Spring Semester, 2008 **B.Y. Choueiry**

CSCE 421/821: Foundations of Constraint Processing

Homework 4
Implementing FC and CBJ

Assigned: Monday, March 3, 2008

Due: Monday, March 24, 2008

Total value: 125 points. Penalty of 20 points for lack clarity and documentation in code. Bonus of 30 points for additional work.

Notes: This homework must be done individually. *If you receive help from anyone, you must clearly acknowledge it.* Always acknowledge sources of information (URL, book, class notes, etc.). Please inform instructor quickly about typos or other errors.

The goal of this exercise is to implement them advanced backtrack search mechanisms and test them on the test cases of Homework 3 choosing one model per problem instance.

- Implementing FC. 30 points
- Implementing CBJ. 30 points
- Implementing the hybrid of the above two mechanisms: FC-CBJ. 30 points
- Reporting the results obtained on the examples of Homework 3, choosing one model per problem instance. 30 points
- Your impressions. 5 points

General indications:

- *Please make sure that you keep your code and protect your files.* Your name, date, and course number must appear in each file of code that you submit.
- All programs must be compiled, run and tested on `cse.unl.edu`. Programs that do not run correctly in this environment will not be accepted.
- You must submit a README file so that we know to run and test your code.

1 Data structures

Create a class objects for storing an instance of an X-solver, where X is FC, CBJ, and CBJ. (For lispers, use `defclass`.) Each class should be a sub-class of BT-solver and should store in its attributes the data structures necessary for particular search mechanism. Create all three classes. For FC-Solver, implement *reductions* as a 2-dimensional array, storing in one entry of a row a variable and in the second entry of the same row the list of list of values removed from the domain of the variable (treated as a stack).

2 Hybrid search

1. Implement the partial look-ahead search algorithm FC: 10 points
   - Taking as parameters an ordering heuristic (implement with LD, degree, and ddr) and 10 points
   - The ordering strategy (i.e., static and dynamic). 10 points
2. Implement the mechanism for reducing the backtracking effort 'conflict-directed backtracking' (CBJ): 10 points
   - Taking as parameters an ordering heuristic (implement with LD, degree, or ddr) and 10 points
   - Using a static ordering of the variables. 10 points
3. Implement the hybrid of the above two mechanisms FC-CBJ: 10 points
   - Taking as parameters an ordering heuristic (implement with LD, degree, and ddr) and 10 points
   - Using a static ordering of the variables. 10 points
4. **Bonus** (20 points): Implement dynamic variable-ordering with FC-CBJ.

3 Performance comparison

For the following working conditions:

- For the specified variable ordering-heuristics (i.e., LD, degree, and/or ddr),
- For the ordering strategies implemented (i.e., either static and/or dynamic), and
- Searching for *one* solution,

  Bonus (10 points): searching for *all* solutions.

Evaluate the four search algorithms you have implemented so far (i.e., BT, FC, CBJ, FC-CBJ) in terms of #CC, #NV, #BT, and CPU time, comparing their performance on the following problem instances: 26 points

1. A chain of 4 variables;
2. Coloring K4 with 2 colors;
3. Map Coloring of Australia;
4. 4 Queens;
5. 5 Queens;
6. 6 Queens;
7. Zebra (binary);
8. A binary CSP problem of your choice from the online library (e.g., Knights, Latin Square);

where the first four problem instances are useful mainly for debugging.

Conclude with your observations. *Hint:* verify Prosser's conclusions and that of Kondrak and van Beek's. Do these conclusions hold under dynamic orderings? 4 points

4 Your impressions

Tell us whether you find the set of Homework 1, Homework 3, and Homework 4 useful or not and how they can possibly be improved. Store your impressions in a separate file.