Due Date: 12 February 2019 at 09:30 Eastern

PSet3

CS 4649/7649 Robot Intelligence: Planning
Instructor: Matthew Gombolay

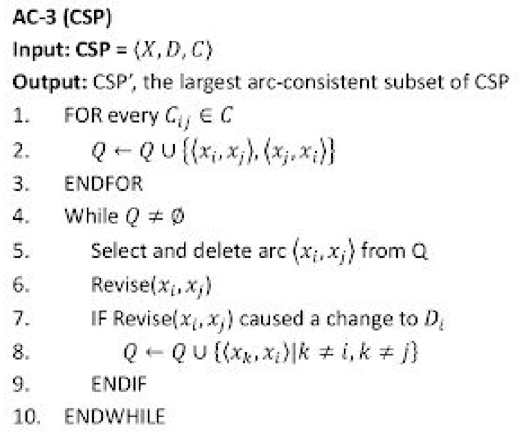
Instructions:

- You may work with one or more classmates on this assignment. However, all work must be your own, original work (i.e., no copy+pasting code). You must list all people you worked with and sources you used on the document you submit for your homework
- Solutions to "by hand" problem must be enclosed by a box and be legible.
- You should submit your solutions to PSet 3 as a single, zipped folder. The folder should contain exactly 1 PDF file for your answers to Problems 1-3 and 4 *.py files for your code to Problem 3.

Problem 1

Prove that the complexity of AC-3 is $O\left(ek^3\right)$. You are welcome to look up sources online to help with the proof, but the proof must be detailed and derived using your own words – no copy+pasting!

Solution:



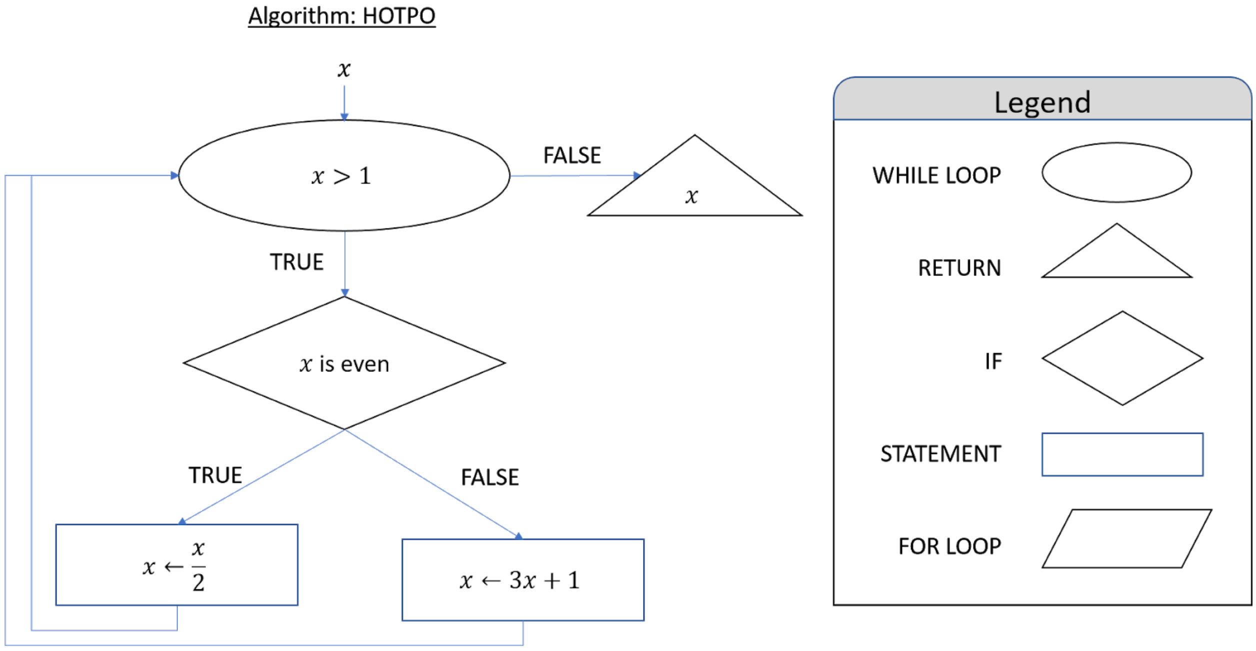
```
AC-3 (CSP)
Input: CSP = $\langle X, D, C \rangle$
Output: CSP', the largest arc-consistent subset of CSP
1.  FOR every $C_{ij} \in C$
2.      $Q \leftarrow Q \cup \{\langle x_i, x_j \rangle, \langle x_j, x_i \rangle\}$
3.  ENDFOR
4.  While $Q \neq \emptyset$
5.      Select and delete arc $\langle x_i, x_j \rangle$ from Q
6.      Revise($x_i, x_j$)
7.      IF Revise($x_i, x_j$) caused a change to $D_i$
8.          $Q \leftarrow Q \cup \{\langle x_k, x_i \rangle | k \neq i, k \neq j\}$
9.      ENDIF
10. ENDWHILE
```

We describe the AC-3 algorithm using the pseudo algorithm presented in the lecture slides and shown in the figure above. The arc, **<x_i,x_j>** is revised only if it is in ***Q***. In the worst-case scenario, every call to the **Revise()** function causes a change in the domain ***D_i***. Thus, in the worst-case scenario arc **<x_i,x_j>** is added to ***Q*** as many as ***k*** times, which is the size of the largest domain. And assuming there are ***e*** binary constraints, according to line 2, there are ***2e*** arcs added to the queue. Therefore the ***while loop*** (lines 4 - 10) executes ***O(ek)*** times in the worst-case scenario. As discussed in class, the **Revise()** function has a complexity of ***O(k^2)***. Therefore the overall complexity of the AC-3 algorithm is ***O(e) + O(ek * k^2) = O(ek^3)***

Reference: Zhang, Yuanlin, and Roland HC Yap. "Making AC-3 an optimal algorithm." *IJCAI*. Vol. 1. 2001.

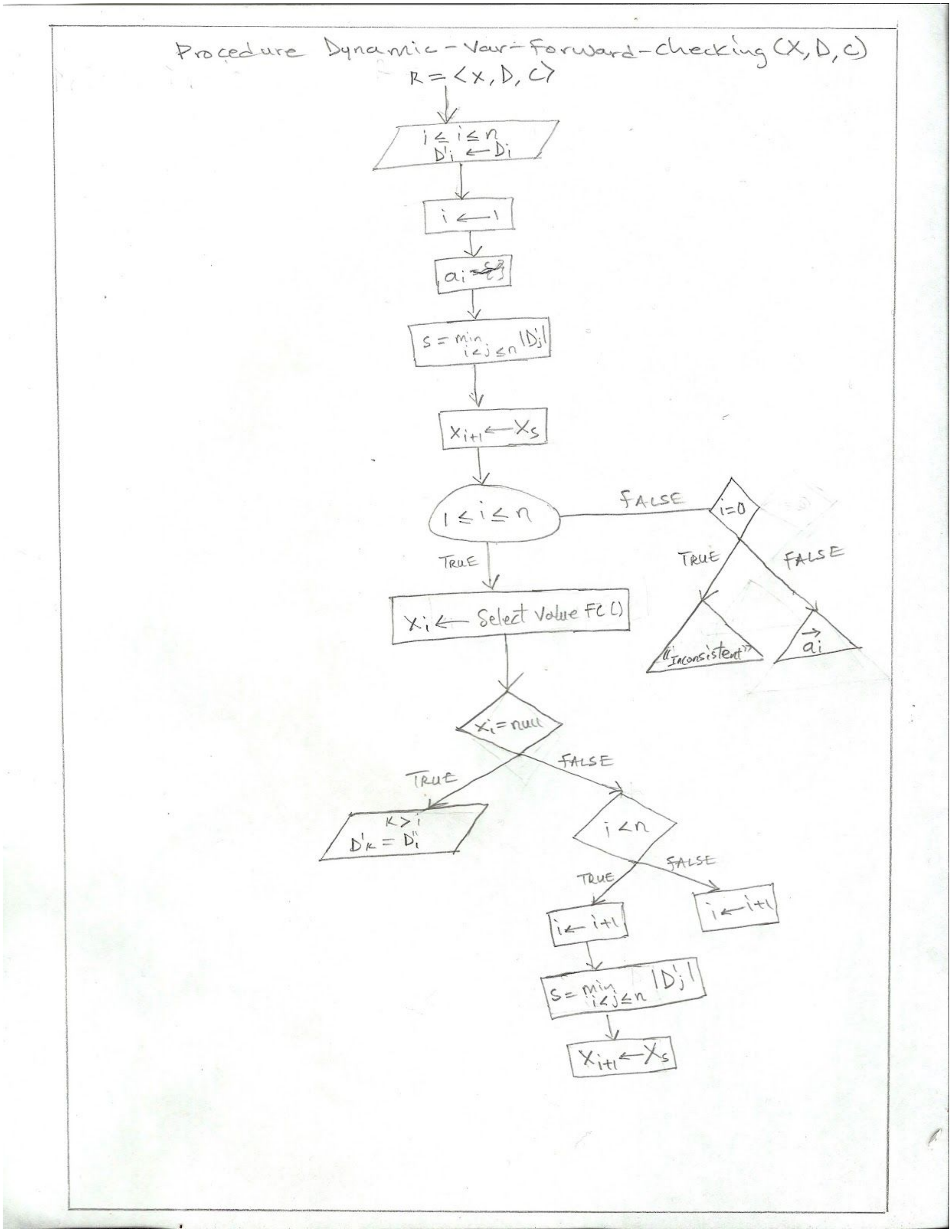
Problem 2

For this problem, you will draw block diagrams describing the flow of Backtrack Search with Forward Checking and Dynamic Variable Ordering. An example block diagram for HOTPO is shown below. Please follow the same conventions for your solutions.



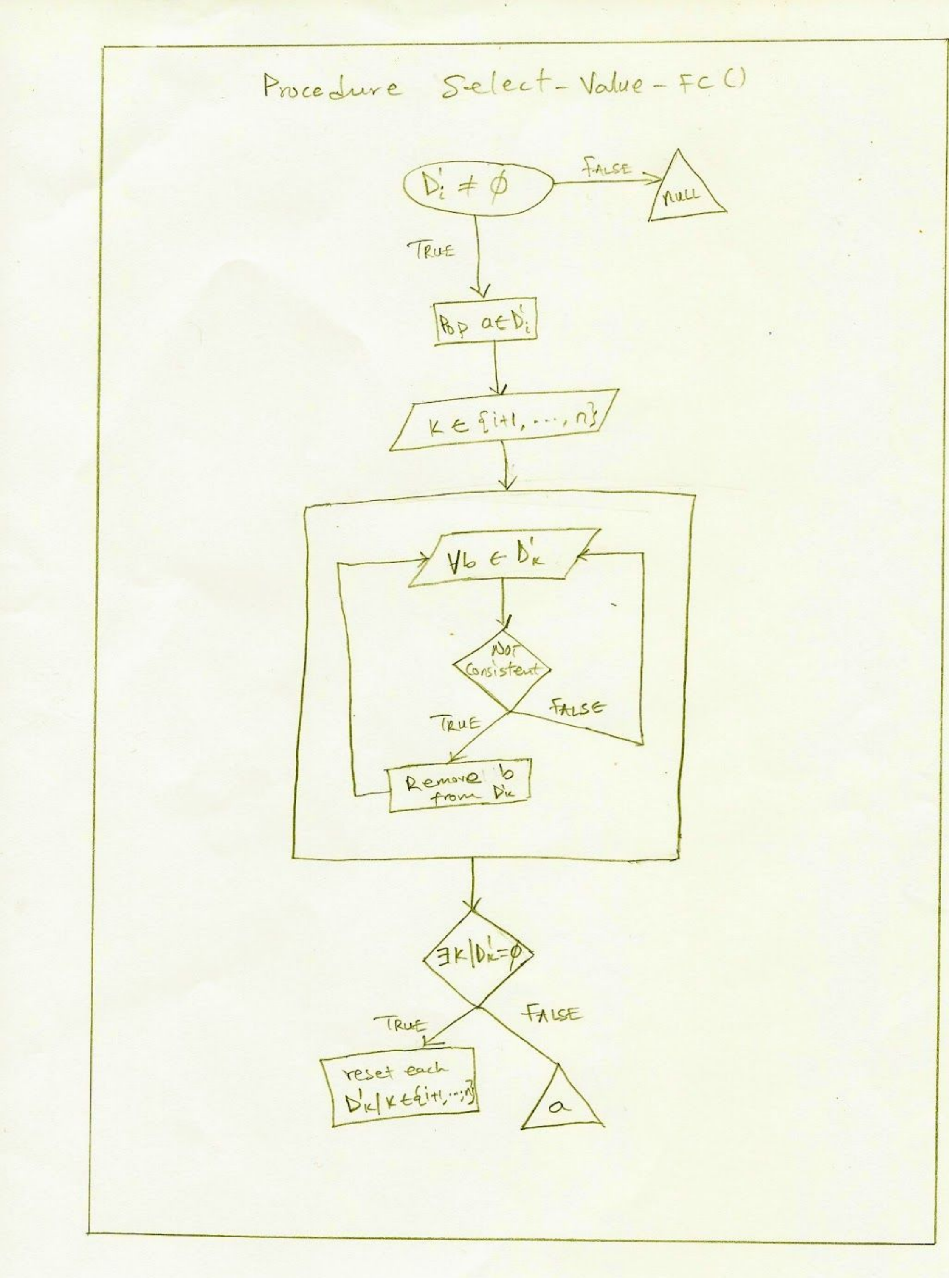
Part A) Draw a block-diagram describing Backtrack Search with Forward Checking and Dynamic Variable Ordering, based upon slide 63 (“Procedure Dynamic-Var_Forward-Checking(x,D,C)”).

Due Date: 12 February 2019 at 09:30 Eastern



Due Date: 12 February 2019 at 09:30 Eastern

Part B) Draw a block-diagram describing Backtrack Search with Forward Checking and Dynamic Variable Ordering, based upon slide 57 (“Procedure Select-Value-FC()”), which is a subroutine for the program you described in Part A.


Procedure Select-Value-FC()
$D'_i \neq \emptyset$
FALSE
NULL
TRUE
Pop $a \in D'_i$
$k \in \{i+1, \dots, n\}$
$\forall b \in D'_k$
NOT Consistent
TRUE
FALSE
Remove b from D'_k
$\exists k | D'_k = \emptyset$
TRUE
FALSE
reset each $D'_k | k \in \{i+1, \dots, n\}$
a

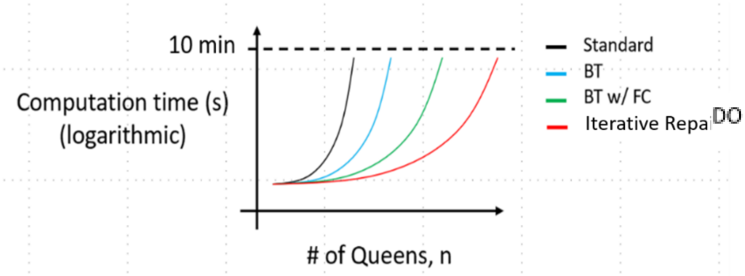
Due Date: 12 February 2019 at 09:30 Eastern

Problem 3

Implement the following algorithms to solve the n-queens problem, which is defined as the problem of placing n queens on a chess board of size n by n such that no queen can "take" another queen (i.e., no two queens should be on the same row, column or diagonal).

● Standard Search	file name: standard.py
● Backtrack Search	file name: standard.py
● Backtrack Search with Forward Checking (FC)	file name: BT_w_FC.py
● Iterative Repair (Min-Conflict Heuristic)	file name: iterative_repair.py

Generate a plot like the one below by trying n = {1,2,…} for each algorithm until the search method can no longer find a solution within 10 minutes of computation time.



Notes:

- Use the statespace convention we described with columns for variables and rows for the domain values.
- For Standard Search, Backtrack Search, and Backtrack Search with FC, use a fixed ordering assigning queens (your variables) based upon their column index, e.g., the queen in column one would be assigned first, the queen in column two would be assigned second, and so forth.
- For iterative repair, use a random initialization as your greedy heuristic. If after performing hill climbing, you cannot find a solution, generate another random solution (i.e., there should be a loop that keeps trying to find a solution for up to 10 minutes of computation).
- You should have one Python file for each type of search, and the code should be clearly documented. Files should be named as shown above. Further, when the grader runs the python code, the script should print to the screen the time taken for each # of queens up to 10 minutes.
- The 10 minute time-out is not a cumulative time but a time per value of n. So, for example, if your first four times are 1 min, 2 min, 3 min, and 5 min for n = 1, 2, 3, and 4, respectively, you should keep going. You should keep going until, for some n large enough, that you cannot find a solution in <= 10 minutes for that single n-Queens problem.