CSCE476/876 **Homework 5** **Spring 2001**

Assigned on: Monday, April 9^{th}, 2001.
Compulsory, Pen and Paper, due on: Friday, April 20^{th}, 2001. Total 60 points.
Bonus, Programming, due on: Wednesday, April 25^{th}, 2001. Total 80 or 120 points.

This homework consists of two parts: Section 1 is a pen-and-paper assignment and is compulsory to all students. Section 2 is a programming assignment in Common Lisp and is optional. *The grades of Section 2 will count as bonus for the homework portion of the final grade.* The goal of Section 2 is to help you improve your grades on the assignments while providing some challenges to students who are excited about implementing algorithms.

1 Required

1. **Axioms in FOL** *(Adapted from AIMA, exercise 7.6)*
   Using the following: `Child(x,y)`, `Sibling(x,y)`, `Female(x)`, `Male(x)`, and `Spouse (x, y)`:
   - (10 points) Write axioms describing the predicates: `GrandChild`, `GreatGrandParent`, `Brother`, `Sister`, `Daughter`, `Son`, `Aunt`, `Uncle`, `BrotherInLaw`, `SisterInLaw`, and `FirstCousin`. We want these axioms to be definitions, so use $\Leftrightarrow$ instead of $\Rightarrow$.
   - (5 points) Knowing that a second cousin is a child of one's parent first cousin, write the definition of a N^{th}-`cousin`, as a recursive expression in terms of the predicates defined above. Hint: Let N^{th}-`cousin` be a ternary predicate, that takes as input n, and two persons p_1 and p_2.
2. **AIMA, exercise 7.9**. Questions a, b, and c.
   Five (5) points per question, total 15 points.
3. **AIMA, exercise 7.11**
   Five (5) points per question, total of 30 points.

2 Optional: Programming

You have the choice between the following three problems. Choose <u>one</u> and implement it. This is a (sweat-intensive) opportunity to catch up on grades for past homework. In all cases, you need to return a README file that explains your design choices, the data structures, your implementation, an input output example, as well as an analysis of the cases for which your algorithm works and those cases it can't handle and a proposal on how to fix them.

1. (80 points) AIMA, exercise 4.13.

2. (80 points) AIMA, exercise 6.15.

3. (120 points) **GTA assignment problem as a CSP**, details to retrieve from instructor.
   Note: This exercise is still at an experimental stage, and is not risk free. You will be graded on your contribution to debug/solve the program and on your report.

   The assignment of graduate teaching assistants (GTAs) to courses at CSE is a combinatorial task that needs to take into account the availability of the GTAs and, to the extent possible, their preferences. The goal of this exercise is to:

   (a) Familiarize yourself with the data structures and functions that your colleague Robert Glaubius has already implemented for this task. These data structures and functions implement the GTA assigment problem as a CSP and models most of the constraints encountered either as unary constraint or as binary ones.

   (b) Implement a depth-first backtrack search with forward-checking that finds one solution to the problem. This task consists of implementing a simple chronological backtracking and a simple forward-checking strategy that re-filters the domains of all future variables according to past variables upon backtracking. (There is a neat specification of the FC algorithm that is a little more complex to implement than this straightforward procedure. It is your choice to decide which FC you want to implement.)

   (c) Define an evaluation function and find the the optimal solution to the problem according to the proposed function.
   Each assignment can be "evaluated" in terms of some combinations of the values of individual preferences expressed by the GTAs, a prefernce is expressed as a value between 1 and 5. You need to propose an evaluation function that returns the value of a given solution as a combination of the individual preferences (e.g., maximize average value, minimize variance, maximize minimum preference, etc.). After you have found one solution, you need to evaluate it according to this evaluation function, choose the solution as an incumbent, then seach for the optimal solution in terms of your evaluation function by generating all solutions and comparing them to the incumbent. When the value of a new partial solution is at least as good as that of the incumbent, you keep extending the solution, otherwise, you prune it, in a Branch and Bound fashion. During this process, whenener you obtain a complete solution that is better than your current incumbent, than it becomes your new incumbent.