NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT FOR
Special Term (Part 1) AY2022/2023

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE
SOLUTIONS

June 16, 2023 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number only. Do not write your name.
2. This assessment contains FOUR (4) questions. All the questions are worth a total of 60 MARKS. It is set for a total duration of 120 MINUTES. You are to complete all 4 questions.
3. This is a CLOSED BOOK assessment. However, you may reference a SINGLE DOUBLE-SIDED A4 CHEAT SHEET.
4. You are allowed to use NUS APPROVED CALCULATORS.
5. 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.
6. You may not communicate with anyone other than the invigilators in any way.

STUDENT NUMBER: ______________________________

EXAMINER'S USE ONLY		
Question	Mark	Score
1	10	
2	17	
3	14	
4	19	
TOTAL	60	

1a. [3 marks] Two search algorithms have the same behaviour if and only if they fulfil the following conditions:

- They expand the same sequence of nodes.
- They return the same answer when the same graph is used.

Define a heuristic function, h, that when used with A* graph search version 3, will behave the same way as a similar graph implementation for Breadth-first Search (BFS) on problems with uniform and positive action costs. Assume the same tie-breaking rules are applied. If you believe that no such heuristic function may be defined, simply answer “not applicable”.

Solution:

Let h(goal) = 0, and h(non-goal) = 1.

// Note that h(n) = 0 for any n does not work because BFS applies an early goal test.
// With such a heuristic A* would expand more nodes than BFS.

1b. [2 marks] Suppose that we run a greedy best-first search algorithm with h(n) = -g(n). What kind of uninformed or informed search algorithm would this emulate when applied to problems with uniform and positive action costs?

Solution:

DFS.

1c. [2 marks] Which of the following is optimal when applied to a problem with an infinite-depth search space that contains exactly one reachable goal, and that also has a space complexity linear in the depth of the search tree? (Assume the use of Graph Search Ver. 2.)

Option **A**: Iterative Deepening Search with backtracking (IDS).
Option **B**: Breadth-first Search (BFS).
Option **C**: Depth-first Search with backtracking (DFS).
Option **D**: Depth-limited Search with backtracking (DLS).
Option **E**: None of the above.

Note that you may pick ***more than one*** *option, except in the case where you pick option* **E**.

Solution:

A.

// While A and B both satisfy the optimality condition, B does not satisfy the space constraint.

1d. [3 marks] Which of the following statements are true?

Option **A**: A consistent heuristic will dominate an admissible but inconsistent heuristic.

Option **B**: A tree search implementation of Uniform Cost Search (UCS) is complete on finite search trees where all action costs are greater than 0.

Option **C**: A tree search implementation of UCS is complete on finite search trees where all action costs are greater than 0, and greater than some small positive constant ε.

Option **D**: Under A* graph search version 2 where an admissible heuristic is utilised, a node can never be repeatedly visited.

Option **E**: Under A* graph search version 3 where a consistent heuristic is utilised, a node may be repeatedly visited.

Option **F**: None of the above.

Note that you may pick ***more than one*** *option, except in the case where you pick option* **F**.

Solution:

B and C.

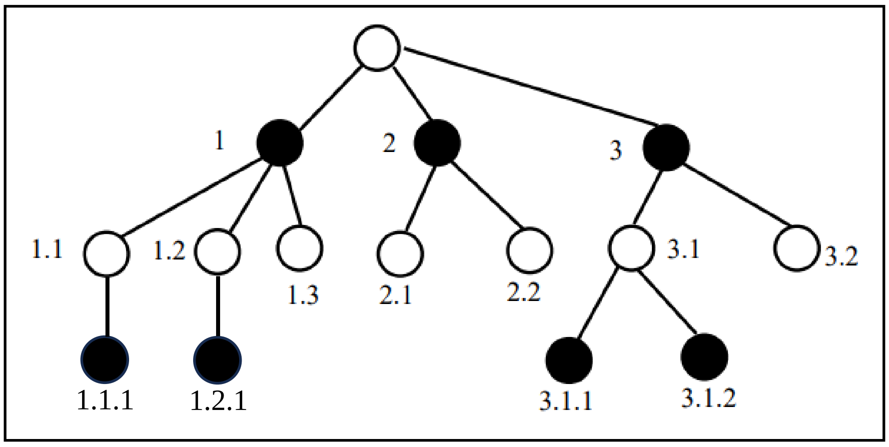
// ¬A & D: simple counterexamples. B & C given finite tree. ¬E since nodes do not repeat.

2a. For Parts **2a(i)** and **2a(ii)** below, assume an Alpha-Beta (α-β) Pruning algorithm implementation that prefers nodes on the left – i.e., that adopts left-to-right ordering.

Consider the following node labelling scheme:

- A node at depth *k* is assigned a ***sequence*** of positive integers of length *k*.
- The root node is labelled using the empty sequence (), since it is at depth 0.
- The label of a child node is the label of the parent node, appended by the move number that brings us from the parent node to the child node.

Example: Consider the figure below. The node labelled "3.1.2" is at depth 3 and can be reached by taking the third move from the root node, followed by the first move, followed by the second move.



(i) [3 marks] A node is said to be **Type 1** if all integers in its label sequence are all 1 (i.e., no other integers appear in its label). Therefore, "1", "1.1", "1.1.1", etc, are all **Type 1** nodes, but "1.2" and "3.2" are not **Type 1** nodes.

For any given tree, ***T***, all **Type 1** nodes will not be pruned.

Is the above statement true or false? Provide a rationale for your answer.

Solution:

Notice that **Type 1** nodes correspond to the first path explored. For all nodes along this first path, the condition $\alpha < \beta$ is always satisfied. On the other hand, the condition for pruning is $\alpha \geq \beta$. Hence, no pruning occurs.

/*
The reason $\alpha < \beta$ always holds is as follows.

– Base case:
    True on initialisation, since $\alpha = -\infty$ and $\beta = \infty$
– Inductive step:
    When the first action is taken, this action has not been explored before. Hence, its value is unknown, thus α and β cannot be updated. Therefore, $\alpha < \beta$ remains unchanged and still holds.
– Termination step:
    At a leaf node, α or β may be updated (depending on if the node in question is a MAX or MIN node). However, the node has already been explored before α or β is updated. Thus, all nodes (including the leaf node) are explored.
*/

(ii) [7 marks] Prove or disprove the following statement.

For any given tree, ***T***, any **Type 1** node with $d > 0$ children will have all its d children explored. (Notice that in the example tree from Part **2a(i)** above, the node "1" has 3 children, namely: "1.1", "1.2", and "1.3". Note that "1.1.1" and "1.2.1" are not child nodes of node "1".)

Solution:

The statement is **true**.

We have established that the first child of a **Type 1** node is also a **Type 1** node, and hence must be explored. We are left to show that the other $d - 1$ child nodes must also be explored. We shall show this by contradiction.

Suppose the contrary, i.e., the k-th successor is not explored.

For this to occur, we must observe that $\alpha \geq \beta$. Recall that the first child is explored. Hence, from among the 2nd, 3rd, …, $(k - 1)$-th successors, exactly one of α or β must be updated such that the k-th successor is not explored.

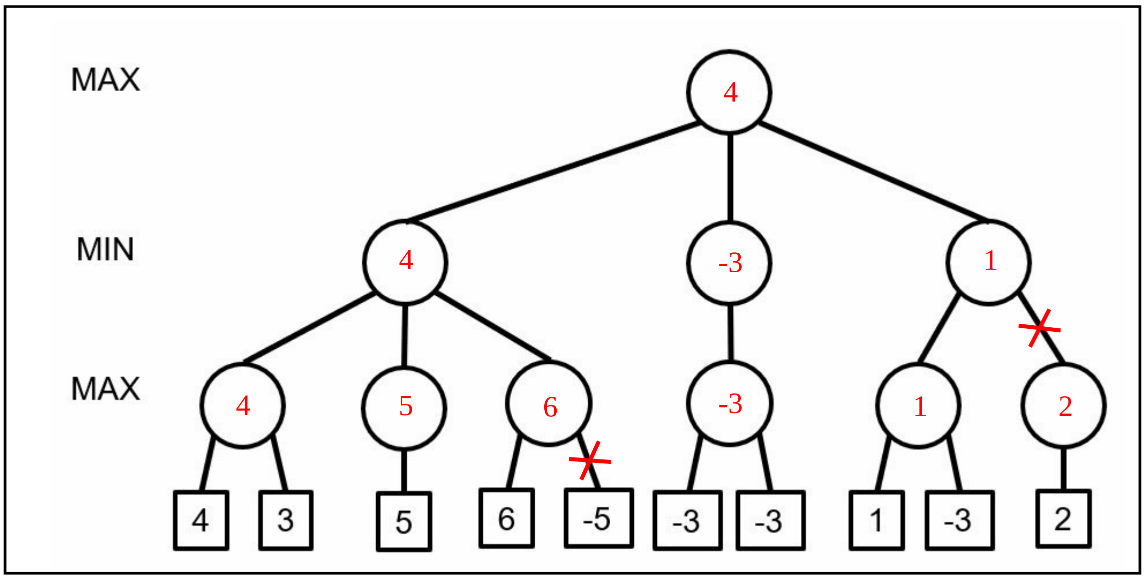
Suppose α is updated. The update rule specifies that α can only be revised upwards (via recursion resolution). However, for **Type 1** nodes, β is ∞. It is not possible to revise α upwards a finite number of times using a finite value, and yet α must become larger than β (which is ∞). Hence, the k-th successor cannot be pruned given this contradiction.

A similar argument can be used to show that this is the case when β is updated (where α would be $-\infty$).

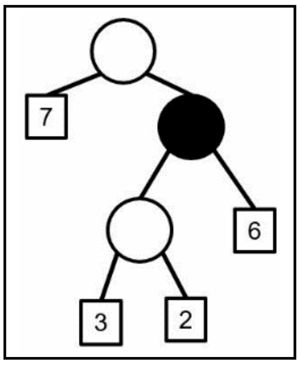
Note that $\alpha = -\infty$ and $\beta = \infty$ follows from the argument made in **2a(i)**.

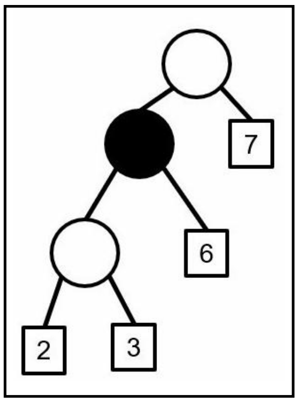
2b. [3 marks] Trace the Alpha-Beta (α-β) Pruning algorithm from left to right on the given tree. Fill all nodes with the appropriate utility values and cross out all pruned edges.

Solution:



2c. [4 marks] A game tree is ***best-score-first*** ordered if for all non-terminal nodes, its first successor represents the best possible move for that position. The tree on the right is an example of a game tree that is ***best-score-first*** ordered. Notice that at the first node (which is a MAX node), the first action is best as it leads to a utility score of 7 while the second action leads to a utility score of 3. In the second node (which is a MIN node), the first action is best as it leads to a utility score of 3 while the second action leads to a utility score of 6. In the third node (which is a MAX node), the first action is once again best. Hence, the above tree is ***best-score-first*** ordered.





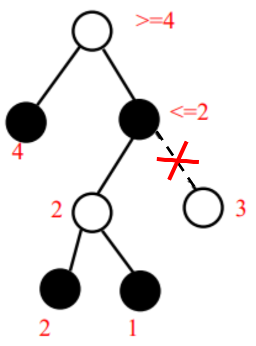
Note that a different agent can use a different move-ordering heuristic to generate a different game tree for the same game. For example, the game tree on the left might be generated instead. This is a permutation of the first tree. Note that there are many possible permutations for a given game tree, all of which corresponds to a slightly different move-ordering heuristic being used. Notice how in the first tree (which is ***best-score-first*** ordered), one branch is pruned by the α-β Pruning algorithm. However, in the second tree (which is a permutation, but not ***best-score-first*** ordered), no branch is pruned by the α-β Pruning algorithm. Hence, fewer nodes are visited with the first tree than the second tree.

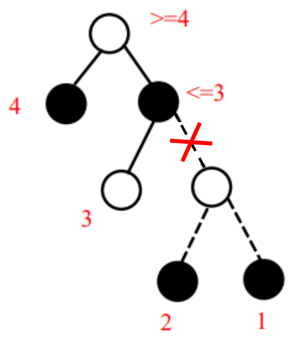
Prove or disprove the following statement.

Let ***T*** be a ***best-score-first*** ordered game tree. Suppose ***T'*** is a permutation of ***T*** such that it is not ***best-score-first*** ordered. If the α-β Pruning algorithm visits N nodes given ***T***, then it will visit $M \geq N$ nodes given ***T'***.

Solution:

False. The following is a counterexample. The tree on the left is ***best-score-first*** ordered, but the tree on the right, which is not ***best-score-first*** ordered, has less nodes explored.





3a. Consider the following constraint satisfaction problem (CSP) for Parts **3a(i)** and **3a(ii)**.

We have 3 variables x_1, x_2, and x_3, with domains corresponding to integers ranging from 1 to 7 (inclusive). We also have the following constraints.

- $x_1 = 2x_2$
- $x_2 = x_3 + 1$

(i) [5 marks] Trace the execution of the AC3 algorithm as a pre-processing step. Assume the following initial arc queue: [(x_1, x_2), (x_2, x_3), (x_3, x_2), (x_2, x_1)]. Note that if an arc ***A*** is already in the queue, we do not re-queue arc ***A***.

Note that there will be no error carried forward (ECF) considered.

Solution:

Arc Popped (x_i, x_j)	Updated Domain of First Variable (x_i)	Arc(s) Enqueued [Write "NA" if no arcs are enqueued]
(x_1, x_2)	x_1 = [2, 4, 6]	NA
(x_2, x_3)	x_2 = [2, 3, 4, 5, 6]	(x_1, x_2)
(x_3, x_2)	x_3 = [1, 2, 3, 4, 5, 6]	NA
(x_2, x_1)	x_2 = [2, 3]	(x_3, x_2)
(x_1, x_2)	x_1 = [4, 6]	NA
(x_3, x_2)	x_3 = [1, 2]	NA

Note that you need not use all the given rows.

(ii) [2 marks] Apply the Backtracking algorithm (with forward-checking) on the pre-processed version of the queue – i.e., to your answer from Part **3a(i)**. Use the ordering x_1, x_2, and x_3 when deciding variable order, and use the largest value in the domain first when deciding value order.

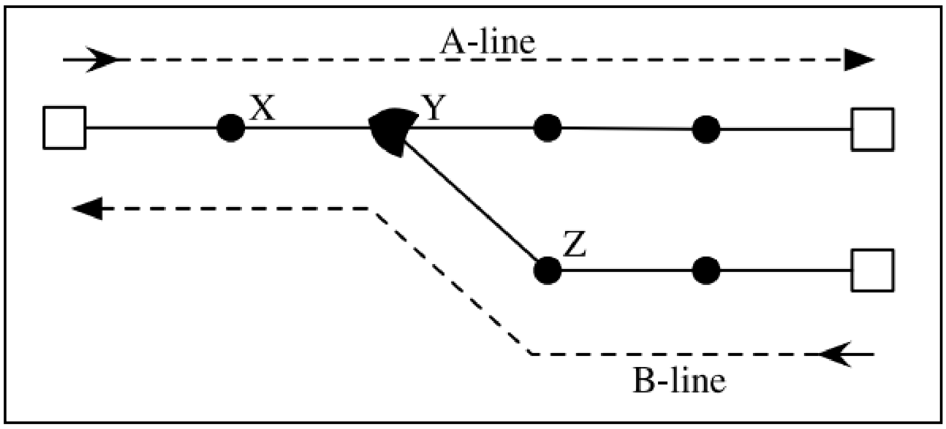
Note that there will be no error carried forward (ECF) considered.

Solution:

Variable to Assign	Assigned Value	Updated Domains [Note that you need only update domains that have changed.]
x_1	6	x_2 = [3]
x_2	3	x_3 = [2]
x_3	2	*Solved*

Note that you need not use all the given rows.

3b. Two trains, ***A-line*** and ***B-line*** use the simple rail network shown below. The objective is to schedule a single daily departure for each train. Each departure will be on the hour (i.e., exactly 1300 hrs, 1400 hrs, 1500 hrs, etc), between 1300 hrs and 1900 hrs (inclusive). The trains run at identical, constant speeds, and each segment takes a train one hour to cover.



We can model this problem as a CSP in which the variables represent the departure times of the trains from their source stations:

- Variables: ***A***, ***B*** // representing the ***A-line*** and ***B-line*** trains, respectively.
- Domains: {1,2,3,4,5,6,7} // 1 signifies 1300 hrs, 2 signifies 1400 hrs, and so on.

For example, if ***A*** = 1 and ***B*** = 1, then both trains leave at 1300 hrs.

The complication is that the trains cannot pass each other in the region of the track that they share and will collide if improperly scheduled. The only points in the shared region where trains can pass or touch without a collision are the terminal stations (squares) and the intersection **Y**.

- ***Example 1***: The ***A-line*** and ***B-line*** both depart at 1600 hrs. At 1800 hrs, the ***A-line*** will have reached **Y**, clearing the shared section of the track. At 1800 hrs, the ***B-line*** will have only reached **Z**. Thus, no collision will occur.
- ***Example 2***: The ***A-line*** leaves at 1600 hrs and the ***B-line*** leaves at 1400 hrs. At 1700 hrs, the ***A-line*** will be at node **X** while the ***B-line*** will be at intersection **Y**. Thus, they will collide at around 1730 hrs.
- ***Example 3***: The ***A-line*** leaves at 1600 hrs and the ***B-line*** leaves at 1500 hrs. At 1700 hrs, the ***A-line*** will be at node **X** while the ***B-line*** will be at node **Z**. At 1800 hrs, they will pass each other safely at the intersection **Y** with no collision.

(i) [2 marks] If the ***B-line*** leaves at 1300 hrs, list the times that the ***A-line*** can safely leave without causing a collision.

Solution:

1300 hrs, 1400 hrs, 1800 hrs, 1900 hrs.

// or $A = 1, 2, 6, 7$

(ii) [4 marks] Define the constraint(s) between the variables ***A*** and ***B*** for this CSP. Your statement should be precise, involving variables and inequalities, not a vague assertion that the trains should not collide.

Solution:

$(A < B + 2) \vee (A > B + 4)$

(iii) [1 mark] Suppose that the ***A-line*** train must leave between 1600 hrs and 1700 hrs (inclusive), and the ***B-line*** train must leave between 1300 hrs and 1900 hrs (inclusive). State the domains for ***A*** and ***B*** after these unary constraints have been imposed.

Solution:

A = {4, 5}
B = {1, 2, 3, 4, 5, 6, 7}

4a. [2 marks] Consider the following Knowledge Base (KB) of a logical agent.

- R_1: x_1 if and only if x_2
- R_2: Either x_1 or x_3 but not both
- R_3: x_1 only if x_3
- R_4: x_2 or not x_3

Let the query, α, be: x_2 implies x_3.

Apply the Truth-table Enumeration algorithm by completing the following truth table, and then prove or disprove the statement: "α is entailed by the KB."

Solution:

x_1	x_2	x_3	R_1	R_2	R_3	R_4	α
T	T	T	T	F	T	T	T
T	T	F	T	T	F	T	F
T	F	T	F	F	T	F	T
T	F	F	F	T	F	T	T
F	T	T	F	T	T	T	T
F	T	F	F	F	T	T	F
F	F	T	T	T	T	F	T
F	F	F	T	F	T	T	T

Notice that determining α is unnecessary since the KB is unsatisfiable.
The query is (vacuously) entailed by the KB.

// There is a False in each row (possible model) of the KB.

4b. [5 marks] Consider the following KB corresponding to a logical agent.

- R_1: x_1 or x_2
- R_2: x_3 or x_4
- R_3: x_5 or x_6
- R_4: Not x_1 or not x_3
- R_5: Not x_1 or not x_5
- R_6: Not x_3 or not x_5
- R_7: Not x_2 or not x_4
- R_8: Not x_2 or not x_6

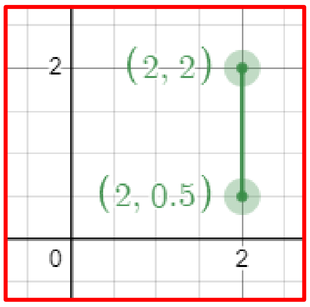
Let the query, α, be: x_4 implies not x_6. Using resolution, show that the query is <u>not entailed</u> by the KB.

Solution:

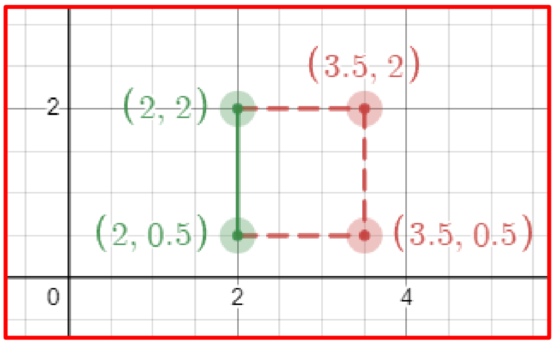
R_0: $\neg x_4 \lor \neg x_6$ // the negation of the negated query, i.e., just the original query $\neg(\neg\alpha) \equiv \alpha$
$R_2, R_6 \Rightarrow R_9$: $x_4 \lor \neg x_5$
$R_3, R_8 \Rightarrow R_{10}$: $\neg x_2 \lor x_5$
$R_1, R_{10} \Rightarrow R_{11}$: $x_1 \lor x_5$
$R_4, R_{11} \Rightarrow R_{12}$: $\neg x_3 \lor x_5$
$R_2, R_{12} \Rightarrow R_{13}$: $x_4 \lor x_5$
$R_9, R_{13} \Rightarrow R_{14}$: $x_4 \lor x_4 \equiv x_4$ // Idempotent law
$R_7, R_{14} \Rightarrow R_{15}$: $\neg x_2$
$R_1, R_{15} \Rightarrow R_{16}$: x_1
$R_5, R_{16} \Rightarrow R_{17}$: $\neg x_5$
$R_3, R_{17} \Rightarrow R_{18}$: x_6
$R_0, R_{18} \Rightarrow R_{19}$: $\neg x_4$
$R_{14}, R_{19} \Rightarrow R_{20}$: $\emptyset$

4c. You are tasked to create a logical agent to solve a puzzle. The objective of the puzzle is to mark k points on a continuous 2-dimensional plane of size N by M, centred at (0, 0) – i.e., to define a set of points, of size k. The puzzle begins with two safe points already marked (note that these initial safe points may differ from one puzzle application to the next).

Other safe points are defined by vertices of squares. For example, consider the following example puzzle, where the initial safe points are (2, 0.5) and (2, 2).



Some additional safe points that may be inferred are thus as follows.



Note that (3.5, 2) and (3.5, 0.5) are also safe points because they form a square with at least two existing safe points.

Consequently, many other safe points may be inferred.

(i) [6 marks] Define the Knowledge Base (KB) for this logical agent.

Solution:

The KB must include rules that allow us to infer two new safe points from two existing ones.

Let S_a denote the safe point.
Let C_b denote the candidate point.
Let $d(p_1, p_2)$ denote the distance between the two points p_1 and p_2.

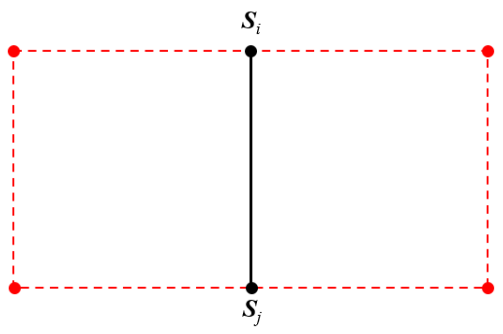
KB:
$d(S_1, S_2) = d(C_1, C_2) \land$
$d(S_1, C_1) = d(S_2, C_2) \land d(S_1, C_2) = d(S_2, C_1) \land$ // possible side or diagonal
$((d(S_1, S_2) = d(S_1, C_1)) \lor (d(S_1, S_2) = d(S_1, C_2))) \land$ // either is a side of the square
$\{[((d(S_1, S_2) = d(S_1, C_1)) \Rightarrow (d(S_1, C_2) = \sqrt{2} \cdot d(S_1, S_2))] \lor$
$[((d(S_1, S_2) = d(S_1, C_2)) \Rightarrow (d(S_1, C_1) = \sqrt{2} \cdot d(S_1, S_2))]\}$ // Pythagorean check

(ii) [6 marks] Design an agent function that utilises this KB and performs inference to determine $k-2$ additional safe squares apart from the initial two safe points. Your agent function specification should be efficient.

Solution:

In each iteration, the logical agent must determine possible candidate points and then use the KB to infer that they are safe points.

For each two safe points, ($\boldsymbol{S}_i$, $\boldsymbol{S}_j$), which have not been expanded before (i.e., implementing graph search), we wish to generate two pairs of candidate points, since the line segment formed by ($\boldsymbol{S}_i$, $\boldsymbol{S}_j$) may form one side of two squares (refer to the diagram below for an example – note that the two safe points may be rotated to any degree and we would still have only two candidate squares).



The determination of these four candidate points, matching the two squares formed using the safe points $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$, is based on a constant time function since this would in turn be based on finding the two perpendicular and one parallel line segment that form each of these squares.

One could formulate a direct prove for the above method to show that it will always generate candidate points that are actually safe points. However, in this application, we will let the KB and inference engine (IE) help us to prove this for us.

END OF PAPER

BLANK PAGE

BLANK PAGE

BLANK PAGE