DM841
Discrete Optimization

Modeling for CP

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Resume

- CP modeling examples
  - Graph labeling with consecutive numbers
  - Send More Money
- Constraint programming:
  representation (modeling language) + reasoning (propagation + search)
  - model
  - propagate, filtering, pruning
  - search = backtracking + branching
- Gecode: model in Script class implementation
  - Variables:
    declare as members
    initialize in constructor
    update in copy constructor
  - Posting constraints (in constructor)
  - Create branching (in constructor)
  - Provide copy constructor (recomputation) and copy function (cloning)

List of Contents

Outline

Constraint Programming

The **domain** of a variable x, denoted $D(x)$, is a finite set of elements that can be assigned to x.

A **constraint** C on X is a subset of the Cartesian product of the domains of the variables in X, i.e., $C \subseteq D(x_1) \times \cdots \times D(x_k)$. A tuple $(d_1, \ldots, d_k) \in C$ is called a solution to C.
Equivalently, we say that a solution $(d_1, ..., d_k) \in C$ is an assignment of the value d_i to the variable x_i for all $1 \leq i \leq k$, and that this assignment satisfies C. If $C = \emptyset$, we say that it is inconsistent.

Extensional: specifies the good (or bad) tuples (values)
Intensional: specifies the characteristic function

Constraint Programming

Constraint Satisfaction Problem (CSP)

A CSP is a finite set of variables $\mathcal{X}$ with domain extension $\mathcal{D} = D(x_1) \times \cdots \times D(x_n)$, together with a finite set of constraints $\mathcal{C}$, each on a subset of $\mathcal{X}$. A **solution** to a CSP is an assignment of a value $d \in D(x)$ to each $x \in \mathcal{X}$, such that all constraints are satisfied simultaneously.

Constraint Optimization Problem (COP)

A COP is a CSP $\mathcal{P}$ defined on the variables $x_1, \ldots, x_n$, together with an objective function $f : D(x_1) \times \cdots \times D(x_n) \rightarrow Q$ that assigns a value to each assignment of values to the variables. An **optimal solution** to a minimization (maximization) COP is a solution d to $\mathcal{P}$ that minimizes (maximizes) the value of $f(d)$.

Task:

- determine whether the CSP/COP is consistent (has a solution):
- find one solution
- find all solutions
- find one optimal solution
- find all optimal solutions

Solving CSPs

- Systematic search:
  - choose a variable x_i that is not yet assigned
  - create a choice point, i.e. a set of mutually exclusive & exhaustive choices, e.g. $x_i = v$ vs $x_i \neq v$
  - try the first & backtrack to try the other if this fails
- Constraint propagation:
  - add $x_i = v$ or $x \neq v$ to the set of constraints
  - re-establish local consistency on each constraint
    $\leadsto$ remove values from the domains of future variables that can no longer be used because of this choice
  - fail if any future variable has no values left

Representing a Problem

- a CSP $\mathcal{P} =< X, \mathcal{D}, \mathcal{C} >$ represents a problem P, if every solution of $\mathcal{P}$ corresponds to a solution of P and every solution of P can be derived from at least one solution of $\mathcal{P}$
- More than one solution of $\mathcal{P}$ can represent the same solution of P or viceversa, if symmetries are present
- The variables and values of $\mathcal{P}$ represent entities in P
- The constraints of $\mathcal{P}$ ensure the correspondence between solutions
- we must make sure that any solution to $\mathcal{P}$ yields exactly one solution to P, and that any solution to P corresponds to a solution to $\mathcal{P}$ or is symmetrically equivalent to such a solution, and that if $\mathcal{P}$ has no solutions, this is because P itself has no solutions.
- The aim is to find a model $\mathcal{P}$ that can be solved as quickly as possible (Note that shortest run-time might not mean least search!)

Interactions with Search Strategy

Whether a model is better than another can depend on the search algorithm and search heuristics

- Let's assume that the search algorithm is fixed
  although different level of consistency can also play a role
- Let's also assume that choice points are always $x_i = v$ vs $x_i \neq v$
- Variable (and value) order still interact with the model a lot
- Is variable & value ordering part of modelling?

In practice it is.
but it depends on the modeling language used

Global Constraint: `alldifferent`

Global constraint:

set of more elementary constraints that exhibit a special structure when considered together.

`alldifferent` constraint

Let $x_1, x_2, \ldots, x_n$ be variables. Then:

$$\texttt{alldifferent}(x_1, ..., x_n) = \{(d_1, ..., d_n) \mid \forall i, d_i \in D(x_i), \quad \forall i \neq j,\ d_i \neq d_j\}.$$

Constraint arity: number of variables involved in the constraint

Note: different notation and names used in the literature

Global Constraint Catalog

http://www.emn.fr/z-info/sdemasse/gccat/sec5.html

Global Constraint Catalog

Corresponding author: Nicolas Beldiceanu nicolas.beldiceanu@emn.fr

Online version: Sophie Demassey sophie.demassey@emn.fr

Google Search ○ Web ⊙ Catalog
○ all formats ⊙ html ○ pdf

Global Constraint Catalog
html / 2009-12-16

Search by:

NAME Keyword Meta-keyword Argument pattern Graph description
Bibliography Index

Keywords (ex: *Assignment, Bound consistency, Soft constraint,...*) can be searched by **Meta-keywords** (ex: *Application area, Filtering, Constraint type,...*)

About the catalogue

The catalogue presents a list of 348 global constraints issued from the literature in constraint programming and from popular constraint systems. The semantic of each constraint is given together with a description in terms of graph properties and/or automata.

Outline

1. Constraint Satisfaction Problem
2. Modeling Examples
   n-Queens, Grocery, Magic Squares
3. Example: Sudoku

Outline

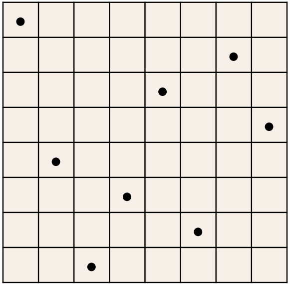
1. Constraint Satisfaction Problem

2. Modeling Examples
   n-Queens, Grocery, Magic Squares

3. Example: Sudoku

8-Queens

Problem Statement



- Place 8 queens on a chess board such that the queens do not attack each other
- Straightforward generalizations
  - place an arbitrary number: *n* Queens
  - place as closely together as possible

What Are the Variables?

- Representation of position on board
- First idea: two variables per queen
  - one for row
  - one for column
  - $2 \cdot n$ variables
- Insight: on each column there will be a queen!

Fewer Variables…

- Have a variable for each column
  - value describes row for queen
  - *n* variables

- Variables: $x_0, \ldots, x_7$

  where $x_i \in \{0, \ldots, 7\}$

Other Possibilities

- For each field: number of queen
  - which queen is not interesting, so…
  - n^2 variables
- For each field on board: is there a queen on the field?
  - 8×8 variables
  - variable has value 0: no queen
  - variable has value 1: queen
  - n^2 variables

Constraints: No Attack

- not in same column
  - by choice of variables
- not in same row
  - $x_i \neq x_j$ for $i \neq j$
- not in same diagonal
  - $x_i - i \neq x_j - j$ for $i \neq j$
  - $x_i - j \neq x_j - i$ for $i \neq j$

- $3 \cdot n \cdot (n - 1)$ constraints

Fewer Constraints…

- Sufficient by symmetry

  $i < j$ instead of $i \neq j$

- Constraints
  - $x_i \neq x_j$ for $i < j$
  - $x_i - i \neq x_j - j$ for $i < j$
  - $x_i - j \neq x_j - i$ for $i < j$

- $3/2 \cdot n \cdot (n - 1)$ constraints

Even Fewer Constraints

- Not same row constraint

  $x_i \neq x_j$ for $i < j$

  means: values for variables pairwise distinct
- Constraints
  - distinct(x_0, …, x_7)
  - $x_i – i \neq x_j - j$ for $i < j$
  - $x_i – j \neq x_j - i$ for $i < j$

Pushing it Further…

- Yes, also diagonal constraints can be captured by distinct constraints
  - ~~see assignment~~

```
distinct(x0,     x1,    ...,  x7)
distinct(x0-0,  x1-1, ..., x7-7)
distinct(x0+0, x1+1, ..., x7+7)
```

Script: Variables

```
Queens(void) : q(*this,8,0,7) {
  …
}
```

Script: Constraints

```
Queens(void) : q(*this,8,0,7) {
  distinct(*this, q);
  for (int i=0; i<8; i++)
    for (int j=i+1; j<8; j++) {
  rel post(*this, x[i]-i != x[j]-j);
      post(*this, x[i]-j != x[j]-i);
    }
  …
}
```

Script: Branching

```
Queens(void) : q(*this,8,0,7) {
  …
  branch(*this, q,
         INT_VAR_NONE,
         INT_VAL_MIN);
}
```

Good Branching?

- Naïve is not a good strategy for branching

- Try the following (see assignment)
  - first fail
  - place queen as much in the middle of a row
  - place queen in knight move fashion

Summary 8 Queens

- Variables
  - model should require few variables
  - good: already impose constraints
- Constraints
  - do not post same constraint twice
  - try to find “big” constraints subsuming many small constraints
    - more efficient
    - often, more propagation (to be discussed)

Grocery

Grocery

- Kid goes to store and buys four items
- Cashier: that makes \$7.11
- Kid: pays, about to leave store
- Cashier: hold on, I multiplied!
  let me add!
  wow, sum is also \$7.11
- You: prices of the four items?

Model

- Variables
  - for each item A, B, C, D
  - take values between {0, …, 711}
  - compute with cents: allows integers
- Constraints
  - A + B + C + D = 711
  - A * B * C * D = 711 * 100 * 100 * 100

The unique solution (upon the symmetry breaking of slide 87) is:
A=120, B=125, C=150, D=316.

Script

```
class Grocery : public Space {
protected:
    IntVarArray abcd;

    const int s = 711;
    const int p = s * 100 * 100 * 100;
public:
    Grocery(void) … { … }

    …
}
```

Script: Variables

```
Grocery(void) : abcd(*this,4,0,711) {
  …
}
```

Script: Sum

```
…
// Sum of all variables is s
linear(this, abcd, IRT_EQ, s);

IntVar a(abcd[0]), b(abcd[1]),
       c(abcd[2]), d(abcd[3]);
```

Script: Product

```
IntVar t1(*this,1,p);
IntVar t2(*this,1,p);
IntVar t3(*this,p,p);

mult(*this, a, b, t1);
mult(*this, c, d, t2);
mult(*this, t1, t2, t3);
```

Branching

- Bad idea: try values one by one
- Good idea: split variables
  - for variable x
  - with $m = (\min(x) + \max(x)) / 2$
  - branch $x < m$ or $x \geq m$
- Typically good for problems involving arithmetic constraints
  - exact reason needs to be explained later

Script: Branching

```
branch(*this, abcd,
       INT_VAR_NONE,
       INT_VAL_SPLIT_MIN);
```

Search Tree

- 2829 nodes for first solution
- Pretty bad…

Better Heuristic?

- Try branches in different order

  split with larger interval first
  - try: `INT_VAL_SPLIT_MAX`
- Search tree: 2999 nodes
  - worse in this case

Symmetries

- Interested in values for A, B, C, D
- Model admits equivalent solutions
  - interchange values for A, B, C, D
- We can add order A, B, C, D:

  $A \leq B \leq C \leq D$
- Called “symmetry breaking constraint”

Script: Symmetry Breaking

```
…
rel(this, a, IRT_LQ, b);
rel(this, b, IRT_LQ, c);
rel(this, c, IRT_LQ, d);
…
```

Effect of Symmetry Breaking

- Search tree size 308 nodes
- Let us try `INT_VAL_SPLIT_MAX` again
  - tree size 79 nodes!
  - interaction between branching and symmetry breaking
  - other possibility: $A \geq B \geq C \geq D$
  - we need to investigate more (later)!

Any More Symmetries?

- Observe: 711 has prime factor 79
  - that is: $711 = 79 \times 9$
- Assume: A can be divided by 79
  - add: $A = 79 \times X$
    for some finite domain var X
  - remove $A \leq B$
  - the remaining B, C, D of course can still be ordered

Any More Symmetries?

- In Gecode

```
IntVar x(*this,1,p);
IntVar sn(*this,79,79);
mult(*this, x, sn, a);
```

- Search tree 44 nodes!
  - now we are talking!

Summary: Grocery

- Branching: consider also
  - how to partition domain
  - in which order to try alternatives
- Symmetry breaking
  - can reduce search space
  - might interact with branching
  - typical: order variables in solutions
- Try to really understand problem!

Domination Constraints

- In symmetry breaking, prune solutions without interest
- Similarly for best solution search
  - typically, interested in just one best solution
  - impose constraints to prune some solutions with same "cost"

Another Observation

- Multiplication decomposed as

  $A \cdot B = T_1 \quad C \cdot D = T_2 \quad T_1 \cdot T_2 = P$

- What if

  $A \cdot B = T_1 \quad T_1 \cdot C = T_2 \quad T_2 \cdot D = P$

  - propagation changes: 355 nodes
  - propagation is not compositional!
  - another point to investigate

Magic Squares

2	9	4
7	5	3
6	1	8

Unique solution for n=3, upon the symmetry breaking of slide 99.

Magic Squares

- Find an $n \times n$ matrix such that
  - every field is integer between 1 and n^2
  - fields pairwise distinct
  - sums of rows, columns, two main diagonals are equal
- Very hard problem for large n
- Here: we just consider the case n=3

Model

- For each matrix field have variable x_{ij}
  - $x_{ij} \in \{1, .., 9\}$
- One additional variable s for sum
  - $s \in \{1, .., 9 \times 9\}$
- All fields pairwise distinct
  - distinct(x_{ij})
- For each row i have constraint
  - $x_{i0} + x_{i1} + x_{i2} = s$
  - columns and diagonals similar

Script

- Straightforward
- Branching strategy
  - first-fail
  - split again: arithmetic constraints
  - try to come up with something that is really good!

- Generalize it to arbitrary *n*

Symmetries

- Clearly, we can require for first row that first and last variable must be in order
- Also, for opposing corners
- In all (other combinations possible)
  - $x_{00} < x_{02}$
  - $x_{02} < x_{20}$
  - $x_{00} < x_{22}$

Important Observation

- We know the sum of all fields

  $1 + 2 + \ldots + 9 = 9(9+1)/2=45$

- We "know" the sum of one row

  s

- We know that we have three rows

  $3 \times s = 45$

Implied Constraints

- The constraint model already implies

  $3 \times s = 45$

  - implies solutions are the same
- However, adding a propagator for the constraint drastically improves propagation
- Often also: redundant or implied constraint

Effect

Simple model	92 nodes
Symmetry breaking	29 nodes
Implied constraint	6 nodes

Summary: Magic Squares

- Add implied constraints
  - are implied by model
  - increase constraint propagation
  - reduce search space
  - require problem understanding
- Also as usual
  - break symmetries
  - choose appropriate branching

Outlook…

- Common modeling principles
  - what are the variables
  - finding the constraints
  - finding the propagators
  - implied (redundant) constraints
  - finding the branching
  - symmetry breaking

Modeling Strategy

- Understand problem
  - identify variables
  - identify constraints
  - identify optimality criterion
- Attempt initial model   simple?
  - try on examples to assess correctness
- Improve model   much harder!
  - scale up to real problem size

Outline

Example: Sudoku

Model and solve the following Sudoku in MIP and CP

	4	3		8		2	5	
6								
					1		9	4
9					4		7	
			6		8			
	1		2					3
8	2		5					
								5
	3	4		9		7	1	

Sudoku: ILP model

Let y_{ijt} be equal to 1 if digit t appears in cell (i, j). Let N be the set $\{1, \ldots, 9\}$, and let J_{kl} be the set of cells (i, j) in the 3×3 square in position k, l.

$$\sum_{j \in N} y_{ijt} = 1, \qquad \forall i, t \in N,$$

$$\sum_{j \in N} y_{jit} = 1, \qquad \forall i, t \in N,$$

$$\sum_{i,j \in J_{kl}} y_{ijt} = 1, \qquad \forall k, l = \{1, 2, 3\}, t \in N,$$

$$\sum_{t \in N} y_{ijt} = 1, \qquad \forall i, j \in N,$$

$$y_{i,j,a_{ij}} = 1, \qquad \forall i, j \in \text{ given instance.}$$

Sudoku: CP model

Model:

$$
\begin{aligned}
&X_{ij} \in N, && \forall i, j \in N,\\
&X_{ij} = a_{ij}, && \forall i, j \in \text{ given instance},\\
&\text{alldifferent}([X_{1i}, \ldots, X_{9i}]), && \forall i \in N,\\
&\text{alldifferent}([X_{i1}, \ldots, X_{i9}]), && \forall i \in N,\\
&\text{alldifferent}(\{X_{ij} \mid ij \in J_{kl}\}), && \forall k, l \in \{1, 2, 3\}.
\end{aligned}
$$

Search: backtracking

Sudoku: CP model (revisited)

$$
\begin{aligned}
&X_{ij} \in N, && \forall i, j \in N, \\
&X_{ij} = a_t, && \forall i, j \in \text{ given instance}, \\
&\text{alldifferent}([X_{1i}, \ldots, X_{9i}]), && \forall i \in N, \\
&\text{alldifferent}([X_{i1}, \ldots, X_{i9}]), && \forall i \in N, \\
&\text{alldifferent}(\{X_{ij} \mid ij \in J_{kl}\}), && \forall k, l \in \{1,2,3\}.
\end{aligned}
$$

Redundant Constraint:

$$
\begin{aligned}
&\sum_{j \in N} X_{ij} = 45, && \forall i \in N, \\
&\sum_{j \in N} X_{ji} = 45, && \forall i \in N, \\
&\sum_{ij \in J_{kl}} X_{ij} = 45, && k, l \in \{1,2,3\}.
\end{aligned}
$$

Viewpoints

Viewpoint $(\mathcal{X}, \mathcal{D})$:

- same solutions
- can be combined
- rule of thumb in choosing a viewpoint:
  it should allow the constraints to be easily and concisely expressed;
  the problem to be described using as few constraints as possible, as long as those constraints have efficient, low-complexity propagation algorithms

Releated concept: auxiliary variables and linking or channelling

Modeling Constraints

Better understood if:

- aware of the range of constraints supported by the constraint solver and the level of consistency enforced on each and
- have some idea of the complexity of the corresponding propagation algorithms.
- combine them
- use global constraints
- extensional constraints
- implied constraints