Massachusetts Institute of Technology

16.410 Principles of Automated Reasoning and Decision Making

Problem Set #4

Background for Problems 1 and 2

In Problems 1 and 2, you will implement backtracking algorithms for solving CSPs. The implementation will be general so that it will be usable for a wide variety of problem types. However, you will test your implementation on a specific kind of CSP: the N-queens problem.

You are provided with several Java classes that will help you get started. The class **CSP** is used to represent the CSP problem, and includes skeleton methods for the backtracking algorithms that you are responsible for implementing. The Java class CSP is supported by the classes **CSP_Variable**, **CSP_Domain**, and **CSP_Constraint**. These classes, along with CSP, provide a general CSP solver framework. Aspects of these classes are then specialized for particular problem types, such as N-queens. We summarize the function of each of these classes below:

Class CSP

This class is used to represent a CSP problem, and includes methods for solving the CSP. Recall that a CSP is represented by a set of variables, a domain for the variables, and a set of constraints. Thus, the CSP class has elements **variables**, **domain**, and **constraints**. The class also includes constructor, initialize, and print methods, which must be overridden by a class that inherits from CSP. Finally, the class provides methods **backtrack** and **backtrack_fc** for solving the CSP; these implement backtrack search, and backtrack search with forward checking, respectively.

In Problem 2 you will implement the method backtrack, and in Problem 3 you will implement the method backtrack_fc. Note that these methods should be written to apply to any CSP; they are to be implemented at the level of the CSP abstraction, in terms of the classes CSP, CSP_Variable, CSP_Domain, and CSP_Constraint. The implementation of these methods should not contain code specific to a particular type of CSP, such as N-queens.

Class CSP_Variable

The class CSP_Variable represents a single variable in a CSP. The class has elements **domain**, **from_arcs**, and **to_arcs**. In this problem, we will limit ourselves to binary constraints that are directional. Thus, from_arcs is the set of constraints that have the variable as its "input" variable, and to_arcs is the set of constraints that have the variable as its "output" variable. The class also defines basic methods for accessing and modifying domain values, and for checking consistency.

Class CSP_Constraint

The class CSP_Constraint represents a single CSP constraint. It has elements for input variable and output variable. It also has a constructor that initializes these elements.

Class CSP_Domain

The class CSP_Domain represents a domain for a variable. It has a list of domain values, and an accessor method that retrieves them.

The N-Queens Problem

You will test your backtrack algorithm implementations on an N-Queens problem. Although the backtrack and backtrack_fc methods will be coded in a problem-independent manner, you will need to implement some code specific to the N-Queens problem. In particular, you will implement an arc consistency check method that is specific to an N-Queens problem constraint.

For this problem, the goal is to place N queens on an NxN chess board so that no queen can capture another. Thus, no two queens can share a common row, column, or diagonal. We will use the following representation for this type of problem.

- Each variable specifies the row on which a particular queen is placed. Each queen is identified with a column of the board. This automatically ensures that the queens will not share columns.
- The variable domains are the available row indices.
- The constraints between the variables specify that the corresponding queens do not share a row or diagonal.

Class NQueens

The class NQueens, which extends CSP, represents an NQueens CSP. It contains initialize and print methods, which we provide. These will help you get started, and will provide guidance for other aspects of the implementation that are specific to the NQueens problem.

Class NqueensVariable

The class NQueensVariable, which extends CSP_Variable contains element col, which indicates the (constant) column position for the variable. The constructor initializes this element and the domain.

Class NQueensDomain

The class NQueensDomain, which extends CSP_Domain, has a constructor that initializes the domain elements.

Class NQueensConstraint

For the class NQueensConstraint, which extends CSP_Constraint, you will provide the implementation for the method **consistent**, which checks whether the constraint is consistent. This method should retrieve the current domains of the input and output variables of the constraint. If it cannot find any combination of input and output values that are consistent, the method should return false. Otherwise, it should return true.

Problem 1: Backtracking Implementation (40 points)

Implementation

Please implement the **backtrack** method of the CSP class. As stated previously, these methods are to be implemented at the level of the CSP abstraction, in terms of the classes CSP, CSP_Variable, CSP_Domain, and CSP_Constraint.

To support the backtrack method, please implement the **consistent** method of the NQueensConstraint class. As stated previously, this method should retrieve the current domains of the input and output variables of the constraint. If it cannot find any combination of input and output values that are consistent, the method should return false; otherwise, it should return true.

The lecture slides provide a description of the backtrack search algorithms. Please refer to these slides when implementing the backtrack method.

Testing

The class CSPTop provides a main method as an entry point. This initializes an NQueens problem of specified size and solves it, printing out the domains before and after the solution.

Test your implementation of backtrack for sizes 5, 10, and 15. What is the number of consistency checks for each? What is the largest problem size that can be solved in about a minute?

Problem 2: Backtracking with Forward Checking Implementation (30 points)

Implementation

In this problem, you will augment your backtrack algorithm with forward checking, by implementing the algorithm backtrack_fc (in CSP_rep.java) and select_from_domain_values_copy_fc (in CSP_Variable.java). The classes and algorithms should be similar to those for the backtrack algorithm. The primary change should be to add a call to a forward checking method from the backtrack method. This

will require management of variable domain pruning and restoration during backtracking, as described in the lecture notes.

Testing

Test your implementation of backtrack_fc on problems with sizes 5, 10, 20, and 30. What is the number of consistency checks for each? What is the largest problem size that can be solved in about a minute?

Problem 3: Using GraphPlan (30 points)

The spare tire problem is a well-known benchmark problem for planning algorithms (see Russell and Norvig, Ch. 11, for description of this problem).

In this exercise, you will use Graphplan to generate plans that solve the spare tire problem from an existing problem formulation. The formulation for the spare tire problem is in the files **fixit_ops**, **fixit_facts1**, and **fixit_facts2**, which are available on the course web site, on the page for this problem set.

Graphplan and the problem formulation are also available at:

http://www.cs.cmu.edu/~avrim/graphplan.html

See the Graphplan web site, at the above url, for detailed information on how to run this software. For MS Windows users, the source code has been compiled for you. You can run graphplan.exe from a DOS command prompt. For Mac OS or Unix/Linux users, you can compile the source code yourselves simply by typing "make" on a terminal. Refer the the link above for more detail.

Part A. Solve this problem using the Graphplan software. Provide the generated plan; that is, the sequence of operators output by Graphplan.

Part B. Hypothesize a change to the problem formulation that will make the problem substantially more difficult for the Graphplan algorithm. Describe this change and the reason why it will be more difficult for Graphplan to solve. Check whether or not your hypothesis is true by testing the change with Graphplan. List the modified problem, report the change in runtime performance, and list the generated plan.

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010