CS227: Assignment 2

The second assignment involves implementing and evaluating **constraint satisfaction algorithms**, and writing up a report on your experiments.

Algorithms: Implement an algorithm that includes the following features. The algorithm must start by using arc consistency to preprocess the constraints. You may use *AC-3*, though other arc consistency algorithms are also permissible. The algorithm must include *forward checking* and *dynamic variable ordering* using the *minimum remaining values* heuristic. The algorithm must backtrack using either *conflict-directed backjumping* or *dynamic backtracking*.

Evaluate the benefits of each of the above features by comparing the version with all features enabled against variants without arc consistency, without forward checking, without dynamic variable ordering, and without either conflict-directed backjumping or dynamic backtracking. You must get variety in your solutions by randomizing the choice of equally good next variables and the order in which you try values for a given variable.

Problems for evaluation: The main problems for evaluation will be the problem of *filling out a crossword puzzle*. The four test puzzles are included with this handout (electronic version available on Coursework site). Use the dictionary posted on the Coursework site, with a little over 20,000 words. It is a slightly modified version of the standard dictionary available on Unix systems. Additional information on constructing crossword puzzles can be found in the paper:

M. Ginsberg, *et al*, 1990. Search lessons learned from crossword puzzles. In *Proc. of AAAI-90*.

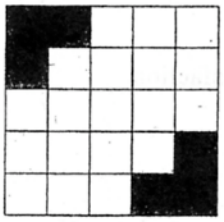
You may also want to run the Zebra problem to help you debug your code. Prosser's version has 11 solutions, while Dechter's version has only 1 (see footnote 15 in [Prosser 1993]). Doing this will also have the side benefit of making your core constraint satisfaction code completely domain independent.

Report: Your report should contain descriptions of the algorithms you are evaluating, including discussions of any optimizations you may have used to make the algorithms run fast. The report should contain the results of running the experiments and a discussion of your conclusions. Your results should be based on averages generated from at least 10 distinct runs for each problem. Additionally, the report should show at least 5 distinct solutions for each problem. Submit all source code electronically. Assignments will be graded on the description of the algorithms and optimizations used, the raw results, and the analysis of your results.

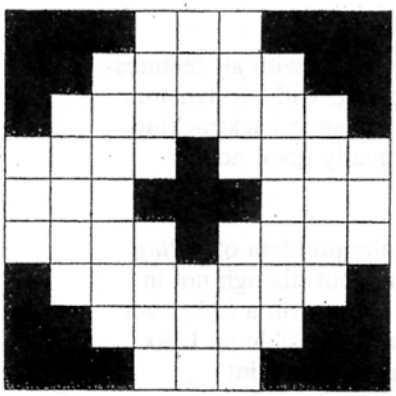
Submission: The report can be submitted electronically, in class, or directly to the TA. Submit source code electronically as a single .tgz or .zip file that unpacks into its own directory. Please include a small README file describing how to build and run your code. Clearly identify all members of the group both on the report, and in the electronic submission. Send electronic submissions to cs227-submit@lists.stanford.edu.

Assignments may be done in groups of 2-3 students. You may choose any programming language for implementation purposes, though we recommend either C or C++.

Assignments are due by noon on **2 May**.



(a)



(b)

Figure 2: Test puzzles

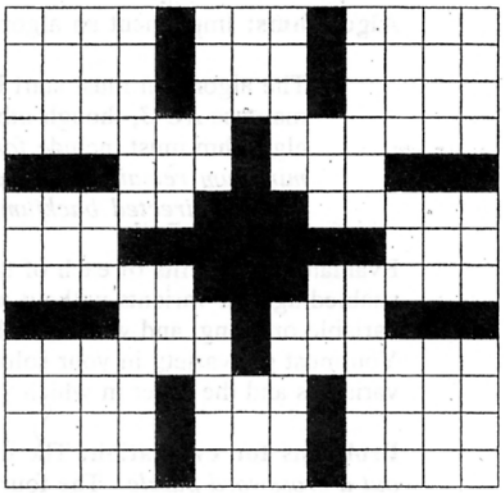
and one of the two words in Figure 1 would be withdrawn immediately.

In practice, this does not work so well. The reason is that the computation involved is a fairly difficult one – we need to look at the possible choices for w_1, check to see which letters are still possible in which spaces (this is the expensive part, since it involves examining each of the choices for w_1), and then to use this information to prune the set of possibilities for w_2. The analysis is expensive enough that the cost incurred is not in general recovered by the associated pruning of the search space. More conventionally put, the forward branching factor for the problem is high enough that additional levels of lookahead draw conclusions no more effectively than their backward counterparts.

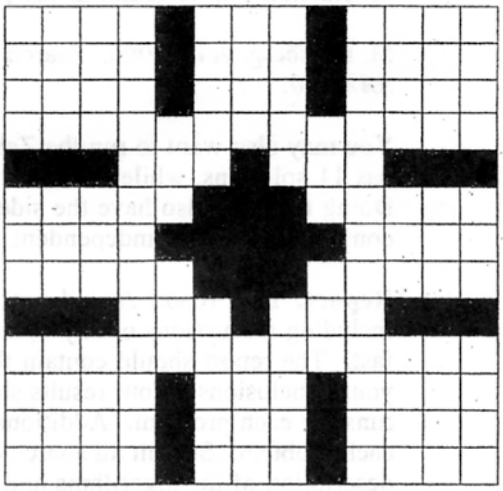
3 Experimental results

Frames used and raw data

In order to evaluate the usefulness of the ideas in the last section, the four puzzles appearing in Figures 2 and 3 were solved by the program. The program always used one level of lookahead (i.e., arc-consistency)



(c)



(d)

Figure 3: Test puzzles (ctd.)