a capacity constraint such that

$$l(e) \leqslant f(e) \leqslant c(e), \forall e \in E$$

Then it suffices to add the lower flow at first. For example, for two nodes with demand 0 and edge with amount in $[4,6]$, we can replace it with

$$4 \xrightarrow{[0,6]} -4$$

4.2.4 Survey Design

Example 4.17 (Survey Design)**.** We ask n_1 customers about n_2 products. Ask customer i the number between $[c_i, c_i']$ products and ask the number between $[p_j, p_j']$ customers questions about product j.

We want to find if there is a feasible survey design.

It is equivalent to give each edge a weight interval, where $s \to i$ with $[c_i, c_i']$, $i \to j$ with $[0,1]$ and $j \to t$ with $[p_j, p_j']$.

4.2.5 Airline Scheduling

Example 4.18 (Airline Scheduling)**.** Flight i from the origin o_i at time s_i to the destination d_i at time f_i.

We want to know what is the minimum number of crews in flights that can be scheduled. A feasible schedule for one crew is a set of flights $\{i_1, i_2, \cdots\}$ such that $f_{i_k} \leqslant s_{i_{k+1}}$, $d_k = o_{k+1}$.

We can construct a graph with nodes $o_i \to d_i$. The edges $o_i \to d_i$ with weight $[1, 1]$. If the schedule from flight i to flight j is feasible *i.e.* $f_i \leqslant s_j$, we let edge $i \to j$ with weight interval $[0, 1]$.

Then a feasible flow gives a feasible schedule. To limit the total amount, we can determine the minimum number.

Remark 4.19. The weight interval have a broader meaning in this problem. With different view of nodes and edges, we can transform it into different limitations.

4.2.6 Image Segmentation

Example 4.20 (Image Segmentation)**.** For an image, $p_{ij} \geqslant 0$ is the separation penalty if neighbors i, j belongs to different partitions.

$a_i \geqslant 0$ is the likelihood that $i \in A$ (foreground) $b_i \geqslant 0$ is the likelihood that $i \in B$ (background)

Our goal is to partition pixels into A, B, to maximize

$$\sum_{i \in A} a_i + \sum_{j \in B} - \sum_{\substack{i,j \text{ neighbors} \\ |\{i,j\} \cap A| = 1}}$$

It is equivalent to

$$\text{minimize } -\sum_{i\in A} a_i - \sum_{j\in B} b_j + \sum_{\substack{i,j \text{ neighbors} \\ |\{i,j\}\cap A|=1}} p_{ij}$$

$$\Leftrightarrow \text{minimize } \sum_{i\in B} a_i \sum_{j\in A} b_j + \sum_{\substack{i,j \text{ neighbors} \\ |\{i,j\}\cap A|=1}} p_{ij}$$

Then we can construct a visual source with edges to all pixels with weight a_i and a visual sink with edges from all pixels with weight b_i. All neighbors of pixel have edges of weight p_{ij} from each other

Remark 4.21. This example focuses on the optimal sum of net flow. To construct visual source, sink and proper edges, we can optimize some sum of structure with related constraints.

4.2.7 Project Selection

Example 4.22 (Project Selection). $v \to w$, v depends on w. Our goal is to find a feasible set S of projects(if $v \in S$, then all prequisites of $v \in S$) to maximize

$$\sum_{v\in S} p(v)$$

- Introduce the virtual source node s and the virtual sink node t.
- Assign a capacity of ∞ to each prerequisite edge.
- Add edge (s, v) with capacity $p(v)$ if $p(v) > 0$.
- Add edge (v, t) with capacity $-p(v)$ if $p(v) < 0$

Then the min-cut (A, B) satisfies:

1) $\forall (u, w) \in E, u \in S \Rightarrow w \in A$

2)

$$
\begin{aligned}
c(A, B) &= \sum_{\substack{v \in B \\ p(v) > 0}} p(v) + \sum_{\substack{v \in A \\ p(v) < 0}} (-p(v)) \\
&= \sum_{v : p(v) > 0} p(v) - \sum_{\substack{v \in A \\ p(v) > 0}} p(v) - \sum_{\substack{v \in A \\ p(v) < 0}} p(v) \\
&= \sum_{v : p(v) > 0} p(v) - \sum_{v \in A} p(v)
\end{aligned}
$$

So it suffices to compute $c(A, B)$!

Remark 4.23. Use edges of capacity ∞, we can reduces some situation we do not want.

4.2.8 Baseball Elimination

Example 4.24. Given set of team S, distinguished team $z \in S$. Team x has won w_x games already. Team x and y play each other r_{xy} additional games.

Given the current standings, is there any outcome of the remaining games in which team z finishes with the most (or tied for the most) wins?

Assume team z wins all remaining games. $M = w_z + \sum_x r_{zx}$.

We want to arrange the remaining games that do not involve team z so that all other teams have $\leqslant M$ wins.

- Construct two virtual nodes s and t.
- Assign a capacity of ∞ to the edge from the match $i - j$ to team i and j.
- Assign a capacity of $M - w_i$ to the edge from team i to t.

- Assign a capacity of r_{ij} to the edge from team s to the match $i-j$.

Then z can possibly win the most games iff the max-flow saturates for all edges from s.

Certificate of Elimination $T \subset \{2,3,\cdots,n\}$, $\omega(T) = \sum_{i\in T}\omega_i, r(T) = \sum_{i<j\in T} r_{ij}$, $\dfrac{\omega(T)+r(T)}{|T|} > M$.

Noticed that if the following equation holds, then team z is theoretically eliminated.

$$\frac{\omega(T)+r(T)}{|T|} > M \tag{4.1}$$

Theorem 4.25. *Team q is theoretically eliminated iff $\exists T \subset \{2,3,\cdots,n\}$ s.t. (4.1) holds*

Proof. If $\forall T \subset \{2,3,\cdots,n\}$, $\dfrac{\omega(T)+r(T)}{|T|} < M$, then

$$\min-\text{cut} \geqslant \sum_{1<i<i\leqslant n} r_{ij} = r(\{2,3,\cdots,n\})$$

That's because, if we consider any $s-t$ cut (A,B) such that $c(A,B) < +\infty$.

1) $i \in B \Rightarrow$ match $i-j \in B$, $\forall j$.

$$\begin{aligned}
c(A,B) &\geqslant \sum_{i\in A}(M-w_i) + \sum_{\substack{i\in B \text{ or } j\in B\\ 1<i<j\leqslant n}} r_{ij}\\
&= r(\{2,3,\cdots,n\}) + |A\backslash\{s\}|\cdot M - \omega(A\backslash\{s\}) - r(A\backslash\{s\})\\
&\geqslant r(\{2,3,\cdots,n\})
\end{aligned}$$

The inverse is trivial. □

Remark 4.26. Note that we use the duality of max-flow and min-cut in this problem.

5 Introduction to Approximation Algorithm

There are plenty of NP-hard optimization problems.

Example 5.1 (3-CNF)**.** For n Boolean variables $\{x_1, x_2, \cdots, x_n\}$.

A *literal* is either a value x_i or negative value $\bar{x}_i$.

A *clause* is a disjunction of literals. *e.g.* $(x_1 \vee x_2 \vee \bar{x_3})$.

A **CNF** is the conjunction of clauses. *e.g.* $(x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_1 \vee \bar{x_2} \vee x_3)$.

A **3-CNF** is a CNF with at most 3 literals in each clause.

Decision problems Check whether we can choose $x_1, \cdots, x_n$

For maximization problems, A is an α**-approximation algorithm** if

$$\mathrm{Val}(A(I), I) \geqslant \alpha \cdot \mathrm{OPT}(I),\ \alpha \in (0, 1]$$

For minimization problem, it will be

$$\mathrm{Val}(A(I), I) \leqslant \alpha \cdot \mathrm{OPT}(I),\ \alpha \in [1, \infty)$$

Max-cut problem: Find the maximal number of bichromatic edges using 2 colors. We have

$$|\mathrm{edge}(A, B)| = \sum_{i=1}^{n} \max\{a_i, b_i\} \geqslant \sum_{i=1}^{n} \frac{a_i + b_i}{2} = \frac{1}{2}|E| \geqslant \frac{1}{2}\mathrm{OPT}$$

So this is a $\frac{1}{2}$-approximation algorithm for max-cut problem.

However, if we consider it as a minimization problem, the value can be 0, so the scale can be $+\infty$. Hence, it is different if we consider the approximation algorithm of min-uncut or max-cut problems.

Decision Problems	**Natural Optimization Problems**
Is there a truth assignment that satisfies the 3-CNF formula?	Maximize the number of clauses satisfied by a truth assignment.
3-coloring (NP-complete)	• Min-coloring • Max-3-cut: Max number of bichromatic edges using 3 colors. • Min-3-Uncut: Min number of monochromatic edges using 3 colors.
2-colorin (P complete)	• Max-cut • Min-Uncut
Vertex Cover: Given $G = (V, E), k$. Decide whether $\exists$ Vertex-Cover using $\leqslant$ vertices.	Min-vertex-Cover: Given $G = (V, E)$. Find $S \subset V$ *s.t.* $\forall e = (u, v) \in E$, $\lvert\{u, v\} \cap S\rvert \geqslant 1$, $\lvert S\rvert$ is minimized.

Table 1: Comparison of Decision Problems and Natural Optimization Problems

c vs. s Decision Problem Decide whether $\mathrm{OPT}(I) \geqslant c$ or $\mathrm{OPT}(I) < s$. If $\mathrm{OPT}(I) \geqslant c$, return YES, else if $\mathrm{OPT}(I) < s$, return NO.

c vs. s Decision Problem Given I *s.t.* $\mathrm{OPT}(I) \geqslant c$, find a solution x *s.t.* $\mathrm{Val}(x; I) \geqslant s$.

Theorem 5.2. *Suppose A solves c vs. s Search problem in poly-time, then $\exists$ poly-time A' that solves c vs. s decision problem.*

The algorithm is as follows:

Algorithm 13 Greedy

```
initiate A, B ← ∅
for i from 1 to n do
    Let a_i ← number of edges between A and i.
    Let b_i ← number of edges between B and i.
    if a_i < b_i then
        A ← A ∪ {i}.
    else
        B ← B ∪ {i}
    end if
end for
```

Algorithm 14 c vs. s Search

```
x ← A(I)
if Val(x; I) ⩾ s then
    return YES
else
    return NO
end if
```

Fact 5.3.

(1) *A is α-approximation algorithm $\Rightarrow$ A is c vs. αc search algorithm $\forall c$.*

(2) *$\exists$ c vs. $s(c)$ search algorithm $\Rightarrow$ $\exists$ α-approximation algorithm where $\alpha = \inf_c \{\frac{s(c)}{c}\}$*

(3) *(contrapositive of A) c vs. s decision problem "hard" $\Rightarrow$ $\frac{s}{c}$-approximation algorithm "hard".*

Remark 5.4. The same c vs. s algorithm in max-cut and min-uncut problems might correspond to quite different approximation ratios.

5.1 Set-Cover

Example 5.5 (Set-Cover). Universe $U = \{1, 2, \cdots, n\}$. $S_1, S_2, \cdots, S_M \subset U$.

Our goal is to find $T \subset \{S_1, \cdots, S_M\}$ such that $\cup_{S\in T} S = U$ and $|T|$ minimized.

Of course, it can be represented as **Max-Coverage** problem: Given additional input k. Find $T \subset \{S_1, \cdots, S_M\}$ *s.t.* $|T| = k$ and $|\cup_{S\in T} S|$ maximized.

It has a greedy algorithm:

Algorithm 15 Greedy

1: $T \leftarrow \varnothing$

2: **repeat**

3: Let S_i be the set that covers the most uncovered elements.

4: $T \leftarrow T \cup \{S_i\}$.

5: $U \leftarrow U \backslash S_i$.

6: **until** $\begin{cases} \text{All elements covered} & \text{set cover} \\ |T| = k & \text{max-coverage} \end{cases}$

Fact 5.6. *Suppose $\exists\ m$ sets covering U. After t choices, T covers $1 - \left(1 - \frac{1}{m}\right)^t$ fraction of elements.*

Corollary 5.7. *The greedy algorithm is $\lceil \ln n \rceil$ approximation for set-cover*

Proof. Let $m = \text{OPT}$. After $t = \lceil \ln n \rceil \cdot m$ choices, number of uncovered elements

$$\left(1 - \frac{1}{m}\right)^{\lceil \ln n \rceil \cdot m} \cdot n \geqslant \frac{1}{n} = 1$$

□

Corollary 5.8. *Greedy is 1 vs. $1 - \frac{1}{e}$ approximation for Max-coverage.*

Proof. Set $m = k$. After $t = k$ choices, coverage of T:

$$1 - (1 - \frac{1}{k})^k \geqslant 1 - \frac{1}{e}$$

□

Fact 5.9. *1 vs. 1-* γ *approximation for ma-coverage* $\Rightarrow \lceil \log_{1-\gamma} \frac{1}{n} \rceil$*-approximation for set-cover.*

Proof. "Guess" $k = \mathrm{OPT}^{\text{set}-\text{cover}}$.

Repeatly invoke $A(k) \cdot \lceil \log_{1-\gamma} \frac{1}{n} \rceil$ times. Then number of uncovered elements

$$n \cdot (1-\gamma)^{\lceil \log_{1-\gamma} \frac{1}{n} \rceil} \geqslant n \cdot \frac{1}{n} = 1$$

□

In fact, we can construct an extreme case for greedy algorithm:

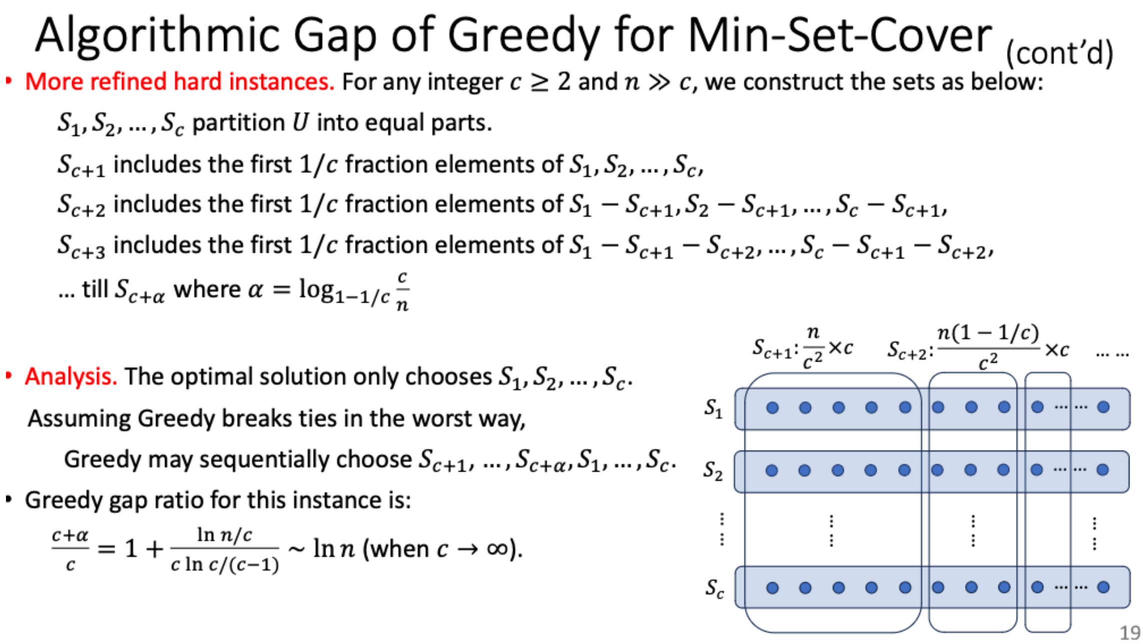


Figure 5.1: Extreme case for greedy algorithm

5.2 Weighted Min Set-cover and Randomized Rounding

Example 5.10 (Weighted Min Set-cover)**.** Given n elements, M sets, $S_1, \cdots, S_M \subset U$. Each set S_i has a weight $w(S_i) > 0$.

Now select $T \subset \{S_1, \cdots, S_M\}$ such that $\sum_{S \in T} w(S)$ minimized.

Integer Program: Minimize $\sum_{i=1}^{M} w(S_i)x_i$. Subject to $\sum_{i:j \in S_i} x_i \geqslant 1, \forall j \in U$, where $x_i \in \{0,1\}, \forall i \in [M]$.

If we relax the integer constraint, we have an LP relaxation: $x_i \in [0,1]$, which can be solved in poly-time since it is a linear program.

We need "rounding" to transform fractional solution to the integer solution.

5.2.1 Randomized Rounding

If $\{x_i^*\}$ is the optimal LP solution. For each s_i, select $s_i \in T$ independently with probability $\min\{\alpha x_i^*, 1\}$. Then

$$\mathbb{E}[w(T)] \leqslant \alpha \sum_{i=1}^{M} w(s_i),\ x_i^* = \alpha \cdot \mathrm{LP} \leqslant \alpha \cdot \mathrm{OPT}$$

Now we want to estimate $\Pr[T \text{ covers } U]$.

If there is some $\alpha x_i^{\geqslant} 1$, then $\Pr(T \text{ covers } U) = 1$. If $\forall i, x_i^* < 1$, then

$$\begin{aligned}
\Pr[T \text{ covers } U] &= 1 - \Pr[\exists j \in U, j \notin T] \\
&\geqslant 1 - \sum_{j \in U} \Pr[j \notin T] \\
&= 1 - \sum_{j \in U} \prod_{i:j \in S_i} (1 - \min\{\alpha x_i^*, 1\}) \\
&\geqslant 1 - \sum_{j \in U} \prod_{i:j \in S_i} \exp(-\alpha x_i^*) \qquad \text{(If some } x_i^* < 1\text{, the probability will be 0)} \\
&= 1 - \sum_{j \in U} \exp(-\sum_{j \in S_i} \alpha x_i^*) \\
&= 1 - \sum_{j \in U} \exp(-\alpha)
\end{aligned}$$

Therefore, we obtain

Claim 5.2.1.

$$\Pr(\text{every element covered}) \geqslant 1 - n \cdot e^{-\alpha}$$

If we set $\alpha = \ln n + \ln\ln n$, then

$$\begin{aligned}
\Pr[\varepsilon_1]\Pr[\text{every element covered}] &\geqslant 1 - n \cdot e^{-\ln n - \ln\ln n} \\
&= 1 - \frac{1}{n \ln n} \\
&\geqslant 1 - \frac{1}{\ln n}
\end{aligned}$$

We should also focus on $\mathbb{E}(w(T)) \leqslant \alpha \cdot \mathrm{OPT}$. Here we use the **Markov Inquality**:

Theorem 5.11 (Markov Inequality)**.** *For Random Variable* $X \geqslant 0$, $\Pr(X \geqslant t\mathbb{E}X) \leqslant \frac{1}{t}$

Proof.

$$\begin{aligned}
\mathbb{E}X &= \mathbb{E}[X|X \geqslant \alpha] \cdot \Pr(X \geqslant \alpha) + \mathbb{E}[X|X < \alpha] \cdot \Pr(X < \alpha) \\
&\geqslant \alpha \cdot \Pr[X \geqslant \alpha]
\end{aligned}$$

Therefore, $\Pr[X \geqslant \alpha] \leqslant \frac{\mathbb{E}X}{\alpha}$ □

So

$$\begin{aligned}
\Pr[\varepsilon_2]\Pr\left[\sum_i w(S_i)y_i \geqslant (\ln n + 2\ln\ln n)\mathrm{OPT}\right] &\leqslant \frac{\ln n + \ln\ln n}{\ln n + 2\ln\ln n} \\
&= 1 - \frac{\ln\ln n}{\ln n + 2\ln\ln n}
\end{aligned}$$

So the union probablity

$$\begin{aligned}
\Pr[\varepsilon \wedge \bar{\varepsilon}_2] &\geqslant \Pr[\varepsilon_1] - \Pr[\varepsilon_2] \\
&\geqslant \frac{\ln\ln n}{\ln n + 2\ln\ln n} - \frac{1}{\ln n} \\
&\geqslant \Omega(\frac{\ln\ln n}{\ln n}) \xrightarrow{\text{boost}} 1 - \frac{1}{n^{100}}
\end{aligned}$$

Theorem 5.12. *With probability $\Omega(\frac{\ln\ln n}{\ln n})$, LP+randomized rounding returns a $(\ln n + 2\ln\ln n)$-approximation solution.*

The question is how to boost the probability. Indeed, if we independently round N times,

$$\begin{aligned}
\Pr[\exists 1 \text{ trial succeeds}] &\geqslant 1 - [1 - \Omega(\frac{\ln\ln n}{\ln n})]^N \\
&\geqslant 1 - \exp(-\Omega(\frac{\ln\ln n}{\ln n} \cdot N)) \\
&\geqslant 1 - \exp(-\Omega(n \cdot \ln n)) \geqslant 1 - e^{-n}
\end{aligned}$$

if we set $N \leftarrow (\ln n) \cdot n$ in the last step.

Apply the method to Max-coverage problem, Integer Problem is to $\max \sum_{j=1}^{n}$ such that

$$\begin{aligned}
&\sum_{i=1}^{M} x_i \leqslant k \\
&y_j \leqslant \sum_{j \in S_i} x_i, \forall j \in [n] \\
&y_j \leqslant 1, \forall j \in [n] \\
&x_i \in \{0, 1\}, \forall i \in [M]
\end{aligned}$$

So the LP relaxation is to relax $x_i \in [0, 1]$.

Repeat k times and select set i with probability $\frac{x_i^*}{k}$.

$$
\begin{aligned}
\mathbb{E}[\text{coverges}] &= \sum_{j=1}^{n} \Pr[j \text{ covered}] \\
&= \sum_{j=1}^{n} \left(1 - \left(1 - \sum_{j \in S_i} \frac{x_i^*}{k}\right)^k\right) \\
&\geqslant \sum_{j=1}^{n} \left(1 - \exp(-\sum_{j \in S_i} x_i^*)\right) \\
&\geqslant \sum_{j=1}^{n} (1 - \exp(-y_i^*)) \\
&\geqslant \alpha \sum_{j=1}^{n} y_j^* = \alpha \cdot \mathrm{LP} \geqslant \alpha \cdot \mathrm{OPT}
\end{aligned}
$$

where $\alpha = 1 - \frac{1}{e}$

The estimation is tight for this rounding if we consider the set $\mathcal{U} = \{0,1\}^k$ and $S_{i,0} = \{a \in U | a_i = 0\}$, $S_{i,1} = \{a \in U | a_i = 1\}$.

5.2.2 Integrality Gap

Instance I is a c vs. s-**Integrality Gap (IG) instance** if $\mathrm{LP}(I) \geqslant c$ and $\mathrm{OPT}(I) \leqslant s$. The **gap ratio** is $\frac{c}{s}$.

1. Large IG $\Rightarrow$ Inaccurate estimation of LP.
2. The estimation of rounding algorithm is usually

$$\text{rounding} \geqslant \cdots \geqslant \alpha \cdot \mathrm{LP} \geqslant \alpha \cdot \mathrm{OPT}$$

Since $\text{rounding} \leqslant \mathrm{OPT}$ in maximization problem, if IG is large, α can be very large and hence the approximation ratio is very bad.

For instance, consider the set-cover problem that is to minimize $\sum_{i=1}^{M} x_i \quad s.t.$

$$\sum_{j\in S_i} x_i \geqslant 1, \forall j \in U, \quad x_i \in [\{0,1\}, \forall i \in [M]]$$

relax the condition $x_i \in [0,1]$ and consider the set $\mathcal{U} = \{0,1\}^q \backslash \{0\}$ and $S_{\vec{\alpha}} = \{l \in U : l^T\alpha = 1 \pmod 2\}$ for $\alpha \in \{0,1\}^q$ with the size $M = 2^q, n = 2^q - 1$.

$$|S_{\vec{\alpha}}| = \begin{cases} 2^{q-1} & \alpha \neq 0 \\ 0 & \alpha = 0 \end{cases}$$

Claim 5.2.2. LP=2

Proof. Take $x_{\vec{\alpha}} = \frac{2}{2^q}$. Then $\sum_{\vec{\alpha}} S_{\vec{\alpha}} = 2$. And the LP constraint met where

$$\forall \vec{e} \in U, \sum_{\vec{e}\in S_{\vec{\alpha}}} \frac{2}{2^q} = 2\mathrm{Pr}_{\vec{\alpha}\in\{0,1\}^q}[\vec{e} \in S_{\vec{\alpha}}] = 2 \times \frac{1}{2} = 1$$

Certainly LP $\geqslant 2$, so LP $= 2$. $\square$

But we also have a claim about OPT:

Claim 5.2.3. OPT $\geqslant q$. So the instance is a 2 vs. q IG with ratio $\frac{q}{2} = \frac{1}{2}\log_2(n+1) = \frac{\ln(n+1)}{2\ln 2}$.

Proof. For any $S_{\vec{\alpha}_1}, \cdots, S_{\vec{\alpha}_{q-1}}$, suppose $S_{\vec{\alpha}_1} \cup \cdots \cup S_{\vec{\alpha}_{q-1}}$ is a cover of U. Then

$$\Leftrightarrow \bar{S}_{\vec{\alpha}_1} \cap \cdots \cap \bar{S}_{\vec{\alpha}_{q-1}} = \{\vec{0}\}$$
$$\Leftrightarrow \{\vec{e} \in \{0,1\}^q : e^T\vec{\alpha}_i = 0 \forall i \in [q-1]\} = \{\vec{0}\}$$

which is impossible! $\square$

I is an α-Integrality Gap instance if

$$\mathrm{LP}(I) \leqslant \frac{1}{\alpha} \cdot \mathrm{OPT}(I)$$

Then we indeed prove that α can be $\frac{1}{2}\log_2(n+1) < \ln(n+1)$ in Min-Set-Cover problem.

Take $U = \{1, 2, \cdots, n\}$. M are $C \cdot k \ln n$ sets, each of which s_i includes each $j \in U$ independently with probability $\frac{1}{k}$

Claim 5.2.4. When $C \geqslant \frac{4}{\varepsilon^2}$, the probability

$$\Pr[\mathrm{LP} \leqslant \frac{k}{1-\varepsilon}]1 - \frac{1}{n}$$

Proof. Consider $x_i = \dfrac{k}{< (1-\varepsilon)}$.

Fix $j \in [n]$.

$$\begin{aligned}
\Pr[\sum_{j \in S_i} x_i \geqslant 1] &= \Pr\left[\sum_{i=1}^{M} \mathbf{1}[j \in S_i] \geqslant \frac{(1-\varepsilon)M}{k}\right] \\
&\geqslant 1 - \left[\frac{e^{-\varepsilon}}{(1-\varepsilon)^{1-\varepsilon}}\right]^{\frac{M}{k}} \\
&= 1 - \exp\left((-\varepsilon - (1-\varepsilon)\ln(1-\varepsilon)) \cdot \frac{M}{k}\right) \\
&\geqslant 1 - \exp(-\frac{\varepsilon^2}{2} \cdot \frac{M}{k}) \\
&\geqslant 1 - \exp(-2\ln n)
\end{aligned}$$

Here we use the Chernoff bound with a high relation of central limit theorem.

Theorem 5.13 (Chernoff Bound). $X_1, X_2, \cdots, X_n \in [0,1]$ *a.s. and* $\mathbb{E}X_i = p_i$. *Let*

$X = X_1 + \cdots + X_n$, $\mathbb{E}X = \mu$. *For any* $\delta > 0$

$$\begin{cases} \Pr[X \geqslant (1+\delta)\mu] \leqslant \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu} \\ \Pr[X \leqslant (1-\delta)\mu] \leqslant \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu} \end{cases}$$

□

Claim 5.2.5. For $k \geqslant \frac{2}{\varepsilon}$ and $n = n(k, \varepsilon, C)$ large enough, we have

$$\Pr[\mathrm{OPT} \geqslant (1-\varepsilon)k \ln n] \geqslant 0.99$$

Proof. Let $z = (1-\varepsilon)k \ln n$.

$\mathrm{OPT} > z \Leftarrow \forall \mathcal{S} \in \binom{[M]}{z}$, $\mathcal{S}$ doesn't cover U. We consider probability of the latter case.

Now fix $\mathcal{S} \in \binom{[M]}{z}$, $\Pr[\mathcal{S}$ cover $U]$ is actually

$$\begin{aligned} \Pr[\mathcal{S} \text{ cover } U] &= \Pr[\forall j \in U, \exists S_i \in \mathcal{S}, j \in S_i] \\ &= \Pr[\exists S_i \in \mathcal{S}, 1 \in S_i]^n \\ &= \left(1 - (1 - \frac{1}{k})^z\right)^n \\ &\leqslant \exp(-n(1-\frac{1}{k})^z) \\ &\overset{k \geqslant 2}{\leqslant} \exp(-n \exp(-(1-\varepsilon)(1+\frac{1}{k}) \ln n)) \\ &\overset{k \geqslant \frac{2}{\varepsilon}}{\leqslant} \exp(-n \exp(-(1-\frac{\varepsilon}{2}) \ln n)) \\ &= \exp(-n \cdot n^{-(1-\frac{\varepsilon}{2})}) \\ &= \exp(-n^{\frac{\varepsilon}{2}}) \end{aligned}$$

So

$$\begin{aligned}\Pr[\exists \mathcal{S} \in \binom{[M]}{z}, \mathcal{S} \text{ cover } U] &\leqslant \binom{M}{z} \cdot \exp(-n^{\frac{\varepsilon}{2}}) \\ &\leqslant \left(\frac{C \cdot e}{1-\varepsilon}\right)^{(1-\varepsilon)k \ln n} \cdot \exp(-n^{\frac{\varepsilon}{2}}) \\ &< \exp(-n^{\frac{\varepsilon}{4}}) \\ &< 0.01\end{aligned}$$

as n large enough. Here we end the proof □

So using Randomized construction, α can approach $(1-\varepsilon)\ln n$ for any ε.

5.3 Hardness of Approximation

5.3.1 P,NP classes

For $\mathcal{L} \in \{0,1\}^*$ is the $0-1$ encoding, a problem is the set of some $0-1$ encoding and a decision problem is to decide whether $\mathcal{L}$ belongs to it.

For instance, $\mathcal{L}_k$ is the set of all ($0-1$ encoding) of set cover instances where U can be covered by k sets. We define

$$P = \{\mathcal{L} : \mathcal{L} \text{ can be poly-time decided by a (deterministic) Turing machine}\}$$

$$NP = \{\mathcal{L} : \mathcal{L} \text{ can be poly-time decided by a non-deterministic Turing machine}\}$$

NP problems are all problems that can be "verified" in poly-time. Explicitly,

for input instance $x \in \{0,1\}^*$, the prover is based on x, providing a "proof" $y \in \{0,1\}^*$ that $|y| \leqslant \mathrm{poly}(|x|)$, however, the verifier is a poly-time algorithm that accepts x, y and outputs YES/NO.

In other words, $\mathcal{L} \in$ NP $\Leftrightarrow \exists$ a prover-verifier system such that

- Completeness: $\forall x \in \mathcal{L}$, $\exists$ proof y *s.t.* verifier returns YES in poly-time.

- Soundness: $\forall x \notin \mathcal{L}$, $\forall$ proof y, verifier returns NO in poly-time.

The equivalence is because we actually can "guess" the proof y in a non-deterministic TM.

If P=NP, then if we can verify proof in poly-time, we can also construct it in poly-time. There isn't innovation anymore!

NP-complete: $\mathcal{L}$ is NPC if

1) $\mathcal{L} \in$ NP

2) $\forall\ \mathcal{L}' \in$ NP, $\mathcal{L}' \geqslant_p \mathcal{L}$ *i.e.* $\mathcal{L}'$ can be reduced to $\mathcal{L}$ in poly-time.

Equivalently, if some NPC problems can be solved in poly-time, then P=NP.

If only 2) in the definition of NPC holds, then it is a **NP-hard** problem.

We define the **polynomial reduction** $M \leqslant_p L$ if

$\exists$ poly-time algorithm A such that $\forall x \in \{0,1\}^*$,

- (completeness) $x \in M \Rightarrow A(x) \in L$.

- (soundness) $x \notin M \Rightarrow A(x) \notin L$

Observed that if M is NP-Complete and $M \leqslant_p \mathcal{L}$, then L is NP-Hard.

Theorem 5.14 (Cock-Levin)**.** *3-SAT is NP-Complete.*

Proof. $\forall L \in NP$, need to show $L \leqslant_p$ 3-SAT.

Let A be the poly-time verifier (DTM) for L.

Now we consider the original DFA, which needs start, process and after, denoted as (s, p, α)

For time t, the tape can be

$$t_{-M}^{(t)}, \cdots, t_M^{(t)}, \alpha^{(t)}, S^{(t)}, p^{(t)}$$

where the transition function is

$$
\begin{aligned}
t_i^{(\tau)} &= g_i(t_i^{(\tau-1)}, \alpha^{(\tau-1)}, s^{(\tau-1)}, p^{(\tau-1)}) \\
s^{(\tau)} &= h_(\alpha^{(\tau-1)}, s^{(\tau-1)}, p^{(\tau-1)}) \\
p^{(\tau)} &= \cdots \\
\alpha^{(\tau)} &= \cdots
\end{aligned}
$$

which is a compose of bool function. So any DFA process can be converted to a 3-SAT instance.

$\square$

Theorem 5.15 (Max-Coverage)**.** *Deciding whether Max-Coverage=100% is NP-Complete.*

Proof. We divide it into two parts:

1. Max-coverage=1 is NP.

2. 3-SAT $\leqslant_p$ Max-coverage=1.

Consider any 3-SAT instance I. We have variables $x_1, \cdots, x_n$ and clauses $c_1, \cdots, c_m$.

Denote $U = \{x_1, \cdots, x_n, c_1, \cdots, c_m\}$ and sets $S_1, S_2, \cdots, S_n, S_{n+1}, \cdots, S_{2n}$. For $i = 1, 2, \cdots, n$,

$$S_i = \{x_i\} \cup \{c_j : c_j \text{ contains } x_i\}$$

$$S_{n+i} = \{x_i\} \cup \{c_j : c_j \text{contains} \bar{x}_i\}$$

Let $k = n$.

Completeness: If I satisfiable, then $\exists \sigma : \{x_i\} \rightarrow \{0, 1\}$, choose

$$\begin{cases} S_i & \text{if } \sigma(x_i) = 1 \\ S_{n+i} & \text{if } \sigma(x_i) = 0 \end{cases}$$

Soundness: If J is YES, for I, let $\sigma(x_i) = \begin{cases} 1 & \text{if } S_i \text{ chosen} \\ 0 & \text{if } S_{i+n} \text{ chosen} \end{cases}$. $\square$

Now we want to consider the approximation problem.

1 vs. 1 Max-Coverage is NP-H but what if decide the gap-version s vs. c.

Observation If we could prove c vs. s M-C is NP-H for $c > s$. Then $\frac{s}{c}$ approximation M-C problem is NP-H.

Theorem 5.16 (PCP theorem)**.** $\exists \varepsilon$ *s.t.* $\mathrm{Max} - 3 - \mathrm{SAT}_{1,1-\varepsilon}$ *is NP-Hard.*

We give an introduction for PCPs.

Definition 5.17 (Probability Checkable Proofs)**.** Verifier: input instance x and proof y.

Reads x, compute a (joint) distribution D over the locations in y, and a Boolean function.

Sample $i, j, k, f \sim D$.

output YES iff $f(y_i, y_j, y_k) = 1$.

- (completeness) If x is YES, then $\exists\, y$ *s.t.* $\Pr[\text{Verifier accepts}] \geqslant c$.
- (soundness) If x is NO, then $\forall y$, 4 $\Pr[\text{Verifier accepts}] \leqslant s$.

Then PCP theorem is equivalent to

Theorem 5.18 (PCP theorem)**.** $\exists \varepsilon > 0$ *s.t. every NP problem has a PCP system with* $c = 1, s = 1 - \varepsilon$.

Definition 5.19. $\mathrm{PCP}_{c,s}[r,q]$ denotes set of languarges that admits a PCP system with c,s,r,q parameters. Explicitly,

- Prover reads input, outputs poly-length proof with unbounded computational prover.
- Verifier in poly-time reads input and r random bits, (deterministically by input and random bits) computes q locations in the proof, reads the q bits in the proof and decides YES/NO.
- The systems satisfies completeness and soundness:
  - (Completeness) Input is YES instance $\Rightarrow \exists$ prover$\Pr$[Verifier accepts] $\geqslant c$.
  - (Soundness) Input is NO instance $\Rightarrow \forall$ prover$\Pr$[Verifier accepts] $\leqslant s$.

Observation 5.20.

- $\mathrm{PCP}_{1,\frac{1}{2}}[0,0] = P$.
- $\mathrm{PCP}_{1,\frac{1}{2}}[0,\mathrm{poly(n)}] = NP$.
- $\mathrm{PCP}_{1,\frac{1}{2}}[O(\log(n)),O(1)] \leqslant NP$

For the final observation, we actually can construct a Verifier to enumerate all possible random bits in $\{0,1\}^r$ to return YES if there is some possibility larger than c.

Indeed, PCP theorem is actually,

Theorem 5.21 (PCP theorem)**.**

$$\mathrm{PCP}_{1,\varepsilon}[O(\log n),O(1)] = \mathrm{PCP}_{1,\frac{1}{2}}[O(\log n),O(1)] = NP$$

Proposition 5.22. *PCP theorem* $\Leftrightarrow \exists s < 1, \mathrm{Gap} - 3\mathrm{MAXSAT}_{1,s}$ *is NP-Hard.*

Proof. " $\Rightarrow$ ": Our goal is to prove $3\mathrm{SAT} \leqslant_p \mathrm{Gap} - 3\mathrm{MAXSAT}_{1,s}$, *i.e.* given a 3-SAT instance ϕ, we can construct an instance Φ in poly-time such that $\mathrm{OPT}(\phi) = 1 \Rightarrow \mathrm{OPT}(\Phi) = 1$.

$3\mathrm{SAT} = \mathrm{GAP} - 3\mathrm{MAXSAT}_{1,\frac{1}{m}}$ is NP-hard. By PCP theorem, $\exists$ a prover-verifier system for 3-SAT that with $c = 1, s = \frac{1}{2}, r = O(\log n), q = O(1)$

Prover provides proof $\vec{x} \in \{0,1\}^N$.

Given ϕ, $\forall \vec{\tau} \in \{0,1\}^r$, verifier computes

$$l_1, \cdots, l_q \in \{1, 2, \cdots, N\}$$

$$f : \{0,1\}^q \to \{0,1\}$$

find $3 - CNF\ g$ over $q + q \cdot 2^q$ variables $\{z_{\vec{\tau}}\}$ and $q \cdot 2^q$ clauses $c_{\vec{\tau}}$ such that $f(\vec{y}) = 1$ iff $\exists \vec{z} \in \{0,1\}^{q \cdot 2^q}, g(\vec{y}, \vec{z}) = 1$.

Construct Φ with variables $\{x_1, x_2, \cdots, x_N\} \cup \bigcup_\tau z_\tau$ and clauses $\bigwedge_\tau c_\tau$.

Completeness: If $\exists \vec{x}$ such that $\Pr_{\vec{\tau}}[\text{Verifier accepts}] = 1$. Then $\forall \vec{\tau}$, $\exists z_{\vec{\tau}}$ such that $c_{\vec{\tau}} = 1$.

Soundness: $\forall \vec{x}, \Pr_{\vec{\tau}}[\text{Verifier accepts}] \leqslant \frac{1}{2}$. Consider a solution σ to Φ. Let $T = \{\vec{\tau} : \text{ Verifier rejects } \sigma(X) \text{ under } \vec{\tau}\}$. Then $T \geqslant \frac{1}{2} \cdot 2^r$.

$\forall \vec{\tau}$, σ doesn't satisfy all clauses in $C_{\vec{\tau}}$ so the number of unsatisfied clauses $\geqslant |T| = \frac{1}{2} \cdot 2^r$.

$$\mathrm{val}(\sigma : \Phi) \leqslant 1 - \frac{|T|}{2^r \cdot q \cdot 2^q} = 1 - \frac{\frac{1}{2} \cdot 2^r}{2^r \cdot q \cdot 2^q} = 1 - \frac{1}{2q \cdot 2^q}$$

" $\Leftarrow$ ": $\forall$ NP language $\mathcal{L} \leqslant_p \mathrm{GAP} - 3\mathrm{SAT}_{1,s}$. Then

$$\mathcal{L} \in \mathrm{PCP}_{1,s}[O(\log n), 3] \leqslant \mathrm{PCP}_{1,\frac{1}{2}}[O(\log(n), O(1))]$$

$\square$

Theorem 5.23. *$\forall \varepsilon > 0$, $\mathrm{GAP} - 3\mathrm{SAT}_{1,\frac{7}{8}+\varepsilon}$ is NP-Hard.*

Corollary 5.24. *$\forall \varepsilon > 0$, $(\frac{7}{8} + \varepsilon)$-approximation Max3SAT is NP-Hard.*

The corollary is equivalent to $\forall \varepsilon > 0, \mathrm{Gap3SAT}_{1-\varepsilon,\frac{7}{8}+\varepsilon}$ is NP-Hard.

Remark 5.25. It implies that it is hard to find an algorithm better than random algorithm. It also shows that perfect completeness is sometimes very hard.

5.4 Label-Cover Games

To prove the theorem, we need to consider a constraint Graph $G = (U, V, E)$, which is a bipartite graph.

Prover is a function $\sigma : U \to [K], V \to [L]$.

Constraints: For each $e = (u, v) \in E$, $\pi_e : [L] \to [K]$.

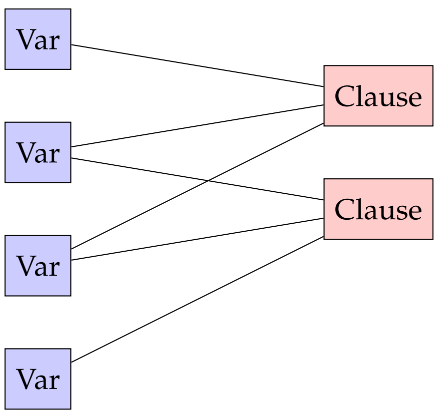
Verifier: Uniformaly sample $e = (u, v) \in E$, accepts only if $\pi_e(\sigma(v)) = \sigma(u)$.

The system is also called "2-Prover-1Verifier Game" or "Projection Game".

More generally, $\pi_e \subset [K] \times [L]$.

Claim 5.4.1. PCP Theorem implies that $\exists \delta > 0$, $\mathrm{Gap} - \mathrm{LC}_{1,1-\delta}^{(K,L)=(2,7)}$ is NP-Hard.

Proof. Reduce from $\mathrm{Gap} - 3\mathrm{SAT}_{1,1-\varepsilon}$.



Variables x_i have $\sigma(x_i) \in \{0,1\}$ and Clauses $c_i = x^1_{j_i} \wedge x^2_{j_i} \wedge x^3_{j_i}$ have $\sigma(c_i) \in [7]$ to represent the state of c_i. $\square$

5.4.1 Paralled Repetition

Given $H = (G(U,V,E), K, L, \{\pi_e\})$.

$$H^{\otimes t} = \left(G^{\otimes t}, K^t, L^t, \{\pi_{e_1,e_2,\cdots,e_t}\}\right)$$

where

$$G^{\otimes t} = \{(u_1, u_2, \cdots, u_t), (v_1, v_2, \cdots, v_T)\}$$

$$\pi_{(u_1,v_1),\cdots,(u_t,v_t)} = \left\{((\alpha_1, \cdots, \alpha_t), (\beta_1, \cdots, \beta_t)) : (\alpha_i, \beta_i) \in \pi_{(u_i,v_i)}\right\}$$

It is easy to check that if H is a projection game, $H^{\otimes t}$ is also a projection game.

We wonder whether the following theorem holds

Theorem 5.26 (not quite true)**.**

$$\mathrm{OPT}(H^{\otimes t}) \leqslant \mathrm{OPT}(H)^t$$

There is a counter-example for $U = \{u_1, u_2\}, V = \{v_1, v_2\}, K = L = 4$ and G is fully connected. $[K], [L] \leftrightarrow \{u, v\} \times \{1, 2\}$

$\pi_{(u_i,v_j)} = \{((u,i),(u,i)), ((v,j),(v,j))\}$

Clearly, $\mathrm{OPT}(H) = \frac{1}{2}$.

However, $\mathrm{OPT}(H^{\otimes 2}) = \frac{1}{2}$.

Let $\sigma((u_{i_1}, u_{i_2})) = (((u, i_1), (v, i_1)))$, $\sigma((v_{j_2}, v_{j_2})) = ((u, j_1), (v, j_2))$.

So verifier accepts if $i_1 = j_2$.

However, the following theorem holds:

Theorem 5.27. *Suppose H has alphabet size less than k and $\mathrm{OPT}(H) \leqslant 1-\delta$. Then*

$$\mathrm{OPT}(H^{\otimes t}) \leqslant \exp(-\Omega(\frac{\delta^3 t}{\log K}))$$

Corollary 5.28. *$\forall \delta > 0$, $\exists\, K, L$ such that $\mathrm{GAP}-\mathrm{LC}(K,L)_{1,\delta}$ is NP-Hard.*

Proof of Theorem 5.23. □

Index

List of Theorems