

## NATIONAL UNIVERSITY OF SINGAPORE

## SCHOOL OF COMPUTING

MIDTERM ASSESSMENT FOR  
Semester 1 AY2023/2024

## CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

October 4, 2023

Time Allowed: 75 Minutes

---

INSTRUCTIONS TO CANDIDATES

1. This assessment contains **THREE (3)** questions. All the questions are worth a total of **30 MARKS**. It is set for a total duration of **75 MINUTES**. You are to complete all 3 questions.
  2. This is a **CLOSED BOOK** assessment. However, you may reference a **SINGLE DOUBLE-SIDED A4 CHEAT SHEET**.
  3. You are allowed to use **NUS APPROVED CALCULATORS**.
  4. If something is unclear, solve the question under reasonable assumptions. State your assumptions clearly in the answer. If you must seek clarification, the invigilators will only answer questions with Yes/No/No Comment answers.
  5. You may not communicate with anyone other than the invigilators in any way.
- 

STUDENT NUMBER:

<b>A</b>								
----------	--	--	--	--	--	--	--	--

---

EXAMINER'S USE ONLY		
Question	Mark	Score
1	10	
2	10	
3	10	
TOTAL	30	

1. There are many variants of the *counting shapes* puzzle. The following is one such puzzle variant, where the objective is to find  $k$  squares (where  $k > 0, k \in \mathbb{Z}$ ).

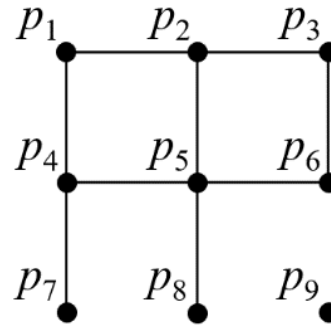
Assume that you are given (1) a set of distinct points,  $\mathbf{P} = \{p_1, p_2, \dots, p_n\}$ , where each point (or vertex),  $p_i = (x_i, y_i)$ , corresponds to a coordinate in a 2-dimensional (2D) Cartesian space, as well as (2) a set of distinct edges  $\mathbf{E} = \{e_1, e_2, \dots, e_n\}$ , where each edge,  $e_j = (p_a, p_b)$ , is an edge, or line segment, that connects  $p_a$  to  $p_b$  (undirected), such that  $p_a, p_b \in \mathbf{P} \wedge p_a \neq p_b$ . Note here that we deliberately define  $|\mathbf{P}| = |\mathbf{E}| = n$ . Further, assume that all points have integer-valued coordinates, i.e.,  $\forall p_i \in \mathbf{P}, x_i, y_i \in \mathbb{Z}$ .

The sets  $\mathbf{P}$  and  $\mathbf{E}$  define a graph, which, when represented on a standard 2D Cartesian space, may include square shapes.

Consider the following example.

Let  $\mathbf{P} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$ , and  $\mathbf{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9\}$ , where each  $p_i \in \mathbf{P}$  and  $e_i \in \mathbf{E}$  are defined in the table below on the left. The resultant graph that is formed based on  $\mathbf{P}$  and  $\mathbf{E}$  is depicted in the figure below on the right.

$i$	$p_i$	$e_i$
1	(0, 2)	$(p_1, p_2)$
2	(1, 2)	$(p_1, p_4)$
3	(2, 2)	$(p_2, p_3)$
4	(0, 1)	$(p_2, p_5)$
5	(1, 1)	$(p_3, p_6)$
6	(2, 1)	$(p_4, p_5)$
7	(0, 0)	$(p_4, p_7)$
8	(1, 0)	$(p_5, p_6)$
9	(2, 0)	$(p_5, p_8)$



Notice that within this graph, there are two squares that may be defined, i.e.,  $(p_1, p_2, p_4, p_5)$ , and  $(p_2, p_3, p_5, p_6)$ . Thus, if  $k = 1$ , then we could return either  $\mathbf{R} = ((p_1, p_2, p_4, p_5))$ , or  $\mathbf{R} = ((p_2, p_3, p_5, p_6))$ ; if  $k = 2$ , then we could only return  $\mathbf{R} = ((p_1, p_2, p_4, p_5), (p_2, p_3, p_5, p_6))$ ; and if  $k \geq 3$ , then there are no viable solutions since we can only define 2 squares (i.e.,  $\mathbf{R} = \emptyset$  or  $()$ ).

Note that with all the return values,  $\mathbf{R}$ , above, we are not concerned with the ordering of the points within each 4-tuple corresponding to a square, nor are we concerned with the ordering of the squares (i.e., 4-tuples) within  $\mathbf{R}$ .

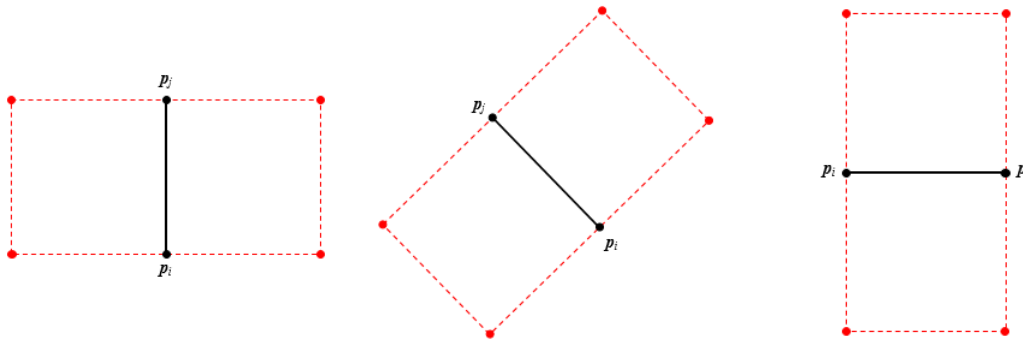
Your task in this question is to define the *search problem* corresponding to the above puzzle, which is to determine the resultant tuple of 4-tuples,  $\mathbf{R}$ , when given  $\mathbf{P}$ ,  $\mathbf{E}$  and  $k$ , such that each 4-tuple,  $(p_a, p_b, p_c, p_d) \in \mathbf{R}$  corresponds to a unique square defined by  $\mathbf{P}$  and  $\mathbf{E}$ , and where  $|\mathbf{R}| = k$ .

*Your search problem specification should be efficient (i.e., better than brute-force search).*

(i) [8 marks] Assume that (1) **Action Costs** are all 1, and that (2) the **Goal Test** simply evaluates the expression  $found = k$  (where  $found$  is a variable that corresponds to the number of unique squares currently found). Complete the search problem formulation by defining the **State**, **Initial State**, **Actions** and **Transition Model**.

### Solution:

The general idea is to have each state correspond to the evaluation of one edge in  $E$ . The search tree is therefore just a list (i.e., one path) of depth  $|E|$ ; there are no branches. With each edge,  $(p_i, p_j)$ , regardless of the orientation of the edge, two squares may potentially be formed – for example cases, refer to the diagram below. Notice that the two squares are formed by drawing edges that are parallel and perpendicular to the given edge.



### Assumptions and globals

- Assume that the set  $E$  is ordered – i.e., there is a lexicographical ordering over the each  $(p_i, p_j) \in E$ . This means that each  $E[i]$  always refers to a one specific, distinct, edge in  $E$ .
- Let  $R$  denote a global variable containing all the squares currently found within the search – i.e.,  $R$  will contain all squares that are consistent with  $V$  and  $E$ .
- Assume hash table access for elements in  $V$ ,  $E$ , and  $R$  – i.e.,  $O(1)$  lookup complexity.

### State

- With each state, we store the following variables:
  - The edge index:  $i \in \mathbb{Z}$ ,  $i \in [0, |E|)$ .
  - The number of squares currently found:  $found = |R|$ .

### Initial state

- $i = NULL$  // generation and testing of potential squares in actions/transition
- $found = \{\}$  // note that each element of  $R$  will be a 4-tuple of vertices defining a square

### Actions

- There will only be one possible action for each state: increment  $i$  – i.e.,  $i = i + 1$ .

### Transition model

- With the updated  $i$ , consider  $E[i] = (p_a, p_b)$ , where  $p_a = (x_a, y_a)$ ,  $p_b = (x_b, y_b)$ .
- Generate the additional two vertices for each of the two potential squares.
  - This is an  $O(1)$  operation given that these vertices can be determine based on 8 possible cases governing the relationship between  $x_a$  and  $x_b$ ,  $y_a$  and  $y_b$  (e.g.,  $x_a > x_b \wedge y_a > y_b$ ). Do note that the 9th case (i.e.,  $x_a = x_b \wedge y_a = y_b$ ) does not need to be considered since for each edge,  $(p_a, p_b)$ ,  $p_a \neq p_b$ . (See Appendix A – pp 12-13.)
- For each of the two squares, check that the two new vertices exist in  $V$ , and if so, that the respective edges exists in  $E$ . These are all assumed to be  $O(1)$  operations.
- If the square is valid, then check that it does not already exist in  $R$  (to avoid double accounting the same square) before then adding it and updating  $found$ . Again, these are  $O(1)$  operations.

/\* A simpler implementation would be to consider all possible combinations of 4 vertices based on  $V$  – i.e., to determine (as a pre-processing step) the  $n$  choose 4 possible distinct 4-tuples comprising vertices from  $V$  (recall that  $n = |V| = |E|$ ).

Each action would simply consist of traversing to the next 4-tuple of vertices, while each transition would involve the traversal of each of the 6 choose 4 possible edges linking the 4 vertices (given that there are 6 possible edges linking those 4 vertices). For each set of 4 edges, we would then have to check (i) if all 4 edges exist (in  $E$ ) – which has  $O(1)$  complexity, and (ii) that they form a square – i.e., an  $O(1)$  operation that checks that the 4 edges have the same length, and that the other 2 missing edges are diagonals that form Pythagorean triplets with edges defining the square.

However, this search would, in the worst case, require the traversal of all  $n$  choose 4 states, which gives us  $O(n^4)$  complexity. In comparison, the given answer, which traverses existing edges instead, has complexity  $O(n)$ . \*/

/\* Further, note that there are two possible interpretations of the formation of a square based on the given description. That is, what we consider as line segment candidates may be as follows.

1. Each side of a square must correspond to a single edge, and not composite ones that are parallel and extensions (e.g.,  $(p_1, p_2)$  and  $(p_2, p_3)$ ).
2. Each side of a square may correspond to the extensions mentioned above.

The answer specified assumes (1). \*/

**(ii) [2 marks]** Define the **runtime** complexities of your **Actions** and **Transition Model** functions defined in (i). *Note that you will **NOT** be awarded any marks for this part if your search specification does not construct a viable search tree, which would allow an uninformed search algorithm to solve the problem.*

### Solution:

The given actions function has  $O(1)$  complexity.

The given transition model has  $O(1)$  complexity.

**2a.** A heuristic,  $h$ , is **2-consistent** if for all pairs of nodes  $(n, n')$  within the graph where  $n'$  is reached from  $n$  via 2 actions, the pair  $(n, n')$  must satisfy  $h(n) \leq c(n, n') + h(n')$ , where  $c(n, n')$  corresponds to the shortest distance from node  $n$  to node  $n'$ .

(i) [3 marks] Prove or disprove the following.

If a heuristic,  $h$ , is **consistent**, then it is **2-consistent**.

**Solution:**

**True.**

Let  $B$  denote the set containing all pairs of nodes that exactly two actions apart.

For each  $(n, n') \in B$ , we have the following.

- As  $n'$  is two actions away from  $n$ , there must exist at least one node  $b$  between  $n$  and  $n'$ .

For any  $b$  between any  $(n, n') \in B$ , asserting consistency of  $h$ , we have:

- A.  $h(b) \leq c(b, n') + h(n')$
- B.  $h(n) \leq c(n, b) + h(b)$

Substituting (A) into (B), which maintains the inequality in (B), we have the following.

- C.  $h(n) \leq c(n, b) + c(b, n') + h(n') \equiv h(n) \leq c(n, n') + h(n')$

Thus, the statement is true.

/\* Also note that the values for  $h(n)$ ,  $h(b)$ , and  $h(n')$  must be constant for any given graph. Consider the following.

Optimal  $b$  cases:

- Let  $b'$  be a node that defines an optimal-cost path between  $n$  and  $n'$ .
- Asserting consistency of  $h$ , we have:
  - A.  $h(b') \leq c(b', n') + h(n')$
  - B.  $h(n) \leq c(n, b') + h(b')$
- Substituting (A) into (B), which maintains the inequality in (B), we have the following.
  - C.  $h(n) \leq c(n, b') + c(b', n') + h(n') \equiv h(n) \leq c(n, n') + h(n')$
- This shows that for each  $(n, n') \in B$ , the given statement is true for  $b$  that forms an optimal-cost path between  $n$  and  $n'$ .

Non-optimal  $b$  cases:

- Asserting optimal-cost path from  $n$  to  $n'$  via  $b'$ , then we know that for all other  $b$  connecting  $n$  and  $n'$ , the following is satisfied.
- $c(n, b') + c(b', n') \leq c(n, b) + c(b, n')$
- Consequently, we have the following.
- $h(n) - h(n') \leq c(n, b') + c(b', n')$  // rearrangement of positive terms
- $h(n) - h(n') \leq c(n, b') + c(b', n') \leq c(n, b) + c(b, n')$  // asserting (D)
- $h(n) \leq c(n, b) + c(b, n') + h(n') \equiv h(n) \leq c(n, n') + h(n')$  // for non-opt.  $b$
- This shows that for each  $(n, n') \in B$ , the given statement is true for  $b$  that forms a path with non-optimal cost between  $n$  and  $n'$ .

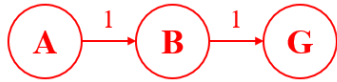
Given the statement holds for any  $b$ , we have shown that it is true. \*/

(ii) [2 marks] Prove or disprove the following.

If a heuristic,  $h$ , is **2-consistent**, then it is **consistent**.

**Solution:**

Consider the following graph where **A** is the initial state and **G** is a goal state.

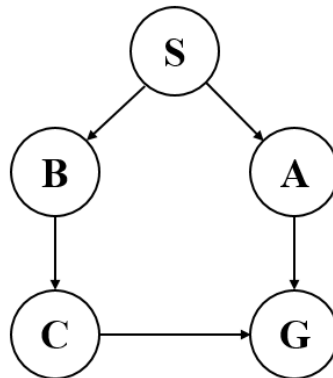


Let  $h(A) = h(B) = 0$ , and  $h(G) = 10$ .

Since  $h(A) \leq c(A, G) + h(G)$ ,  $h$  is 2-consistent.

However,  $h(G) = 10$  implies that  $h$  is inadmissible, and thus, not consistent.

2b. [2 marks] Consider the directed graph depicted in the figure below. Let **S** be the initial state and **G** be the goal state.



Trace the **Iterative Deepening Search (IDS) tree search** algorithm for the above graph. Specify the order in which the nodes are popped off the frontier. In cases where a tie-breaker is necessary, assume that nodes are pushed onto the frontier in alphabetical order.

**Solution:**

**S-S-B-A-S-B-C-A-G** (if starting at depth 0)

**S-B-A-S-B-C-A-G** (if starting at depth 1)

Note that both answers above are accepted.

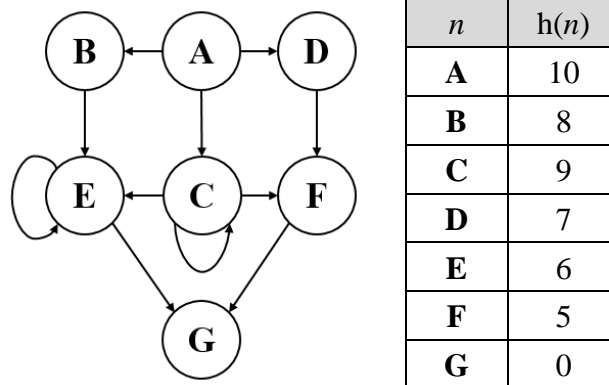
// The working is as follows.

// Depth 0: S

// Depth 1: S, B, A

// Depth 2: S, B, C, A, G (goal found; return)

**2c. [3 marks]** Consider the directed graph depicted in the figure below on the left, and the associated heuristic values as defined by the heuristic function,  $h$ , in the table below on the right. Assume that the **Goal Test** will return *False* on all nodes except node **G**.



Trace the **Local Beam Search** algorithm with beam size,  $k = 3$ . Specifically, write down the contents of the beam at each iteration. Start your beam with nodes **A**, **B**, and **C**.

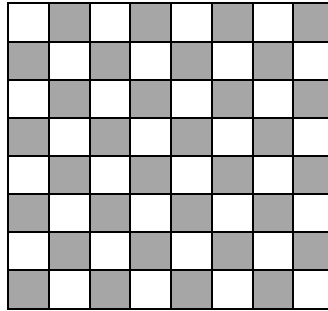
**Solution:**

Beam trace is as follows.

- **Iteration 1: A, B, C**
  - Candidates via A: B(8), C(9), D(7)
  - Candidates via B: E(6)
  - Candidates via C: C(9), E(6), F(5)
- **Iteration 2: E, E, F (if duplicates allowed) or D, E, F (if no duplicates allowed)**
  - Candidates via E: E(6), G(0)
  - Candidates via F: G(0)
  - Candidates via D: F(5)
- **Iteration 3: Goal found (via E)**
  - E occurs in both versions of the beam after iteration 2
  - This means that G will be generated and found with either beam

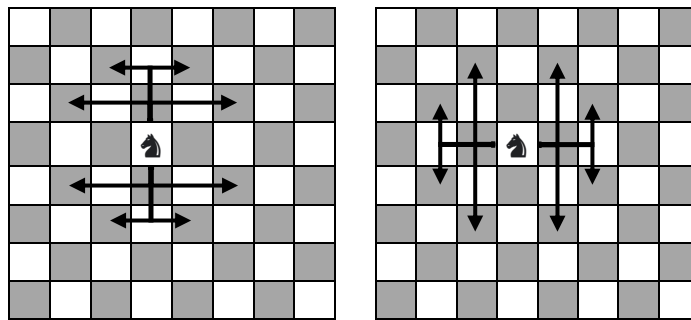
3. Consider the following puzzle that is played out on a Chess board.

Chess is typically played on an 8-by-8 board, which is depicted below.



However, assume that the current puzzle is played on a Chess board of infinite size.

A Knight (♠) is a Chess piece that moves in an “L” shape. More precisely, the Knight piece can move two cells vertically followed by one square horizontally, or two cells horizontally followed by one cell vertically. The Knight piece’s movement is summarised in the figures below.



In this puzzle, a Knight starts at a given Chess board cell,  $(x_s, y_s)$ , and must find the **shortest path** to a specified *goal* cell at  $(0, 0)$  using the **A\* Search** algorithm.

Do note that the Chess board in this puzzle will contain several obstacles. The Knight may **NOT** occupy any square that is occupied by an obstacle, although it may jump over the obstacles. For example, a Knight is allowed to move from cell  $(x, y)$  to cell  $(x+2, y+1)$  even if square  $(x+1, y)$  contains an obstacle.

Assume that the **cost** of any action taken by the Knight is equal to 1.

Further, assume that **two heuristics**,  $h_1$  and  $h_2$ , have been defined.

The heuristic,  $h_1$ , is defined as follows.

- $h_1(n) = 0$  if  $n = \text{goal}$ , where  $n$  and  $\text{goal}$  are coordinates on the Chess board
- $h_1(n) = 1$  if  $(n \neq \text{goal}) \wedge (\text{ManhattanDistance}(n, \text{goal}) \% 2 = 1)$
- $h_1(n) = 2$  if  $(n \neq \text{goal}) \wedge (\text{ManhattanDistance}(n, \text{goal}) \% 2 = 0)$

The heuristic,  $h_2$ , is defined as follows.

- $h_2(n) = \lceil \text{ManhattanDistance}(n, \text{goal}) / 3 \rceil$



Note that the **Manhattan distance** (i.e., the output of the *ManhattanDistance* function) between two cells,  $(x_1, y_1)$  and  $(x_2, y_2)$ , is given by  $|x_2 - x_1| + |y_2 - y_1|$ .

Consequently, it should be noted that  $h_1$  is admissible since every Knight's move will change the Manhattan distance to the goal from odd to even, or from even to odd. The heuristic  $h_2$  is also admissible since every Knight's move covers a Manhattan distance of 3, but not every cell that has a Manhattan distance of 3 away may be reached via a Knight's move.

(i) [3 marks] Prove or disprove the **admissibility** and **consistency** of the following heuristic.

$$h_3(n) = h_1(n) + h_2(n)$$

**Solution:**

The heuristic  $h_3$  is **not admissible**.

Consider the following counterexample.

Let  $n$  correspond to a state where a Knight is at square (2, 1). Consequently, we have  $h^*(n) = 1$ . However, we have  $h_3(n) = h_1(n) + h_2(n) = 1 + 1 = 2$ . Since  $h_3(n) > h^*(n)$ ,  $h_3$  is not admissible.

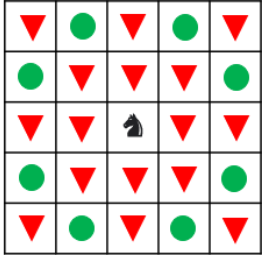
Since  $h_3$  is not admissible,  $h_3$  is **not consistent**.

(ii) [3 marks] Let the **Chebyshev distance** between two cells,  $(x_1, y_1)$  and  $(x_2, y_2)$ , be defined by  $\text{MAX}(|x_2 - x_1|, |y_2 - y_1|)$ . Assuming that the function  $\text{ChebyshevDistance}(a, b)$  determines the Chebyshev distance between cells  $a$  and  $b$ , prove or disprove the **admissibility** and **consistency** of the following heuristic.

$$h_4(n) = \lceil \text{ChebyshevDistance}(n, \text{goal}) / 2 \rceil$$

### Solution:

#### Proof of admissibility

- Every Knight's move under  $h_4$  covers a Chebyshev distance of 1 or 2, whereas every actual move that a Knight may make under the original rules only covers a subset of destination cells have a Chebyshev distance of 2. We loosen the Knight's move constraint so that every square a Chebyshev distance of 1 or 2 away can be reached in 1 move. Refer to the diagram on the right. The cells containing circles depict where the Knight may move to under the original rules. However, under the relaxed rules, the Knight may also move to the cells containing triangles.
- 
- With this relaxed Knight move constraint, every cell of Chebyshev distance  $2k$  away from the goal can be reached in  $k$  moves, where  $k \in \mathbb{Z}^+$ . Every cell of distance  $2k+1$  away from the goal requires at least  $k+1$  moves to reach.
  - The above is a mathematical description of the  $h_4$  heuristic.
  - As this heuristic **represents a relaxation** of the Knight's move constraint, this would not overestimate the true number of moves required to reach the goal for any square, and hence the heuristic is **admissible**.

#### Proof of consistency

- Under the relaxation given by  $h_4$ , **every Knight's move covers a Chebyshev distance of 1 or 2**, with all moves made having a uniform cost of 1 – i.e., each move between a parent and its successor has a cost of 1. (Note the distinction between distance and cost.)
- Given a state  $n'$ , which is a successor of state  $n$ . The following general cases are possible.
  - $\text{ChebyshevDistance}(n, \text{goal}) - \text{ChebyshevDistance}(n', \text{goal}) = 2$ .
    - Given the above, we have  $h_4(n) - h_4(n') = \lceil a/2 \rceil - \lceil (a-2)/2 \rceil = 1$ , where  $a = \text{ChebyshevDistance}(n, \text{goal})$ . (Assume the same about  $a$  below.)
    - Consequently,  $\text{cost}(n, n') + h_4(n') = 1 + (h_4(n) - 1) = h_4(n)$ .
  - $\text{ChebyshevDistance}(n, \text{goal}) - \text{ChebyshevDistance}(n', \text{goal}) = 1$ .
    - Here, we have  $h_4(n) - h_4(n') = \lceil a/2 \rceil - \lceil (a-1)/2 \rceil = 1$  when  $a$  is odd, or else when  $a$  is even, we have  $h_4(n) - h_4(n') = \lceil a/2 \rceil - \lceil (a-1)/2 \rceil = 0$ .
    - Consequently, when  $a$  is odd, we have the same outcome as Case (A).
    - And, when  $a$  is even,  $\text{cost}(n, n') + h_4(n') = 1 + h_4(n) > h_4(n)$ .
  - $\text{ChebyshevDistance}(n, \text{goal}) - \text{ChebyshevDistance}(n', \text{goal}) = 0$ .
    - Here, we have the same outcome as Case (B) above, when  $a$  is even.
  - $\text{ChebyshevDistance}(n, \text{goal}) - \text{ChebyshevDistance}(n', \text{goal}) = -1$  or  $-2$ .
    - When the result is  $-2$ , we have  $h_4(n) - h_4(n') = \lceil a/2 \rceil - \lceil (a+2)/2 \rceil$ , which results in  $\text{cost}(n, n') + h_4(n') = 1 + (h_4(n) + 1) > h_4(n)$ . And when the result is  $-1$ , we similarly have  $\text{cost}(n, n') + h_4(n') > h_4(n)$ .

In all the possible cases, we see that the **consistency constraint**  $h_4(n) \leq \text{cost}(n, n') + h_4(n')$  **holds..**

(iii) [4 marks] Define an admissible heuristic  $h_5$  that dominates  $h_4$ . The heuristic must be defined mathematically, and given an infinite board, must differ from  $h_4$  in an infinite number of cells. Provide the justification that (1)  $h_5$  is admissible, (2)  $h_5$  is dominant over  $h_4$ , and (3)  $h_5$  differs from  $h_4$  in an infinite number of cells (i.e., you should show that  $h_5$  does not, trivially, only improve on  $h_4$  for a finite number of points). *Note that your heuristic must not be the (abstract) optimal heuristic,  $h^*$  or any function over it.*

**Solution:**

$$h_5 = \text{MAX}(h_2, h_4)$$

**Proof of admissibility**

- As  $h_2$  and  $h_4$  are both admissible, for any state  $n$ ,  $\text{MAX}(h_2(n), h_4(n)) \leq h^*(n)$ .

**Proof of dominance**

- Given that  $h_2 \neq h_4$ , for any state  $n$ ,  $\text{MAX}(h_2(n), h_4(n)) \geq h_2(n) \wedge \text{MAX}(h_2(n), h_4(n)) \geq h_4(n)$ .
- Therefore, for any state  $p$ , where  $h_2(p) > h_4(p)$ ,  $\text{MAX}(h_2(p), h_4(p)) > h_4(p)$ , while for all other states,  $\text{MAX}(h_2(p), h_4(p)) = h_4(p)$ . Thus,  $h_5 = \text{MAX}(h_2, h_4)$  dominates  $h_4$ .

**Proof of  $h_4$  and  $h_5$  differing for infinitely many states**

- Consider the cells in the set  $S$ , which have coordinates in the form  $(6k, 6k)$ , where  $k \in \mathbb{Z}$ .
- Here, for any  $s \in S$ , we have  $h_4(s) = 3k$ , while  $h_5(s) = \text{MAX}(3k, 4k) = 4k$ . As  $k$  ranges from  $(-\infty, \infty)$ ,  $h_4$  and  $h_5$  differ in infinitely many cells.

END OF PAPER

## BLANK PAGE

**Appendix A**

There are 8 possible cases to consider, with one case omitted since, for each edge  $(p_a, p_b)$ , we have  $p_a, p_b \in P \wedge p_a \neq p_b$ .

**Case 1:**  $x_i = x_j \wedge y_i < y_j$  (full example in **Appendix A.1**)

**Case 2:**  $x_i = x_j \wedge y_i = y_j$  (omitted case given constraint above)

**Case 3:**  $x_i = x_j \wedge y_i > y_j$

**Case 4:**  $x_i > x_j \wedge y_i < y_j$  (full example in **Appendix A.2**)

**Case 5:**  $x_i > x_j \wedge y_i = y_j$

**Case 6:**  $x_i > x_j \wedge y_i > y_j$

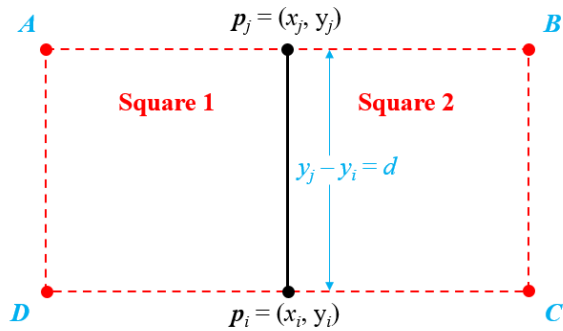
**Case 7:**  $x_i < x_j \wedge y_i < y_j$

**Case 8:**  $x_i < x_j \wedge y_i = y_j$  (full example in **Appendix A.3**)

**Case 9:**  $x_i < x_j \wedge y_i > y_j$

Do note that these cases may be further generalised (into fewer cases). However, one could argue that such a generalisation simply condenses the algorithm specification, and not complexity (since each case already has  $O(1)$  complexity).

The only requirement for the question is the functional abstraction about the generation of the vertices and edges corresponding to the two potential squares, and that this functionality has  $O(1)$  complexity.

**Appendix A.1 – Case 1:**  $x_i = x_j \wedge y_i < y_j$ 

**Square 1** new vertices:

**A:**  $(x_j - d, y_j)$

**D:**  $(x_i - d, y_i)$

**Square 1** new edges:

$(p_j, A)$

$(A, D)$

$(D, p_i)$

**Square 2** new vertices:

**B:**  $(x_j + d, y_j)$

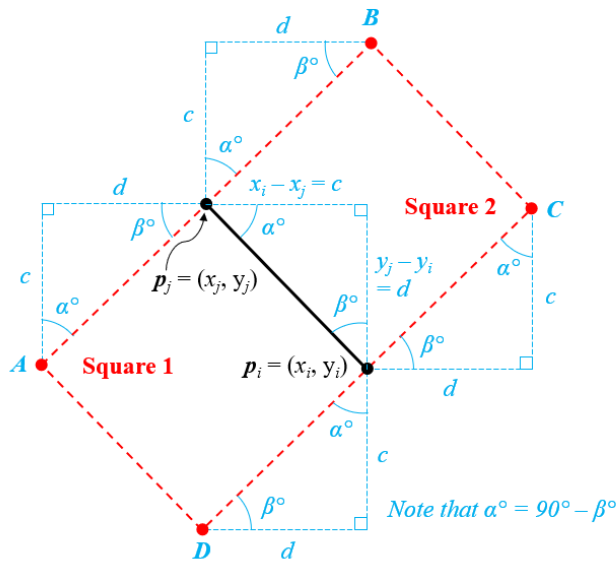
**C:**  $(x_i + d, y_i)$

**Square 2** new edges:

$(p_j, B)$

$(B, C)$

$(C, p_i)$

**Appendix A.2 – Case 4:**  $x_i > x_j \wedge y_i < y_j$ 

**Square 1** new vertices:

**A:**  $(x_j - d, y_j - c)$

**D:**  $(x_i - d, y_i - c)$

**Square 1** new edges:

$(A, p_j)$

$(A, D)$

$(D, p_i)$

**Square 2** new vertices:

**B:**  $(x_j + d, y_j + c)$

**C:**  $(x_i + d, y_i + c)$

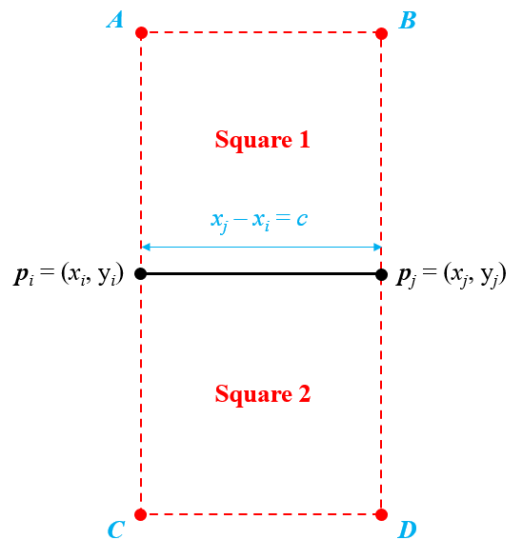
**Square 2** new edges:

$(p_j, B)$

$(B, C)$

$(p_i, C)$

## BLANK PAGE

Appendix A.3 – Case 8:  $x_i < x_j \wedge y_i = y_j$ **Square 1** new vertices: $A: (x_i, y_i + c)$  $B: (x_j, y_j + c)$ **Square 1** new edges: $(p_i, A)$  $(A, B)$  $(B, p_j)$ **Square 2** new vertices: $C: (x_i, y_i - c)$  $D: (x_j, y_j - c)$ **Square 2** new edges: $(p_i, C)$  $(C, D)$  $(D, p_j)$

BLANK PAGE

---