電腦對局理論 Homework 1

蔡平樂

October 15, 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：

---

procedure DFS(pos, $threshold$)
    $Stack\_init(S)$
    $Stack\_init(Prv\_States)$
    $Push(S, pos)$
    $depth \leftarrow 0$
    while not $S.empty()$ **do**
        $current \leftarrow S.pop()$
        if $current == Null$ **then**
            $depth - -$
            continue
        $record(current, limit\_depth - depth)$
        $NX \leftarrow next\_states\_gen(current.pos)$
        $Next \leftarrow sort\_and\_erase(NX, maximum = threshold - depth - 1)$
        $Push(S, Null)$
        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 $sort\_and\_erase$

$sort\_and\_erase(NX, maximum)$ 使用 Heuristic value 排序 NX，並移除所有 $value > maximum$ 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 測資上能夠將約 60 秒的時間壓縮至 15 秒內，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

procedure MINIMUMMOVEESTIMATE(pos)
  $Priority\_Queue\_init(PQ)$
  for b in $Black\_Pieces(pos)$ **do**
    for r in $Red\_Pieces(pos)$ **do**
      if b can capture r **then**
        $EnPriority\_Queue(PQ, dis(b, r))$
  for (r_1, r_2) be any pair in $Red\_Pieces(pos)$ **do**
    for r_2 in $Red\_Pieces(pos)$ **do**
      $EnPriority\_Queue(PQ, dis(r_1, r_2))$
  $N = |Red\_Pieces(pos)|$
  $p = sum$\{first N elements in PQ\}
  return x

$dis(sq\_a, sq\_b)$ 為位置 sq_a 到位置 sq_b 的最短路徑，在沒有 Duck 的情形下為 Euclidean distance，在有 Duck 時使用 Floyd-Warshall algorithm 計算。

2.2 admissible proof

Fix a board state pos, suppose OPT is the exact minimum step to capture all red pieces.
First, we transform a Board pos into a weighted, edge-colored graph G by following:

Let $B = \{b_1, b_2, ..., b_n\}$ be the set of all black pieces, and $R = \{r_1, r_2, ..., r_m\}$ be the set of all red pieces. Then $G = (V, E)$ with $E = B \cup R$, node of B and

- if $b_i \in B$ can capture $r_j \in R$, then $(b_i, r_j) \in E$ with color Black and weight $dis(b_i.location, r_j.location)$
- foreach $r_i, r_j \in R$, then $(r_i, r_j) \in E$ with color Red and weight $dis(r_i.location, r_j.location)$

Suppose we have an optimal solution.
Foreach single b_k, define M_k as the move of b_k in the optimal solution, and $R_k = \{r_1^k, r_2^k, ..., r_{n_k}^k\}$ be the red pieces captured by b_k sequentially in the optimal solution, then by the proposition of optimal solution, we have following proposition

- $(b_k, r_1^k) \in E, (r_i^k, r_{i+1}^k) \in E \forall i < n_k + 1$
- $\bigcup_k R_k = R$, each R_k disjoint.
- Let $P_k = \{(b_k, r_1^k), (r_1^k, r_2^k), ..., (r_{n_k-1}^k, r_{n_k}^k)\}$ be the path, then the union of all paths will cover all red nodes. i.e. $R \subseteq \cup_{k=1}^{n} P_k$

from triangular equation we have

$$M_k \geq weight(b_k, r_1^k) + \sum_{j=1}^{n_k-1} weight(r_j^k, r_{j+1}^k)$$

Therefore

$$Optimal\ solution\ steps = \sum_{k=1}^{n} M_k \geq \sum_{k=1}^{n} weight(b_k, r_1^k) + \sum_{j=1}^{n_k-1} weight(r_j^k, r_{j+1}^k)$$

$\sum_{k=1}^{n} weight(b_k, r_1^k) + \sum_{j=1}^{n_k-1} weight(r_j^k, r_{j+1}^k)$ is the sum of $|R|$ edges in G, and it will smaller or equal than the sum of $|R|$ edges with minimum weight at G.

2.2.1 Variant of the algorithm