Assignment 3

CSG120, Fall 2003
Due: Thursday, Oct. 16

Part I. Propositional Logic Inference Using Exhaustive Model Checking

(Taken from Problem 7.9, p. 238, in the textbook.) Consider the following knowledge base:

- If the unicorn is mythical, then it is immortal.
- If the unicorn is not mythical, then it is a mortal mammal.
- If the unicorn is either immortal or a mammal, then it is horned.
- If the unicorn is horned, then it is magical.

1. Create propositional symbols to represent each of the relevant assertions, (e.g., "the unicorn is mythical"), and then represent the knowledge given above as a single propositional logic sentence. You may use either Lisp-style syntax or the mathematical logic syntax used in the book.

2. Use the function *exhaustive-check-satisfiability* in the file "satisfiability.lisp" (from the /programs/logic/ subdirectory under the course web page) to determine whether each of the following assertions are entailed by the above knowledge base:

- The unicorn is mythical.
- The unicorn is magical.
- The unicorn is horned.

(To run this program you will, of course, need to express your answer to problem 1 using Lisp-style syntax.) Print out and turn in the dribble output. (Also, be sure to state explicitly for each of the 3 assertions whether it is or is not entailed by the given knowledge base, which you will determine by interpreting this output appropriately.)

3. Convert the knowledge base from problem 1 into CNF. (The result should contain 6 clauses.)

Part II. Finding Satisfying Assignments Using WalkSAT

4. Create a Lisp implementation of the WalkSAT algorithm (Figure 7.17, p. 223), but include code that prints out how many iterations it takes before finding a satisfying assignment on any particular run. You are encouraged to borrow code (or more simply, call functions) from "satisfiability.lisp" where helpful. Make up some very simple test cases to demonstrate that your code works. (You may just use $p = 0.5$ in all your tests or experiment with different values; it's up to you.) Turn in both the source code and dribble files with your tests.

5. Consider the following simple logic puzzle: Jones, Smith, and Clark hold the jobs of programmer, knowledge engineer, and manager (not necessarily in that order). Jones owes the programmer \$10. The

manager's spouse prohibits borrowing money. Smith is not married. Your task is to figure out which person has which job.

Use nine propositional symbols to represent the possible person/job assignments, and represent the given facts in propositional logic, together with all the constraints that you know must hold since there is a 1-1 correspondence between names and jobs. For example, you might use the symbol SM to represent the assertion "Smith is the manager." You do not need to represent the relation between owing and borrowing, or being married and having a spouse; you can just use these to draw conclusions. (E.g., from "Smith is not married" and the mention of "the manager's spouse" we know that Smith can't be the manager, which we would represent in Lisp-style syntax as (not SM).) Express this knowledge base as a collection of clauses (i.e., in CNF form with an implicit AND joining the clauses).

Hint: Your CNF knowledge base should contain 24 clauses, most of which are required to represent the 1-1 correspondence constraint.

6. Use your WalkSAT program (with $p = 0.5$) to find a satisfying assignment for these clauses. Is this the only satisfying assignment? (You may run the exhaustive search program to check this since the truth table is not impractically large in this case.) Hand in corresponding dribble output, and summarize clearly the solution found. Comment on how hard or difficult it is for WalkSAT to find a solution to this puzzle (based on the number of iterations it took), compared to how much effort must be expended by the exhaustive search program.

7. [Extra Credit] Here you are to consider to what extent the "density" of solutions in an underlying search space contributes to the relative efficiency of local search methods like WalkSAT (or hill-climbing or simulated annealing or genetic algorithms, etc.) in finding solutions. For example, on p. 150 of the textbook, it is mentioned that the runtime of finding solutions to the n-queens problem (using CSP-specific local search methods) has been found to be essentially independent of problem size n. It is also stated there that "Roughly speaking, n-queens is easy for local search because solutions are densely distributed throughout the state space."

It seems reasonable to conjecture that for such search methods to run in essentially *constant* time, the density of solutions should be essentially constant. But what if the density of solutions does decrease significantly with the size of the search space? As an extreme case, suppose the search space grows exponentially with problem size n but for each n there is exactly one solution. Does this doom local search methods?

In particular, consider a satisfiability problem having n variables and exactly one satisying assignment. (A good example of this would be a propositional logic representation for a logic puzzle taken from a typical puzzle book; designers of such puzzles generally work to create puzzles with unique solutions.) In this case the solution density as a function of n is 2^{-n}.

Show that even in this case there are situations where local search methods can be very efficient. In particular, construct a satisfiability problem with n variables (where n is arbitrary) having only one satisfying assignment but for which WalkSAT (with any value of p) always finds a solution in at most n iterations. Prove your answer.