Combinatorial Optimization

Applications: resource allocation, scheduling, routing

Properties: computationally difficult, require technical expertise to solve, but important in practice

Solution Techniques:

- Integer Programming
- Specialized Methods
- Metaheuristics
- *Constraint Programming*

Constraint programming is an approach to solving combinatorial optimization problems with roots in computer science. For some complex problems, constraint programming vastly outperforms integer programming techniques.

Constraint Programming

- Improved language for specifying constraints and search procedures
- Much easier to code for some problems than integer programming
- Supported in IBM OPL

Constraint programming is supported in IBM Optimization Studio. IBM Optimization Studio has two computing engines, the CPLEX engine for solving linear programming problems and a CP engine for solving combinatorial optimization problems using constraint programming. The structure of an OPAL CP program is very similar to the structure used in an OPAL model that uses CPLEX to solve a linear programming problem.

The Details of Constraint Programming

- Algorithms based on branching and pruning
- Careful use of constraints to reduce possible values for variables
- Can shape how the search proceeds through the possible solutions
- **FOCUS:** Find feasible solutions
- **Secondary:** Optimize by trying all of the feasible solutions

Constraint programming algorithms are based on branching and pruning, which is similar to branch and cut algorithms for integer programming. The constraints are used to remove or prune branches of solutions which violate the constraints. The emphasis in constraint programming is not on optimization, but instead on feasibility. We merely want to identify a feasible solution or possibly all of the feasible solutions. If we do need to do optimization, then constraint programming first identifies the feasible solutions and then evaluates the objective function for each feasible solution.

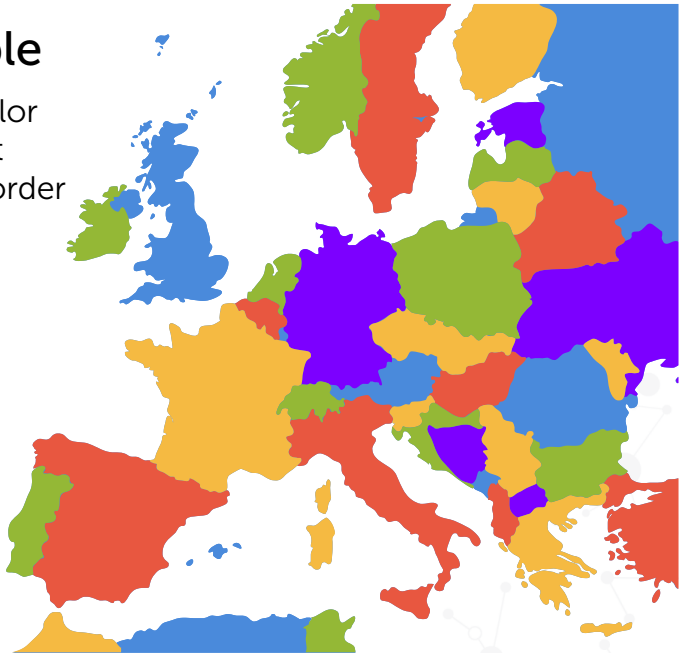
Lesson Goals

- Provide an introduction
  - Not comprehensive
  - Make you curious
- Illustrate rich language via examples in OPL

We'll spend just one week getting a taste of constraint programming. And while this certainly won't make you an expert, hopefully it will make you aware of some of the applications and make you curious to learn more. We'll see just a few examples to illustrate some of the ideas of constraint programming, and we'll focus on how to use the language to specify constraints. We won't be looking at how to shape or customize the search through the possible solution space. To get a few more details about constraint programming, a good place to start is the Wikipedia page.

A Coloring Example

Use at most 4 colors to color countries on a map so that countries which share a border are colored differently.



Map coloring is a famous mathematical problem, and it is a mathematical theorem that any planar map can be colored with at most four colors in such a way that any two adjacent countries are assigned different colors. Adjacency means that the countries share a boundary longer than a single point. Most map makers actually use more than four colors to color their maps, like in the map of Europe shown here. There's a nice bit of trivia that the Four Color Theorem was the first mathematical theorem to be proved by a computer in 1976. On the next slide, we'll show you an OPAL program for solving a small map coloring problem.

An OPL Coloring Program

```
range r = 0..3;
string Names[r] = ["blue", "white", "yellow", "green"];
dvar int Belgium in r;
dvar int Denmark in r;
dvar int France in r;
dvar int Germany in r;
dvar int Luxembourg in r;
dvar int Netherlands in r;

subject to {
    Belgium != France;
    Belgium != Germany;
    Belgium != Netherlands;
    Belgium != Luxembourg;
    Denmark != Germany;
```


This is a simple example to illustrate a constraint program. We're going to look at a small version of the Europe coloring problem with just six countries. We can always get away with using just four colors, thanks to our mathematical theorem.

A little of the program doesn't fit on the side, but you can see it's quite similar to the programs we wrote to solve linear programs in OPAL. It begins with the variable declarations and then has the constraints. Notice, there is no minimize or maximize here, since our only concern is finding a feasible map coloring. In the next slide, we'll demonstrate running the program in CPLEX Studeio.

Demonstrate the Color Problem in OPL

Video

Global and Specialized Constraints

- Global constraints effect all of the variables at once
  - *All-different* constraint ensures that the values assigned to some variables are all unique
- Specialized constraints for a variety of common problems
  - *Pack* for packing items into containers
  - *Lexicographic* enforces lexical ordering
  - Precedence constraints for scheduling problems

The main advantage of constraint programming is that certain kinds of constraints can be specified quite simply, which allows us to sometimes write simple and efficient programs. Some constraints are called global constraints, as they constrain all of the variables simultaneously. We'll make heavy use of the all different constraint, for instance.

Other constraints are specialized constraints that apply to certain categories of problems. For instance, we'll use precedence constraints to identify a feasible schedule for a construction project. In the next slide we'll present a simple problem that will utilize the all different constraint.

Example One – A Simple Problem

Find a feasible solution with the following constraints:

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$

$x_3 \in \{1,2,3\}$

$x_4 \in \{1,2,3,4,5\}$

$x_1 + x_3 = 4$

x_1, x_2, x_3, x_4 are all different

This problem is similar to an integer programming problem, except that we aren't trying to optimize anything. Also, the last constraint requires all the variables to have different values, which is not a typical integer programming constraint, but is simple to handle in most constraint programming engines. The next slide is a video where we'll show you how to solve this problem using constraint programming in OPAL.

Example One

Video

Using Decision Variables to Index Arrays

- An integer valued decision variable can index an array
- Textbook calls this an *element* constraint (page 529)

```
dvar int x1 in 1..3;
dvar int x2 in 1..3;
int z1[1..3] = [5,10,20];
int z2[1..3] = [3,7,12];

maximize z[x1] + z[x2];
subject to{
    z1[x1] + z2[x2] <= 30
}
```

The max value is 27 and occurs when `x1 = 3 (z1 = 20)` and `x2 = 2 (z2 = 7)`.

One of the more powerful elements of constraint programming is that we can use integer value decision variables to index arrays. We'll make use of this capability in several places. Here, we want to maximize the sum of the z variables, but they aren't simple functions of x1 and x2. There's no linear relationship between the values.

They don't go consecutive, like 5, 10, 15, and so on. And they don't fall into consecutive integer values like 1, 2, 3. So instead, what we're going to do is make x1 and x2 act as indices for z1 and z2. x1 and x2 will be our decision variables. If you stare at this for a bit, you'll see that maximum value is 27, and it occurs when the decision variables x1 and x2 are 3 and 2, respectively.

Example Two – Optimizing Advertising Profit

- Buy 5 total ad spots: x1, x2, and x3 for products 1, 2, and 3
- No more than 3 ad spots for any one product
- Profit varies nonlinearly with the number of ad spots with no simple formulas
- Maximize profit

In this example. We want to optimize or maximize advertising profit. Unfortunately, the amount of profit does not vary linearly with the number of ad spots that we buy. The formulas are more complex. You can read about the details in the textbook, but I'll try to explain the problem in more detail as we go through an OPAL program for solving this problem in the next slide.

Example Two

Video

Example Three – Using General Values from Sets

Maximize $10x_1 + 2x_2 - 223$

Subject to

$x_1 \in \{10,20,30\}$

$x_2 \in \{20,30,40\}$

$x_3 \in \{10,30,50\}$

x_1, x_2 and x_3 are all different.

While this example looks very much like example one, example one was easier because each variable x1, x2, was restricted to be in a range of consecutive integer values, whereas here they come from more general sets. For instance, x1 can be 10, 20, or 30. Moreover, we've added an optimization element to this problem, and look at the optimization function. We want to maximize 10x1 plus 2x2 minus x3 squared.

We can do any kind of optimization we wish here, because we're simply going to look at all the feasible solutions and compute this value 10x1 plus 2x2, et cetera, for each of those feasible solutions. Finally, we want x1, x2, and x3 to be all different. We'll see how we do this in our program in the next video.

Example Three

Video

The Traveling Salesman Problem

Video

Scheduling Problems

- Have a sequence of tasks that each take a certain amount of time
- Some must be completed before others, but some may overlap
- e.g. A before B, A before C, but B and C together OK
- Goal: Feasible schedule

One of the strengths of constraint programming is its ability to tackle complex scheduling problems. We're going to see only a very basic example, and you'll extend it just a little in the homework. In the download folder for the week, you'll find a document called "Getting started with scheduling in CPLEX Studio."

It's written in the form of a tutorial. If you find you need to work on scheduling problems in the future, then that tutorial is a great place to start. In the next slide, we'll show you a video of where we describe the introductory example in that tutorial. The homework problem is quite similar to the second example in the tutorial, so you'll likely want to go through that on your own.

Scheduling Problem Example

Video