Constraint Programming

## What is Constraint Programming?

Constraint programming is the study of computational models and systems based on constraints. The idea is to model and solve a problem by exploring the constraints that can fully characterize the problem, hence any solution of the problem must satisfy all of the stated constraints. Constraint programming is one of the most exciting developments in programming languages in the last decade. It is attracting widespread commercial interest and is now becoming the method of choice for modeling any types of optimization problems.

As stated by Eugene C. Freuder,

*"*Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Constraints have recently emerged as a research area that combines researchers from a number of fields, including Artificial Intelligence, Programming Languages, Symbolic Computing and Computational Logic. Constraint networks and constraint satisfaction problems have been studied in Artificial Intelligence starting from the seventies. Systematic use of constraints in programming has started in the eighties. In constraint programming the programming process consists of the generation of requirements (constraints) and solution of these requirements, by specialized constraint solvers.

Constraint programming is intended to solve computationally hard problems. These are combinatorial problems, graph-theoretic problems, optimization problems, and problems involving scheduling and planning.

Constraint programming has been successfully applied in numerous domains. Recent applications include computer graphics (to express geometric coherence in the case of scene analysis), natural language processing (construction of efficient parsers), database systems (to ensure and/or restore consistency of the data), operations research problems (like optimization problems), molecular biology (DNA sequencing), business applications (option trading), electrical engineering (to locate faults), circuit design (to compute layouts), etc.

**Example.** Cryptarithmetic puzzles

The problem is to assign distinct (decimal) digits to letters so that adding two words yields the third. E.g.

S E N D  
 + M O R E  
 --------------  
 M O N E Y  
  
  
 D O N A L D  
 + G E R A L D  
 --------------  
 R O B E R T

For problems like this we can always guess a solution (an assignment of digits to letters in this case). This is the so-called *generate and test* approach. However, the number of possibilities may be too large to be manageable.  
  
For instance, in the second example above 10 letters need to be assigned. Each letter can be assigned one of the 10 digits. Thus, there are potentially 10! = 3628800 possible assignments to check.  
  
A constraint logic program can solve these problems much more efficiently.

**Example.** Games

The N-queens problem: there is an N X N board on which to place one queen on each row so that no queens are at the same column or the same diagonal.

The search space for the problem is N!, a very large number for not a very large N. E.g., for 20 queens, there is a huge number of possible board positions, for  each of which we can decide whether it is a solution.

A constraint logic program can solve the queens problem for 20 queens in a fraction of a second. Someone has solved this problem using constraint programming for several hundred queens.

For 20 queens, a naive Prolog program (e.g. the one found in [these notes](https://eclass.srv.ualberta.ca/pluginfile.php/2478062/mod_page/content/13/lectures/sample-prog.pl)) would need about 1 hour to get a solution.

**Example** NP-Complete Problems

This is a large list. For example,

-****The Travelling Salesman Problem (TSP)****

Given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly one.  There are many different versions of TSP.

TSP has several applications even in its simplest formulation, such as planning, logistics, and the manufacture of microchips.   
  
-****Map Coloring****  
Given a map and a number of colors, whether the map is colorable such that any adjacent regions are colored with different colors.   
  
-****Boolean Satisfiability (SAT)****  
  
A clause in propositional logic is a disjunction of literals  l\_1 v ... l\_n, where each l\_i is a proposition or negation of a proposition (a proposition is called a Boolean variable, and sometimes we also call it an atom).  Given a set of clauses in propositional logic, determine whether there is a truth value assignment of Boolean variables so that every clause is satisfied (i.e., every clause evaluates to true).  SAT has been a major tool for circuit verification.

**Example.** Planning

The planning problem is: given an initial state and a finite state, and a number of actions, get a sequence of actions that lead from the initial state to the final state. There are preconditions which decide when each action can be executed, and effects of each action.

Planning Example: The blocks world

A simple yet computationally challenging planning problem. We are given a table and a configuration of a number of blocks, each of which is either on the table or on another block. Suppose the action that we can take is to move a block X to Y which is another block or the table, denoted by *move(X,Y)*. There are usually conditions under which an action can take place. E.g. the action move(X,Y) can take place if there is no block on X or Y.

A plan is a sequence of actions. The problem we want to solve is to generate a plan so that a configuration can be reached from some initial state.

**Example:** Optimization

There is a long list of important real world optimization problems. Most are NP-hard or worse.

Example, taken from https://www.mat.unical.it/aspcomp2011/FinalProblemDescription /FastfoodOptimalityCheck

The fastfood chain [McBurger](https://www.mat.unical.it/aspcomp2011/McBurger) owns several restaurants along a highway. Recently, they have decided to build several depots along the highway, each one located at a restaurant and supplying several of the restaurants with the needed ingredients. Each of the restaurants will be supplied by the nearest depot. If two or more depots are equidistantly nearest to a restaurant, it is supplied by exactly one of these. Naturally, the depots should be placed so that the average distance between a restaurant and its assigned depot is minimized, or equivalently the sum of supply distances is minimized.

Many constraint optimization problems are studied in mathematics and engineering. This field is called "operations research".

## **Approaches to Solving Constraints**

Given a problem, you can develop an algorithm for it (hope it is efficient), and implement the algorithm in a computer language, and run it. Or you can try one of the constraint programming languages, in which case you are responsible, not for developing an algorithm, but for modeling the problem in the underlying language.   
  
There are two main approaches to Constraint Programming: one is based on SAT, and the other on Constraint Satisfaction.  In this course, we will study the latter approach.

## Constraint Satisfaction

One way to solve a constraint problem is to formalize it as a constraint satisfaction problem.

A constraint satisfaction problem (CSP) consists of three parts:

1. A set of variables X={x1,...,xn}  
2. For each variable xi, a finite set Di of possible values (its domain)   
3. A set of constraints restricting the values that the variables can take  
simultaneously.

Note that values need not be a set of consecutive integers (although often they are), they need not even be numeric.

A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. We may want to find:

* just one solution, with no preference as to which one,
* all solutions,
* an optimal, or at least a good solution, given some objective functiondefined in terms of some or all of the variables.

**Example**

Suppose we have variables X,Y,Z, and their domains are

D\_X = D\_Y = {1,2,3}, D\_Z = {2,3,4}

and the constraints are

X < Y and Y < Z.

It's clear that this CSP is satisfiable. For example, the assignment

X=1, Y=2, Z=3

is a solution.

**Example**

Using the 4-queen problem as an example, we can use a variable to represent each queen, thus we have 4 variables, say,

x1, x2, x3, x4

Assume x1 is placed on the first row, x2 on the second row, and so on. The domain for each queen then consists of 4 columns. Thus, we have domains for the variables as

D\_x1 = {1,2,3,4},   
 D\_x2 = {1,2,3,4},   
 D\_x3 = {1,2,3,4},   
 D\_x4 = {1,2,3,4},

The constraints that must be satisfied can be stated as relations. E.g. the constraint that says x1 and x2 cannot be on the same column can be expressed by the set of tuples satisfying it:

r1(x1,x2) ={(1,2),(1,3),(1,4),  
 (2,1),(2,3),(2,4),  
 (3,1),(3,2),(3,4),  
 (4,1),(4,2),(4,3)}

Stating satisfied tuples is tedious. In a suitable language, one may express this by

r1(x1,x2) if x1 is not equal to x2, and  
 x1 takes a value from D\_x1 and x2 takes a value from D\_x2

We can express constraints mathemathically, or by using Prolog-like syntax. For example, to express that all queens are on different rows, we can specify that all variables are pairwise distinct (i.e., the values they get are distinct).

x1 =/= x2, x1 =/= x3, x1 =/= x4

x2 =/= x3, x2 =/= x4

x3 =/= x4

To specify that no queens can be on the same diagonal, we can write

|xi - xj| =/= |i - j|,  where i =/= j

That is, the difference of the rows of any two queens is not the same as the difference of their columns. For example,  if x1=3 and x3=1, then the two queens on the board are on the same diagonal.

We need a language to express all three parts of a CSP. Logic programming has been extended to constraint logic programming (CLP), where language constructs are provided for us to model a constraint problem as a CSP.

## Constraint Solving Methods

A constraint is local but a solution to all constraints is global. For example, consider X < Y and Y < Z over the domain {1,2,3} for all the variables. X=1 and Y=3 is a solution to the constraint X < Y, but cannot give a solution when Y < Z is also considered.

### **Systematic Search Algorithms**

A CSP can be solved using generate-and-test paradigm (GT) that systematically generates each possible value assignment and then tests it to see if it satisfies all the constraints. A more efficient method uses the backtracking paradigm (BT), one of the most common algorithms for performing systematic search. Backtracking incrementally attempts to extend a partial solution toward a complete solution, by repeatedly choosing a value for another variable.

**Example**

Consider X < Y and Y < Z over the domain {1,2,3}. We assign a value to a variable, one at a time. Whenever a variable is assigned, it's checked against all previous assignments to see whether it's "so far so good" (the constraints involving these assigned variables are not violated).

X assigned X=1   
 (so far so good) X < Y is not violated   
 / \  
Y assigned Y=1 Y=2   
 (no good) (so far so good) X < Y and Y < Z are not violated  
 / | \  
Z assigned Z=1 Z=2 Z=3   
 (no good)(no good) All var's are assigned

### **Consistency Techniques**

Consider X < Y and Y < Z over the domain {1,2}. There is no solution for this CSP, but we only discover that when an assignment for Z is needed.

The late detection of inconsistency is a major disadvantage of BT. Thus, consistency techniques are applied initially before any variable is assigned, and during the backtrack search, after each variable is assigned. The idea is to discover values which are not possible, so they can be safely removed for the assignment so far (they are not removed permanently; they will be recovered upon backtracking).

The most common techniques are *node consistency* and *arc consistency*.

Node consistency only applies to constraints with one unassigned variable, since it's trivial to check which values for the variable are not possible, and therefore should be removed.

For example, consider X > 2 over the domain {1,2,3,4} for X. After we apply node consistency, the domain for X is reduced to {3,4}. The CSP where the constraint is X > 2 over the domain {3,4} for X is called *node consistent*.

Arc consistency only applies to a constraint with two unassigned variables, say X and Y. If for a value of X, there is no value of Y such that the constraint is satisfied, then this value for X can be removed. This is done repeatedly until no domain reduction is possible.

For example, consider X < Y and Y < Z over the domain Dx = Dy = Dz = {1,2,3}.

Consider the constraint X < Y first. Apparently, X=3 cannot give a solution for any Y. So Dx is reduced to Dx1 = {1,2}. Similarly, we get Dy1={2,3}. Consider Y < Z. Given Dy1={2,3}, the only value for Z that can satisfy the constraint is Z=3, thus Dz={3}. Similarly, Dy1 is reduced to Dy2={2}. And finally, Dx1 is reduced to Dx2={1}.

The CSP with constraints X < Y and Y < Z over the domain Dx = {1}, Dy = {2}, and Dz = {3} is called *arc consistent*.

### **Additional Space Pruning Techniques**

If you want to more information on CSP related topics, you can start from [here](http://en.wikipedia.org/wiki/Constraint_satisfaction_problem).

Last modified: Monday, 21 March 2016, 8:53 PM