Algorithm Design and Analysis

Vinsong

December 17, 2025

Abstract

The lecture note of 2025 Fall Algorithm Design and Analysis by professor 呂學一. 希望我可以活著度過這學期∼∼∼∼

Contents

Chapter 1

Graph Theory: Path and Shortest Path Problems

Lecture 8

Definition 1.0.1 (path). Let G be an n-vertex m-weighted directed graph with weight w (which can be positive, negative or zero). The weight of a path P of G is defined as 30 Oct. 14:20

$$w(P) = \sum_{xy \in E(P)} w(xy)$$

For vertices u and v of G, we call a path of G from u to v a **uv-path** of G.

Definition 1.0.2 (distance). For vertices u and v of G, the **distance** from u to v in G, denoted by $d_G(u, v)$, is defined as

$$d_G(u,v) = \begin{cases} \infty & \text{if there is no uv-path in } G \\ w(P) & \forall Q \in \text{uv-paths},\ w(P) \leq w(Q) \\ -\infty & \text{otherwise} \end{cases}$$

Comment (1). 這個 P 就叫做 **shortest** uv**-path**

Comment (2). 真正的 path 是不允許重複經過點的。這裡定義的 path 其實在真正的 graph theory 裡面叫做 **walk**。$V(P)$, $E(P)$ 都是 multiset。

1.1 Single-Source Shortest Path Problem

Problem 1.1.1 (Single-Source Distance Problem). Given

- Input: a directed graph G with edge weights $w : E(G) \to \mathbb{R}$ and a **source** vertex $r \in V(G)$.
- Output: $d_G(r, v)$ for all vertices $v \in V(G)$.

Note. 我們可以用下面這個問題可以規約 (reduce) 到上面的問題

Problem 1.1.2 (Single-Source Shortest Path Problem). Given

- Input: a directed graph G with edge weights $w : E(G) \to \mathbb{R}$ and a **source** vertex $r \in V(G)$.
- Output: a (shortest-path) tree T of G rooted at r such that if G contains a shortest rv-path of G, then rv-path of T is a shortest rv-path of G.

所以我們應該要解決 Single-Source Distance Problem，先做兩個假設 $m = \Omega(n)$

Comment (1). 我們可以用 DFS 先處理掉 r 無法到達的點，所以可以假設

$$d_G(r, v) < \infty, \ \forall v \in V(G)$$

Comment (2). r 固定，簡寫 $d(v) := d_G(r, v)$

1.1.1 Bellman-Ford Algorithm

Algorithm. For each vertex $v \in V(G)$, we use $d[v]$ to estimate $d(v)$.

- Initialization

$$d[i] = \begin{cases} 0 & i = r \\ \infty & \text{otherwise} \end{cases}$$

- Repeat $n - 1$ times relaxation step: for each edge $uv \in E(G)$, 更新

$$d[v] = \min\{d[v], \ d[u] + w(uv)\}$$

- Relaxation 結束後，For each edge $uv \in E(G)$, if $d[v] > d[u] + w(uv)$, then

$$d[v] = -\infty$$

- For each vertex $v \in V(G)$, 如果他可以被任何 u which $d[u] = -\infty$ reach（DFS $O(m + n)$），則

$$d[v] = -\infty$$

Note. The running time is $O(mn)$.

Proof. 我們先做一些觀察

Observation (1). 在 $n - 1$ 次 relaxation 後，$\forall v \in V(G)$, $d[v] \geq d(v)$，永遠不會小於真正的 $d(v)$.

Observation (2). If P is a shortest rs-path of G for some $s \in V(G)$,

- 在這條 rs-path 上的每一個 v of P，這條 rv-path 也會是 shortest rv-path of G.
- 對於每一個 edge uv of P, if 在先前的 relaxation step 後，會有

$$d[u] = d(u)$$

在這次 relaxation step 後，會有

$$d[v] = d(v)$$

現在我們來證明 Bellman-Ford Algorithm 的正確性。我們分三種情況討論，Case 1 已經在之前就證明可以用 DFS 處理掉了。

- Case 2: $d(v) \neq -\infty$. Let P be a shortest rv-path of G. for each vertex u_j of P, where $j = 0, 1, \ldots, |V(P)| - 1$, $u_0 = r$ and $u_{|V(P)|-1} = v$. 根據我們的 Obs.2，我們知道在第 i 次 relaxation step 後

$$d[u_j] = d(u_j) \quad \forall j \in \{0, \cdots, \min(i, |V(P)| - 1)\}$$

- Case 3: $d(v) = -\infty$: 因為到達不了的點一經被處理掉了，因此必定存在 rv-path P of G, which contain a cycle C such that $w(C) < 0$. 所以我們可以

Claim. At the end of n-th round,

$$d[u] = -\infty \quad \forall u \in V(C)$$

By $u \in V(P)$, we have $d[v] = -\infty$ at the end.

To prove this claim, we assume for contradiction. The n-th of round 並沒有成功更新 $d[u]$ for all $u \in V(C)$. 我們嘗試對每一個邊做 relaxation 都應該失敗。Thus,

$$d[x] + w(xy) \geq d[y] \quad \forall xy \in E(C)$$

把 C 上所有這種 inequality 全部加起來，我們會得到

$$\sum_{xy \in E(C)} w(xy) \geq 0$$

contradiction to $w(C) < 0$.

Hence, Bellman-Ford Algorithm is correct. ■

1.1.2 Lawler's Algorithm

Remark. 針對 Acyclic Graph 的 Algorithm，Since the input graph has no cycle, it has no negative cycle.

Algorithm. 只需要 One Relaxation Step 就可以了

- 用 $O(m+n)$ 做一次 Topological Sort on the input directed acyclic graph G to get a topological order u_i, $\forall i \in \{1, \ldots, n\}$

- Initialization

$$d[u_i] = \begin{cases} 0 & i = 0 (d[r] = 0) \\ \infty & \text{otherwise} \end{cases}$$

- For i from 1 to n, we do relaxation step for each $u_i v$

$$d[v] = \min\{d[v],\ d[u_i] + w(u_i v)\}$$

Note. The running time is $O(m + n)$.

Proof. 因為這是一個 DAG，所以做完一次 Topological Sort 後，我們就可以知道每個點的 outgoing edge 順序，也就是知道他們在 shortest path 裡面的順序，因此即便我們不知道這條 $u_i v$-path 在哪裡，但我們可以保證在我們處理到 u_i 的時候，$d[u_i]$ 已經是正確的 $d(u_i)$ 了。因此我們只需要做一次 relaxation step 就可以了。 ■

1.1.3 Dijkstra's algorithm

Remark. 本質上是針對 Non-Negative Weighted Graph 的 Greedy Algorithm，可以做更近一步的簡化，Since the input graph has no negative edge, it has no negative cycle.

Algorithm. One round of estimate improvement suffices, although we cannot rely on topological sort (since G may contain cycles).

- Initialization

$$d[v] = \begin{cases} 0 & v = r \\ \infty & \text{otherwise} \end{cases}$$

- 有 n 次 iteration，每次 iteration 從還沒被處理過的點中選出 $d[u]$ 最小的點 u，並對 u 的每一個 outgoing edge uv 做 relaxation

Note. The running time is $O(m + n \log n)$.

Proof. Let's prove the correctness by contradiction.

1° Let v be the first vertex selected in S with $d[v] \neq d(v)$. We have

$$d[v] > d(v)$$

接下來我們考慮 v 被加入 S 的情況

2° Let P be the shortest rv-path of G.

3° Let xy be an arbitrary edge of P such that $x \in S$ and $y \notin S$. (必定存在因為 $r \in S$ 而 $v \notin S$)

4° 因為我們正要處理 v，所以 xy is already processed, we have

$$d[y] = d(y),\ y \neq v$$

5° Since G are nonnegative and y precedes v in P, we have

$$d[y] \leq d(v)$$

6° By 1°, 5°, and 4° we have

$$d[v] > d(v) \geq d[y] = d(y)$$

which contradicts the selection of v. 我們就不該選到 v 因為他並不是最小的那個，還有一個 y 更小

Hence, Dijkstra's algorithm is correct. ■

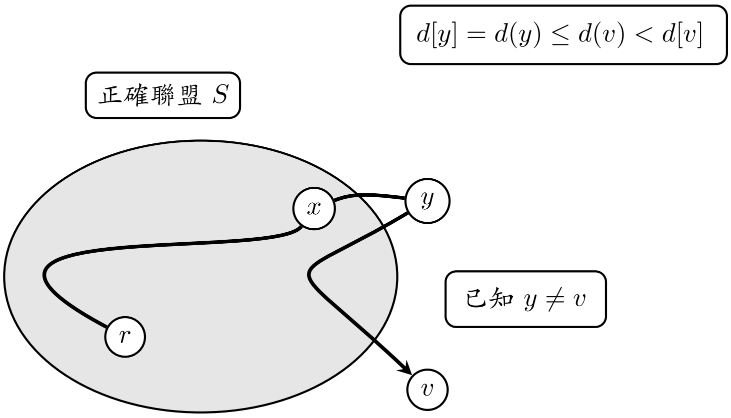


Figure 1.1: Dijkstra's Algorithm Correctness

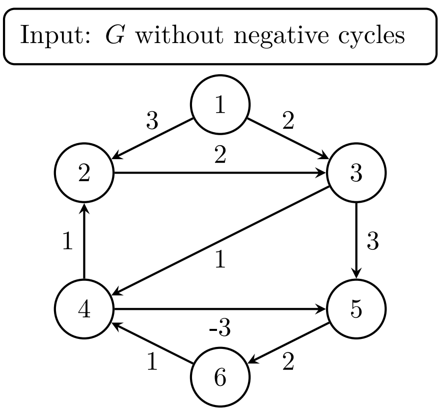
Lecture 9

13 Nov. 14:20

1.2 All-Pairs Shortest Path Problem

Problem 1.2.1 (All-Pairs Distance Problem). Given

- Input: an edge-weighted directed graph G with $V(G) = \{1, 2, \cdots, n\}$ edge weights $w : E(G) \to \mathbb{R}^+$, without negative cycles.
- Output: $d_G(i, j)$ for all $i, j \in V(G)$.



$\Longrightarrow$

Output: $d_G(i, j)$ for all $i, j \in V(G)$

d	1	2	3	4	5	6
1	0	3	2	3	0	2
2	∞	0	2	1	-2	0
3	∞	∞	0	1	-2	0
4	∞	∞	∞	0	-3	-1
5	∞	∞	∞	∞	0	2
6	∞	∞	0	∞	∞	0

Figure 1.2: All-Pairs Distance Problem Example

Algorithm (Naive Solution). Sloving the single-source shortest path problem for each vertex using Dijkstra, Lawler, or Bellman-Ford algorithm.

1.2.1 A Naive DP Solution

Definition 1.2.1. Let $w_k(i,j)$ be the length of the shortest ij-path in G having at most k edges. It will be ∞ if no such path exists.

$$\begin{cases} w_1(i,j) & = w(ij) \\ w_{n-1}(i,j) & = d_G(i,j) \end{cases}$$

Algorithm. Use the recurrence relation for $w_k(i,j)$ is

$$\begin{cases} w_1(i,j) & = w(ij) \\ w_{2k}(i,j) & = \min_{1\le t\le n} (w_k(i,t) + w_k(t,j)) \end{cases}$$

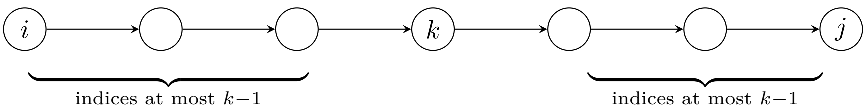
- For each (i,j,k), take $O(n)$ time to compute $w_{2k}(i,j)$ from $w_k(i,j)$.
- For each k, there are n^2 pairs of (i,j), using $O(n^3)$ time to compute all w_{2k} from all w_k.
- It take $O(\log n)$ iterations to compute $d_G = w_{n-1}$ from w_1.

Note. The running time is $O(n^3 \log n)$.

1.2.2 Floyd and Warshall's DP algorithm

Definition 1.2.2. Let $d_k(i,j)$ be the length of the shortest ij-path in G whose intermediate vertices are at most k. It will be ∞ if no such path exists.

$$\begin{cases} d_0(i,j) & = w(ij) \\ d_n(i,j) & = d_G(i,j) \end{cases}$$



Algorithm (Floyd and Warshall's DP Algorithm). Using the recurrence relation for $d_k(i,j)$ is

$$\begin{cases} d_0(i,j) & = w(ij) \\ d_k(i,j) & = \min\{d_{k-1}(i,j),\ d_{k-1}(i,k) + d_{k-1}(k,j)\} \end{cases}$$

- For each (i,j,k), take $O(1)$ time to compute $d_k(i,j)$ from $d_{k-1}(i,j)$.
- For each k, there are n^2 pairs of (i,j), using $O(n^2)$ time to compute all d_k from all d_{k-1}.
- It take n iterations to compute $d_G = d_n$ from d_0.

Note. The running time is $O(n^3)$.

1.2.3 Johnson's Reweighting Technique

Algorithm (Naive Solution with Dijkstra). 如果我們可以拿到一個 nonnegative edge-weight 的 graph，我們就可以簡單地用 Dijkstra's algorithm 來解 All-Pairs Shortest Path Problem

- For each vertex i of G, run Dijkstra's algorithm + Quake heap in $O(m + n \log n)$ time to compute $d_G(i, j)$ for all $j \in V(G)$.

Note. The running time is $O(nm + n^2 \log n)$.

所以我們需要一個方法把有負邊權的 graph 轉換成 nonnegative edge-weight 的 graph，Reweighting w into $\hat{w}$ such that

- $\hat{w}$ is nonnegative
- If $\hat{w}$ is the reweighted shortest ij-path, then the original shortest ij-path is w.

Algorithm (Johnson's Reweighting Technique). Following these steps:

- Assign a weight $h(i)$ to each vertex i of G.
- Let
$$\hat{w}(ij) = w(ij) + h(i) - h(j)$$
- Then for any ij-path P, we have
$$\hat{w}(P) = w(P) + h(i) - h(j)$$

Remark. P is a shortest ij-path in G with respect to $\hat{w}$ if and only if it is a shortest ij-path in G with respect to w.

$(4 + 5 + 3 - 1) + 3 - 2 = (8 + 2 + 1 + 1)$ $\qquad$ $(7 + 2 + 3) + 3 - 2 = (0 + 2 + 1)$

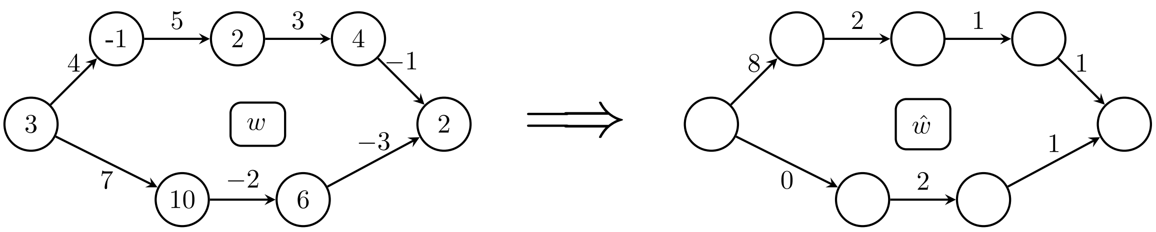


Figure 1.3: Reweighting Example

挑戰是在哪裡找 $h(i)$ 使得 $\hat{w}$ nonnegative？如果有了，我們就可以用 Dijkstra's algorithm 來解 All-Pairs Shortest Path Problem。

Algorithm (Johnson's Technique: Finding $h(i)$)**.** Following these steps:

- Let graph H be obtained by adding a new vertex s to G and adding an edge s_i of weight 0 for each vertex i of G.

  > **Note.** H has no negative cycle iff G has no negative cycle.

- Let $h(i)$ be the distance from s to i in H, i.e.

$$h(i) = d_H(s, i)$$

- The $d_H(s, i)$ can be computed using Bellman-Ford algorithm in $O(m + n)$ time.

Proof. To see that $\hat{w}$ is nonnegative, observe the Figure 4.4.

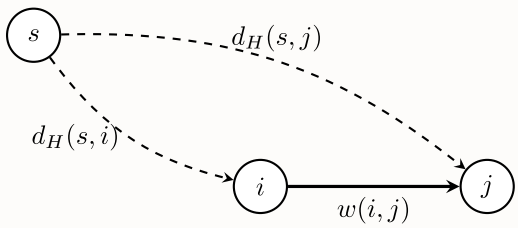


Figure 1.4: Proof of correctness of Johnson's Reweighting Technique

By observation, we have

$$\begin{aligned} \hat{w}(ij) &= w(ij) + h(i) - h(j) \\ &\geq \underbrace{(w(ij) + d_H(s, i))}_{\text{shortest } sj\text{-path which contain } i} - \underbrace{d_H(s, j)}_{\text{shortest } sj\text{-path}} && \text{(Triangle Inequality)} \\ &= 0 \end{aligned}$$

■

Recall. Using Naive Solution with

- General edge weights: Bellman-Ford algorithm in $O(mn^2)$ time, which can be $\Theta(n^4)$ when $m = \Theta(n^2)$.
- Acyclic edge weights: Lawler's algorithm in $O(mn + n^2)$ time.
- Nonnegative edge weights: Dijkstra's algorithm in $O(mn + n^2 \log n)$ time.

Using Floyd–Marshall ' s DP algorithm in for general edge weights in $O(n^3)$ time.

Algorithm (Johnson's algorithm)**.** Using Johnson's Reweighting Technique + Dijkstra's algorithm

- Obtain $h(i)$ for all vertex i using Bellman-Ford algorithm in $O(mn)$ time, and get $\hat{w}$ from w in $O(m)$ time.
- For each vertex i of G, run Dijkstra's algorithm + Quake heap in
$$O(m + n \log n)$$
time on G with edge weights $\hat{w}$ to compute $d_{\hat{G}}(i, j)$ for all $j \in V(G)$. Then obtain a shortest-paths tree of $G(\hat{G})$ rooted at i.
- Compute $d_G(i, j)$ for all $j \in V(G)$ using
$$d_G(i, j) = d_{\hat{G}}(i, j) + h(j) - h(i)$$
in $O(n^2)$ time.

Note. The running time is $O(mn + n^2 \log n)$.

Chapter 2

Graph Theory: Maximum Flow Problem

Problem 2.0.1 (Maximum Flow Problem). Give

- Input: A directed graph G with edge capacities

$$c : E(G) \to \mathbb{R}^+$$

  , and two distinct vertices $s, t \in V(G)$ called **source** and **sink** respectively.

- Output: A "*st*-flow" with maximum "(flow) value".

Comment. 在這個問題下我們允許 multiple/parallel edges 不需要合併成 simple network

Definition. Here are some definitions related to flows:

Definition 2.0.1 (*st*-flow). A *st*-flow is a function

$$f : E(G) \to \mathbb{R}^+ \cup \{0\}$$

that satisfies the following two conditions:

- Capacity constraint:

$$f(e) \leq c(e) \quad \forall e \in E(G)$$

- Conservation law:

$$\sum_{uv \in E(G)} f(uv) = \sum_{vu \in E(G)} f(vu) \quad \forall v \in V(G) \setminus \{s, t\}$$

Definition 2.0.2 (Flow value). The flow value of a flow f is defined as

$$|f| = \sum_{sv \in E(G)} f(sv) - \sum_{us \in E(G)} f(us)$$

2.1 Ford–Fulkerson's Algorithm

Intuition. We can reduce the Maximum Flow Problem into reachability problem for a sequence of **residual graphs** R.

Definition 2.1.1 (Residual Graph)**.** The residual graph $R(f)$ with respect to a flow f of G with $V(G) = V(R(f))$ is defined as follows for each $uv \in E(G)$:

- If $f(uv) < c(uv)$, then $R(f)$ contains an edge uv with capacity

$$c_{R(f)}(uv) = c(uv) - f(uv)$$

- If $f(uv) > 0$, then $R(f)$ contains a reverse edge vu with capacity

$$c_{R(f)}(vu) = f(uv)$$

Comment (1)**.** $R(f)$ 跟 G 一樣，所有 $c(uv)$ 都會是正的，不會是 0 或負的。

Comment (2)**.** G 最多讓 $flow$ 增加 2 倍，因為每條邊 uv 在 $R(f)$ 裡面最多會有兩條邊：uv 和 vu，只要兩個條件都達成。

Lemma 2.1.1. For any st-flow f in G, we have the following properties:

- If $d_{R(f)} = \infty$, then f is a maximum st-flow in G.
- If $d_{R(f)} < \infty$, and g is an st-flow in $R(f)$, then $f + g$ remains an st-flow in G, where

$$(f + g)(uv) = f(uv) + g(uv) - g(vu), \quad \forall uv \in E(G)$$

Note. 這裡的 $g(uv), g(vu)$ 都是由原始的 uv-edge 產生的，因此原始圖若有 vu edge 必須分開處理，不能混在上面兩個式子裡面。

Proof. Let f' be the maximum st-flow in G, but not f. We defined h as follows:

$$h(uv) = f(uv) - f'(uv), \quad \forall uv \in E(G)$$

Since f and f' are both st-flows in G, we have conservation law for f and f', so h satisfies conservation law as well.

$$\sum_{uv \in E(G)} h(uv) = \sum_{vu \in E(G)} h(vu) \quad \forall v \in V(G) \setminus \{s, t\}$$

Now consider some vertex $x, y, z \in V(G) \setminus \{s, t\}$. If $h(xy) > 0$, because h satisfies conservation law, there must exist some $h(yz) > 0$. Continuing this process, we can find a path P,

$$P = s \to v_1 \cdots \to v_k \to t \quad \text{such that } h(v_i v_{i+1}) > 0 \quad \forall i = 0, 1, \cdots, k$$

If $h(uv) > 0$, we have

$$f(uv) > f'(uv) \geq 0 \tag{1}$$

we know $f'(uv)$ can not exceed $c(uv)$, so

$$f'(uv) \leq c(uv) \tag{2}$$

by (1) and (2), we have

$$f(uv) < c(uv)$$

which means

$$c_{R(f)}(uv) = c(uv) - f(uv) > 0$$

Therefore, all edges in $R(f)$ along path P have positive capacities. Which is a st-path in $R(f)$, contradicting the assumption that $d_{R(f)} = \infty$.

Now we start to prove the second property. We need to show that $f+g$ satisfies capacity constraint and conservation law.

- Capacity constraint: For any $uv \in E(G)$, we havee some constraint:
  - $g(uv) < c_{R(f)}(uv) = c(uv) - f(uv) \leq c(uv)$
  - $g(vu) \leq c_{R(f)}(vu) = f(uv)$

  to maximize $(f+g)(uv)$, we set $g(uv)$ to its maximum and $g(vu)$ to its minimum, so we have

  $$(f+g)(uv) = f(uv) + g(uv) - g(vu) \leq f(uv) + (c(uv) - f(uv)) - 0 = c(uv)$$

- Conservation law: For any $v \in V(G) \setminus \{s,t\}$, we have

$$\begin{aligned}
\sum_{uv \in E(G)} (f+g)(uv) &= \sum_{uv \in E(G)} (f(uv) + g(uv) - g(vu)) \\
&= \sum_{uv \in E(G)} f(uv) + \sum_{uv \in E(G)} g(uv) - \sum_{uv \in E(G)} g(vu) \\
&= \sum_{uv \in E(G)} f(uv) + 0 && \text{(by conservation law of } g\text{)} \\
&= \sum_{vu \in E(G)} f(vu) && \text{(by conservation law of } f\text{)} \\
&= \sum_{vu \in E(G)} (f(vu) + g(vu) - g(uv)) && \text{(by conservation law of } g\text{)} \\
&= \sum_{vu \in E(G)} (f+g)(vu)
\end{aligned}$$

■

Algorithm 2.1: Ford-Fulkerson Algorithm

Input: A flow network $G = (V, E)$ with capacity $c(u, v)$; source s; sink t.

Output: A maximum flow f.

Initialize $f(u, v) \leftarrow 0$ for all $(u, v) \in E$

Compute residual capacity

$$c_{R(f)}(uv) = \begin{cases} c(uv) - f(uv) & \text{if } f(uv) < c(uv) \\ f(uv) & \text{if } f(uv) > 0 \end{cases}$$

while $\exists$ *st − augmenting path P in R(f)* **do**

  Obtain an st-path P of $R(f)$, let $q = \min_{uv \in P} c_{R(f)}(u, v)$

  Obtain a st-flow g of $R(f)$ by setting

$$g(uv) = \begin{cases} q & \text{if } uv \in P \\ 0 & \text{otherwise} \end{cases}$$

  Update flow $f \leftarrow f + g$

end

return f

correctness. We separately prove three things:

- Initialization: f is a valid flow in G with value 0.
- According to Lemma 5.1.1（關鍵觀察）：In every round g is a valid flow in $R(f)$, so $f + g$ is a valid st-flow in G.
- Termination: When the algorithm terminates, $d_{R(f)}(s, t) = \infty$, so by Lemma 5.1.1, f is a maximum st-flow in G.

Proof complete. ■

Definition 2.1.2 (augmenting path)**.** In Ford-Fulkerson algorithm, obtain a st-path P of $R(f)$, let $q = \min_{uv \in P} c_{R(f)}(u, v)$. The path P is called an **augmenting path** with respect to flow f.

Definition 2.1.3 (saturating flow)**.** In Ford-Fulkerson algorithm, obtain a st-flow g of $R(f)$ by setting

$$g(uv) = \begin{cases} q & \text{if } uv \in P \\ 0 & \text{otherwise} \end{cases}$$

. The flow g is called a **saturating flow** with corresponding to P.

Lecture 10

20 Nov. 14:20

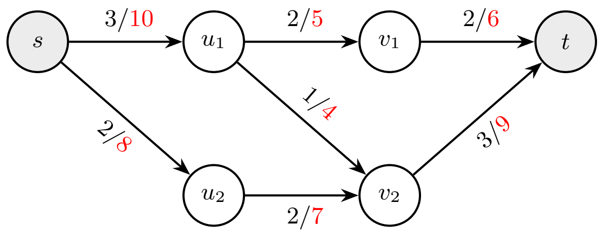


Figure 2.1: Example of Maxflow Problem

考慮 Ford-Fulkerson algorithm 的複雜度分析，如果所有容量都是整數，則每次增廣至少增加 1 單位的流量，而 m 條管線的容量總和是 $C = \sum_{e\in E} c(e)$，因此最多增廣 C 次，每次找增廣路徑花費 $O(m)$ 的時間，總複雜度是

$$T(m, C) = O(C) \cdot O(m) = O(mC)$$

但，這個算是「多項式時間」(polynomial time) 演算法嗎？還是是「指數時間」(exponential time) 演算法？

Remark. Complexity is according to the input size of an instance.

Example. For an $n \times n$ matrix multiplication problem, the input size is $N = \Theta(n^2)$ (set all the number is of size $O(1)$).

The complexity is

- Linear-time if $T(N) = O(N) = O(n^2)$.
- Polynomial-time if $T(N) = O(N)^{O(1)} = O(n)^{O(1)}$.
- Exponential-time if $T(N) = O(1)^N = O(1)^{n^2}$ or more.

Definition 2.1.4 (Complexity of Linear/Quadratic/Polynomial-time Algorithms)**.** For any instance I，define its input size as a non-negative integer function

$$N = \text{size}(I),$$

其中 N 表示描述輸入實例所需的位元數或其他適當的度量方式。令 $T(N)$ 為某演算法在輸入大小為 N 時的最壞情況執行時間。我們對時間複雜度作如下分類：

- Linear-time algorithm：若 $T(N) = O(N)$。
- Quadratic-time algorithm：若 $T(N) = O(N^2)$。
- Polynomial-time algorithm：若存在常數 k 使得 $T(N) = O(N)^{O(1)}$。
- Exponential-time algorithm：若存在常數 $c > 1$ 使得 $T(N) = O(1)^N$ or more。

Example. For a prime testing problem, given an integer N as input.

We have to consider the input size.

- If input size is $N = O(1)$, then the method of checking all integers from 2 to $\sqrt{N}$ is

$$O(\sqrt{N}) = O(1)$$

  which is a linear-time algorithm.

- If the size of N is not constrainted, then the input size is $\Theta(\log N)$ (bits to represent N). The time of checking all integers from 2 to $\lfloor \sqrt{N} \rfloor$ is $\Omega(\sqrt{N})$

  - According to

$$(\log N)^{O(1)} = o(\sqrt{N}) = o(N^{1/2})$$

    this algorithm is not polynomial-time.

  - According to

$$O(N^{1/2}) = O(1)^{O(\log N)}$$

    this algorithm is singly exponential-time.

Note. 所以根據 maximum flow problem 的 input size 是

- 若 $C = O(1)$，input size 是 $O(m) \cdot O(1) = O(m)$，所需要花的時間是 $O(mC) = O(m)$，演算法是 linear-time。

- 若無大小限制，input size 是 $O(m) \cdot O(\log C) = O(m \log C)$，所需要花的時間是

$$O(mC) \neq O(m \log C)^{O(1)}$$

  因此演算法不是 polynomial-time。

Remark. Ford-Fulkerson algorithm 還會出現無限迴圈的問題，例如下圖的圖，設計出非整數的容量，會導致無限迴圈。Ford-Fulkerson 的論文就有提出其他反例。

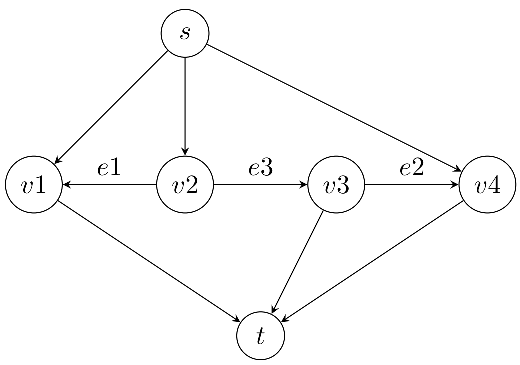


Figure 2.2: A graph that may cause infinite loop in Ford-Fulkerson algorithm

2.2 Edmonds-Karp Algorithm

這是史上第一個被證明是 polynomial-time 的 maximum flow algorithm。

Theorem 2.2.1. If one make sure that the augmenting st-path in $R(f)$ is always the shortest path from s to t (in terms of number of edges), then the Ford-Fulkerson algorithm runs in $O(m^2 n)$ time.

Comment (1). 在保證每次 augmenting path 都是 shortest path 的前提下，Edmonds-Karp algorithm 保證在 mn round 內結束。

Comment. 無需假設 G 所有容量都是整數。

We need two lemmas to prove the above theorem.

Notation. $d^*_f(s,u)$ is the shortest distance from s to u in unweighted version of $R(f)$.

Lemma 2.2.1 (現邊觀察). If in $R(f+g)$ exists uv edge which is not in $R(f)$, then

$$d^*_{R(f+g)}(s,u) = d^*_{R(f)}(s,v) + 1$$

Proof. Let P be the shortest augmenting st-path of $R(f)$. If $R(f)$ don't have the uv edge, P can't go through uv edge. If it does not go through vu edge, too. Then $g(uv) = g(vu) = 0$. We get

$$(f+g)(uv) = f(uv) + g(uv) - g(vu) = f(uv)$$
$$(f+g)(vu) = f(vu) + g(vu) - g(uv) = f(vu)$$

Then we get $R(f+g) = R(f)$ which can not have uv edge, which is contradiction. So P must go through vu edge. Then we have

$$d^*_{f+g}(s,u) = d^*_f(s,v) + 1$$

Proof complete. ∎

Lemma 2.2.2 (遞增觀察). Let P be the shortest augmenting st-path of $R(f)$. Let g be the saturating flow for $R(f)$ correspond to P. Then for any $v \in V(G)$ we have

$$d^*_{R(f+g)}(s,v) \geq d^*_{R(f)}(s,v)$$

Proof. Assume for contradiction that there exists some $v \in V(G)$ such that

$$d^*_{R(f+g)}(s,v) < d^*_{R(f)}(s,v) \tag{1}$$

Thus, $d^*_{f+g}(s,v) \neq \infty$. Let v be such vertex with the smallest $d^*_{f+g}(s,v)$. We know $v \neq s$ since $d^*_{f+g}(s,s) = d^*_f(s,s) = 0$. Let Q be the unweighted shortest sv-path in $R(f+g)$ (u could be s). Let uv be the last edge of Q. We have

$$d^*_{R(f)}(s,u) \leq d^*_{R(f+g)}(s,u) \tag{2}$$

If $uv \subseteq R(f)$, then equation (2), and $uv \in Q$ imply

$$d^*_{R(f)}(s,v) \leq d^*_{R(f)}(s,u) + 1 \leq d^*_{R(f+g)}(s,u) + 1 = d^*_{R(f+g)}(s,v)$$

which is contradiction to equation (1).
If $uv \not\subseteq R(f)$, by 現邊觀察 ($uv \not\subseteq R(f)$ and $uv \subseteq Q \subseteq R(f+g)$), then equation (2) and $uv \subseteq Q$ imply

$$d^*_{R(f)}(s,v) = d^*_{R(f)}(s,u) - 1 \leq d^*_{R(f+g)}(s,u) - 1 = d^*_{R(f+g)}(s,v) - 2$$

which is contradiction to equation (1), too. ■

Now let compute the time complexity of Edmonds-Karp algorithm.

Time Complexity. Since each augmenting path can be found by BFS in $O(m)$ time, we only need to proof that algorithm halt in $O(mn)$ rounds.

Claim. Every round "saturates" at least one edge in the shortest st-path P found in that round, which is $O(m)$ edges in $G \cup G^r$, causing them to be removed from the residual graph of the next round. Thus, we can just show that each $uv \subseteq G \cup G^r$ being removed $O(n)$ times in total.

Suppose that an edge uv of $G \cup G^r$ is not in $R(f)$

- appears in $R(f+g)$ and
- removed in $R(f+g+\cdots+g'+h)$ for the first time after $R(f+g)$

where h is the saturating flow of $R(f+g+\cdots+g'+h)$ corresponding to the shortest augmenting st-path in $R(f+g+\cdots+g')$ saturates uv. Thus, $uv \in E(P)$. We have

$$\begin{aligned} d^*_{R(f)}(s,v) &= d^*_{R(f)}(s,u) - 1 && \text{by 現邊觀察} \\ &\leq d^*_{R(f+g+\cdots+g')}(s,v) - 1 && \text{by 遞增觀察} \\ &= d^*_{R(f+g+\cdots+g'+h)}(s,u) - 2 && uv \in E(P) \end{aligned}$$

Since $d^*_H(s,v) \in \{0, 1, \cdots, n-1, \infty\}$ for any residual graph H, uv can at most appear and disappear $O(n)$ times in the residual graphs throughout the algorithm. Thus, the algorithm halts in $O(mn)$ rounds. and thus run in $O(m^2 n)$ time. ■

Edmonds-Karp 的分析是用邊來看：

- 每條邊 uv 只要「出現之後消失」一次，就會讓 $d^*_R(s,u)$ 增加,
- 每個節點 u 的 $d^*_R(s,u)$ 只有 $O(n)$ 個可能的值.

所以每個邊「出現之後消失」$O(n)$ 次，每回合至少消失一條邊，一共 $O(mn)$ 回合。如果用節點 u 來看，能不能根據一樣的證明，得知

- 節點 u 的任一 outgoing edge「出現之後消失」一次，則 $d^*_R(s,u)$ 的值會增加，
- 節點 u 的 $d^*_R(s,u)$ 只有 $O(n)$ 個可能的值，

所以 u 的所有 outgoing edges 全部只能「出現之後消失」$O(n)$ 次，而每回至少讓一條邊消失，所以總共只有 $O(n^2)$ 回合？

答案是錯誤的，

因為節點 u 的任一 outgoing edge『出現之後消失』一次，則 $d^*_{R(f)}(s,u)$ 的值會增加至少 2 是錯的，邊消失一次，$d^*_{R(f)}(s,u)$ 不一定會增加，因為可能有其他替代路徑，不一定是 shortest path

2.3 Bipartite Matching via Maximum Flow

Problem 2.3.1 (Maximum matching of bipartite graph). Given

- Input: An undirected "bipartite" graph G
- Output: A matching $M \subseteq E(G)$ with maximum $|M|$.

Definition 2.3.1 (Bipartite Graph). A graph G is **bipartite** if there are disjoint vertex subsets U, V of G with $U \cup V = V(G)$ such that every edge of G has one endpoint in U and the other in V.

Definition 2.3.2 (matching). A edge subset $M \subseteq E(G)$ is a **matching** of G if $M = \emptyset$ or the minimal subgraph H of G with $E(H) = M$ which has $\max_{e \in E(H)} \deg(H) = 1$.

We can reduce the maximum matching problem of bipartite graph to the maximum flow problem by the below construction.
Let $G(s,t)$ be the unit-capacity graph obtained from G by adding

- new source vertex s with edges su for all $u \in U$
- new sink vertex t with edges vt for all $v \in V$

Observation. G has a maximum matching with k edges if and only if $G(s,t)$ has a maximum flow with value k.

We separately prove the two directions.

- ($\Rightarrow$) Let M be a maximum matching of G we have to make sure it follow capacity constraints and Conservation law.
  - Capacity constraints: 因為是一對一 matching，所以每個 $u \in U$ 和 $v \in V$ 最多只有一條 flow 1 的邊經過。
  - Conservation law: For each vertex $u \in U$, at most one edge su has flow 1, and for each vertex $v \in V$, at most one edge vt has flow 1, and for each $uv \in M$, at most exists one flow from $U \to V$.
- ($\Leftarrow$) Define $M = \{uv \in E(G) : f(uv) = 1\}$. We have to make sure M is a matching. And by the proposition below, $|M| = k$.

Proposition 2.3.1. If $G(s,t)$ has a maximum flow with value k, then $G(s,t)$ has an "integral" flow with value k. Given that each edge of $G(s,t)$ has unit capacity, the set of edges in the middle part of $G(s,t)$ with non-zero flow forms a matching of G with k edges. Due to f is closed under $\{+,-\}$.

Chapter 3

Computational Geometry

3.1 Nearest Point Pair

Problem 3.1.1 (Nearest Point Pair Problem). Given

- Input: A set P of n points in the plane.
- Output: A pair of points $p, q \in P$ such that the Euclidean distance $d(p, q)$ is minimized.

Comment. Not losing generality, we can assume that

$$|P| = 2^k, \quad k \in \mathbb{N}$$

A naive algorithm is to compute the distance of each pair of points, then solve a 老大問題，which takes $O(n^2)$ time.

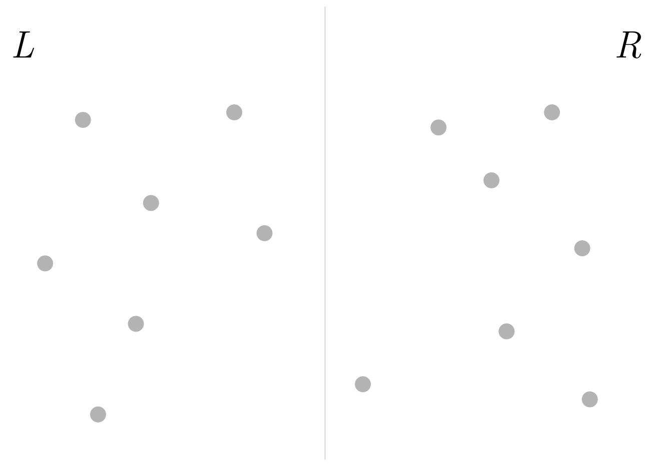


Figure 3.1: Divide-and-Conquer Strategy for Nearest Point Pair Problem

We can use divide-and-conquer strategy to solve this problem follow the above graph.

Algorithm. Pre sort P by y-coordinate as P_y, which would take

$$O(n \log n)$$

Then, use divide-and-conquer strategy to solve the problem

1° Spend $O(n)$ time to split P into two halves P_L and P_R,

$$P_L = \{p \mid p \in P : x(p) \leq x_{\text{mid}}\}, \quad P_R = \{p \mid p \in P : x(p) > x_{\text{mid}}\}$$

with x_{mid} being the median x-coordinate of points in P. We can use the minimum-selection algorithm to find the median in $O(n)$ time.

2° Output a closest pair among the following three pairs:

- The closest pair in P_L (recursively solved).
- The closest pair in P_R (recursively solved).
- The closest pair (p, q) with $p \in P_L$ and $q \in P_R$, which can be solved in $O(n)$ time as below.

Note. Follow the graph below, the node will exist in the vertical strip with width $2d$ centered at the dividing line. For each point p in the strip, we only need to check at most 8 points in the box. So the complexity is

$$O(n) \times O(1) = O(n)$$

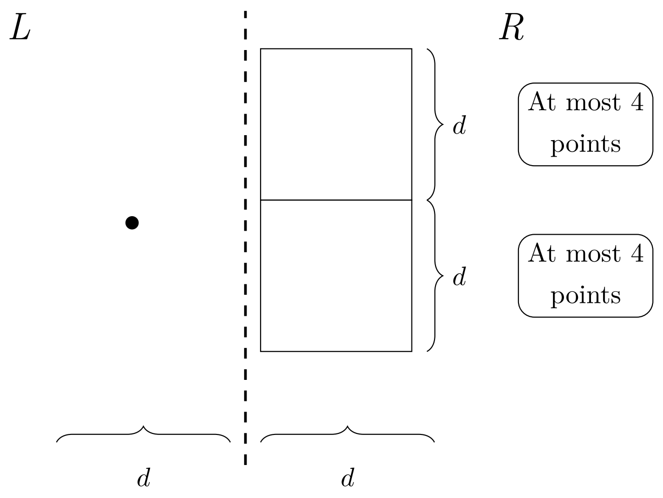


Figure 3.2: Finding Closest Pair Across the Dividing Line

把所有距離分割線不超過 d 的點挑出來，根據演算法一開始的準備動作，我們只要花 $O(n)$ 時間就可以把這些點根據它們的 y 座標排好。我們稱這個 sorted list 為 M^*。

- M^* 裡面在 L 中的點的形成 L^*, 在 R 中的點形成 R^*.

再花 $O(n)$ 時間就能夠整理出資料結構使得每個點 $p \in M^*$ 都能在 $O(1)$ 時間查得：

- 在 L^* 中 y 座標與 p 的 y 座標最近的上下（按照 M^* 的順序）各一個節點。

- 在 R^* 中 y 座標與 p 的 y 座標最近的上下（按照 M^* 的順序）各一個節點。

For each point p in L^*,

- check four points of R^* above p in M^*.
- check four points of R^* below p in M^*.

只需要在 M^* 中由上而下走過每個點，每個點在 M^* 的順序中上下各取八個點一定會找到在對面那一側的 $2d \times d$ 的方格內跟自己最近的點（如果格子內有點的話）。

Note. By master theorem, the time complexity of this divide-and-conquer algorithm is

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

which gives

$$T(n) = O(n \log n)$$

Therefore, the overall time complexity including the initial sorting step is

$$O(n \log n) + O(n \log n) = O(n \log n)$$

3.2 Convex Hull

Problem 3.2.1 (Convex Hull)**.** Given

- Input: A set P of n points in the plane.
- Output: The convex polygon with a minimum perimeter enclosing all points in P.

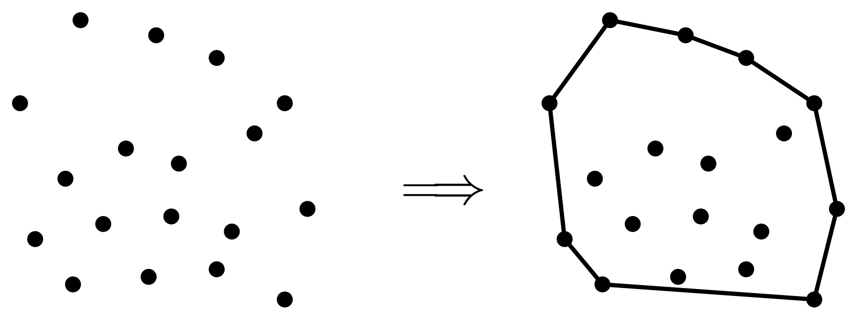


Figure 3.3: Convex Hull Example

A naive algorithm is to check the $p \in P$ with minimum x-coordinate, which must be a vertex on the convex hull. Then, chose the next vertex with maximum slope to the current vertex. Then, rotate the whole graph until we return to the starting vertex. This algorithm takes $O(n^2)$ time.

Algorithm. Now we introduce a new method with $O(n \log n)$ time complexity.

1° Find the point p^* with the lowest x-coordinate in $O(n)$ time (老大問題)

2° Sort all the points in P by the slope of the line segment p^*p for each $p \in P \setminus \{p^*\}$ in $O(n \log n)$ time.

3° Chose the initial convex hull H to be the triangle formed by p^* and the first two points in the sorted list.

4° For each remaining point p in the sorted list, do:

- Beacuse the one we chose the points based on the slope from p^*, the point p must be outside the current convex hull H.
- Move along the boundary of H in clockwise direction from the initial point, and remove all vertices q of H such that the line segment qp makes a left turn with respect to the edge of H incident to q.
- Do this until we reach a vertex r of H such that the line segment rp makes a right turn with respect to the edge of H incident to r.
- Add the edge rp^* and p^*p to H to form the new convex hull.

This step takes $O(n)$ time in total because each point is added and removed at most once.

Therefore, the overall time complexity is

$$O(n) + O(n \log n) + O(n) = O(n \log n)$$

■

3.2.1 Application of Convex Hull: Farthest Point Pair

Problem 3.2.2 (Farthest Point Pair Problem). Given

- Input: A set P of n points in the plane.
- Output: A pair of points $p, q \in P$ such that the Euclidean distance $d(p, q)$ is maximized.

We can use the convex hull to solve this problem by doing the bitonic boss problem.

Problem 3.2.3 (Bitonic Boss Problem). Given

- Input: A bitonic sequence $A[1],\ A[2],\ \cdots,\ A[n]$ of distinct positive integers.
- Output: the index i with $1 \leq i \leq n$ such that

$$A[i] = \max_{1 \leq j \leq n} A[j]$$

在 convex hull 上的點依照極角排序後，距離最大的兩個點一定會是 bitonic boss problem 的解。因此，我們可以先用

$$O(n \log n)$$

時間求出 convex hull，然後在 convex hull 上的點用 bitonic boss problem 找出距離最大的兩個點，which takes

$$O(h)$$

time, where h is the number of points on the convex hull. Therefore, the overall time complexity is

$$O(n \log n) + O(h) = O(n \log n)$$

Chapter 4

B-tree, 23-tree, 234-tree, RB-tree

Lecture 10

4.1 RB-tree

28 Nov. 14:20

Definition 4.1.1 (RB-tree). RB-tree (Red-Black tree) — binary search tree with additional properties:

- For black node,
  (1) Root is black.
  (2) All paths from leaf to root contain the same number of black nodes.
- For red node,
  (1) No red node has a red child.
- BST property holds.
  (1) Every non-leaf node have two children.
  (2) The whole tree are increasing in in-order traversal.

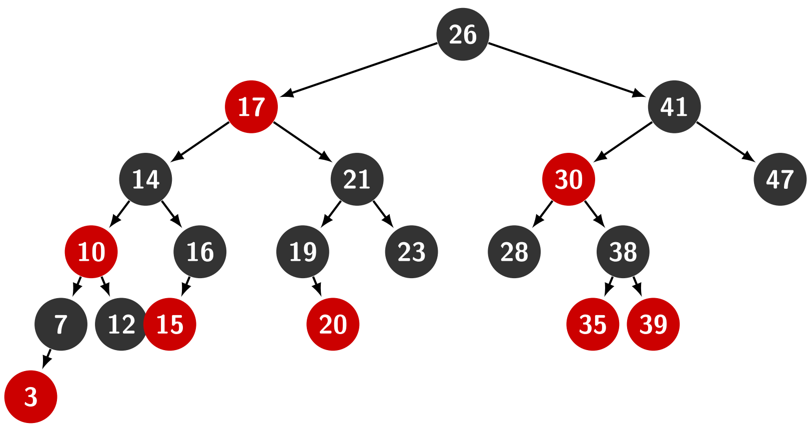


Figure 4.1: An example of RB-tree

Note. The height of RB-tree with n nodes is $O(\log n)$, which is use to guarantee the time complexity of search, insert and delete operation is $O(\log n)$ with balancing the tree.

4.2 Balance Tree (B-tree)

Definition 4.2.1 (B-tree)**.** B-tree of order t is a tree with the following properties:

- Every node has at most t children.
- Every non-leaf node (except root) has at least $\lceil t/2 \rceil$ children.
- A non-leaf node with k children contains $k-1$ keys.
- All leaves appear on the same level.
- The keys in each node are sorted in increasing order.

Definition 4.2.2 (23-tree)**.** 23-tree is a B-tree of order 3.

Definition 4.2.3 (234-tree)**.** 234-tree is a B-tree of order 4.

Note. The height of B-tree with n nodes is $O(\log n)$, which is use to guarantee the time complexity of search, insert and delete operation is $O(\log n)$ with balancing the tree.

Remark. We can convert RB-tree to 234-tree by merging red nodes to their black parent nodes in $O(n)$, which means they are equivalent in terms of search, insert and delete operations.

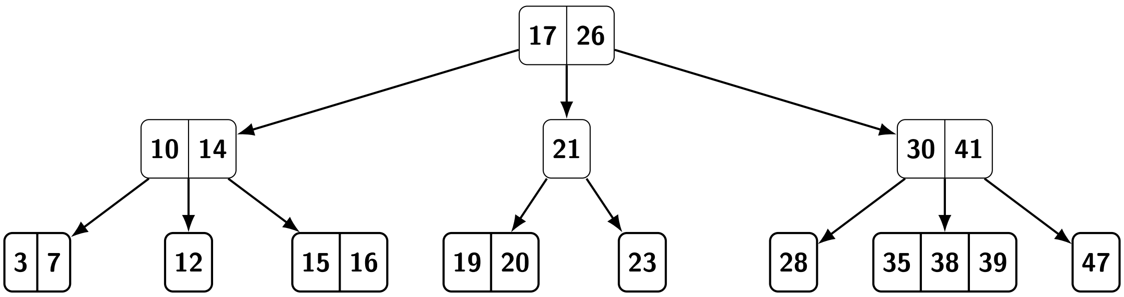


Figure 4.2: Convert RB-tree to 234-tree

Note. To convert RB-tree to 23-tree, we can follow the same procedure as converting to 234-tree, but we need to ensure we always combine the right child red node with its black parent node.

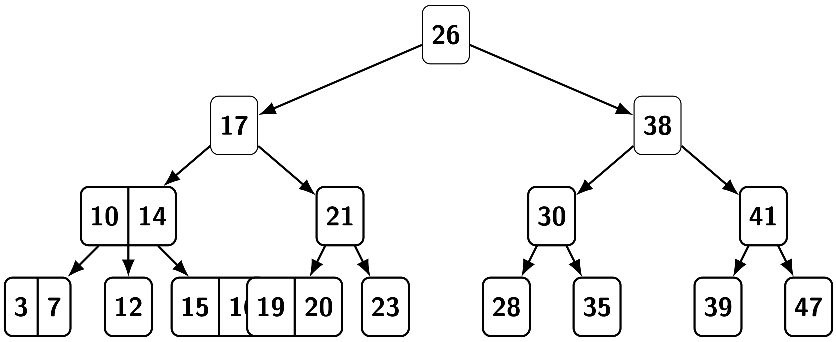


Figure 4.3: Convert RB-tree to 23-tree

Chapter 5

Hashing, Randomized Algorithm & Communication Complexity

5.1 Hashing

Question. Suppose that we want to represent an initially empty set S of at most n numbers such that each element of S is a positive integer no more than n. We would like to support the following operations on S for any given integer i with $1 \leq i \leq n$:

- membership: determining whether i belongs to S.
- insertion: inserting i into S.
- deletion: deleting i from S.

To implement the above operations alll in $O(1)$ time, we can use an array of size n (let's call it 'A') initialized to all zeros. Each index of the array corresponds to an integer from 1 to n.

Intuition. We can use a sorted array or a balanced search tree to store the elements of S. However, both approaches would require $O(\log n)$ time for membership, insertion, and deletion operations. But it may be possible to achieve $O(1)$ time.

Algorithm. Keep all the elements of S in a binary array $C[1 \ldots, n]$. Specifically, for each $j = 1, \ldots, n$, we maintain the condition that $C[j] = 1$ holds if and only if S contains element j.

- Space: $O(n)$.
- Time:
  - creation and initialization: $O(n)$.
  - membership: $O(1)$ (direct look-up).
  - insertion: $O(1)$.
  - deletion: $O(1)$.

Remark. This method still requires $O(n)$ for initialization time.

Algorithm. Maintain a dense array $D[1 \dots n]$ storing the elements contiguously, a sparse array $S[1 \dots U]$, and a size counter k. The condition that $x \in Set$ holds if and only if $1 \leq S[x] \leq k$ and $D[S[x]] = x$.

- Space: $O(n)$ (requires memory for the universe size).
- Time:
  - creation and initialization: $O(1)$ (**no need to zero out arrays**).
  - membership: $O(1)$ (via double-check logic).
  - insertion: $O(1)$.
  - deletion: $O(1)$.

If we want to store a set of n numbers are all integers in the range 1 to m where m is much larger than n, the above methods would be inefficient in terms of space. If we can have a finger printing function that maps the large universe of size m to a smaller range of size $O(n)$, we can then use the above methods to store the set efficiently.

Intuition. We want to find a function that maps a large universe of size m to a smaller range of size $O(n)$ in $O(1)$ time.

$$H : \{1, 2, \dots, m\} \to \{1, 2, \dots, k\} \quad \text{where } k = O(n)$$

where

$$\forall x \neq y, H(x) \neq H(y)$$

5.2 Randomized Algorithm & Communication Complexity

Question. Given

- Input: two $n \times n$ matrices A and B.
- Output: determine whether $A = B$

Comment (1). The time complexity is $\Theta(n^2)$ by checking each entry one by one.

Comment (2). But we are more interested in communication complexity. To minimize the communication between two parties (Alice and Bob), we can use a randomized algorithm.

Algorithm. We can use the following randomized algorithm:

- Alice and Bob agree on a vector r with n entries where each entry is chosen uniformly at random from $\{0, 1\}$.
- Compare the vectors Ar and Br.

Comment. The communication complexity of this algorithm is $O(n)$ if we don't count the initial agreement on p.

We can do some analysis on the error probability of this algorithm. If $A = B$, then $Ar = Br$ always holds. If $A \neq B$, we want to bound the probability that $Ar = Br$.

Proposition 5.2.1.

$$\Pr[Ar = Br \mid A = B] = 1 \tag{1}$$

$$\Pr[Ar = Br \mid A \neq B] \leq 0.5 \tag{2}$$

Proof. Equation (1) is trivial. For Equation (2), let $C = A - B$. Since $A \neq B$, we can show that

$$\Pr\left[Cr = 0^n \mid C \neq 0^{n\times n}\right] \leq 0.5$$

Assuming $C \neq 0^{n\times n}$, there is a row i of C whose nonzero entries are $c_{i1}, c_{i2}, \ldots, c_{ik}$ for a $k \geq 1$. Thus, it suffices to show that

$$\Pr\left[\sum_{j=1}^{k} c_{ij_k} r_{j_k} = 0 \middle| c_{ij} \neq 0 \text{ for some } j\right] \leq 0.5$$

Let r_{j_k} be the last random variable in the summation. When all the other elements of r determined, at most one choice of r_{j_k} can satisfy the equation.

$$\sum_{j=1}^{k} C_{ij_k} r_{j_k} = 0 \Rightarrow r_{j_k} = -\frac{\sum_{j=1}^{k-1} c_{ij_k} r_{j_k}}{c_{ij_k}}$$

Thus, the probability is at most 0.5. ■

If we think about the error probability is too high, we have two solutions:

Algorithm. We can repeat the above algorithm $t = O(1)$ to reduce the error probability to 2^{-t} but the communication complexity would increase to $O(tn) = O(n)$.

Algorithm. We can use the following randomized algorithm:

- Alice and Bob agree on a large prime number p (e.g., $p > n^2$).
- Alice randomly selects a vector $r \in \mathbb{Z}_p^n$ where each entry is chosen uniformly at random from $\{0, 1, \ldots, p-1\}$.
- Alice computes the vector Ar and sends it to Bob.
- Bob computes the vector Br and compares it with the received vector Ar.
- If $Ar = Br$, Bob concludes that $A = B$; otherwise, he concludes that $A \neq B$.

Chapter 6

P & NP

6.1 P-class & NP-class

Definition 6.1.1 (P-class)**.** P-class is the class of decision problems that can be solved by a deterministic Turing machine in polynomial time.

Definition 6.1.2 (NP-class)**.** NP-class is the class of decision problems for which a given solution can be verified by a deterministic Turing machine in polynomial time. Or equivalently, NP-class is the class of decision problems that can be solved by a non-deterministic Turing machine in polynomial time.

Remark. To see the definition of deterministic and non-deterministic Turing machine, please refer to the note on Introduction to the Theory of Computation (CSIE 3110) or the note I made.

Question. Given

- Input: a graph G and an integer k.
- Output: determine whether G contains a clique of size k.

Algorithm (Non-deterministic Algorithm for Clique Problem)**.** Let set S be the empty vertex set. For each vertex x of G, (non-deterministically) either insert x into S or do nothing. If $|S| \leq k$ and S is a vertex cover of G, then output yes. Otherwise, output no.

Note (Correctness of Non-deterministic Algorithm)**.** The correctness of the non-deterministic algorithm should be checked by

- If YES, then there is a computation path of the algorithm that leads to yes.
- If NO, then all computation paths of the algorithm lead to no.

Note (Time Complexity of Non-deterministic Algorithm)**.** We say that a nondeterministic algorithm N runs in polynomial time if for any input x of N, any computation of N on x takes time polynomial in the size of x.

We have more type then we can discuss about P-class and NP-class.

- P-class $\subseteq$ NP-class.
- Each NP problem can be solved by a deterministic Turing machine in exponential time.
- If $P = NP$, then all NP problems can be solved by a deterministic Turing machine in polynomial time.
- Whether $P = NP$ or not is still an open problem.
  - $P = NP$: All hardest problems in NP can be solved in polynomial time.
  - $P \neq NP$: There are some problems in NP that cannot be solved in polynomial time.

Lecture 12

有一種等價的表達 NP 的方法，就是驗證器 (verifier) 的方法。 5 Dec. 14:20

Definition 6.1.3. A decision problem L is in NP if there exists a polynomial-time algorithm V such that for every instance x,

$$x \in L \Leftrightarrow \exists\, c, |c| \leq |x|^{O(1)} \text{ such that } V(x, c) = \text{YES}$$

Here, c is called a certificate (or witness) for the instance x.

6.2 NP-Hard and NP-Complete

Definition 6.2.1 (NP-Hard). A problem H is NP-Hard if for every problem L in NP, there is a polynomial-time reduction from L to H.

$$\forall L \in \text{NP}, \;\; L \leq_p H$$

or equivalently, they are at least as hard as the all problems in NP.

Definition 6.2.2 (NP-Complete). A problem C is NP-Complete if

1. C is in NP, and
2. C is NP-Hard.

NP-Complete 就是 NP 裡面最難的問題，如果我們能找到一個多項式時間的演算法來解決 NP-Complete 問題，那麼所有 NP 的問題都可以在 polynomial time 內解決，也就是說 NP = P。

6.3 The question of P vs NP

Question (Satisfiability (SAT) Problem). Given

- **Input:** A Boolean formula ϕ in CNF (conjunctive normal form).
- **Output:** Is there a truth assignment to the variables that makes ϕ true?

Theorem 6.3.1 (Cook-Levin Theorem). The Boolean satisfiability problem (SAT) is NP-Complete.

緊接著 Richard Karp 在 1972 年提出了 21 個 NP-Complete 問題，並且證明這些問題都是 NP-Complete 的，採用的是 polynomial-time reduction to SAT

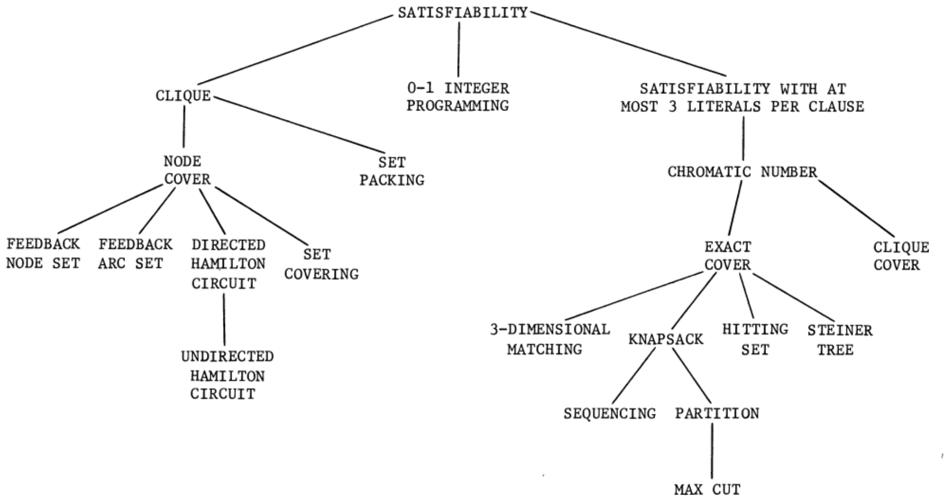


Figure 6.1: All NP-Complete Problems in Karp's 21 NP-Complete Problems

6.4 Reduction

As previously seen. Problem A can be reduced (in polynomial time) to Problem B if the following condition holds: if problem B has a polynomial-time algorithm, then so does problem A. which is denoted as

$$A \leq_p B$$

6.4.1 Hamiltonian Cycle Problem

Question (Hamiltonian Cycle Problem). Given

- **Input:** An undirected graph $G = (V, E)$.
- **Output:** Is there a simple cycle in G that visits every vertex exactly once, i.e. contains all vertices in V.

Note. 這並不是 Euler's tour，因為 Hamiltonian cycle 只需要經過每個點一次，而 Euler's tour 則是需要經過每條邊一次。

Question (Hamiltonian Path Problem). Given

- **Input:** An undirected graph $G = (V, E)$ and two vertices $s, t \in V$.
- **Output:** Whether there is a simple path in G from s to t that visits every vertex exactly once, i.e. contains all vertices in V.

我們可以將 Hamiltonian Cycle Problem reduce 到 Hamiltonian Path Problem，

Corollary 6.4.1. Hamiltonian Cycle Problem $\leq_p$ Hamiltonian Path Problem.

Proof. Let $G = (V, E)$ be an instance of Hamiltonian Cycle Problem. Suppose there is an polynomial-time algorithm B for Hamiltonian Path Problem. We can get an new algorithm A for Hamiltonian Cycle Problem as follows:

Algorithm $A(G)$:

We run $B(G, s, t)$ for each edge $st \in E(G)$. If all iteration return NO instance, then return NO; otherwise return YES.

Note. 我們必須確認三件事情

1° G 有一個包含所有頂點的 Hamiltonian cycle $A(G)$ 是否 return YES？

2° G 沒有包含所有頂點的 Hamiltonian cycle $A(G)$ 是否 return NO？

3° Algorithm A 是否在 polynomial time 內完成？

1° 如果 G 有一個包含所有頂點的 Hamiltonian cycle，則對於 cycle 上的任意一條邊 st，將 st 移除後，剩下的路徑就是一個從 s 到 t 的 Hamiltonian path。因此，當我們執行 $B(G, s, t)$ 時，會回傳 YES，進而使得 $A(G)$ 回傳 YES。

2° 如果 G 沒有包含所有頂點的 Hamiltonian cycle，則對於任意一條邊 st，將 st 移除後，剩下的路徑不可能是從 s 到 t 的 Hamiltonian path。因此，當我們執行 $B(G, s, t)$ 時，會回傳 NO，進而使得 $A(G)$ 回傳 NO。

3° Algorithm A 需要對每一條邊執行一次 $B(G, s, t)$。假設圖 G 有 m 條邊，而算法 B 在 polynomial time 內完成，即存在一個多項式函數 $p(n)$ 使得對於圖中有 n 個頂點的情況下，算法 B 的運行時間為 $O(n)^{O(1)}$。因此，算法 A 的總運行時間為 $O(m \cdot \text{poly}(n))$。由於在一般情況下，邊數 m 至多為 $\frac{n(n-1)}{2}$，因此總運行時間仍然是 polynomial time。

Proof complete. ■

因為 Hamiltonian Cycle Problem 是 NP-Complete 的 (Karp's 21 NP-Complete Problems 之一)，所以 Hamiltonian Path Problem 也是 NP-Complete 的（我們把 Hamiltonian Cycle Problem reduce 到 Hamiltonian Path Problem）。

Question (Longest Path Problem). Given

- **Input:** A graph G and two vertices $u, v \in V(G)$.
- **Output:** a longest simple uv-path in G.

這是一個 NP 問題，可以找一個 NP verifier，只要這個 verifier 是 longest path，我就可以在 polynomial time 內驗證這個 path 是否是 longest path。

Corollary 6.4.2. Hamiltonian Path Problem $\leq_p$ Longest Path Problem.

Proof. Let (G, u, v) be an instance of Hamiltonian Path Problem. Suppose there is a polynomial-time algorithm B for Longest Path Problem. We can get a new algorithm A for Hamiltonian Path Problem as follows:

Algorithm $A(G, u, v)$:

We run $B(G, u, v)$ to get a longest simple uv-path P of G. If P passes through all vertices of G, then return YES; otherwise return NO.

1° 如果 G 有一個包含所有頂點的 Hamiltonian path，則當我們執行 $B(G, u, v)$ 時，longest simple uv-path P 必定包含所有頂點，$A(G, u, v)$ 回傳 YES。

2° 如果 G 沒有包含所有頂點的 Hamiltonian path，則當我們執行 $B(G, u, v)$ 時，回傳的 longest simple uv-path P 不可能包含所有頂點，因此 $A(G, u, v)$ 回傳 NO。

3° Algorithm A 只需要執行一次 $B(G, u, v)$，假設圖 G 有 n 個頂點，而算法 B 在 polynomial time 內完成，即存在一個多項式函數 $p(n)$ 使得對於圖中有 n 個頂點的情況下，算法 B 的運行時間為 $O(n)^{O(1)}$。因此，算法 A 的總運行時間為 $O(\text{poly}(n))$，仍然是 polynomial time。

Proof complete. ■

6.4.2 Vertex Cover Problem

Question (Vertex Cover Problem). Given

- **Input:** A graph $G = (V, E)$ and an integer k.
- **Output:** Determine whether G admits a set of at most k vertices that cover all edges in $E(G)$.

Question (Independent Set Problem). Given

- **Input:** A graph $G = (V, E)$ and an integer k.
- **Output:** Determine whether G contains a subset S of $V(G)$ with $|S| \geq k$ such that two vertices in S are not adjacent in G.

我們可以將 Vertex Cover Problem reduce 到 Independent Set Problem，藉由下面的觀察：

Corollary 6.4.3. Vertex Cover Problem $\leq_p$ Independent Set Problem and vice versa.

Proof. Fir we claim that

Claim. For each subset $S \subseteq V(G)$, S is a vertex cover of G if and only if $V(G) \setminus S$ is an independent set of G.

Proof. We proof the claim as follows:

"$\Rightarrow$" 對於 S 每個 $uv \in E(G)$ 都至少有一個端點在 S 中。假設 $V(G) \setminus S$ 不是 independent set，則存在 $x, y \in V(G) \setminus S$ 使得 $xy \in E(G \setminus S)$，但 xy 的兩個端點都不在 S 中，與 S 是 vertex cover 矛盾。

"$\Leftarrow$" 對於 $V(G) \setminus S$ 中任意兩個頂點 x, y，$xy \notin E(G)$。假設 S 不是 vertex cover，則存在 $uv \in E(G \setminus S)$，使得 $u, v \in V(G) \setminus S$，與 $V(G) \setminus S$ 是 independent set 矛盾。

Let (G, k) be an instance of Independent Set Problem. Suppose there is a polynomial-time algorithm B for Vertex Cover Problem. We can get a new algorithm A for Independent Set Problem as follows:

Algorithm $A(G, k)$:

We run $B(G, |V(G)| - k)$. If B returns YES, then return YES; otherwise return NO.

1° By the claim, they are equivalent.

2° Algorithm A only needs to execute once $B(G, |V(G)| - k)$, and since B runs in polynomial time, so does A.

Proof complete. ■

6.4.3 3-SAT Problem

Definition. Here is the definition of some variants of Satisfiability Problem, suppose x is a Boolean formula in CNF.

Definition 6.4.1 (literal). A literal is either a variable x_i or its negation $\neg x_i$ or we denote as $\overline{x_i}$.

Definition 6.4.2 (clause). If $x_1, x_2, \ldots, x_k$ are literals, then

$$x_1 \vee x_2 \vee \cdots \vee x_k$$

is a clause.

Definition 6.4.3 (CNF formula). If $C_1, C_2, \ldots, C_m$ are clauses, then

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

is a CNF (Conjunctive Normal Form) formula.

Definition 6.4.4 (k-CNF formula). A k-CNF is a CNF each of whose clauses has k literals.

Question (3-SAT Problem). Given

- **Input:** A k-CNF ϕ.
- **Output:** Determine whether ϕ is satisfiable.

Definition 6.4.5. Given a 3-CNF ϕ, for each clause

$$C_i = \alpha \vee \beta \vee \gamma$$

we construct a triangle with vertices labeled α, β, and γ. For any two vertices x and $\overline{x}$ of the same variable x that are complement to each other, we add an edge $x\overline{x}$ between them. The resulting graph is denoted as $G(\phi)$.

Theorem 6.4.1. The 3-CNF ϕ is satisfiable if and only if the graph $G(\phi)$ has an independent set of size n.

Proof. We proof the theorem as follows:

"⇒" If a truth assignment T satisfies ϕ, then T satisfy at least one literal in each clause. We choose an arbitrary one statisfied literal from each clause(triangle). Since no two chosen literals are adjacent in $G(\phi)$, the set of chosen literals forms an independent set of size n in $G(\phi)$.

"⇐" Let S be an independent set of size n in $G(\phi)$. Each triangle in $G(\phi)$ contributes exactly one vertex to S. For each $\alpha \in S$, we assign $T(\alpha) = \text{true}$ and $T(\overline{\alpha}) = \text{false}$. Since no two vertices in S are adjacent, this assignment is consistent. Moreover, since S contains one vertex from each triangle, T satisfies at least one literal in each clause of ϕ. Therefore, T is a satisfying assignment for ϕ.

Proof complete. ■

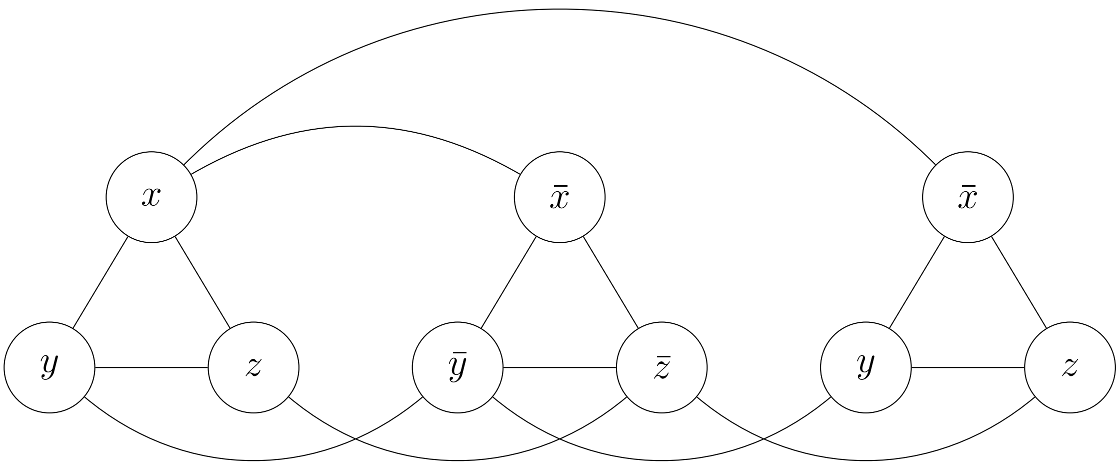


Figure 6.2: An example of constructing $G(\phi)$ from
$\phi = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$

Corollary 6.4.4. 3-SAT Problem $\leq_p$ Independent Set Problem.

Proof. Let ϕ be an instance of 3-SAT Problem. Suppose there is a polynomial-time algorithm B for Independent Set Problem. We can get a new algorithm A for 3-SAT Problem as follows:

Algorithm $A(\phi)$:

Let m be the number of clauses in ϕ. Obtain the graph $G(\phi)$. We run $B(G(\phi), m)$. If B returns YES, then return YES; otherwise return NO.

1° By the theorem, they are equivalent.

2° Constructing $G(\phi)$ from ϕ can be done in polynomial time, as it involves creating a triangle for each clause and adding edges between complementary literals. Algorithm A only needs to execute once $B(G(\phi), m)$, and since B runs in polynomial time, so does A.

Proof complete. ■

Chapter 7

Approximation

Lecture 13

7.1 Approximation Algorithm

12 Dec. 14:20

對於 NP-Hard 問題，我們通常無法在多項式時間內找到最佳解，因此我們可能可以將問題簡單化，或是使用啟發式演算法 (heuristic algorithm) 來找到近似解 (approximate solution)，犧牲掉一些的最佳性，來換取更快的運算時間。但 approximate algorithm 並不是 heuristic algorithm，而針對 approximate algorithm，我們有一些要求：

- 針對最佳化問題（optimization problems）：目標是尋找最佳解（optimal solution），而非回答「是」或「否」的 decision problems
- 必須是 polynomial-time algorithm
- 必須保證解的品質（quality of solution）：這是與 heuristic algorithm 最大的不同點

衡量 approximate algorithm 的標準是 approximation ratio，定義如下：

Definition 7.1.1 (Approximation Ratio). For a minimization problem, let C be the cost of the solution returned by an approximate algorithm, and C^* be the cost of an optimal solution. An algorithm is said to have an approximation ratio of $\rho(n)$ if for all instances of size n, it holds that

$$\frac{C}{C^*} \leq \rho(n)$$

For a maximization problem, the approximation ratio is defined as

$$\frac{C^*}{C} \leq \rho(n)$$

Remark. 通常我們希望 $\rho(n)$ 越接近 1 越好，若 $\rho = 1$ 則這是最佳演算法

Note. 也存在 additive guarantee 的定義方式

$$|C - C^*| \leq k$$

但很難達到

7.1.1 Vertex Cover Problem

Question (Vertex Cover (Optimization Version)). Given

- **Input**: A graph $G = (V, E)$.
- **Output**: A vertex cover S of G with minimum $|S|$.

Algorithm 7.1: Approximation Algorithm for Vertex Cover

```
Input: A graph G = (V, E)
Output: A vertex cover C of G
C ← ∅
E′ ← E(G)
while E′ ≠ ∅ do
    (u, v) ← an arbitrary edge from E′
    C ← C ∪ {u, v}
    E′ ← E′ \ {edges incident to u or v}
return C
```

對於每個邊 (u, v)，我們都將 u 和 v 加入 vertex cover C 中，但對於 optimal cover C^* 來說，只需要 u 或 v 即可，因此我們可以得到

$$|C| \leq 2 \cdot |C^*|$$

並且他的運行時間是 $O(|V| + |E|)$，所以這是一個 **2-approximation algorithm**

Note. 兩倍其實沒有高估，對於 worst-case 來說

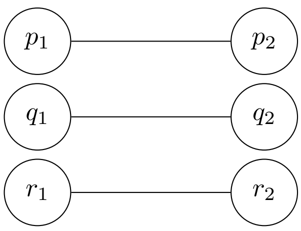


Figure 7.1: Worst-case for Approximation Algorithm for Vertex Cover

optimal cover 只需要選擇 $\{p_1, q_1, r_1\}$，但 approximation algorithm 可能會選擇 $\{p_1, p_2, q_1, q_2, r_1, r_2\}$

Remark (Upper Bound and Lower Bound). 目前已知的 vertex cover approximation algorithm 的最佳 approximation ratio 為 $2 - \epsilon$ for $\epsilon = 1$ 例如 Karakostas 的

$$2 - \Theta\left(\frac{1}{\sqrt{\log n}}\right)$$

在 $P \neq NP$ 的前提下，已知的 lower bound 為 P-time approximation ratio 最高的是

$$\sqrt{2} \approx 1.414$$

有時候 Greedy Algorithm 並不一定是好的 approximate algorithm，如果我們採用 Greedy Algorithm 來解決 vertex cover 問題，會發現他並不是一個好的 approximate algorithm，因為在 worst-case 下，他的

approximation ratio 可能會高達 $\Theta(\log n)$，並且是 tight 的分析，如果我們採用的 Greedy 策略是

Repeatedly select the vertex v with

$$\max_{v \in V} \deg(v)$$

The worst-case can be get by an Bipartite Graph ($G = (L \cup R, E)$) as follows:
Build a bipartite graph $G = (L \cup R, E)$ where

$$L : |L| = k$$

and

$$R : R = \bigcup_{i=1}^{k} R_i, \text{ where } |R_i| = \left\lfloor \frac{k}{i} \right\rfloor$$

在這個 case 底下，optimal cover 只需要選擇 L，但 Greedy Algorithm 會先選擇 R_1，接著是 R_2，依此類推，直到所有的 L 都被覆蓋為止，因此 Greedy Algorithm 最終會選擇 $|R|$ 個頂點

$$|R| = \sum_{i=1}^{k} \left\lfloor \frac{k}{i} \right\rfloor \approx k \cdot H_k = \Theta(k \log k)$$

且

$$n = |L| + |R| = \theta(k + H_k) = \theta(k \log k)$$

我們可以得知 Greedy Algorithm 的 approximation ratio 為

$$\frac{|R|}{|L|} \geq \Omega\left(\frac{H_k}{k}\right) = \Omega(\log k) = \Omega(\log n)$$

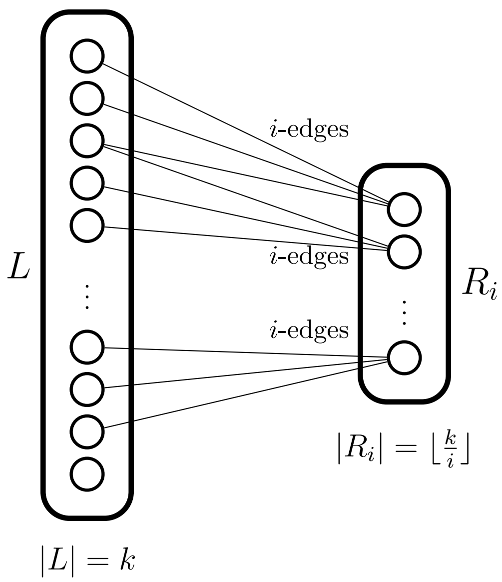


Figure 7.2: Worst-case for Greedy Algorithm for Vertex Cover: The R_i to L Connections

所以，並非所有的 Greedy Algorithm 都是好的 approximate algorithm，需要針對不同的問題設計不同的 approximate algorithm。

Note. 現在有一個新的問題，我們已經得知了 vertex cover 問題有一個 2-approximation algorithm，那對於他的 complement 問題 Maximum Independent Set 我們是否可以透過相同的 approximate algorithm，假設總共 100 個 vertex，而 optimal cover 需要 49 個 vertex，那 optimal independent set 最多只能有 51 個 vertex，而 2-approximation algorithm 最多會選擇 98 個 vertex，因此 approximation ratio 為

$$\frac{98}{49} = 2$$

但是直接根據 vertex cover 的 approximate solution 來找 independent set 的話，會得到 2 個 vertex

$$\frac{51}{2} = 25.5$$

顯然是一個極差的 approximate algorithm。

7.1.2 Traveling Salesman Problem (TSP) in Metric Space

Definition 7.1.2 (Metric Space)**.** A metric space is a set M together with a distance function $d : M \times M \to \mathbb{R}$ such that for all $x, y, z \in M$, the following properties hold:

- Non-negativity: $d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$.
- Symmetry: $d(x, y) = d(y, x)$.
- Triangle Inequality: $d(x, z) \leq d(x, y) + d(y, z)$.

Question (Metric TSP)**.** Given

- **Input**: A complete graph $G = (V, E)$ with non-negative edge weights that satisfy the triangle inequality for any three vertices $u, v, w \in V(G)$

$$w(uw) \leq w(uv) + w(vw)$$

- **Output**: A minimum-weight Hamiltonian cycle in G.

我們可以用以下方法來解決 Metric TSP 問題：

Algorithm 7.2: Approximation Algorithm for Metric TSP

Input: A complete graph $G = (V, E)$ with non-negative edge weights that satisfy the triangle inequality

Output: A Hamiltonian cycle in G

1. $T \leftarrow$ Minimum Spanning Tree of G
2. $H \leftarrow$ Preorder traversal of T
3. $C \leftarrow$ Hamiltonian cycle obtained by visiting vertices in the order of H. Skipping repeated vertices
4. **return** C

對於 Minimum Spanning Tree T，因為 MST 是連接所有 vertex 的最小權重樹，而 Hamiltonian cycle 只要去掉其中一條邊就可以變成一棵樹，並且是一個 spanning tree，所以我們可以得到

$$w(T) \leq w(C^* \setminus \{e\}) \leq w(C^*)$$

而 Preorder Traversal H 的 total weight 是

$$w(H) = 2 \cdot w(T)$$

而 Hamiltonian cycle C 是透過跳過 H 重複的 vertex 得到的，因此根據 triangle inequality，我們可以得到

$$|w(C)| \leq |w(H)| = 2 \cdot |w(T)| \leq 2 \cdot |w(C^*)|$$

所以這是一個 2-approximation algorithm
還有一個更好的 approximate algorithm 是 Christofides Algorithm，可以達到 1.5-approximation ratio

Algorithm 7.3: Christofides Algorithm for Metric TSP

Input: A complete graph $G = (V, E)$ with non-negative edge weights that satisfy the triangle inequality

Output: A Hamiltonian cycle in G

$T \leftarrow$ Minimum Spanning Tree of G
$M \leftarrow$ Minimum-weight perfect matching on the odd-degree vertices of T
$H \leftarrow$ Eulerian tour of $T \cup M$
$C \leftarrow$ Short cut of H
return C

Note (Perfect Matching)**.** Perfect matching 是指在一個圖中，每個頂點都恰好被匹配到另一個頂點的邊集合

- Edmonds 有一種 $O(mn^2)$ 的 polynomial-time algorithm
- Gabow and Tarjan 有一種 $O(n^3 \log n)$ 的 polynomial-time algorithm

Let C^* be the optimal Hamiltonian cycle. We can derive the following inequalities:

$$w(T) \leq w(C^*) \tag{1}$$

$$w(M) \leq \frac{1}{2} \cdot w(C^*) \tag{2}$$

Combining (1) and (2), we get

$$w(C) \leq w(H) = w(T) + w(M) \leq w(C^*) + \frac{1}{2}w(C^*) = \frac{3}{2}w(C^*)$$

Remark (Inequality (2))**.** Let C^*_X be a minimum-weight Hamiltonian cycle for the subgraph induced by the odd-degree vertices X in T, which denote as $G[X]$. By the triangle inequality, we have

$$w(C^*_X) \leq w(C^*) \tag{3}$$

Observe that C^*_X can be partitioned into two perfect matchings M_1 and M_2. (交錯匹配，先配左再配右或先配右再配左) Thus,

$$w(M) \leq 0.5 \cdot (w(M_1) + w(M_2)) = 0.5 \cdot w(C^*_X) \tag{4}$$

Combining (3) and (4), we obtain inequality (2).

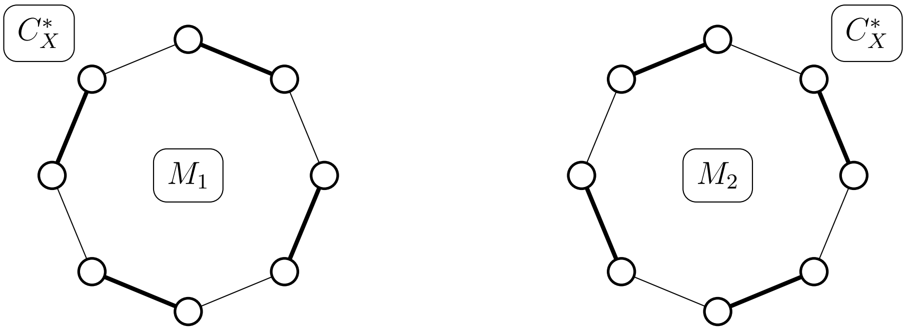


Figure 7.3: Partitioning C_X^* into Two Perfect Matchings M_1 and M_2

Approximation 分為很多種：

- Polynomial-time Approximation Scheme (PTAS)：對於任意的 $\epsilon > 0$，存在一個多項式時間的演算法，可以找到一個 $(1+\epsilon)$-approximation 的解：Euclidean TSP
- Constant Factor Approximation：存在一個常數 c，使得演算法可以找到一個 c-approximation 的解：Vertex Cover
- Logarithmic Approximation：存在一個對數函數 $\log(n)$，使得演算法可以找到一個 $\Theta(\log n)$-approximation 的解：Set Cover
- No Approximation Possible：除非 P = NP，否則不存在任何多項式時間的近似演算法：General TSP

7.2 No Approximation Possible

7.2.1 General TSP

Question (General TSP). Given

- **Input**: A graph $G = (V, E)$ (not necessarily complete) with non-negative edge weights. (w may not satisfy the triangle inequality)
- **Output**: A minimum-weight Hamiltonian cycle in G.

這個問題就變成 NP-Complete 問題，就算我們的近似是非常差的近似，例如

$$f(n) = n^n$$

但尋找 $f(n)$-approximation algorithm 仍然是 NP-Complete 問題，因為如果有一個 $f(n)$-approximation algorithm B for TSP，我們就可以拿來解 Hamiltonian Cycle on unweighted Graph 問題：

- 如果 $B(G)$ 算出某個 Hamiltonian cycle C 滿足

  $$w(C) \le f(n) \cdot w(C^*)$$

  就知道 G 有 Hamiltonian cycle

- 如果 $B(G)$ 算不出任何 Hamiltonian cycle C 滿足

  $$w(C) \le f(n) \cdot w(C^*)$$

則代表 G 沒有 Hamiltonian cycle，因為如果有任何 Hamiltonian cycle C，則

$$w(C) = n = w(C^*)$$

簡而言之，如果有 $f(n)$-approximation algorithm for General TSP，我們就可以用它來解決是否有 Hamiltonian cycle 的問題，因為有 Hamiltonian cycle 的話，approximation algorithm 一定可以找到一個比 $f(n) \cdot w(C^*)$ 還小的解（用原本就有的邊），但如果沒有 Hamiltonian cycle 的話，approximation algorithm 一定找不到比 $f(n) \cdot w(C^*)$ 還小的解（因為沒有解）。

7.2.2 Logarithmic Approximation

Question (Set Cover Problem). Given

- **Input**: k subsets $S_1, S_2, \ldots, S_k$ of a Universe set $U = \{1, 2, \ldots, n\}$.
- **Output**: A minimum-size index set I such that

$$\bigcup_{i \in I} S_i = \{1, 2, \ldots, n\}$$

Theorem 7.2.1. Vertex Cover Problem $\leq_p$ Set Cover Problem.

Proof. Given a graph $G = (V, E)$, we can construct an instance of Set Cover Problem as follows:

- Universe: $U = E$
- Subsets: For each vertex $v \in V$, define a subset $S_v = \{e \in E : e \text{ is incident to } v\}$
- Index Set: The index set I corresponds to the selected vertices in the vertex cover.

A vertex cover in G corresponds to a set cover in the constructed instance, and vice versa. Therefore, solving the Set Cover Problem on this instance will yield a solution to the Vertex Cover Problem.

■

所以，Set Cover Problem 是一個 NP-Hard 問題，我們可以採用以下的 Greedy Approximation Algorithm 來解決 Set Cover Problem：

Algorithm 7.4: Greedy Approximation Algorithm for Set Cover Problem

Input: A universe set $U = \{1, 2, \ldots, n\}$ and k subsets $S_1, S_2, \ldots, S_k$ of U

Output: An index set I such that $\bigcup_{i \in I} S_i = U$

```
I ← ∅
C ← ∅
while C ≠ U do
    i ← arg max_i |S_i \ C|
    C ← C ∪ S_i
    I ← I ∪ {i}
return I
```

Theorem 7.2.2. The Greedy Algorithm is an $O(\log n)$-approximation algorithm for the Set Cover Problem.

Proof. We now denote the i-th set chosen by the greedy algorithm as S_i, and let C_i be the set of elements covered before the S_i is chosen. Then we can let the price of choose element j in the i-iteration is

$$\text{price}(j) = \frac{1}{|S_i \setminus C_i|}$$

Which will let the cost of choosing all integer be

$$\sum_{j \in U} \text{price}(j) = |I|$$

While j is about to put into C, there are at least $n-j+1$ elements not covered yet. I^* is a collection of sets that cover all elements, so there is at least one set $t \in I^*$ that S_t cover at least

$$\frac{n-j+1}{|I^*|}$$

, which is the average number of elements that each set in I^* can cover.

> **Note.** 因為 Greedy Algorithm 每次都選擇可以覆蓋最多未覆蓋元素的集合，我們已經證明一定存在起碼一個以上 cover 超過平均值的集合 S_t，因此
>
> $$|S_i \setminus C_i| \geq |S_t \setminus C_i|$$

Therefore, we have

$$|S_i \setminus C_i| \geq \frac{n-j+1}{|I^*|}$$

, which implies that

$$\text{price}(j) = \frac{1}{|S_i \setminus C_i|} \leq \frac{|I^*|}{n-j+1}$$

Since the total cost is

$$\sum_{j \in U} \text{price}(j) = |I| \leq \sum_{j=1}^{n} \frac{|I^*|}{n-j+1} = |I^*| \cdot H_n = O(\log n) \cdot |I^*|$$

, we conclude that the Greedy Algorithm is an $O(\log n)$-approximation algorithm for the Set Cover Problem. ■

7.3 Deterministic Rounding

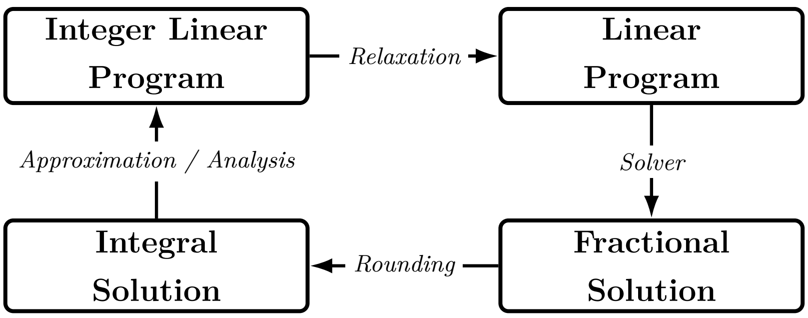


Figure 7.4: The procedure of Approximation by LP Relaxation and Rounding

我們可以透過以下步驟來設計一個 approximate algorithm：

- 將最佳化問題轉換成 Integer Linear Program (ILP)
- 將 ILP 放寬成 Linear Program (LP)
- 使用 polynomial-time LP solver 來求解 LP，得到 fractional solution
- 將 fractional solution 透過 rounding 技術轉換成 integral solution
- 分析 integral solution 的品質，證明其 approximation ratio

以 Vertex Cover Problem 為例，我們可以將其轉換成以下的 ILP：

Question (Vertex Cover ILP). Let

$$x_i = \begin{cases} 1 & \text{node } i \text{ is coverd} \\ 0 & \text{otherwise} \end{cases}$$

The ILP formulation is as follows:

$$\min \sum_{i \in V} x_i \quad \text{s.t.} \quad \begin{cases} x_i + x_j \geq 1 & \text{for each edge } (i, j) \in E \\ x_i \in \{0, 1\} & \text{for each node } i \in V \end{cases}$$

然後我們可以將 ILP 放寬成一般的 LP：

Question (Vertex Cover LP). The LP formulation is as follows:

$$\min \sum_{i \in V} x_i \quad \text{s.t.} \quad \begin{cases} x_i + x_j \geq 1 & \text{for each edge } (i, j) \in E \\ 0 \leq x_i \leq 1 & \text{for each node } i \in V \end{cases}$$

接著，我們可以使用 polynomial-time LP solver 來求解 LP，得到 fractional solution $x^* = \{x_i^*\}$

Note. Linear Programming 可以在多項式時間內被求解，常見的演算法有 Simplex Method、Ellipsoid Method 和 Interior Point Method 這些 LP solver，但在本課程並沒有介紹，因此假設我們有一個黑盒子可以在多項式時間內求解 LP。

最後，我們可以透過以下的 rounding 技術來將 fractional solution 轉換成 integral solution：
Let

$$C = \{i \in V \mid x_i^* \geq 0.5\}$$

Remark. 對於每個 approximation algorithm 我們必須檢查三件事：

- **Feasibility**: C 是否為一個合法的 solution
- **Tractability**: 運行時間是否為多項式時間
- **Approximation Ratio**: $|C|$ 與 $|C^*|$ 之間的關係

Theorem 7.3.1. The above rounding algorithm is a 2-approximation algorithm for the Vertex Cover Problem.

Proof. 我們證明以下三件事

- **Feasibility**: 對於每個 edge $(i,j) \in E$，根據 LP 的限制條件，我們有

$$x_i + x_j \geq 1$$

因此至少有一個 x_i 或 x_j 大於等於 0.5，所以 C 中至少包含一個端點，因此 C 是一個合法的 vertex cover。

- **Tractability**: 求解 LP 的時間是多項式時間，而 rounding 的過程只需要遍歷所有的頂點，因此整體運行時間是多項式時間。

- **Approximation Ratio**: 由於 rouding solution $\hat{x}_i$ 有以下性質

$$\hat{x}_i = \begin{cases} 1 & \text{if } x_i \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

因此我們得到

$$\sum_{i \in V} \hat{x}_i \leq 2 \cdot \sum_{i \in V} x_i$$

並且我們知道

$$\sum_i x_i \leq |C^*| = \sum_i x_i^* \leq \sum_i \hat{x}_i \leq 2 \cdot \sum_i x_i^*$$

也就是

$$|C_{\text{LP}}| \leq |C_{\text{ILP}}| \leq |C_{\text{Rounding}}| \leq 2 \cdot |C_{\text{LP}}|$$

所以我們得到

$$|C_{\text{Rounding}}| \leq 2 \cdot |C^*|$$

綜合以上三點，我們可以得知這是一個 2-approximation algorithm for Vertex Cover Problem。 ■

7.4 Randomized Rounding

Notation. Suppose there are n variables $1, 2, \ldots, n$. For each clause C_j, we denote the set as ordered pairs (C_j^+, C_j^-) of disjoint subsets of variables, where

$$C_j^+ = \{i \mid \text{variable } i \text{ appears as a positive literal in clause } C_j\}$$

and

$$C_j^- = \{i \mid \text{variable } i \text{ appears as a negative literal in clause } C_j\}$$

and

$$|C_j| = |C_j^+| + |C_j^-|$$

我們要解決的問題是 Max SAT Problem，因為原始的 K-SAT Problem 是 decision problem，因此我們將其轉換成 optimization problem：

Question (Max SAT Problem). Given

- **Input**: m clauses $C_1, C_2, \ldots, C_m$ with $|C_j| \leq k$ over n variables $1, 2, \ldots, n$.
- **Output**: A truth assignment that maximizes the number of satisfied clauses.

Note. 2-SAT, MAX 1-SAT 都是 P 問題，而 3-SAT, MAX 2-SAT, MAX 3-SAT 都是 NP-Hard 問題

對於這個問題，有一種 0.75-approximation algorithm，精準一點來說，我們會介紹

1° Expected ratio ≥ 0.5

2° Expected ratio $\geq 1 - 1/e \approx 0.63$

3° Expected ratio ≥ 0.75

7.4.1 Expected Ratio ≥ 0.5

我們從第一種方法開始：

Algorithm 7.5: Random assign

Input: m clauses $C_1, C_2, \ldots, C_m$ with $|C_j| \leq k$ over n variables $1, 2, \ldots, n$

Output: A truth assignment that maximizes the number of satisfied clauses

for *i from* 1 *to* n **do**

  Set

$$x_i = \begin{cases} \text{True} & \text{with } \Pr[\text{True}] = 0.5 \\ \text{False} & \text{with } \Pr[\text{False}] = 0.5 \end{cases}$$

return Assignment X

我們來分析這個演算法的 approximation ratio：

Lemma 7.4.1. Since all variables are assigned independently, the probability of clause C_j being satisfies is

$$1 - \left(\frac{1}{2}\right)^{|C_j|}$$

which is at least $\frac{1}{2}$.

所以 the expected number of satisfied clauses is at least

$$\sum_{j=1}^{m} \left(1 - \left(\frac{1}{2}\right)^{|C_j|}\right) \geq \sum_{j=1}^{m} \frac{1}{2} = \frac{m}{2}$$

因此，這是一個 expected ratio ≥ 0.5 的 approximate algorithm for Max SAT Problem。

7.4.2 Expected Ratio $\geq 1 - 1/e$

這個方法其實只在在 Rouning 的部分做了 Randomized Rounding：

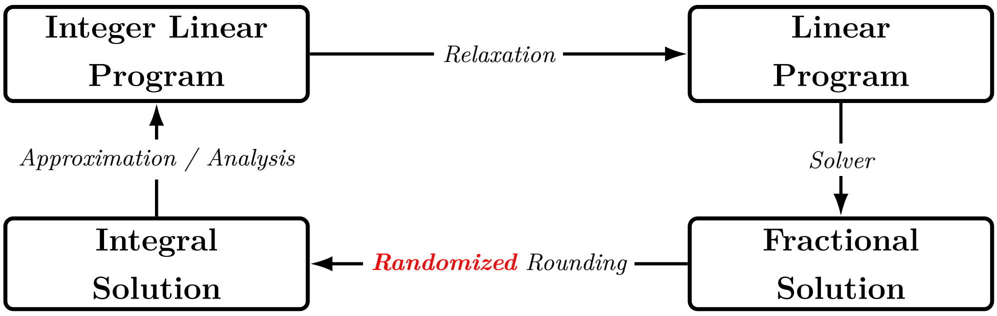


Figure 7.5: The procedure of Approximation by LP Relaxation and Rounding

We first let

$$x_i = \begin{cases} 1 & \text{variable } i \text{ is True} \\ 0 & \text{variable } i \text{ is False} \end{cases}$$

and

$$y_j = \begin{cases} 1 & \text{clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

Question (Max SAT ILP)**.** The ILP formulation is as follows:

$$\max \sum_{j=1}^{m} y_j \quad \text{s.t.} \quad \sum_{i \in C_j^+} x_i + \sum_{i \in C_j^-} (1 - x_i) \geq y_j$$

with

$$\begin{cases} x_i \in \{0, 1\} \\ y_j \in \{0, 1\} \end{cases}$$

Then do the relaxation:

Question (Max SAT LP)**.** The LP formulation is as follows:

$$\max \sum_{j=1}^{m} y_j \quad \text{s.t.} \quad \sum_{i \in C_j^+} x_i + \sum_{i \in C_j^-} (1 - x_i) \geq y_j$$

with

$$\begin{cases} 0 \leq x_i \leq 1 \\ 0 \leq y_j \leq 1 \end{cases}$$

Lemma 7.4.2. Let (x^*, y^*) be the optimal solution to the LP. The maximum number of satisfied clauses (actual maximum) is at most $\sum_{j=1}^{m} y_j^*$.

Proof. Since the LP is a relaxation of the ILP, any feasible solution to the ILP is also a feasible solution to the LP. Therefore, the optimal value of the LP is at least as large as the optimal value of the ILP. ■

Algorithm 7.6: Randomized Rounding

for i *from* 1 *to* n **do**
  Set x_i to be TRUE with probability x_i^*
return Assignment X

Lemma 7.4.3. We have the inequality:

$$\Pr[C_j \text{ is satisfied}] \geq \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) y_j^*$$

, which is at least $(1 - 1/e)\, y_j^*$.

Proof. We start from the inequality:

$$\Pr[C_j \text{ is satisfied}] \geq \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) y_j^*$$

$$\begin{aligned}
1 - \text{LHS} &= \left(\prod_{i \in C_j^+} (1 - x_i^*)\right)\left(\prod_{i \in C_j^-} x_i^*\right) \\
&\leq \left(\frac{\sum_{i \in C_j^+}(1 - x_i^*) + \sum_{i \in C_j^-} x_i^*}{|C_j|}\right)^{|C_j|} && \text{(by AM-GM Inequality)} \\
&= \left(1 - \frac{\sum_{i \in C_j^+}(x_i^*) + \sum_{i \in C_j^-}(1 - x_i^*)}{|C_j|}\right)^{|C_j|} \leq \left(1 - \frac{y_j^*}{|C_j|}\right)^{|C_j|} && \text{(by LP constraint)}
\end{aligned}$$

Claim. Let $f(r) = 1 - (1 - r/k)^k$. Then, for any $k \in \mathbb{Z}^+$, $f(r)$ is concave in $r \in [0, 1]$, we have

$$f(r) \geq \left(1 - (1 - 1/k)^k\right) r$$

Proof. One can verify that the claim holds at the endpoints $k = 1, 2$. Now, assume that $k > 2$. We have

$$\begin{cases} f(0) = 0 \\ f(1) = 1 - (1 - 1/k)^k \end{cases}$$

and

$$\frac{\mathrm{d}}{\mathrm{d}r} f(r) = (1 - r/k)^{k-1}$$

$$\frac{\mathrm{d}^2}{\mathrm{d}r^2} f(r) = -\frac{k-1}{k}(1 - r/k)^{k-2} \leq 0$$

which is concave in $r \in [0, 1]$.

By the claim, we have

$$\text{LHS} \geq 1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|} \geq \text{RHS}$$

■

7.4.3 Expected Ratio ≥ 0.75

我們可以將前兩種方法結合起來，得到更好的 approximation ratio：

Algorithm 7.7: Combined Algorithm

```
Let a boolean variable b be TRUE with probability 0.5
if b = TRUE then
    run the Random Assign Algorithm
else
    run the Randomized Rounding Algorithm
return Assignment X
```

Theorem 7.4.1. The above combined algorithm is a $\frac{3}{4}$-approximation algorithm for the Max SAT Problem.

Proof. By lemma 10.4.1, 10.4.3, we have

$$\begin{aligned}
2 \cdot \Pr[C_j \text{ is satisfied}] &\geq 1 - \left(\frac{1}{2}\right)^{|C_j|} + \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) y_j^* \\
&\geq 1 - \left(\frac{1}{2}\right)^{|C_j|} + \left(1 - \frac{1}{e}\right) y_j^* && \text{(since lemma 10.4.3)} \\
&\geq \frac{3}{2} y_j^* && \text{(since lemma 10.4.1 and } 1 - 1/e > 1/2) \\
\Rightarrow \Pr[C_j \text{ is satisfied}] &\geq \frac{3}{4} y_j^*
\end{aligned}$$

■

7.5 Derandomized

我們將使用條件期望值來進行 derandomization：

7.5.1 Derandomized for Random Assign Algorithm

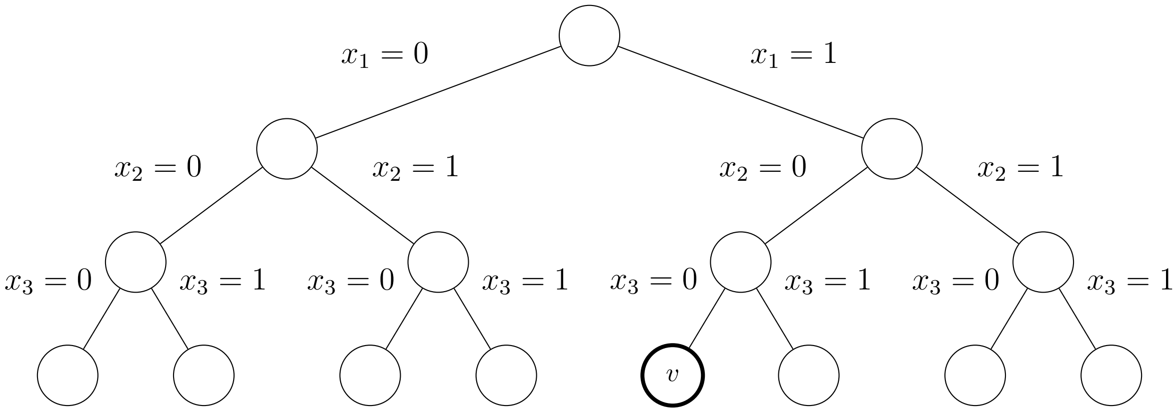


Figure 7.6: Truth Assignment Tree for SAT

根據上面的 Truth Assignment Tree，我們可以拿到所有可能的 truth assignment 的 decision path，而我們可以使用條件期望值來選擇每個變數的值：例如，

$$v : x_1 = 1, x_2 = 0, x_3 = 0$$

對每個 v 來說，都可以對應一個 partial truth assignment $f(v)$

Definition 7.5.1 (Expected Satisfied Clauses). For each boolean variable x_i, there are 0.5 probability to be TRUE or FALSE.

Example. For

- $C_1 = \neg x_1$: satisfied T with probability 1/2
- $C_2 = x_2 \vee x_3$: satisfied T with probability 3/4
- $C_3 = x_1 \vee \neg x_2 \vee \neg x_3$: satisfied T with probability 7/8

because Expected number of satisfied clauses is a linear function, so we have

$$\mathbb{E}[\text{number of satisfied clauses}] = \frac{1}{2} + \frac{3}{4} + \frac{7}{8} = \frac{13}{8}$$

Expected Satisfied Clauses is the expected number of satisfied clauses.

Corollary 7.5.1. Since Expected Satisfied Clauses is a linear function, we have

$$\begin{aligned}\mathbb{E}[\text{number of SC}] = \Pr[x_1 = F] \cdot \mathbb{E}[\text{number of SC} \mid x_1 = F] \\ + \Pr[x_1 = T] \cdot \mathbb{E}[\text{number of SC} \mid x_1 = T]\end{aligned}$$

So the conditional expectation is set to

Example. If x_1 is set to FALSE

- $C_1 = \neg x_1$: satisfied T with probability 1
- $C_2 = x_2 \vee x_3$: satisfied T with probability 3/4
- $C_3 = x_1 \vee \neg x_2 \vee \neg x_3$: satisfied T with probability 3/4

Under this condition, we have

$$\mathbb{E}[\text{number of SC} \mid x_1 = F] = 1 + \frac{3}{4} + \frac{3}{4} = 5/2$$

Corollary 7.5.2. If v and w are the children of u, where

- uv corresponds to setting $x_i = 0$
- uw corresponds to setting $x_i = 1$

then

$$\mathbb{E}[u] \leq \max\{\mathbb{E}[v], \mathbb{E}[w]\}$$

Proof. Follow the definition of conditional expectation.

$$\begin{aligned}\mathbb{E}[u] &= \Pr[x_i = 0] \cdot \mathbb{E}[v] + \Pr[x_i = 1] \cdot \mathbb{E}[w] \\ &= \frac{1}{2} \cdot (\mathbb{E}[v] + \mathbb{E}[w]) \\ &\leq \max(\mathbb{E}[v], \mathbb{E}[w])\end{aligned}$$

Proof complete. ■

每個 node 的期望值都可以用 polynomial time 計算出來，因此找到最大期望值的路徑也可以在 polynomial time 內完成。

Algorithm 7.8: Derandomized for Random Assign Algorithm

Build the Truth Assignment Tree
$u \leftarrow$ root of the tree
while u *is not a leaf* **do**
    $v, w \leftarrow$ children of u
    if $\mathbb{E}[v] \geq \mathbb{E}[w]$ **then**
        $u \leftarrow v$
    else
        $u \leftarrow w$
return Assignment $f(u)$

7.5.2 Derandomized for Randomized Rounding Algorithm

這根第一個 derandomization 類似，我們也是建立 Truth Assignment Tree，然後使用條件期望值來選擇每個變數的值。

所以我們必須考慮的問題是，如何計算每個 node 的期望值：

$$\Pr[C_j \text{ is satisfied}] = 1 - \left(\prod_{i \in C_j^+} (1 - x_i^*)\right)\left(\prod_{i \in C_j^-} x_i^*\right)$$

所以每個點的左右 probability 分別是

- left child: $x_i = 0$ with probability $1 - x_i^*$
- right child: $x_i = 1$ with probability x_i^*

可以透過 linear programming 來算出來，都在 polynomial time 內。

Algorithm 7.9: Derandomized for Randomized Rounding Algorithm

Build the Truth Assignment Tree
$u \leftarrow$ root of the tree
while *u is not a leaf* **do**
  $v, w \leftarrow$ children of u
  if $\mathbb{E}[v] \geq \mathbb{E}[w]$ **then**
    $u \leftarrow v$
  else
    $u \leftarrow w$
return Assignment $f(u)$

可以發現完全一樣的演算法，只是計算每個 node 的期望值的方式不同而已。

7.5.3 Derandomized for Combined Algorithm

這次只是在最一開始的時候，選擇要跑哪一個 derandomized algorithm，我們一樣可以用條件期望值來決定：

$$\begin{aligned}\mathbb{E}[\text{number of SC}] &= \Pr[b = F] \cdot \mathbb{E}[\text{number of SC} \mid b = F] \\ &\quad + \Pr[b = T] \cdot \mathbb{E}[\text{number of SC} \mid b = T]\end{aligned}$$

因此我們可以選擇 $\mathbb{E}[\text{number of SC} \mid b = F]$ 或 $\mathbb{E}[\text{number of SC} \mid b = T]$ 較大的一個來執行，因為 root node 的 expected ratio 是 0.75，所以他的兩個 child node 至少有一個的 expected ratio 也會是 ≥ 0.75。

Algorithm 7.10: Derandomized for Combined Algorithm

$m_1 \leftarrow \mathbb{E}[\text{number of SC} \mid b = F]$
$m_2 \leftarrow \mathbb{E}[\text{number of SC} \mid b = T]$
if $m_1 \geq m_2$ **then**
  run Derandomized for Random Assign Algorithm
else
  run Derandomized for Randomized Rounding Algorithm
return Assignment X