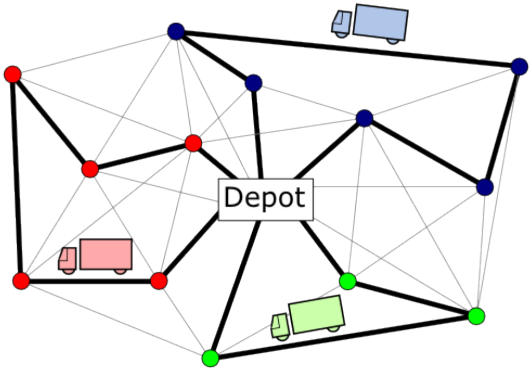
Constraint Programming in a Nutshell

Pierre Flener

ASTRA Research Group
on Combinatorial Optimisation
Department of Information Technology
Uppsala University
Sweden
www.it.uu.se/research/group/astra

Joint ACP and GdR RO Summer School 2017
2017-09-18&19

Optimisation



Optimisation is a science of service:
to scientists, to engineers, to artists, and to society.

Outline

Outline

Example (Agricultural experiment design)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley							
corn							
millet							
oats							
rye							
spelt							
wheat							

Constraints to be **satisfied**:

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.

Example (Agricultural experiment design)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

Constraints to be **satisfied**:

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.

Example (Doctor rostering)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Doctor A							
Doctor B							
Doctor C							
Doctor D							
Doctor E							

Constraints to be **satisfied**:

1. #doctors-on-call / day $= 1$
2. #operations / workday ≤ 2
3. #operations / week ≥ 7
4. #appointments / week ≥ 4
5. day off after operation day
6. ...

Objective function to be **minimised**:

- Cost: ...

Example (Doctor rostering)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Doctor A	call	–	oper	–	oper	–	–
Doctor B	app	call	–	oper	–	–	call
Doctor C	oper	–	call	app	app	call	–
Doctor D	app	oper	–	call	oper	–	–
Doctor E	oper	–	oper	–	call	–	–

Constraints to be **satisfied**:

1. #doctors-on-call / day $= 1$
2. #operations / workday ≤ 2
3. #operations / week ≥ 7
4. #appointments / week ≥ 4
5. day off after operation day
6. ...

Objective function to be **minimised**:

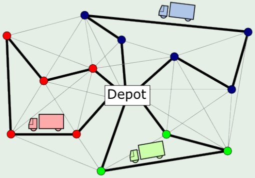
- Cost: ...

Example (Vehicle routing: parcel delivery)

Given a depot with a vehicle fleet and parcels for clients, **find** which vehicle brings which parcel to which client when.

Constraints to be **satisfied**:

1. All parcels are delivered on time.
2. No vehicle is overloaded.
3. Driver regulations are respected.
4. …



Objective function to be **minimised**:

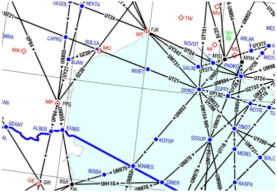
- Cost: the total fuel consumption and driver salary.

Example (Travelling salesperson: optimisation TSP)

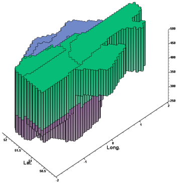
Given a map and cities, **find** a **shortest** route visiting each city once and returning to the starting city.

Applications in Air Traffic Management

Demand vs capacity



Airspace sectorisation



Contingency planning

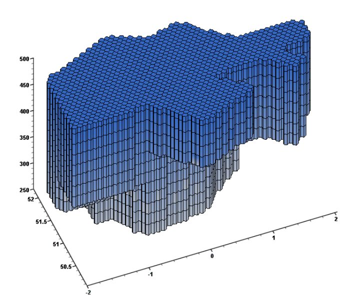
Flow	Time Span	Hourly Rate
From: Arlanda To: west, south	00:00 – 09:00 09:00 – 18:00 18:00 – 24:00	3 5 2
From: Arlanda To: east, north	00:00 – 12:00 12:00 – 24:00	4 3
...	...	...

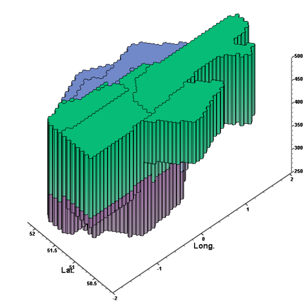
Workload balancing

Example (Airspace sectorisation)

Given an airspace split into c cells, and a targeted number s of sectors.

Find a colouring of the cells into s connected convex sectors, with minimal imbalance of the workloads of their air traffic controllers.

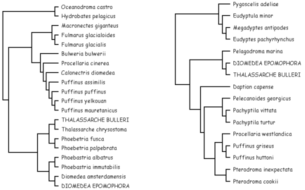




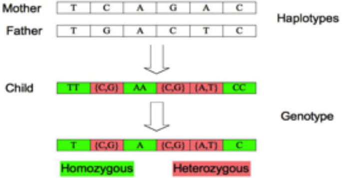
There are s^c possible colourings, but very few optimally satisfy the constraints: is intelligent search necessary?

Applications in Biology and Medicine

Phylogenetic supertree



Haplotype inference



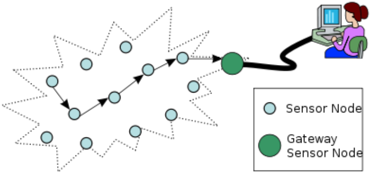
Medical image analysis

Doctor rostering

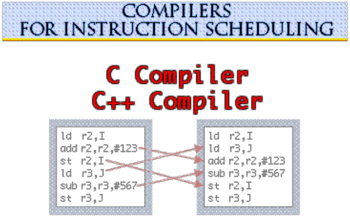
Applications in Programming and Testing

Robot-task sequencing

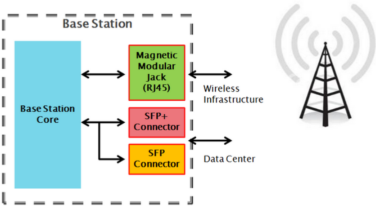
Sensor-net configuration



Compiler design

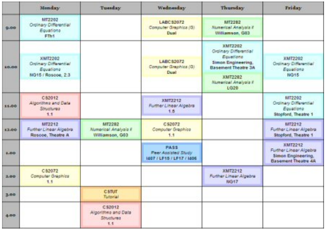


Base-station testing



Other Application Areas

School timetabling



	Monday	Tuesday	Wednesday	Thursday	Friday
9.00	MT2202 *Ordinary Differential Equations* FTh1		LABCS2072 *Computer Graphics (G)* Dual	MT2282 *Numerical Analysis II* Williamson, G03	
10.00	XMT2202 *Ordinary Differential Equations* NG15 / Roscoe, 2.3		LABCS2072 *Computer Graphics (G)* Dual	XMT2202 *Ordinary Differential Equations* Simon Engineering, Basement Theatre 3A XMT2282 *Numerical Analysis II* LG20	XMT2202 *Ordinary Differential Equations* NG15
11.00	CS2012 *Algorithms and Data Structures* 1.1		XMT2212 *Further Linear Algebra* 1.5		MT2202 *Ordinary Differential Equations* Stopford, Theatre 1
12.00	MT2212 *Further Linear Algebra* Roscoe, Theatre A	MT2282 *Numerical Analysis II* Williamson, G03	CS2072 *Computer Graphics* 1.1		MT2212 *Further Linear Algebra* Stopford, Theatre 1
1.00			PASS *Peer Assisted Study* I407 / LF15 / LF17 / I406		XMT2212 *Further Linear Algebra* Simon Engineering, Basement Theatre 4A
2.00	CS2072 *Computer Graphics* 1.1			XMT2212 *Further Linear Algebra* NG17	
3.00		CSTUT *Tutorial*			
4.00		CS2012 *Algorithms and Data Structures* 1.1			

Sports tournament design

Security: SQL injection?

Container packing

Definition

In a constraint problem, values have to be **found** for all the unknowns, called variables (in the mathematical sense) and ranging over **given** sets called domains, so that:

- All the given constraints on the variables are **satisfied**.
- Optionally: A given objective function on the variables has an optimal value: **minimal** cost or **maximal** profit.

Definition

A candidate solution to a constraint problem assigns to each variable a value within its domain.
The search space consists of all candidate solutions.

Example (Optimisation TSP over n cities)

A brute-force algorithm evaluates all $n!$ candidate routes:

- A computer of today evaluates 10^6 routes / second:

n	time
11	40 seconds
14	1 day
18	203 years
20	77k years

- Planck time is shortest useful interval: $\approx 5.4 \cdot 10^{-44}$ s; a Planck computer would evaluate $1.8 \cdot 10^{43}$ routes / s:

n	time
37	0.7 seconds
41	20 days
48	$1.5 \cdot$ age of universe

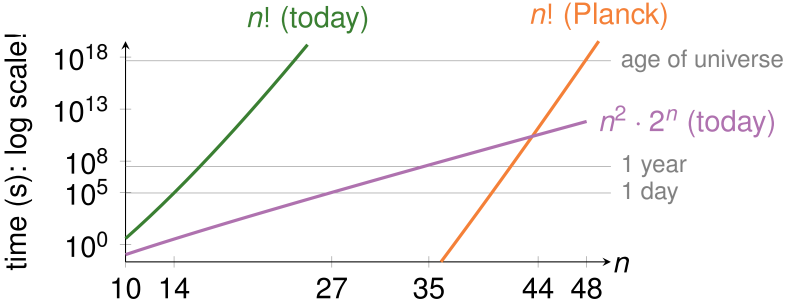
The dynamic program by Bellman-Held-Karp “only” takes $\mathcal{O}(n^2 \cdot 2^n)$ time: a computer of today takes a day for $n = 27$, a year for $n = 35$, the age of the universe for $n = 67$, and it beats the $\mathcal{O}(n!)$ algo on the Planck computer for $n \geq 44$.

Search spaces are often larger than the universe!

Many important real-life problems are NP-hard and can only be solved exactly & fast enough by intelligent search, unless P $=$ NP:

NP-hardness is not where the fun ends, but where it begins!

Do not give up but try to stay ahead of the curve: there is an instance size until which an **exact** algorithm is fast enough!



The Concorde TSP Solver beats the Bellman-Held-Karp exact algo: it uses approximation & local-search algorithms, but it can sometimes prove the exactness (optimality) of its solutions. The largest instance it has solved exactly, in 136 CPU years in 2006, has 85,900 cities! ☞ Let the fun begin!

Outline

A solving technology offers methods and tools for:

what: **Modelling** constraint problems in declarative language.

and / or

how: **Solving** constraint problems intelligently:

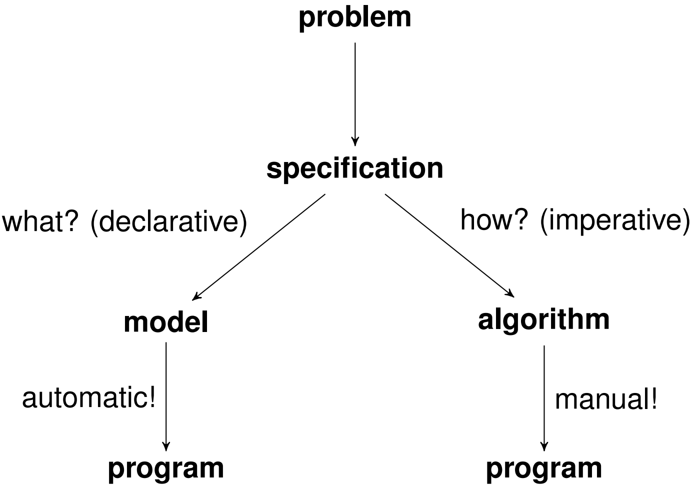
- Search: Explore the space of candidate solutions.
- Inference: Reduce the space of candidate solutions.
- Relaxation: Exploit solutions to easier problems.

A solver is a software that takes a model as input and tries to solve the modelled problem.

Combinatorial (= discrete) optimisation covers satisfaction *and* optimisation problems, for variables over *discrete* sets.

The ideas in this lecture extend to continuous optimisation, to soft optimisation, and to stochastic optimisation.

Modelling vs Programming



Example (Solving technologies)

With general-purpose solvers, taking a model as input:

- Boolean satisfiability (SAT)
- SAT modulo theories (SMT)
- (Mixed) integer linear programming (IP and MIP)
- Constraint programming (CP)
- ...
- Hybrid technologies

Techniques, *usually without* modelling and solvers:

- Dynamic programming (DP)
- Greedy algorithms
- Approximation algorithms
- Stochastic local search (SLS)
- Genetic algorithms (GA)
- ...

Constraint Programming Technology

Constraint programming (CP) offers methods & tools for:

what: Modelling constraint problems in a high-level language.

and

how: **Solving** constraint problems intelligently by:

- either default search upon pushing a button
- or systematic search guided by user-given strategies
- or local search guided by user-given (meta-)heuristics
- or hybrid search

plus inference, called propagation, but little relaxation.

Slogan of CP:

Constraint Program = Model [+ Search]

Outline

Example (Agricultural experiment design, AED)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

Constraints to be **satisfied**:

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.
General term: balanced incomplete block design (BIBD).

Example (Agricultural experiment design, AED)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	1	1	1	0	0	0	0
corn	1	0	0	1	1	0	0
millet	1	0	0	0	0	1	1
oats	0	1	0	1	0	1	0
rye	0	1	0	0	1	0	1
spelt	0	0	1	1	0	0	1
wheat	0	0	1	0	1	1	0

Constraints to be **satisfied**:

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 grains, 7 plots, 3 plots/grain, 3 grains/plot, balance 1.
General term: balanced incomplete block design (BIBD).

Example (BIBD *integer* model: $\checkmark \rightsquigarrow 1$ and $- \rightsquigarrow 0$)

```
int: nbrVarieties; int: nbrBlocks;
set of int: Varieties = 1..nbrVarieties;
set of int: Blocks = 1..nbrBlocks;
int: sampleSize; int: blockSize; int: balance;
array[Varieties,Blocks] of var 0..1: BIBD;
solve satisfy;
constraint forall(v in Varieties)
 (sampleSize = sum(BIBD[v,..]));
constraint forall(b in Blocks)
 (blockSize = sum(BIBD[..,b]));
constraint forall(v1, v2 in Varieties where v1 < v2)
 (balance = sum(b in Blocks)(BIBD[v1,b]*BIBD[v2,b]));
```

Example (Instance data for our AED)

```
nbrVarieties = 7; nbrBlocks = 7;
sampleSize = 3; blockSize = 3; balance = 1;
```

Reconsider the model fragment:

```
constraint forall(v1, v2 in Varieties where v1 < v2)
 (balance = sum(b in Blocks)(BIBD[v1,b]*BIBD[v2,b]));
```

This constraint is declarative (and by the way non-linear), so read it using only the verb “to be” or synonyms thereof:

> *For all two ordered varieties* `v1` *and* `v2`*,*
> *the sum over all blocks* `b` *of the products*
> `BIBD[v1,b]*BIBD[v2,b]` *must equal* `balance`

The constraint is not procedural:

> *For all two ordered varieties* `v1` *and* `v2`*,*
> *we first add up, over all blocks* `b`*, the products*
> `BIBD[v1,b]*BIBD[v2,b]`*, and then we check*
> *whether that sum is equal to* `balance`

The latter reading is appropriate for solution checking, but solution finding performs no such procedural summation.

Example (Idea for another BIBD model)

barley	{plot1, plot2, plot3}
corn	{plot1, plot4, plot5}
millet	{plot1, plot6, plot7}
oats	{plot2, plot4, plot6}
rye	{plot2, plot5, plot7}
spelt	{plot3, plot4, plot7}
wheat	{plot3, plot5, plot6}

Constraints to be **satisfied**:

1. Equal sample size: Every grain is grown in 3 plots.
2. Equal growth load: Every plot grows 3 grains.
3. Balance: Every grain pair is grown in 1 common plot.

Example (BIBD *set* model: a block *set* per variety)

```
...
...
...
...
array[Varieties] of var set of Blocks: BIBD;
...
...
  (sampleSize = card(BIBD[v]));
...
  (blockSize = count(BIBD,b));
...
  (balance = card(BIBD[v1] inter BIBD[v2]));
```

Example (Instance data for our AED)

```
...
...
```

Example (Doctor rostering)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Doctor A	call	–	oper	–	oper	–	–
Doctor B	app	call	–	oper	–	–	call
Doctor C	oper	–	call	app	app	call	–
Doctor D	app	oper	–	call	oper	–	–
Doctor E	oper	–	oper	–	call	–	–

Constraints to be **satisfied**:

1. #doctors-on-call / day $= 1$
2. #operations / workday ≤ 2
3. #operations / week ≥ 7
4. #appointments / week ≥ 4
5. day off after operation day
6. ...

Objective function to be **minimised**:

- Cost: ...

Example (Doctor rostering)

```
set of int: Days = 1..7;
set of int: Mon2Fri = 1..5;
enum: Doctors = {Dr A, Dr B, Dr C, Dr D, Dr E};
enum: ShiftTypes = {app, call, oper, none};

array[Doctors,Days] of var ShiftTypes: Roster;

solve minimize ...;  % objective function

constraint forall(d in Days)
  (count(Roster[..,d],call) = 1);
constraint forall(w in Mon2Fri)
  (count(Roster[..,w],oper) <= 2);
constraint count(Roster,oper) >= 7;
constraint count(Roster,app) >= 4;
constraint forall(d in Doctors)
  (regular(Roster[d,..], (oper none|app|call|none)*);
...  % other constraints
```

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

Example (Sudoku)

```
array[1..9,1..9] of var 1..9: Sudoku;
...  % load the hints
solve satisfy;
constraint forall(row in 1..9)
    (alldifferent(Sudoku[row,..]));
constraint forall(col in 1..9)
    (alldifferent(Sudoku[..,col]));
constraint forall(i,j in {1,4,7})
    (alldifferent(Sudoku[i..i+2,j..j+2]));
```

Using variables as indices: black magic?!

Example (Job allocation at minimal salary cost)

Given jobs `1..n` and the salaries of workers `1..w`,
find a worker for each job,
such that some constraints (on the qualifications of the workers for the jobs, on workload distribution, etc) are satisfied and the total salary cost is minimal:

```
array[1..w] of int: Salary;
array[1..n] of var 1..w: Worker; % job j by Worker[j]
solve minimize sum(j in 1..n)(Salary[Worker[j]]);
constraint ...;         % qualifications, workload, etc
```

Example (Travelling salesperson over cities `1..n`)

```
array[1..n,1..n] of float: Distance;  % instance data
array[1..n] of var 1..n: Next; % go from c to Next[c]
solve minimize sum(c in 1..n)(Distance[c,Next[c]]);
constraint circuit(Next);
constraint ...;                  % side constraints, if any
```

Constraint-Based Modelling

The given models are expressed in a high-level constraint-based modelling language, typical of CP:

- There are several **types** for variables: integers (`int`), reals (`float`), Booleans (`bool`), strings (`string`), integer sets (`set`), and matrices thereof (`array`).
- There is a nice vocabulary of **predicates** (<, <=, =, !=, >=, >, `alldifferent`, `circuit`, `regular`, ...), **functions** (+, −, *, `card`, `count`, `inter`, `sum`, ...), and **connectives** (/\, \/, ...).
- There is support for *both* constraint **satisfaction** (`satisfy`) *and* constrained **optimisation** (`minimize` and `maximize`).

Most modelling languages are (much) lower-level than this!

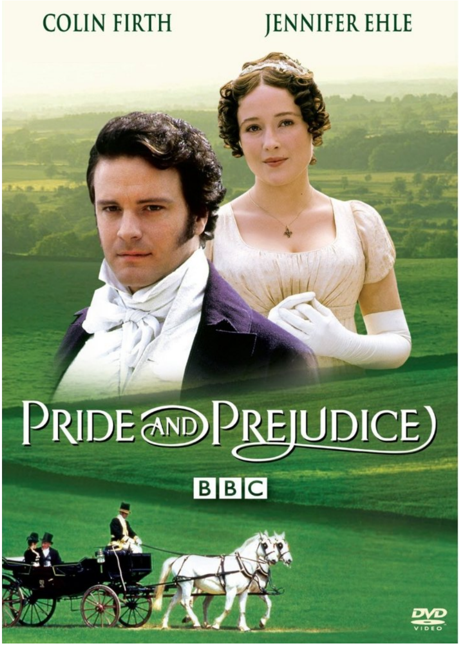
Got the Moves, But Can’t Show It?!

The constraint predicates (`alldifferent`, `circuit`, `regular`, ...) and structured variable types (sets, ...) allow us both to model the structure of a constraint problem and to exploit that structure when solving it.

Dozens of constraint predicates (see the Catalogue) declaratively encapsulate complex propagation algorithms, including for $\sum_{i=1}^{n} a_i \cdot x_i \sim b$, where $\sim \in \{<, \leq, =, \neq, \geq, >\}$.

If the scope of a predicate is an unfixed number of variables (an array of variables, a set variable, or a string variable), then one speaks of a global constraint in a CP model.

There is no standardised CP modelling language: distinct CP solvers may support distinct predicates, possibly under distinct names and signatures, as well as distinct types.
☞ But see the MiniZinc.org language & toolchain, which extends the spirit of AMPL and GAMS.


COLIN FIRTH
JENNIFER EHLE
PRIDE AND PREJUDICE
BBC
DVD
VIDEO

UPPSALA UNIVERSITET

Pride:

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.

— Eugene Freuder, a CP pioneer

Pride:

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.
— Eugene Freuder, a CP pioneer

Prejudice:

The contribution of the article should be the reduction of an engineering problem to a known optimization format. [. . .] showcases pseudo code [. . .] submit this work to a journal interested in code semantics [. . .].
— Reviewer of a paper of mine at a prestigious OR journal

Prejudice:

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.

— Eugene Freuder, a CP pioneer

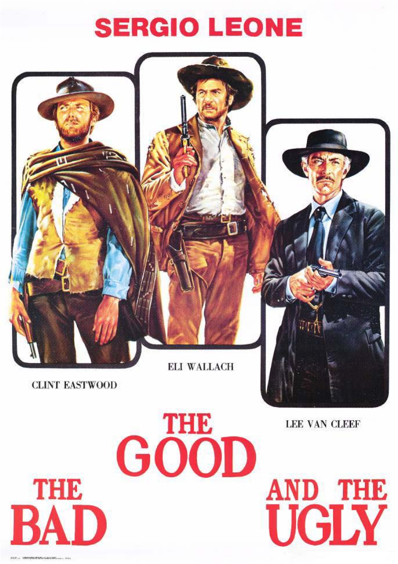
Pride:

The contribution of the article should be the reduction of an engineering problem to a known optimization format. [. . .] showcases pseudo code [. . .] submit this work to a journal interested in code semantics [. . .].

— Reviewer of a paper of mine at a prestigious OR journal

UPPSALA UNIVERSITET

Correctness Is Not Enough for Models



Modelling is an Art!

There are good & bad models for each constraint problem:

- Different models of a problem may take different time on the same solver for the same instance.
- Different models of a problem may scale differently on the same solver for instances of growing size.

A tiny model change may accelerate the solving manyfold!

Good modellers are worth their weight in gold!

Use solvers, based on decades of cutting-edge research: you reuse hundreds of thousands of lines of highly tuned code that is very hard to beat on exact solving.

Outline

CP Technology (reminder)

Constraint programming (CP) offers methods & tools for:

what: Modelling constraint problems in a high-level language.

and

how: **Solving** constraint problems intelligently by:

- either default search upon pushing a button
- or systematic search guided by user-given strategies
- or local search guided by user-given (meta-)heuristics
- or hybrid search

plus inference, called propagation, but little relaxation.

Slogan of CP:

Constraint Program = Model [+ Search]

Outline

CP Solving = Propagation + Search

A CP solver conducts search interleaved with propagation:

Each constraint has a propagator.

Propagation of one Constraint: Propagator

Example

Consider the constraint CONNECTED($[C_1, \ldots, C_n]$), which enforces max one stretch per colour among the n variables.

From

. . . ? . . .			?	?	?			. . . ? . . .

the propagator of the CONNECTED(. . .) constraint infers

no redyellow								no redblack

☞ Propagation is the elimination of ~~the~~ impossible values from the current domains of the variables, and thereby accelerates otherwise blind search.

Example

Consider the n-ary predicate `alldifferent`, with $n = 4$:

$$\texttt{alldifferent([}a, b, c, d\texttt{])} \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\ \ b \in \{4, 5\},\ \ c \in \{3, \cancel{4}\},\ \ d \in \{1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$$

No propagation by (2).

Example

Consider the n-ary predicate `alldifferent`, with $n = 4$:

$$\texttt{alldifferent([}a, b, c, d\texttt{])} \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \land a \neq c \land a \neq d \land b \neq c \land b \neq d \land c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\ \ b \in \{4, 5\},\ \ c \in \{3, \cancel{4}\},\ \ d \in \{1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$$

No propagation by (2).

Example

Consider the n-ary predicate `alldifferent`, with $n = 4$:

$$\texttt{alldifferent([}a, b, c, d\texttt{])} \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \land a \neq c \land a \neq d \land b \neq c \land b \neq d \land c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\ \ b \in \{4, 5\},\ \ c \in \{3, 4\},\ \ d \in \{1, 2, 3, 4, 5\}$$

No propagation by (2).

Example

Consider the n-ary predicate `alldifferent`, with $n = 4$:

$$\texttt{alldifferent}([a, b, c, d]) \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\ \ b \in \{4, 5\},\ \ c \in \{3, \cancel{4}\},\ \ d \in \{1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$$

No propagation by (2). But perfect propagation by (1)!

Example

Consider the n-ary predicate `alldifferent`, with $n = 4$:

$$\texttt{alldifferent([}a, b, c, d\texttt{])} \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\ b \in \{4, 5\},\ c \in \{3, \cancel{4}\},\ d \in \{1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$$

No propagation by (2). But perfect propagation by (1)!
Search: The `alldifferent` propagator is suspended, as its constraint currently does not surely hold.

Example

Consider the n-ary predicate `alldifferent`, with $n = 4$:

$$\texttt{alldifferent}([a, b, c, d]) \tag{1}$$

Modelling: (1) is equivalent to $\frac{n(n-1)}{2}$ binary constraints:

$$a \neq b \wedge a \neq c \wedge a \neq d \wedge b \neq c \wedge b \neq d \wedge c \neq d \tag{2}$$

Inference: (1) propagates much better than (2). Example:

$$a \in \{4, 5\},\; b \in \{4, 5\},\; c \in \{3, \cancel{4}\},\; d \in \{1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$$

No propagation by (2). But perfect propagation by (1)!
Search: The `alldifferent` propagator is suspended, as its constraint currently does not surely hold. If search or another propagator infers $a = 4$, then the `alldifferent` propagator is awakened: it infers $b = 5$ and is disposed of, as its constraint then surely holds, for any d.

Example (Propagator for `alldifferent`)

Solutions to `alldifferent`$([u, v, w, x, y, z])$ map to maximum matchings in a bipartite graph for the domains:

$u \in \{0, 1\}$

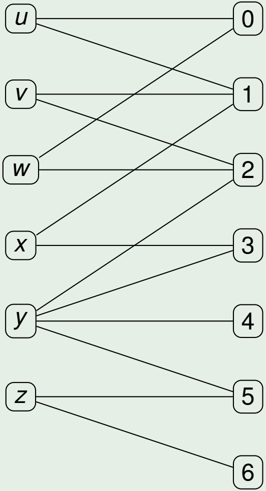
$v \in \{1, 2\}$

$w \in \{0, 2\}$

$x \in \{1, 3\}$

$y \in \{2, 3, 4, 5\}$

$z \in \{5, 6\}$



Example (Propagator for `alldifferent`)

Mark all edges of *some* maximum matching; Hopcroft-Karp algorithm takes $O(m\sqrt{n})$ time for n variables and m values:

$u \in \{0, 1\}$

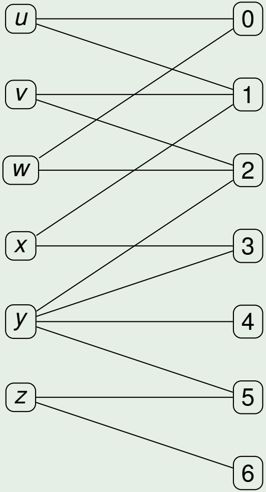
$v \in \{1, 2\}$

$w \in \{0, 2\}$

$x \in \{1, 3\}$

$y \in \{2, 3, 4, 5\}$

$z \in \{5, 6\}$



Example (Propagator for `alldifferent`)

Mark all edges of *some* maximum matching; Hopcroft-Karp algorithm takes $O(m\sqrt{n})$ time for n variables and m values:

$u \in \{0, 1\}$

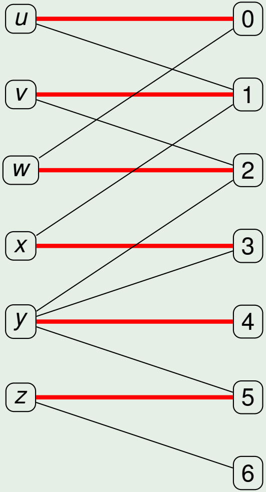
$v \in \{1, 2\}$

$w \in \{0, 2\}$

$x \in \{1, 3\}$

$y \in \{2, 3, 4, 5\}$

$z \in \{5, 6\}$



Example (Propagator for `alldifferent`)

Mark all other edges in *all* other maximum matchings, exploiting a result by J. Petersen in *Acta Mathematica* 1891:

$u \in \{0, 1\}$

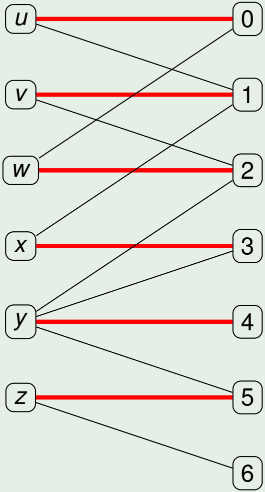
$v \in \{1, 2\}$

$w \in \{0, 2\}$

$x \in \{1, 3\}$

$y \in \{2, 3, 4, 5\}$

$z \in \{5, 6\}$



Example (Propagator for `alldifferent`)

Mark all other edges in *all* other maximum matchings, exploiting a result by J. Petersen in *Acta Mathematica* 1891:

$u \in \{0, 1\}$

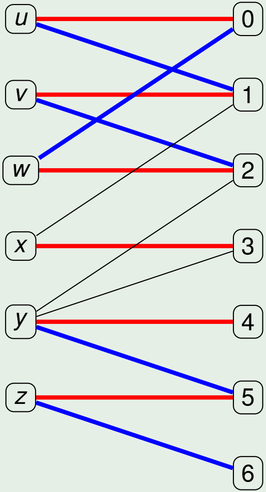
$v \in \{1, 2\}$

$w \in \{0, 2\}$

$x \in \{1, 3\}$

$y \in \{2, 3, 4, 5\}$

$z \in \{5, 6\}$



Example (Propagator for `alldifferent`)

Every still unmarked edge is in *no* maximum matching.
Propagate accordingly within the current domains:

$u \in \{0, 1\}$

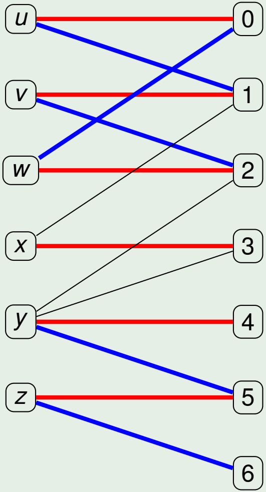
$v \in \{1, 2\}$

$w \in \{0, 2\}$

$x \in \{1, 3\}$

$y \in \{2, 3, 4, 5\}$

$z \in \{5, 6\}$



Example (Propagator for `alldifferent`)

Every still unmarked edge is in *no* maximum matching.
Propagate accordingly within the current domains:

$u \in \{0, 1\}$

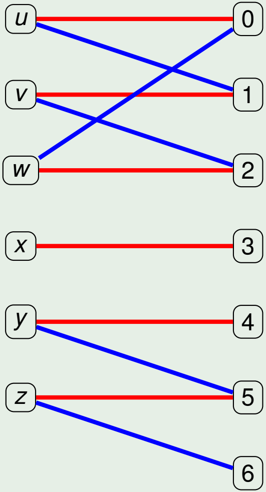
$v \in \{1, 2\}$

$w \in \{0, 2\}$

$x \in \{\cancel{1}, 3\}$

$y \in \{\cancel{2}, \cancel{3}, 4, 5\}$

$z \in \{5, 6\}$



Search + Propagation of All Constraints

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	?						
spelt							
wheat							

Search + Propagation of All Constraints

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	?						
spelt							
wheat							

But plot1 cannot grow rye as that would violate the second constraint (every plot grows 3 grains).

Search + Propagation of All Constraints

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–						
spelt							
wheat							

But plot1 cannot grow rye as that would violate the second constraint (every plot grows 3 grains).

Search + Propagation of All Constraints

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–						
spelt							
wheat							

But plot1 cannot grow rye as that would violate the second constraint (every plot grows 3 grains). Actually, plot1 cannot grow oats, spelt, or wheat either, for the same reason, and this was already propagated when trying the search guess that plot1 grow millet!

Search + Propagation of All Constraints

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–						
spelt	–						
wheat	–						

But plot1 cannot grow rye as that would violate the second constraint (every plot grows 3 grains). Actually, plot1 cannot grow oats, spelt, or wheat either, for the same reason, and this was already propagated when trying the search guess that plot1 grow millet!

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	?					
spelt	–						
wheat	–						

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	?					
spelt	–						
wheat	–						

Guess: Let plot2 grow rye. (Strategy: ✓ guesses first.)

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓					
spelt	–						
wheat	–						

Guess: Let plot2 grow rye. (Strategy: ✓ guesses first.)

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓					
spelt	–						
wheat	–						

Propagation: plot2 cannot grow spelt and wheat as otherwise the second constraint (every plot grows 3 grains) would be violated for plot2.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓					
spelt	–	–					
wheat	–	–					

Propagation: plot2 cannot grow spelt and wheat as otherwise the second constraint (every plot grows 3 grains) would be violated for plot2.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓					
spelt	–	–					
wheat	–	–					

Propagation: plot3, plot4, and plot6 cannot grow rye as otherwise the third constraint (every grain pair is grown in 1 common plot) would be violated.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–		–	
spelt	–	–					
wheat	–	–					

Propagation: plot3, plot4, and plot6 cannot grow rye as otherwise the third constraint (every grain pair is grown in 1 common plot) would be violated.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–		–	
spelt	–	–					
wheat	–	–					

Propagation: plot5 and plot7 must grow rye as otherwise the first constraint (every grain is grown in 3 plots) would be violated for rye.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–					
wheat	–	–					

Propagation: plot5 and plot7 must grow rye as otherwise the first constraint (every grain is grown in 3 plots) would be violated for rye.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–					
wheat	–	–					

Propagation: plot3 must grow spelt and wheat as otherwise the second constraint (every plot grows 3 grains) would be violated for plot3.

UPPSALA UNIVERSITET

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓				
wheat	–	–	✓				

Propagation: plot3 must grow spelt and wheat as otherwise the second constraint (every plot grows 3 grains) would be violated for plot3.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓				
wheat	–	–	✓				

Common fixpoint reached: No more propagation possible.

Example (BIBD: AED partial assignment)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓			
wheat	–	–	✓				

Guess: Let plot4 grow spelt. (Strategy: ✓ guesses first.)

Propagation: etc.

Systematic search, for a satisfaction problem:

1: propagate all constraints; **backtrack** if empty domain
2: **if** only fixed variables, **then** show solution & **backtrack**
3: **while** there is at least one suspended propagator **do**
4: select unfixed variable, v, of current domain $\mathrm{dom}(v)$
5: partition $\mathrm{dom}(v)$ using guesses (say $v = d$ & $v \neq d$, or $v > d$ & $v \leq d$, for a selected value $d \in \mathrm{dom}(v)$)
6: **for each** guess: **recurse** upon adding it as constraint

For an optimisation problem: before backtracking at line 2 add the constraint that any next solution must be better.

Strategies:

- Line 4: variable selection strategy: smallest domain, ...
- Line 5: value selection strategy: maximum, median, ...
- Line 5: guess selection strategy: equality, bisection, ...
- Tree exploration: depth-first search, ...

Example (Search for the BIBD *integer* model)

```
solve :: int_search(BIBD,input_order,indomain_max) satisfy;
```

Outline

Example (BIBD: AED assignment after *i* moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	✓	–	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal sample size: Every grain is grown in 3 plots. Satisfied at initialisation and by each move: invariant.
2. Equal growth load: Every plot grows 3 grains. Currently satisfied: zero violation.
3. Balance: Every grain pair is grown in 1 common plot. But, e.g., oats & rye are grown in $2 > 1$ common plots.

Example (BIBD: AED assignment after *i* moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	✓	–	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal sample size: Every grain is grown in 3 plots. Satisfied at initialisation and by each move: invariant.
2. Equal growth load: Every plot grows 3 grains. Currently satisfied: zero violation.
3. Balance: Every grain pair is grown in 1 common plot. But, e.g., oats & rye are grown in $2 > 1$ common plots.

Selected move: let plot6 instead of plot5 grow oats.

Example (BIBD: AED assignment after *i* moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal sample size: Every grain is grown in 3 plots. Satisfied at initialisation and by each move: invariant.
2. Equal growth load: Every plot grows 3 grains. Currently satisfied: zero violation.
3. Balance: Every grain pair is grown in 1 common plot. But, e.g., oats & rye are grown in $2 > 1$ common plots.

Selected move: let plot6 instead of plot5 grow oats.

Example (BIBD: AED assignment after $i + 1$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal sample size: Every grain is grown in 3 plots. Satisfied at initialisation and by each move: invariant.
2. Equal growth load: Every plot grows 3 grains. But plot5 grows $2 < 3$ grains; plot6 grows $4 > 3$ grains.
3. Balance: Every grain pair is grown in 1 common plot. But, e.g., corn & oats are grown in $2 > 1$ common plots.

Example (BIBD: AED assignment after $i+1$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	–	✓	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal sample size: Every grain is grown in 3 plots. Satisfied at initialisation and by each move: invariant.
2. Equal growth load: Every plot grows 3 grains. But plot5 grows $2 < 3$ grains; plot6 grows $4 > 3$ grains.
3. Balance: Every grain pair is grown in 1 common plot. But, e.g., corn & oats are grown in $2 > 1$ common plots.

Selected move: let plot5 instead of plot6 grow corn.

UPPSALA UNIVERSITET

Example (BIBD: AED assignment after $i + 1$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal sample size: Every grain is grown in 3 plots. Satisfied at initialisation and by each move: invariant.
2. Equal growth load: Every plot grows 3 grains. But plot5 grows $2 < 3$ grains; plot6 grows $4 > 3$ grains.
3. Balance: Every grain pair is grown in 1 common plot. But, e.g., corn & oats are grown in $2 > 1$ common plots.

Selected move: let plot5 instead of plot6 grow corn.

Example (BIBD: AED assignment after $i + 2$ moves)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

1. Equal sample size: Every grain is grown in 3 plots. Satisfied at initialisation and by each move: invariant.
2. Equal growth load: Every plot grows 3 grains. Currently satisfied: zero violation.
3. Balance: Every grain pair is grown in 1 common plot. Currently satisfied: zero violation.

Stop search: All constraints are satisfied.

Local search:

1: let s and s^* be the same computed initial assignment
2: **while** there are violated constraints & iterations left **do**
3:     select a move on s; let s' be the reached assignment
4:     **if** s' is better than s^* **then** $s^* := s'$
5:     $s := s'$
6: **return** s^*

Heuristics: What move to select?

- Line 3: assign, flip, swap, add, drop, transfer, ...
- Line 3: best / first / random improvement, ...

Meta-heuristics: How to escape local optima?

- Lines 2 to 5: simulated annealing, tabu search, ...

Heuristics are an Art!

There are good & bad heuristics for each problem model:

- Different heuristics for a model may take different time on the same solver for the same instance.
- Different heuristics for a model may scale differently on the same solver for instances of growing size.

A tiny heuristic tweak may accelerate the solving manyfold!

Good heuristicians are worth their weight in gold!

Use solvers, based on decades of cutting-edge research: you reuse hundreds of thousands of lines of highly tuned code that is very hard to beat.

Outline

Stand-alone languages:

- ALICE by Jean-Louis Laurière, France, 1976
- CHIP at ECRC, Germany, 1987 – 1990,
  then marketed by Cosytec, France
- OPL, by P. Van Hentenryck, USA, and ILOG, France: front-end to both ILOG CP Optimizer and ILOG CPLEX
- Comet, by P. Van Hentenryck and L. Michel, USA
- MiniZinc, at U. of Melbourne and Monash U., Australia
- ...

Libraries (the ones listed before ";" are open-source):

- Prolog: ECLiPSe, . . . ; SICStus Prolog, . . .
- C++: Gecode, OR-Tools; IBM CP Optimizer, CHIP, . . .
- Java: Choco, Google OR-Tools, JaCoP, . . . ; . . .
- Scala: OscaR; . . .
- ...

Outline

Success Stories by CP Users and Contributors:

AIMMS

AMPL

BOUYGUES

cadence

CISCO

FICO

Google

IBM

intel

JEPPESEN
A BOEING COMPANY

NASA

ORACLE

QUINTIQ

RedPrairie

RENAULT

SAP

SIEMENS

TACTON

THALES

XEROX

. . .

Success stories: CP = **technology of choice** in scheduling, configuration, personnel rostering, timetabling, . . .

Outline

Scope of Constraint Programming

CP has a wide scope, as it addresses:

- satisfaction problems and optimisation problems
- discrete variables and continuous variables
- linear constraints and non-linear constraints

in principle in any combinations thereof, by:

- systematic search, if optimality more crucial than speed
- local search, if speed is more crucial than optimality

Common Misconceptions about CP

- CP is for experts: the predicate vocabulary is *large*.

  But there are tools helping you identify predicates, and about a dozen predicates take you a very long way!

- CP is for experts: the search strategy is *mandatory*.

  No, it is optional: there is adaptive autonomous search!

- CP'ers claim CP is a silver bullet for NP-hard problems.

  No: CP solvers are complementary in strength to MIP, SAT, SMT, ... solvers and to local search, which leads to hybrid optimisation technologies: LCG, ... !

Bluffer's Guide to the C$\ast$ Alphabet Soup

- CP solvers $=$ constraint solvers

  MIP, SAT, ... solvers are also constraint solvers, whether known as such in those communities or not!

- CP $=$ CLP $=$ constraint *logic* programming

  Many modern CP solvers are not Prolog libraries!

  $$\text{CLP}(\mathbb{Q}, \mathbb{R}) \text{ solvers} \subsetneq \text{CLP solvers} \subsetneq \text{CP solvers}$$

  CP is a granddaughter of logic programming (LP)!

- CP $=$ CSP $=$ constraint satisfaction *programming*

  CSP $=$ constraint satisfaction *problem*, an AI term.

  MIP, SAT, ... solvers also solve constraint problems, whether those communities use that term or not!

Bluffer's Guide to the C$\ast$ Alphabet Soup

- CP solvers $\subsetneq$ constraint solvers

  MIP, SAT, ... solvers are also constraint solvers, whether known as such in those communities or not!

- CP $=$ CLP $=$ constraint *logic* programming

  Many modern CP solvers are not Prolog libraries!

  $$\text{CLP}(\mathbb{Q},\mathbb{R}) \text{ solvers} \subsetneq \text{CLP solvers} \subsetneq \text{CP solvers}$$

  CP is a granddaughter of logic programming (LP)!

- CP $=$ CSP $=$ constraint satisfaction *programming*

  CSP $=$ constraint satisfaction *problem*, an AI term.

  MIP, SAT, ... solvers also solve constraint problems, whether those communities use that term or not!

Bluffer's Guide to the C$*$ Alphabet Soup

- CP solvers $\subsetneq$ constraint solvers

  MIP, SAT, ... solvers are also constraint solvers, whether known as such in those communities or not!

- CP $=$ CLP $=$ constraint *logic* programming

  Many modern CP solvers are not Prolog libraries!

  $$\text{CLP}(\mathbb{Q}, \mathbb{R}) \text{ solvers} \subsetneq \text{CLP solvers} \subsetneq \text{CP solvers}$$

  CP is a granddaughter of logic programming (LP)!

- CP $=$ CSP $=$ constraint satisfaction *programming*

  CSP $=$ constraint satisfaction *problem*, an AI term.

  MIP, SAT, ... solvers also solve constraint problems, whether those communities use that term or not!

Bluffer's Guide to the C$*$ Alphabet Soup

- CP solvers $\subsetneq$ constraint solvers

  MIP, SAT, … solvers are also constraint solvers, whether known as such in those communities or not!

- CP $\neq$ CLP $=$ constraint *logic* programming

  Many modern CP solvers are not Prolog libraries!

  $$\text{CLP}(\mathbb{Q},\mathbb{R}) \text{ solvers} \subsetneq \text{CLP solvers} \subsetneq \text{CP solvers}$$

  CP is a granddaughter of logic programming (LP)!

- CP $=$ CSP $=$ constraint satisfaction *programming*

  CSP $=$ constraint satisfaction *problem*, an AI term.

  MIP, SAT, … solvers also solve constraint problems, whether those communities use that term or not!

Bluffer's Guide to the C$*$ Alphabet Soup

- CP solvers $\subsetneq$ constraint solvers

  MIP, SAT, ... solvers are also constraint solvers, whether known as such in those communities or not!

- CP $\neq$ CLP $=$ constraint *logic* programming

  Many modern CP solvers are not Prolog libraries!

  $$\text{CLP}(\mathbb{Q}, \mathbb{R}) \text{ solvers} \subsetneq \text{CLP solvers} \subsetneq \text{CP solvers}$$

  CP is a granddaughter of logic programming (LP)!

- CP $=$ CSP $=$ constraint satisfaction *programming*

  CSP $=$ constraint satisfaction *problem*, an AI term.

  MIP, SAT, ... solvers also solve constraint problems, whether those communities use that term or not!

Bluffer's Guide to the C$*$ Alphabet Soup

- CP solvers $\subsetneq$ constraint solvers

  MIP, SAT, … solvers are also constraint solvers, whether known as such in those communities or not!

- CP $\neq$ CLP $=$ constraint *logic* programming

  Many modern CP solvers are not Prolog libraries!

  $$\text{CLP}(\mathbb{Q},\mathbb{R}) \text{ solvers} \subsetneq \text{CLP solvers} \subsetneq \text{CP solvers}$$

  CP is a granddaughter of logic programming (LP)!

- CP $=$ CSP $=$ constraint satisfaction *programming*

  CSP $=$ constraint satisfaction *problem*, an AI term.

  MIP, SAT, … solvers also solve constraint problems, whether those communities use that term or not!

Bluffer's Guide to the C$\ast$ Alphabet Soup

- CP solvers $\subsetneq$ constraint solvers

  MIP, SAT, ... solvers are also constraint solvers, whether known as such in those communities or not!

- CP $\neq$ CLP $=$ constraint *logic* programming

  Many modern CP solvers are not Prolog libraries!

  CLP$(\mathbb{Q}, \mathbb{R})$ solvers $\subsetneq$ CLP solvers $\subsetneq$ CP solvers

  CP is a granddaughter of logic programming (LP)!

- CP $\neq$ CSP $\neq$ constraint satisfaction *programming*

  CSP $=$ constraint satisfaction *problem*, an AI term.

  MIP, SAT, ... solvers also solve constraint problems, whether those communities use that term or not!

Opportunities for CP

Rapid prototyping, with high solving performance, when:

- Constraints are, still or again, subject to experiments
- Partition into hard & soft constraints yet undetermined

Combinatorial structure is impure, due to side constraints.

It is time to consider all or more problem constraints.

Domain knowledge exploitable for problem-specific search.

It is a configuration problem.

It is a personnel rostering problem.

It is a scheduling problem.

It is a time-tabling problem.

Outline

C. Allignol, N. Barnier, P. Flener, and J. Pearson.
Section 2 of Constraint programming [. . .].
Knowledge Engineering Review, 27(3):361–392, 2012.

K. R. Apt.
Principles of Constraint Programming.
Cambridge University Press, 2003.

F. Rossi, P. van Beek, and T. Walsh (editors).
Handbook of Constraint Programming.
Elsevier, 2006.

E. C. Freuder and M. Wallace.
Constraint technology and the commercial world.
IEEE Intelligent Systems, 15(1):20–23, 2000.

M. Milano, P. Van Hentenryck, et al.
The future of constraint programming.
Constraints, special issue, 19(2), 2014.

M. Wallace.
Constraint programming – The paradigm to watch.
Constraint Programming Letters, 1:7–13, 2007.

Ph. Baptiste, C. Le Pape, and W. Nuijten.
Constraint-Based Scheduling.
Kluwer Academic Publishers, 2001.

P. Van Hentenryck and L. Michel.
Constraint-Based Local Search.
The MIT Press, 2005.