Massachusetts Institute of Technology

16.410 Principles of Automated Reasoning and Decision Making

Problem Set #6 Due in class and online: Session 13

Constraint Satisfaction Problems

Objectives

In this problem set we examine the solution to constraint satisfaction problems (CSPs). A CSP involves finding a consistent set of assignments for a set of **variables**, $X_0,...,X_{n-1}$, where each variable, X_i, takes its values from a **domain** of values, D_i. The variable assignments are constrained to satisfy a set of **constraints** between some or all of the variables. In this problem set, we restrict ourselves to **binary constraints**, which constrain two variables.

We begin with a set of short-answer questions that cover basic concepts. Then, we proceed to a number of coding exercises, in which you implement a number of constraint processing algorithms in Java. The algorithms are to be tested on a set of problems of different sizes, in order to evaluate their relative performance.

As with all programming problem sets you will be expected to provide, in addition to your software, a set of junit tests, a main method that demonstrates the software applied to the problems specified in the problem set, and a document explaining the design of your solution and the results obtained from it.

Readings

Lecture notes L10 and L11.
AIMA Chapter 5.

Problem 1 – Backtracking on a Four-variable Problem (25 points)

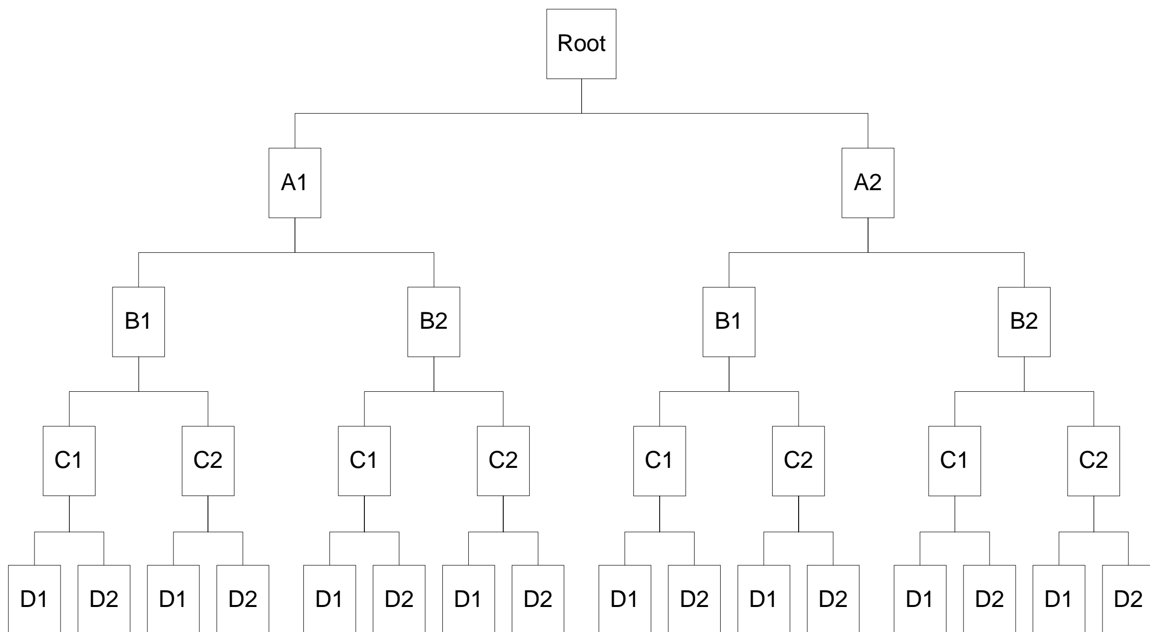
In this problem, we consider a very simplified situation in which there are only four variables: A, B, C and D, and each of them have only two legal values, which we will write as: A1, A2 (for variable A), B1, B2 (for variable B), C1, C2 (for variable C) and

D1, D2 (for variable D). Binary constraints between variable pairs are expressed as a set of legal assignments:

C_{AB}: {(A1, B1) (A2, B1)}
C_{AC}: {(A1, C1) (A2, C2)}
C_{BD}: {(B1, D1)}
C_{CD}: {(C2, D1)}
C_{BC}: no constraint
C_{AD}: no constraint

We say that “an assignment is generated" every time a variable in the problem gets a new (candidate) assignment. We assume that the variables are examined in alphabetical order and the values in numerical order. Below, we ask you to solve this problem using pure backtracking and also by using backtracking with forward checking. Stop when a valid solution is found.

The search tree for this problem is given below. Each node (except the root) is labeled with the value involved in the assignment; the variable is the first letter of the value. Your answers will consist of a space-separated sequence of values corresponding to assignments as they are generated during the search. For example, {A1 B1}, etc.



6.1.1 – Using pure backtracking, how many assignments are made before the answer is found?

What is the assignment sequence?

6.1.2 – Using backtracking with forward checking, how many assignments are made before the answer is found?

What is the assignment sequence?

Problem 2 – CSP True/False (25 points)

Please indicate whether the following statements are true or false.

6.2.1 - Constraint propagation, with no search, will find a satisfactory way to color all planar maps with four colors, if such a coloring exists.

6.2.2 – Given a constraint graph with e edges and d values per domain, constraint propagation takes $O(ed^2)$ arc tests for a single pass through all of the arcs in the graph.

6.2.3 – In backtracking with forward checking, after assigning a value to a variable, does the search loop needs to test the new assignment against past assignments.

6.2.4 – A backtracking search that performs a full constraint propagation after each assignment in place of forward checking generally results in an answer that uses fewer arc-tests than search with forward-checking only.

6.2.5 – A dynamic variable ordering heuristic based on domain size is generally an effective strategy to speed up solving CSP.

6.2.6 – A dynamic value ordering heuristic, which measures the influence of neighboring variables, is generally an effective strategy to speed up solving a CSP, when all the answers are desired.

6.2.7 – If constraint propagation leaves all variables with non-empty domains, there is at least one solution.

6.2.8 – If constraint propagation leaves at least one variable with an empty domain, there is no solution.

6.2.9 – If constraint propagation leaves all variables with singleton domains, there is a unique solution.

6.2.10 – If constraint propagation leaves some variable with more than one value in its domain, there is not a single unique solution.

6.2.11 – Backtracking with forward checking never explores more possible variable assignments than pure backtracking.

6.2.12 – Backtracking with forward checking never tests more constraint arcs than pure backtracking.

Background for Problems 3 and 4

For problems 3 and 4, you will implement backtracking algorithms for solving CSP's. 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 a number of classes that will help you get started. The class CSP is used to represent the CSP problem, and also has methods for the backtracking algorithms, which you will write. The 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.

Class CSP

This class is used to represent a CSP problem, and also has methods for solving it. 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 has constructor, initialize, and print methods, which must be overridden in a class that inherits from CSP. Finally, the class has methods **backtrack**, and **backtrack_fc**, for solving the CSP, using, respectively, backtrack search, and backtrack search with forward checking.

You will implement the method backtrack in problem 3, and the method backtrack_fc in problem 4. Note that 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. 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 the arc consistency check method for 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.

- The variables are the rows where queens are to be 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 are 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 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 3 – Backtracking Implementation (25 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 4 – Backtracking with Forward Checking Implementation (25 points)

Implementation

In this problem, you will augment your backtrack algorithm with forward checking, by implementing the algorithm backtrack_fc. 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?

16.413 Students Only

The pseudocode provided in the lecture for backtracking with forward checking uses an approach in which variable domains are copied in order to preserve the domains prior to pruning, so that they can be restored when backtracking. Discuss advantages and disadvantages of this approach. Propose an alternative approach, that does not use copying, and discuss its advantages and disadvantages.

For optional extra credit (25 points), implement this approach.

Problem 5: Time

Please let us know the amount of time it took you to complete this problem set.