電腦對局理論 Homework 1

蔡平樂

October 16, 2025

1 Algorithm design

主體搜尋使用 IDA^* 配合 $DFS1_{cost}$，其中 IDA^* 實作完全與課堂中相同，因此不另外說明。DFS 演算法在一些細節上有修改，具體演算法如下：

Algorithm 1 DFS

```
procedure DFS(pos, threshold)
    Stack_init(S)
    Stack_init(Prv_States)
    Push(S, pos)
    depth ← 0
    while not S.empty() do
        current ← S.pop()
        if current == Null then
            depth − −
            continue
        record(current, limit_depth − depth)
        NX ← next_states_gen(current.pos)
        Next ← sort_and_erase(NX, max = threshold − depth − 1)
        Push(S, Null)
        for nx in Next do
            if is_visited(nx) then continue
            if is_win(nx) then return true
            Push(S, nx)
        depth + +
    return false
```

細節將於其後說明

1.1 *sort_and_erase*

$sort\_and\_erase(NX, max)$ 使用 Heuristic value 排序 NX，
並移除所有 $value > max$ 的元素，因此無須在 push 階段進行剪枝。

實作上，最初是使用標準算法的使用 std::sort 後於 push 過程中剪枝，但使用 gprof 時發現 heuristic 時間佔比相當高，也發現 std::sort 基本上只調用 insertion sort 進行排序，考量到實作 insertion sort 時間成本較低，且可以大幅降低 heuristic function 的呼叫次數，因此自行實作 sort 函數排序 NX，使用略帶修改的 insertion sort，使其直接忽略 $value > threshold$ 的元素並能在同時計算 $\leq threshold$ 的元素數量，實測在公佈的 3-3 測資上能夠將時間壓縮約 4 倍，常數上有相當大的優化。

1.2 *record / is_visited*

在搜索完每個 pos 後，將 $(pos, remain\_depth)$ 紀錄，代表 pos 在深度 $remain\_depth$ 下不可能解開。$is\_visit(pos, remain\_depth)$ 則會判斷是否盤面 pos 是否已搜索過深度 $\leq remain\_depth$ 的狀態，若是則 return true。具體實作如下：

---

procedure RECORD(pos, remain_depth)
    if $pos \in recorder.keys$ **then**
        $recorder[pos] \leftarrow \max(recorder[pos], remain\_depth)$
    else
        $recorder[pos] \leftarrow remain\_depth$
procedure $is\_visited(pos, remain\_depth)$
    if $pos \notin recorder.keys$ **then return false**
    else return $recorder[pos] \geq remain\_depth$

---

2 Heuristic function design

2.1 algorithm

earlybird 繳交的 heuristic 為下列版本。$dis(sq\_a, sq\_b)$ 為棋盤上 sq_a 到

Algorithm 2 Minimum Move Estimate 1

procedure MinimumMoveEstimate(pos)
  $PQ = PriorityQueue()$//sort by weight
  $Edges \leftarrow \{\}$
  for b in $Black\_Pieces(pos)$ **do**
    for r in $Red\_Pieces(pos)$ **do**
      if b can capture r **then**
        $E = edge(begin = b, end = r, weight = dis(b, r))$
        $PQ.push(E)$
  for (r_1, r_2) be any pair in $Red\_Pieces(pos)$ **do**
    for r_2 in $Red\_Pieces(pos)$ **do**
      $E = edge(begin = r_1, end = r_2, weight = dis(r_1, r_2))$
      $PQ.push(E)$
  $N = |Red\_Pieces(pos)|$
  $x \leftarrow 0$
  for $i = 1$ to N **do**
    $x \leftarrow x + PQ.pop()$
  return x

位置 sq_b 的最短路徑，在沒有鴨子的情形下為 Euclidean distance，在有鴨子時使用 Floyd-Warshall algorithm 計算。此外，黑方有車時 $dis(a, b)$ 不使用前述的最短距離，而是根據 a, b 是否在同行列直接定為 1 或 2。

而本次繳交略有修改，使用下列版本：

Algorithm 3 Minimum Move Estimate 2

```
1:  procedure MinimumMoveEstimate(pos)
2:      Edges ← {}
3:      for b in Black_Pieces(pos) do
4:          for r in Red_Pieces(pos) do
5:              if b can capture r then
6:                  E = edge(begin = b, end = r, weight = dis(b, r))
7:                  Edges.push(E)
8:      for (r_1, r_2) be any pair in Red_Pieces(pos) do
9:          for r_2 in Red_Pieces(pos) do
10:             E = edge(begin = r_1, end = r_2, weight = dis(r_1, r_2))
11:             Edges.push(E)
12:     N = |Red_Pieces(pos)|
13:     sort(Edges) // sort by weights
14:     DSU ← DisJoint_SET
15:     DSU.init()
16:     x ← 0
17:     cnt ← 0
18:     for e ∈ Edges do
19:         if e.begin is Red then // 19-21 does not exist in the earlybird
20:             if DSU.at_the_same_set(e.begin, e.end) then
21:                 continue
22:         x ← x + e.weight
23:         DSU.union(e.begin, e.end)
24:         cnt++
25:         if cnt ≥ |Red_pieces| then break
26:     return x
```

簡單來說，此算法

- 將估計值初始化為 0
- 計算所有 red, red 與 $black, red$ pair 的最短距離並排序
- 每次挑出最小距離的兩點，若其中一子為黑方或兩點位於同一個集合中則跳過，若非則將估計值加上兩點距離並合併這兩點所在集合
- 重複上述步驟直到挑出數量等同紅子的邊為止，並回傳估計值

2.2 Design criteria and admissible proof

首先，假設 $B = \{b_1, b_2, ..., b_N\}$ 為目前棋盤上的黑子，$R = \{r_1, ..., r_M\}$ 為棋盤上的紅子。若目前有一組最佳解，共需 OPT 步數，且

- b_1 需循序吃掉 $r^1_1, r^1_2, \ldots, r^1_{n_1}$, 使用步數 M_1
- b_2 需循序吃掉 $r^2_1, r^2_2, \ldots, r^2_{n_2}$, 使用步數 M_2
- $\ldots$
- b_N 需循序吃掉 $r^N_1, r^N_2, \ldots, r^N_{n_N}$, 使用步數 M_N

(n_k 可能為 0)

由最佳解的性質，可以發現

- $\sum_{k=1}^{N} n_k = M$
- $\cup_{k=1}^{N} \{r^k_1, \ldots, r^k_{n_k}\} = R$
- $M_k \geq dis(b_k, r^k_1) + \sum_{j=1}^{n_k} dis(r^k_j, r^k_{j+1})$
  ($dis(a, b)$ 為棋盤上 a 到 b 的最短路徑)

因此，若將整個棋盤視為以 $R \cup B$ 為 node 的全連通圖 G，u, v 兩點間邊長為 $dis(u, v)$，$P_k = (b_k, r^k_1, \ldots, r^k_{n_k})$ 視為圖上的路徑，此時 $P_1, P_2, ..., P_N$ 相當於 G 上的一組路徑覆蓋。除此之外，M_k 的最小值將會是 P_k 的權重和（若 P_k 長度為 1 則權重和視為 0），因此 $OPT \leq \sum_{k=1}^{N} weighs(P_k)$

因此我們可將下列問題視為對最佳解的下界估計

Problem 2.1

Construct $G = (V, E)$ with

- $V = R \cup B$
- $E = \{(r_1, r_2) : r_1, r_2 \in R, weight = dis(r_1, r_2)\} \cup \{(b, r) : b \in B, r \in R \text{ } and \text{ } b \text{ can capture } r, weight = dis(b, r)\}$

find Paths $P_1, \ldots P_N$ with minimum total weights w.r.t

- P_k start with b_k, and all other nodes from R
- $P_1, ..., P_N$ cover $R \cup B$

此外，我們可省略所有長度為 1 的 P_k，因此問題也有下列等價形式

Problem 2.2

find Paths $P_1, \ldots P_n, n \leq N$ with minimum total weights w.r.t

- P_k start with $\hat{b}_k \in B$, and all other nodes from R
- $P_1, ..., P_N$ cover R

在足夠寬敞的盤面下，通常可以透過錯開各子的移動使 M_k 能夠直接等於 P_k 的權重和。換句話說，problem 2.1與 2.2在許多狀況下都會是相當準確的估計。此外，problem 2.1與 problem 2.2的解皆會小於原始問題的解，因此也自動滿足 admissible 性質，因此若能快速估計其解答，則我們即可獲得相當好的 admissible heuristic。

然而此問題為 TSP 問題的推廣，因此目前尚無快速解決的演算法，因此我們同樣使用簡化問題的解來逼近 2.2。Earlybord 提交的是直接累加權重最低的 M 條邊用作近似解。

而本報告使用的則是下述問題的解：

Problem 2.3

Find a subset E of edges V, $|E| = M$, s.t. the total weights is minimum and **does not cause cycle in** R

很明顯的，2.3的答案 $\leq$2.2的答案。此外，我們可以證明??使用的類 Kruskal 演算法能夠得出 2.3之解答

具體來說，??使用以下方式求解

$sol \leftarrow 0$
$DSU \leftarrow disjoint\_set(all\ nodes)$
for $e \in Sorted(Edges)$ **do**
  if $e.begin$ is red and $e.end$ is red **then**
    if $DSU.root(e.begin) == DSU.root(e.end)$ at same set **then**
      continue
  else
    $sol \leftarrow sol + e.weight$
    $DSU.union(e.begin, e.end)$

簡單來說便是不斷提取最小邊，若會構成環則放棄，否則就累加。由於 Cut Theorem 在問題 2.3成立，因此該演算法正確

Theorem 2.1: Cut-theorem

if (u, v) is an edge with minimum weight of G, then $\exists$ optimal solution of 2.3, calling E, s.t. $(u, v) \in E$

Proof. 若不存在如此的 E，則從 2.3的最佳解中隨機提取一個 E

- 若 $u \in B$ 或 $v \in B$，則隨機刪除 E 中一條邊並加入 (u, v)，此操作必不會在 R 中製造環
- 若 u, v 在 E 中屬於同一個連通塊，則存在唯一一條 u, v 路徑，隨機刪除其中一條邊並將 (u, v) 接上，此操作後 E 總權重不變大。
- 若 u, v 屬於不同連通塊，則隨機刪除 E 中一條邊並加入 (u, v)，此操作必不會在 R 中製造環

$\square$

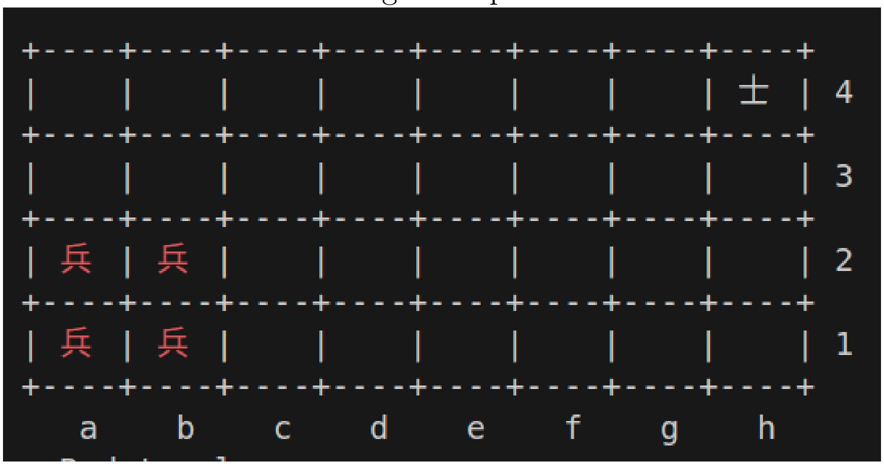
此算法設計思路與 Kruskal 相同，根據 2.1，每次取最小邊，其後將最小邊連成的兩點合併為一個點，再取新圖中的最小邊，重複此步驟直至找到 M 條邊，此算法將可求得 2.3的最佳解，即為??實作之內容。
而由於 solution of 2.3 $\leq$ solution of 2.2 $\leq OPT$，因此 admissible 性質可被保證。

3 Performance analysis and benchmark

3.1 heuristics

原先使用的 algorithm 2同樣是對於問題 2.3的逼近，且都是排序邊後取最小相加，然而遇到形如下述盤面時容易失效。如上圖所示，當所有紅子集中

Figure 1: problem board



時，algorithm 2取到的所有邊都將是紅子間計算得出，導致即使右上角的士正確往左下移動時，algorithm 2的估計依然不會減少，失去作為 heuristic 的效果。

因此 algorithm 3加入跳過重複點的機制後，可以保證至少有一條邊是取自黑子-> 紅子，對上述問題有相當程度的改善。

以下為兩種 heuristic 在 task 3-1 至 task 3-3 的執行結果

Heuristic	3-1	3-2	3-3
algorithm 2			
algorithm 3	$8 \times 10^{-4}/452$	1.13 / 3220270	4.80 / 13496393

Table 1: performance(time / heuristic calls)

3.2 record

Heuristic	3-1	3-3	SH-3
without record	8×10^{-4}/452 / 3916	4.80 / 1.3×10^8 / 3840	388 /1.1×10^9 / 4028
with record	0.004/445/3884	0.006/5917/4168	0.08/ 89156/ 4844

Table 2: performance(time / heuristic calls / memory used)