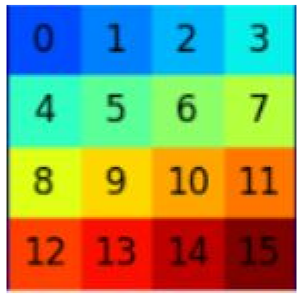
CS 561
Artificial Intelligence
Assignment

Prepared By: Dhruv Kohli (120123054)

Exercise 1 & Exercise 2

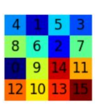
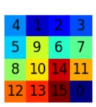
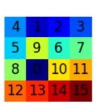
Problem: Consider the 15-puzzle problem with Manhattan Heuristic Distance with the following goal state (0 represents the BLANK):

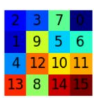
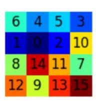
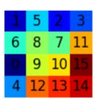


Apply A*, IDA*, RBFS and BFS with custom weighted heuristic such that the cost function is non-monotonic over 20 random initial states and compare the the length of optimal solution, number of nodes generated, comment and plot.

Solution:

Algorithms: in Appendix.

No.	Initial State	Length of Optimal Solution			Number of Nodes Generated		
		A*	IDA*	RBFS	A*	IDA*	RBFS
1		15	15	15	80	45	84
2		15	15	15	127	81	68
3		7	7	7	16	7	16
4		4	4	4	10	4	10

5		15	15	15	67	30	64
6		17	17	17	185	151	191
7		12	12	12	35	18	28
8		13	13	13	57	32	66
9		11	11	11	33	15	25
10		18	18	18	229	151	137
11		18	18	18	36	25	50
12		19	19	19	115	43	61
13		17	17	17	91	50	72
14		8	8	8	17	8	17
15		11	11	11	25	11	25
16		12	12	12	47	29	54
17		16	16	16	230	167	248

18		22	22	22	1163	845	873
19		16	16	16	71	30	58
20		11	11	11	46	29	25

Bonus Problem:

Inconsistent Heuristic Detail:

Let B be:

-	*	-	-
-	-	*	*
*	-	-	*
*	-	-	*

$H(n) = w_1 * H_1(n) + w_2 * H_2(n)$

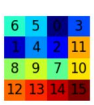
Where,

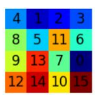
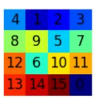
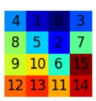
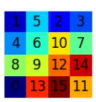
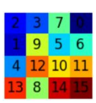
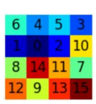
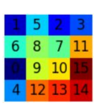
$H_1(n)$ = Manhattan Distance of n from goal state

$H_2(n) = 0$

w_1 = 1 if BLANK is at one of the cell marked with * in B else 0

$w_2 = 1 - w_1$

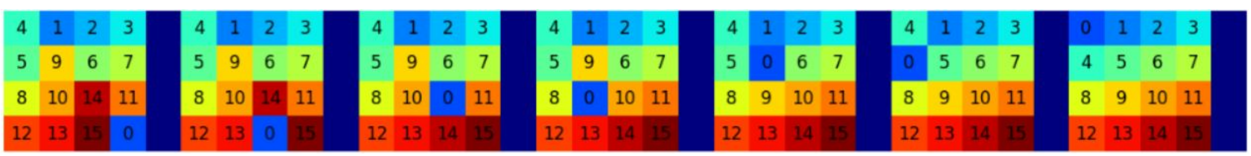
No.	Initial State	Length of Optimal Solution			Number of Nodes Generated		
		A*	IDA*	RBFS	A*	IDA*	RBFS
1		15	15	15	130	82	132

2		15	15	15	242	146	184
3		7	7	7	47	56	33
4		4	4	4	20	10	12
5		15	15	15	117	120	61
6		17	17	17	257	284	324
7		12	12	12	54	45	42
8		13	13	13	142	136	61
9		11	11	11	53	32	43
10		18	18	18	452	308	216
11		18	18	18	88	46	90
12		19	19	19	251	56	73
13		17	17	17	143	91	54
14		8	8	8	40	31	21

15		11	11	11	43	47	35
16		12	12	12	104	80	94
17		16	16	16	386	232	440
18		22	22	22	2358	1336	1696
19		16	16	16	140	93	57
20		11	11	11	62	70	60

Optimal solution for:

Sample #3:



Sample #5:

Comments:

- **As expected the solution lengths are same with A*, IDA* and RBFS.**
- **With the inconsistent heuristic, the number of nodes (configurations of 15-puzzle) generated is much higher as compared with the consistent heuristic of Manhattan distance.**

Exercise 3 & Exercise 4

Problem: **. Generate at least 20 instances of 8-queens and solve them using random restart hill-climbing, simulated annealing and genetic algorithm. Measure the percentage of solved problems. Compare results with various instances of initial population in genetic algorithm.**

Solution:

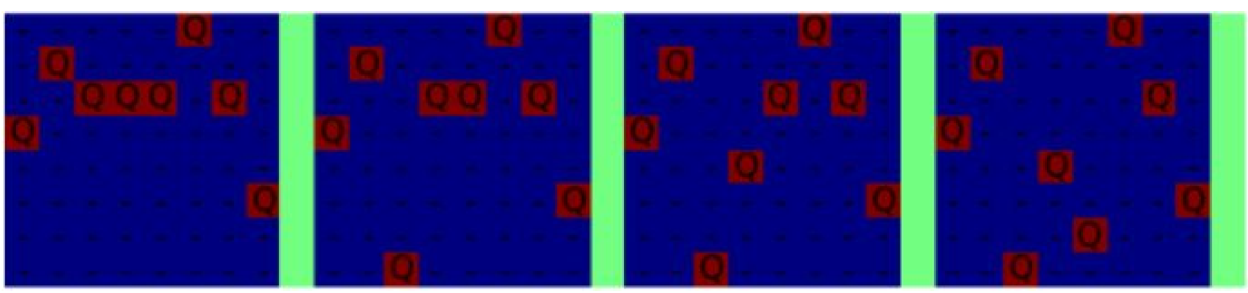
Algorithms: in Appendix.
Temperature Schedule in Simulated Annealing is T(t) = 1/t

	Percentage of solved problems
Random Restart Hill Climbing	0.132
Simulated Annealing	0.976

*** Note that random restart hill climbing solves every problem since it never stops until it finds the solution. In the above table we have measured the percentage of instances where the initial state lead to a solution using hill-climbing subroutine of random restart hill climbing.**

Solution sample

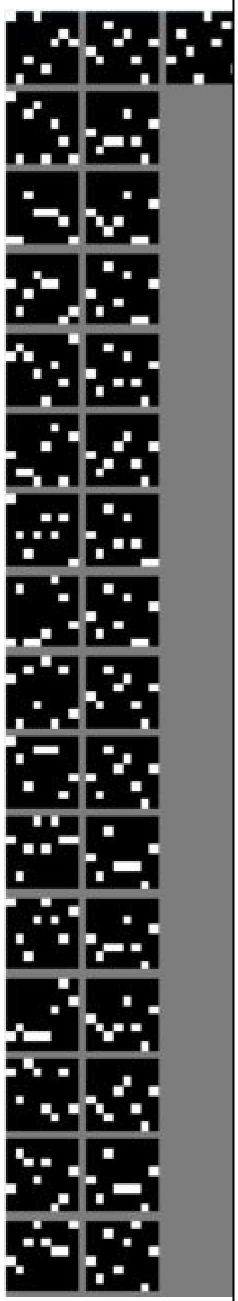
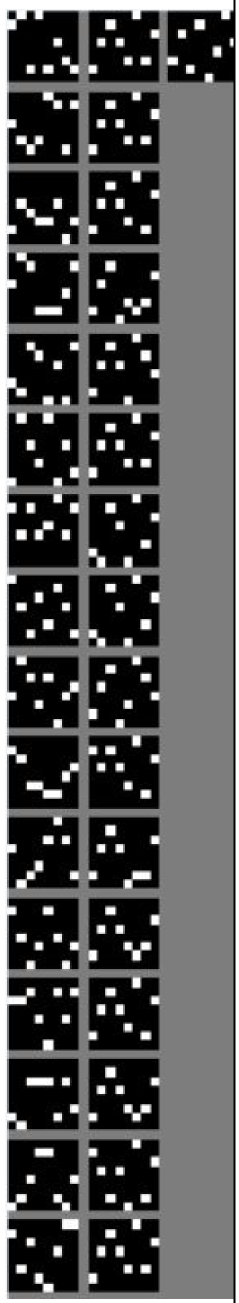
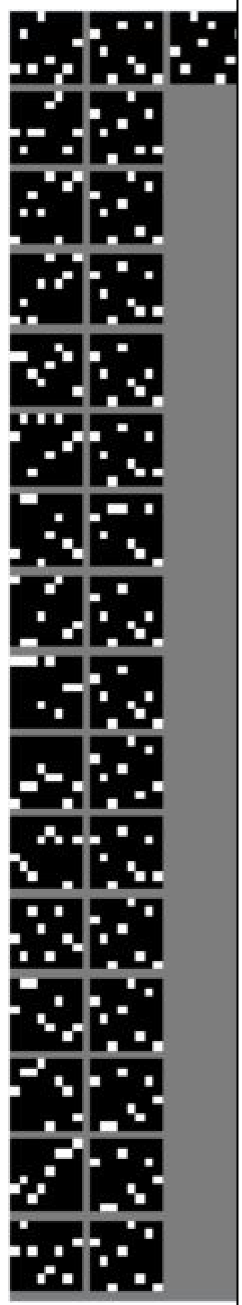
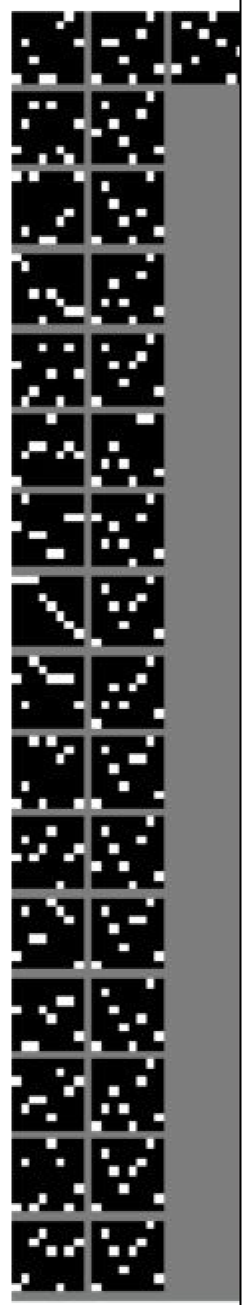
- **Using hill-climbing subroutine of random restart hill climbing:**



Genetic Algorithm with various initial instances of initial populations:
Parameters:

- **Initial Population size: 16**
- **Mutation Probability: 0.1**

Regarding plots in the table below:
The first column in the plot represents initial population where white cell represents queen and black cell represents empty cell, the second column represents final population which contains a solution state which is shown in third column.

Plots				

#iterations inside genetic algorithm	**4475**	**17431**	**1657**	**19946**

Comments:

- **The percentage of solved problems in case of random restart hill climbing and simulated annealing are consistent with values mentioned in the book.**
- **The initial population in the genetic algorithm has drastic effect on the number of iterations required inside the genetic algorithm.**

Exercise 5

Problem: Construct a game playing agent using alpha-beta pruning for tic-tac-toe game. Assume your own move generators and evaluation function. What is the effective branching factor? Can you improve this by improving the move ordering? Comment on your results.

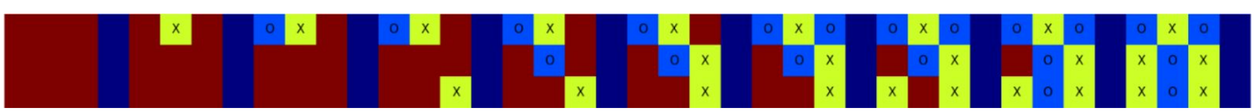
Solution:

Algorithms: in Appendix.

Root is BLANK state (No cell filled).
Here, Move Ordering corresponds to Random Successor Examination.

	Alpha-Beta Pruning	Alpha-Beta Pruning with Move ordering
Nodes Explored (N)	30710	28726
Max Depth (d)	9	9
Effective Branching Factor ($N^{1/d}$)	3.1520	3.1286

Single run of Tic-Tac-Toe where,
X: Computer
O: Human



Comments:

- **As described in the textbook (in move ordering section) randomly examining the successors does decrease the number of nodes explored, hence decreasing the branching factor.**

Appendix

A* Algorithm:

```
function A*(start, goal)
    // The set of nodes already evaluated.
    closedSet := {}
    // The set of currently discovered nodes still to be evaluated.
    // Initially, only the start node is known.
    openSet := {start}
    // For each node, which node it can most efficiently be reached from.
    // If a node can be reached from many nodes, cameFrom will eventually contain the
    // most efficient previous step.
    cameFrom := the empty map

    // For each node, the cost of getting from the start node to that node.
    gScore := map with default value of Infinity
    // The cost of going from start to start is zero.
    gScore[start] := 0
    // For each node, the total cost of getting from the start node to the goal
    // by passing by that node. That value is partly known, partly heuristic.
    fScore := map with default value of Infinity
    // For the first node, that value is completely heuristic.
    fScore[start] := heuristic_cost_estimate(start, goal)

    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, goal)

        openSet.Remove(current)
        closedSet.Add(current)
        for each neighbor of current
            if neighbor in closedSet
                continue        // Ignore the neighbor which is already evaluated.
            // The distance from start to a neighbor
            tentative_gScore := gScore[current] + dist_between(current, neighbor)
            if neighbor not in openSet  // Discover a new node
                openSet.Add(neighbor)
            else if tentative_gScore >= gScore[neighbor]
                continue        // This is not a better path.

            // This path is the best until now. Record it!
            cameFrom[neighbor] := current
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)

    return failure

function reconstruct_path(cameFrom, current)
    total_path := [current]
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.append(current)
    return total_path
```

*** source: wikipedia**

IDA* Algorithm:

```
node              current node
g                 the cost to reach current node
f                 estimated cost of the cheapest path (root..node..goal)
h(node)           estimated cost of the cheapest path (node..goal)
cost(node, succ)  step cost function
is_goal(node)     goal test
successors(node)  node expanding function

procedure ida_star(root)
  bound := h(root)
  loop
    t := search(root, 0, bound)
    if t = FOUND then return bound
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure

function search(node, g, bound)
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    t := search(succ, g + cost(node, succ), bound)
    if t = FOUND then return FOUND
    if t < min then min := t
  end for
  return min
end function
```

*** source: wikipedia**

RBFS Algorithm:

function RECURSIVE-BEST-FIRST-SEARCH($problem$) **returns** a solution, or failure
  return RBFS($problem$, MAKE-NODE($problem$.INITIAL-STATE), ∞)

function RBFS($problem, node, f\_limit$) **returns** a solution, or failure and a new f-cost limit
  if $problem$.GOAL-TEST($node$.STATE) **then return** SOLUTION($node$)
  $successors \leftarrow [\,]$
  for each $action$ **in** $problem$.ACTIONS($node$.STATE) **do**
    add CHILD-NODE($problem, node, action$) into $successors$
  if $successors$ is empty **then return** $failure, \infty$
  for each s **in** $successors$ **do** /* update f with value from previous search, if any */
    $s.f \leftarrow \max(s.g + s.h,\ node.f))$
  loop do
    $best \leftarrow$ the lowest f-value node in $successors$
    if $best.f > f\_limit$ **then return** $failure, best.f$
    $alternative \leftarrow$ the second-lowest f-value among $successors$
    $result, best.f \leftarrow$ RBFS($problem, best, \min(f\_limit, alternative)$)
    if $result \neq failure$ **then return** $result$

*** source: AI: A Modern Approach**

Random Restart Hill Climbing Algorithm:

```
While True:
        Result = HILL-CLIMBING(Random instance of problem)
        If GOAL-TEST(Result)
                Return Result
```

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
     neighbor ← a highest-valued successor of current
     if neighbor.VALUE ≤ current.VALUE then return current.STATE
     current ← neighbor
```

*** source: AI: A Modern Approach**

Simulated Annealing Algorithm:

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to “temperature”

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
     T ← schedule(t)
     if T = 0 then return current
     next ← a randomly selected successor of current
     ΔE ← next.VALUE – current.VALUE
     if ΔE > 0 then current ← next
     else current ← next only with probability e^(ΔE/T)
```

*** source: AI: A Modern Approach**

Genetic Algorithm:

function GENETIC-ALGORITHM($population$, FITNESS-FN) **returns** an individual
  inputs: $population$, a set of individuals
      FITNESS-FN, a function that measures the fitness of an individual

  repeat
    $new\_population \leftarrow$ empty set
    for i = 1 **to** SIZE($population$) **do**
      $x \leftarrow$ RANDOM-SELECTION($population$, FITNESS-FN)
      $y \leftarrow$ RANDOM-SELECTION($population$, FITNESS-FN)
      $child \leftarrow$ REPRODUCE(x, y)
      if (small random probability) **then** $child \leftarrow$ MUTATE($child$)
      add $child$ to $new\_population$
    $population \leftarrow new\_population$
  until some individual is fit enough, or enough time has elapsed
  return the best individual in $population$, according to FITNESS-FN

---

function REPRODUCE(x, y) **returns** an individual
  inputs: x, y, parent individuals

  $n \leftarrow$ LENGTH(x); $c \leftarrow$ random number from 1 to n
  return APPEND(SUBSTRING($x, 1, c$), SUBSTRING($y, c + 1, n$))

*** source: AI: A Modern Approach**

Alpha Beta Pruning Algorithm:

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

*** source: AI: A Modern Approach**