

1 Problem

1.1 General Problem

We formulate our problem with the following five concepts:

- **State**(\mathbb{S}) including a **initial state** (s_0) the agent begins with
- **Actions**(\mathbb{A}) available to agent at each State
- **Transition function** ($F : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{S}$) that takes a (state, action) pair and return a new state
- **Goal Test**($GT : \mathbb{S} \rightarrow \{True, False\}$) that takes a state s and return true if $s \in S_{goal}$, the **Goal State** of the problem.
- **Path Cost** ($C : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}$) That takes two states and return cost moving from one to another

1.2 Single-player Chexers

In this section, we formally describe how the above framework fits the Chexers game.

- Denote set of pieces with $\mathbb{P} = \{(r, q, t)\}$.
 where

$r, q, -(r + q) \in [-3, 3]$ denotes the location on board.
 $t \in \{red, blue, green, block\}$ stands for type of piece.

- $\mathbb{S} := \{p_i \in \mathbb{P}, t_b | i \leq n\}$ where n is number of pieces on board, where t_b is the searching type.*
- $\mathbb{A} := \{(Move, p_i), (Jump, p_i), (Exit, p_i)\}, \forall p_i$ where $t(p_i) = t_b, |\mathbb{A}_s| \leq 6n_p \forall s$.
- $f(s, a) := s'$. where s' differs exactly by one piece $r(p_{i,s}), q(p_{i,s}) \neq r(p_{i,s'}), q(p_{i,s'})$ or $p_i \notin s'$
- $c(s, s') := 1, \forall s, s'$ if $\exists a$ such that $f(s, a) = s'$
- $gt(s) = True$ if $\nexists p_i$ such that $t(p_i) = t_b$

*Initial State s_0 given by problem specification.

2 Search

2.1 preliminary

We use the following Algorithm 1, to search for our goal. Based on our problem specification, we have all required components except for **H**(node), so we will propose one type of **H** we found to be most effective of all and compare it against **Null** $H = 0$ and **bad H** $H_{bad}(node)$.

2.2 Heuristic

As a Problem We propose the following problem definition for mapping a good heuristic value for each pieces and we define $H(s) = \sum_{p_i} h(p_i), \forall p_i$ that $t(p_i) = t_b$ after relaxing the problem to be: a piece can choose freely between **move**, **jump** regardless normal constrains and **cannot** move on blocks.

- $\mathbb{S} := \{(cost_i, position_i) | location_i \in board\}^*$,
- \mathbb{A} : for pos_i with $cost_i = \min(cost_n), \forall pos_{j \neq i}$ reachable from pos_i and $cost_j > cost_i + 1$. Update $(pos_j, cost_i + 1)$. If $\nexists pos_j$, remove $(pos_i, cost_i)$ from s and put $h(pos_i) = cost_i$
- $c(s, s') = 0$ and $f(s, a)$ follows definition in \mathbb{A}
- $gt(s') = True$ if $s' = \emptyset$

* $s_0 = \{(1, pos_i) | can_exit(pos_i) = True\} \cup \{(\infty, pos_i) | can_exit(pos_i) = False\}$

Algorithm 1 General A* algorithm

PRIORITY-QUEUE	▷ Min Priority Queue (min key)
ADD(q , key , $value$),	▷ Add (key , $value$) to q . If $value$ exists, update the key
POP(q)	▷ Pop the value with least key
GET(q , $value$)	▷ Get the key associated with a value
	▷ All above Queue operations can operate in $\Theta(1)$
NODE	
Require: EXPAND($node$)	▷ stores associated state and its parent
Require: G($node$)	▷ expand a node to get it's children
Require: H($node$)	▷ get total cost arriving this node
Require: C($node1$, $node2$)	▷ Computes an admissible estimation of cost to goal state
	▷ Give Path cost arriving node 2 from node 1

```
1: procedure A*( $problem$ ,  $initial$ )
2:    $openSet \leftarrow$  PRIORITY-QUEUE(H( $initial$ ),  $initial$ )
3:    $closedSet \leftarrow \{\}$ 
4:   while  $openSet$  is not empty do
5:      $node \leftarrow$  POP( $openSet$ )
6:     ADD( $closedSet$ ,  $node$ )
7:     if GOAL-TEST( $node$ ) is True then
8:       return  $node$ 
9:     end if
10:    for  $child$  in EXPAND( $node$ ) do
11:       $cost = G(node) + C(node, child) + H(child)$ 
12:      if  $child$  in  $closedSet$  then continue
13:      else if  $child$  not in  $openSet$  or  $cost < GET(openSet, child)$  then
14:        ADD( $openSet$ ,  $cost$ ,  $child$ )
15:      end if
16:    end for
17:  end while
18:  return no solution
19: end procedure
20: All operations  $O(1)$  unless explicitly stated
```

Heuristic problem solution Finding heuristic is straight forward with the given above problem definition. Supporting all the given operations to 1, and you should end up with a complete heuristic map.

Admissible? The algorithm is guaranteed to provide us with a admissible cost estimation for each game state for the following reason:

- The relaxed rule let pieces always able to jump. This will in all cases reduce the number of action taken.

2.3 The search