114-2 演算法期中量子退火報告

111502032 鄧博元

問題說明

Vertex Cover Problem 點覆蓋問題

- 圖的覆蓋是一個頂點的集合，使圖中的每一條邊都至少連結該集合中的一個頂點。尋找最小的頂點覆蓋的問題稱為頂點覆蓋問題，屬於NPC問題。
- 假設有一無向圖 $G = (V, E)$，其中 V 為頂點集合，E 為邊集合。頂點覆蓋是指從 V 中選出一個子集 $C \subseteq V$，使得圖中每條邊 $e = (u, v)$ 至少有一個端點屬於 C。也就是說，對於所有邊 $(u, v) \in E$，必有 $u \in C$ 或 $v \in C$（或兩者皆屬）。
- **簡單來說就是找最少數量的點集合C讓每條邊都至少有一點屬於C**

Qubo 公式設計

為了將 Vertex Cover 問題轉換成適合量子退火求解的 QUBO形式，我們要做兩件事：

1. 定義一組二元變數表示每個點是否被選進覆蓋集合。
2. 設計一個「能量函數」，讓最小化這個函數時，同時達到「邊都有被覆蓋」且「選到的點盡可能少」。

一、變數定義

假設圖中有 n 個節點，對每個節點 v_i 建立一個二元變數 x_i：

- $x_i = 1$ 表示節點 i 被選進點覆蓋集合。
- $x_i = 0$ 表示節點 i 沒有被選。

二、QUBO 目標函數設計

QUBO 目標函數由兩部分組成：

(1) Cost Term：想要選得越少越好

$$\text{Cost} = \sum_{i=1}^{n} x_i$$

這代表我們希望選進點覆蓋集合的節點數量越少越好。

(2) Penalty Term：強迫每條邊至少有一端點被選到

對於每一條邊 (u, v)，如果 $x_u = 0$ 且 $x_v = 0$，代表邊沒被覆蓋，需要加懲罰。

為了表示這個限制，可以對每條邊加入以下懲罰項：

$$\text{Penalty}_{(u,v)} = (1 - x_u - x_v + x_u \cdot x_v)$$

這個式子的特性是：

- 若 $x_u = 1$ 或 $x_v = 1$，值為 0（代表沒違規）
- 若 $x_u = 0$ 且 $x_v = 0$，值為 1（違規，需要懲罰）

三、完整的 QUBO 式子

將兩部分加起來，加上一個懲罰係數 λ，我們就得到完整的 QUBO：

$$\text{QUBO} = \sum_{i=1}^{n} x_i + \lambda \cdot \sum_{(u,v)\in E} (1 - x_u - x_v + x_u \cdot x_v)$$

其中 λ 是一個用來平衡「覆蓋條件」與「選點數量」的重要參數。如果 λ 太小，模型會傾向違規但少選點；太大則會過度選點。

對於所有資料集的實驗 & 最終結果

上網 survey 每一個資料集的最佳解

dataset	|V|	已知最小頂點覆蓋 (MVC)	出處
Keller4	171	160	-
keller5	776	749	PCI Cyber
keller6	3361	3302	PCI Cyber& 某lab 2005 年的論文
p_hat300-1	300	292	PCI Cyber
p_hat700-1	700	689	PCI Cyber
p_hat1500-1	1500	1488	PCI Cyber

先定義我實作過程中用到的所有參數

參數名稱	說明
reads	每個 sampler 讀取幾次樣本，值越大結果越穩定，但耗時越多。
sweeps	每個樣本要執行幾次內部變數更新（只對 SA ，SimulatedAnnealingSampler 有效），控制退火的「深入程度」。

λ	QUBO 的懲罰權重，用來平衡「選少一點點」和「邊要都被覆蓋」， 我一開始是手動對每一個資料集調整對應的值， 但是我到後面使用一個公式 λ = (邊數E/節點數V)* 一個長數c
c	如上λ提及的
beta_range	SA 的退火溫度範圍（初始溫度到結束溫度），低溫區會讓解趨於穩定。
beta_schedule_type	SA 的降溫方式，我實驗了 linear 跟 geometric， 差異在於冷卻速度快慢。

先完成 keller4 資料集

- 一開始就先設定 reads = 1000 (跟範例題的Subset Sum Problem一樣)，sweeps = 1000 (套件預設值)，λ = 250
- 但是結果非常不穩定，有時候160，有時候161，所以我把 read 調成 10000，就穩定都可以得到最佳解 160 了
- ***但是當我對其他資料集做同樣的方法之後發現要跑超級久且效果離我先前survey 到的最佳解有一段差距，所以我做了一些實驗試圖減少時間並接近最佳解***

實驗0: 在量子退火結束之後再過一個 greedy 來做簡單排除篩選，減少數量

- 在某一些狀況下的 keller4 資料集， qubo 的結果僅為161，而對其做 greedy 之後即可降為160
- 其他 dataset 有時候也可以
- 成功降低的機率不高，但是放進流程裡面沒甚麼損失，可以增加得到最小解的機率，且為線性時間

實驗1: 調整 reads

- 只對 keller6 ，因為節點數最多，在我的觀察下是最不穩定的一個
    - 固定 c = 5 ，而 λ = |E| / |V| * c ，所以等同固定 lambda
    - 固定 sweeps = 100 ，不想設定太大不然會跑太久
    - 固定不使用 beta 參數
    - reads = [1, 5, 10, 100, 500, 1000, 5000, 10000]

- 結果:

reads	MVC (after greedy)	MVC_raw (before greedy)	valid	time_sec
1	3323	3323	1	42.6
5	3318	3318	1	47.1
10	3319	3319	1	48.1

100	3314	3314	1	74.0
150	3317	3317	1	60.5
1000	3315	3315	1	53.0
5000	3314	3314	1	96.2
10000	3312	3312	1	117.4

- 分析:
  - 從實驗結果可以發現，reads 增加會讓最終結果越接近最佳解。
  - 從 reads = 1 得到 MVC = 3323，增加到 reads = 10000 時可以下降到 3312，改善 11 點。
  - 但提升效果在 reads 達到一定數量後會逐漸趨緩，例如從 1000 到 5000、甚至到 10000，差距縮小。
  - 綜合來看，reads = 1000～5000 是一個效果與時間的折衷點，再更高 reads 效益變小但耗時變多。

實驗2: 調整 sweeps

- 只對 keller4
  - 固定 c = 5 ，而 $\lambda = |E| / |V| * c$ ，所以等同固定 lambda
  - 固定 reads = 100 ，不想設定太大不然會跑太久
  - 固定 beta_range=[0.01, 20]
  - 固定 beta_schedule_type="linear"
  - sweeps = [1, 5, 10, 100, 500, 1000, 5000, 10000]

- 結果:

sweeps	MVC (after greedy)	MVC_raw (before greedy)	valid	time_sec
1	164	166	1	0.0
5	162	162	1	0.0
10	162	162	1	0.0
100	160	160	1	0.0
500	160	160	1	0.0
1000	160	160	1	0.1
5000	160	160	1	0.2

10000	160	160	1	0.3

- 分析:
  - sweeps 代表每一個 read 內部退火過程跑幾輪，也就是每個樣本優化的「深度」。
  - 這邊可以看到 greedy 有用，把 sweep=1 的 MVC 答案 -2
  - 可以看到，在 sweeps 很小（例如 1）時，結果偏離正確解，且 greedy 有明顯作用（因為初始解選點太多）
  - 當 sweeps 達到 500 或 1000 時，結果已穩定收斂到最佳解 160，增加 sweeps 對最終 MVC 幾乎沒影響，但時間變長
  - 因此，sweeps 提升初期幫助較大，超過一定程度後變成浪費資源。所以我後續 sweeps 控制在 1000~2000

實驗3: 加入 beta_range & beta_schedule_type

- 只針對 keller5
  - 固定 c = 5 ，而 $\lambda = |E| / |V| * c$ ，所以等同固定 lambda
  - 固定 reads = 1000 ，sweeps=2000
  - 比較
    1. beta_range=[0.01, 20], beta_schedule_type="linear"
    2. beta_range=[0.01, 20], beta_schedule_type="geometric"
    3. 都沒有

- 結果:

method	MVC	MVC_raw	valid	time_sec
geometric + beta_range=[0.01, 20]	749	749	1	2.9
linear + beta_range=[0.01, 20]	749	749	1	8.8
neither	757	758	1	67.0

- 分析:
  - 這邊可以看到greedy 有用，把 neither 的 MVC 答案 -1
  - 不設 beta 參數會導致退火過程不穩，造成結果高達 758（未經 greedy 處理），相對較差
  - 加上 beta_range 與 beta_schedule 後，馬上可以找到最小值 749
  - 比較 linear 與 geometric 排程：
    - 兩者都可以達到最小解，但 geometric 所花時間更短。
    - 這代表 geometric 的冷卻速度更快，可以用更少 sweeps 達到同樣效果。

- 加入 beta 參數可能事成功的關鍵，所以我後面都有加

實驗4: 調整λ（c）

一開始我並不知道 λ 應該設定多少，因此我先對多個資料集進行了固定值的測試，例如 λ = 1、100、500、1000 等。

結果發現：

- 當 λ 設為 1 時，部分資料集會出現 invalid 解（違反邊覆蓋條件）；
- 但當 λ 提升至 100 以上時，大多數圖都能得到合法解（valid = True）；
- 不過在 λ = 100、500、1000 間，**最小頂點覆蓋的結果差距並不大**，代表這些值的效果都接近飽和。

從這個觀察中我意識到，圖的結構可能會影響對懲罰強度的需求。
於是我進一步思考：**如果一張圖的邊密度越高，模型就越容易因為選太少點而違反邊的覆蓋條件，這時候 λ 就應該設得更大。**

因此我設計了一個簡單又通用的公式：

$$\lambda = \frac{|E|}{|V|} \cdot c$$

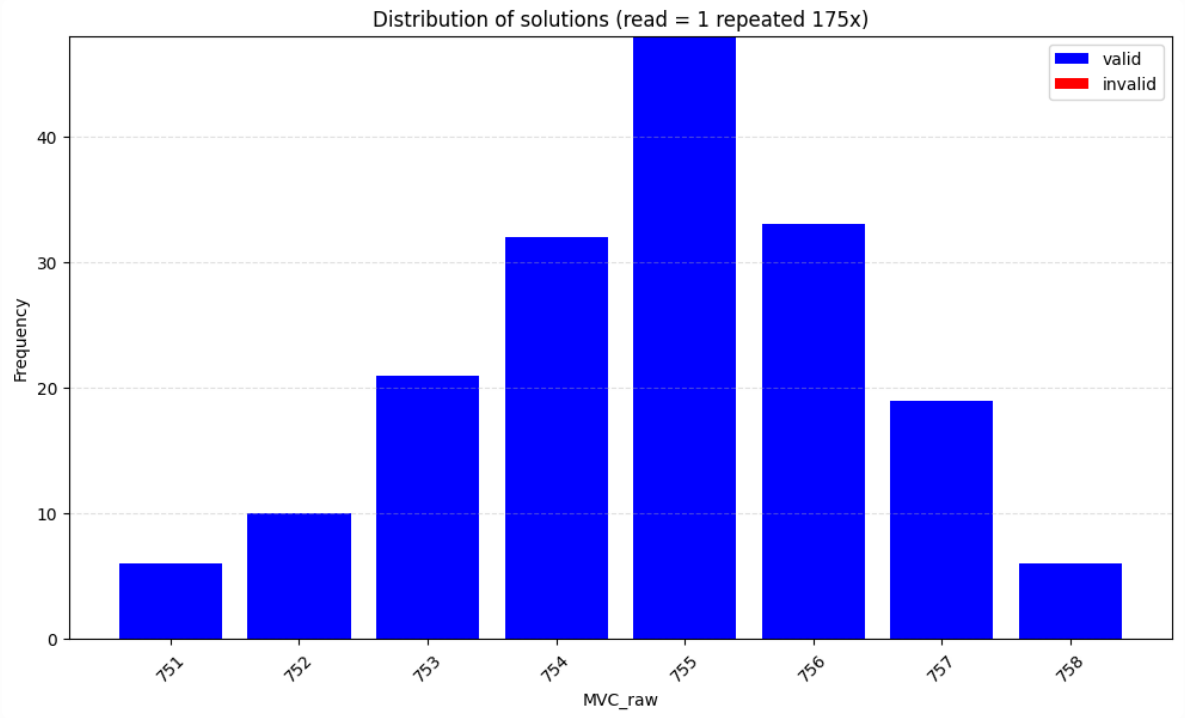
這個公式利用圖的平均度（edge density）來動態調整 λ，使模型能針對不同圖自動調整懲罰強度。
經過實測，在多個資料集中，**當 $c = 5$ 時就能穩定產出合法且接近最佳的解**。

這也成為我後續所有資料集實驗時的統一 λ 設定方式，既簡單，又具有跨圖適應性。
但我無法驗證他是最好的。

奇妙的實驗5: 答案分布圖

- 鑒於套件限制，我們每一次去跑一個資料集的時候都只能在一開始訂好 read 數量(假設 =10000)，並且過程是不可見的，最終只會輸出那 10000 次 read 的最佳解，這對於調參或分析演算法表現來說，是一種資訊上的黑箱。
- 因此我進行了一個有趣的小實驗：將 read 數設定為 1，並重複進行 175 次個別退火過程，讓每一次 read 的結果都能被收集與觀察，進而繪製出 Minimum Vertex Cover (MVC_raw) 的出現頻率分布圖。下圖是以 keller5.mis 為資料集所繪製的結果：

- 挺有趣的



從圖中可以看出：

- 絕大多數的解集中在 754–756 之間，呈現一個左右對稱但稍微偏右的鐘形分布；
- 所有結果皆為合法解（valid），未觀察到因 λ 設定過低而出現 invalid 的狀況，顯示懲罰項設計成功；
- 此實驗某種程度上也印證了：模擬退火會隨機探索局部空間，但出現全域最小解的機率較低，需仰賴大量 reads 或更佳初始溫度策略以提升穩定性與表現。
- MVC_raw 最低只有751，這是因為總共只有跑175 次 read * 1，遠遠達不到我觀察到最佳解的2000次read

最終結果

- **這邊我實測了很多次，結果基本上是穩定的**
- MVC 代表最終 (包含 greedy 篩除) 的結果節點數
- MVC_raw 代表 不包含 greedy 篩除 的結果節點數

name	MVC	MVC_raw	reads	sweeps	λ	c	valid	time
keller4.mis	160	160	2000	1000	149.12	5	1	0.6
keller5.mis	749	749	2000	2000	481.38	5	1	5.4
keller6.mis	3306	3306	5000	5000	1527.2	5	1	178.6
p_hat300-1.mis	292	292	1000	2500	565.28	5	1	1.3

p_hat700-1.mis	689	689	1000	4000	1311.79	5	1	6.3
p_hat1500-1.mis	1488	1488	1000	6000	2797.76	5	1	44.9

與網路上找到的最佳解比較

dataset	|V|	已知最小頂點覆蓋	我的結果
Keller4	171	160	160
keller5	776	749	749
keller6	3361	3302	3306
p_hat300-1	300	292	292
p_hat700-1	700	689	689
p_hat1500-1	1500	1488	1488

- keller6 是裡面唯一沒有達到最佳解的，仍有 4 點的誤差，但該圖節點與邊數皆為六組資料中最多，且其邊密度極高，使得解空間變得更加複雜與難以收斂，因此即便達不到全域最小值，也仍屬於相對高品質的解。

一些觀察

- 實驗中發現，**程式運行時間與以下因素成正比**：

$$T \propto \text{reads} \times \text{sweeps} \times |E|$$

其中 $|E|$ 是圖的邊數，reads 與 sweeps 是我們設定的參數。

dataset	|V|	|E|	reads	sweeps	預估複雜度比 (vs keller4)	實際時間 (秒)
keller4	171	5100	2000	1000	1.0（基準）	0.6
keller5	776	22599	2000	2000	約 8.86 倍	5.4
keller6	3361	153729	5000	5000	約 226 倍	178.6
p_hat300-1	300	10933	1000	2500	約 1.5 倍	1.3
p_hat700-1	700	48644	1000	4000	約 19 倍	6.3
p_hat1500-1	1500	568960	1000	6000	約 334 倍	44.9

→ 可見 keller6 與 p_hat1500-1 執行時間長，並非單純因為節點數多，而是 **邊數（密度）暴增** 導致。

心得

本次期中報告讓我實際動手將經典的 NPC 問題「頂點覆蓋轉化為 QUBO 形式，並透過模擬退火法（Simulated Annealing）進行最佳化。整個過程不只是程式的實作，更是一次對數學建模與參數調校耐心與邏輯的考驗。我逐步學會如何在不同資料集之間做出合理的超參數設計、驗證解的正確性，並透過系統性實驗觀察各種因素（如 reads、sweeps、beta schedule）如何影響結果品質與執行效率。

我在做之前也一直思考：為什麼要用量子退火？有比它更快或更準的方法嗎？我做他的意義是什麼？
根據我上網搜尋結果，若改用傳統演算法（如 networkx 內建的 2-approximation greedy algorithm 或 ILP 求解器），對於中小型圖（如 keller4、p_hat300-1）大多能在數十毫秒到幾秒內完成，甚至直接產出近似或最佳解。而量子退火模擬器在這些情況下通常需要數秒到數分鐘，且仍需搭配後處理（如 greedy），才能達到相同水準。
然而，模擬退火或量子退火法的優勢不在於「解小圖」，而是在高維、解空間極大、限制條件多且不易求解的問題上，能夠以近似方法探索可能解、快速收斂，尤其在整數規劃無法直接套用時提供了一個新選項。

我也覺得量子退火有點像是機器學習再 Gredient descent 找全域最小值的過程，在極度複雜的損失空間中尋找較佳解，只是退火不依賴梯度，而是靠隨機性與冷卻過程進行探索。

這次實作也讓我誤打誤撞地踏進效能最佳化的領域。當程式執行時間異常緩慢時，我曾懷疑是否需要使用 GPU 或平行運算加速，後來才發現，其實只是我一開始把參數設太高（例如 reads 與 sweeps 同時拉到上萬），導致運算量暴增。這段插曲也提醒我：不是所有慢速都需要硬體解決，有時候回頭檢查邏輯與參數才是關鍵。

總之，這次作業讓我從純粹理解 QUBO 的理論，進一步走向實務解題的全流程，相當奇妙有趣，即使未來不一定會直接用到這套技術，這次的經驗仍開啟了我對於組合最佳化問題與其他方法的新視角，說不定握未來在面對類似複雜問題時，能從不同層面思考與拆解。