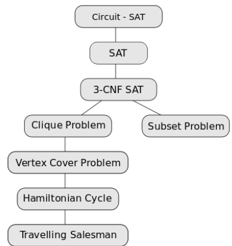
NP問題

1. **選定問題:** N-Puzzle
2. **屬於哪種:** NP-Complete
3. **定義/解釋:**

[1] NP-Complete是NP與NP-Hard的交集，是NP中最難的決定性問題，所有NP問題都可以被快速歸化為NP-Complete問題。因此NP完備問題應該是最不可能被化簡為P（多項式時間可決定）的決定性問題的集合。若任何NPC問題得到多項式時間的解法，那此解法就可應用在所有NP問題上。[1]

1. **假設是NPC=>如何證明?**

[2] 若欲證明一個問題是NPC,最簡單的方法是先證明它屬於NP,然後將「某個已知是NPC的問題變換成它。

1. 先證明該問題為NP類問題

2. 透過右圖的轉換方式,可以轉換成任何其中一個.(ex: 歸約成SAT or 3-CNF SAT...)

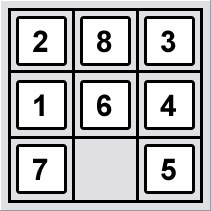
3. 由於SAT與3-CNF SAT都是NP-Complete問題,

於是就可以證明它也是NP-Complete [2]

1. **目前已有解法=>透過什麼演算法=>程式:**

(1). 廣度優先搜尋演算法BFS

(2). 深度優先搜尋演算法DFS

(3). 登山搜尋演算法

(4). 最佳優先搜尋演算法

範例: 8 puzzle

**解法(BFS):**

[3](1). 先完成移動判斷和節點狀態的儲存結構。

(2). 用窮舉狀態，並把當前狀態記錄下來篩選重複路徑。

(3). 只要有一組路徑符合結果，該路徑就是最佳解(因BFS的特性是每次平均向外擴展，所以最先碰到終點的就是最佳解之一)。

(4). 否則搜尋完畢後皆沒有找到符合的狀態，無解。

(5). 最後透過父節點的回溯取得輸出路徑，即可得到最短路徑和分解步驟。[3]

**程式(C#):**<http://davidhsu666.com/archives/csharp-bfs-solve-8-puzzle/>

**參考網址:**

1. <https://zh.wikipedia.org/zh-tw/NP%E5%AE%8C%E5%85%A8>
2. <https://drive.google.com/file/d/1jRpo9I9FDzRaNPEf_xS1YHzbHFl7aWIo/view>
3. <http://davidhsu666.com/archives/csharp-bfs-solve-8-puzzle/>