Constraint Programming

In Pursuit of The Holly Grail

Roman Barták
Charles University in Prague

`bartak@ktiml.mff.cuni.cz`
`http://ktiml.mff.cuni.cz/~bartak`

"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."

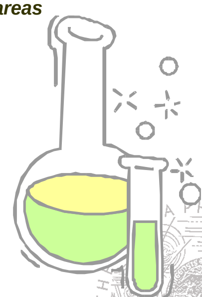
Eugene C. Freuder, Constraints, April 1997

Talk outline

What is constraint programming?
- *a bit of history and application areas*

How to solve constraints?
- *filtering algorithms*
- *constraint propagation*
- *labelling*
- *over-constrained problems*

Conclusions
- *benefits*
- *systems and resources*



What is a constraint?

Constraint is an arbitrary relation over the set of variables.
- every variable has a set of possible values - a domain
  - this talk covers discrete finite domains only
- the constraint restricts the possible combinations of values

Some examples:
- the circle C is inside a square S
- the length of the word W is 10 characters
- X is less than Y
- a sum of angles in the triangle is 180°
- the temperature in the warehouse must be in the range 0-5°C
- John can attend the lecture on Wednesday after 14:00

Constraint can be described:
- intentionally (as a mathematical/logical formula)
- extensionally (as a table describing compatible tuples)

What is constraint programming?

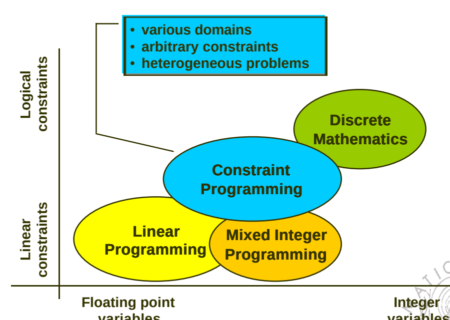
A technology for solving (combinatorial) problems described as:
- *a set of variables*
  `S,E,N,D,M,O,R,Y`
- *domains for variables* (sets of allowed values)
  `E,N,D,O,R,Y::0..9, S,M::1..9`
- *constraints* (relations restricting combinations of values)

```
              1000*S + 100*E + 10*N + D
+             1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y
(SEND+MORE=MONEY)
```

The task is to find a value for each variable satisfying all the constraints.

☞ **User states the problem, computer solves it!** ☜

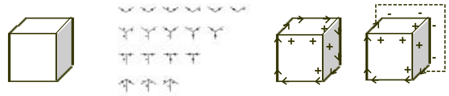
CP and others



A bit of history

Scene labelling (Waltz 1975)

feasible interpretation of 3D lines in a 2D drawing



Interactive graphics (Sutherland 1963, Borning 1981)

geometrical objects described using constraints

Logic programming (Gallaire 1985, Jaffar, Lassez 1987)

from unification to constraint solving

7

Application areas

All types of hard combinatorial problems

Molecular biology

- DNA sequencing
- determining protein structures

Interactive graphic

- web layout

Network management and configuration

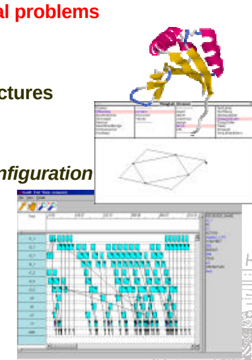
Assignment problems

- personal assignment
- stand allocation

Timetabling

Scheduling

Planning



8

Inside Constraint Programming

A Technology Overview

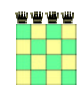
Constraint programming at glance

Modelling (problem formulation)

N-queens problem

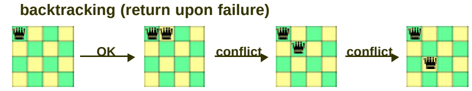
queen = column (looking for row) r(i):: 1..N

non-attack constraints $\forall i \neq j\ \ r(i) \neq r(j)\ \&\ |i-j| \neq |r(i)-r(j)|$



Labelling (search)

backtracking (return upon failure)



Propagation (domain filtering)

remove inconsistencies in advance

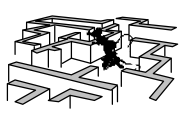
10

Solving constraints by enumeration

Constraints are used only as a test

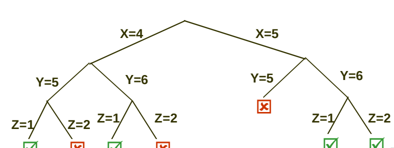
assign values to variables ...

... and see what happens



Example:

X in [4,5]
Y in [5,6]
Z in [1,2]
X<Y
Z<X-2



Some improvements:

- backjumping (jump to a conflicting variable)
- backchecking, backmarking (remember the conflicts)

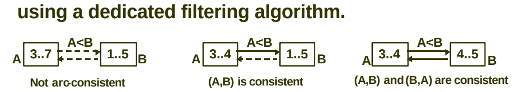
11

Arc consistency

Example:

A in [3,..,7], B in [1,..,5], A<B

Constraint can be used to prune the domains actively using a dedicated filtering algorithm.



Definitions:

The constraint C is arc consistent iff for every variable *i* constrained by C and for every value $v \in D_i$ there is an assignment of the remaining variables in C such that the constraint is satisfied.

The problem is arc consistent if every constraint is arc consistent

12

Constraint propagation

How to establish arc consistency among the constraints?

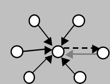
Example: X in [1,..,6], Y in [1,..,6], Z in [1,..,6], X<Y, Z<X-2

X in [1,..,6] Y in [1,..,6] Z in [1,..,6]	$\xrightarrow{X<Y}$	X in [1,..,5] Y in [2,..,6] Z in [1,..,6]	$\xrightarrow{Z<X-2}$	X in [4,5] Y in [2,..,6] Z in [1,2]	$\xrightarrow{X<Y}$	X in [4,5] Y in [5,6] Z in [1,2]

Make all the constraints consistent until any domain is changed (AC-1)

Why we should revise the constraint X<Y if domain of Z is changed?

```
procedure AC-3(C)
    Q ← C                    % a list of constraints for revision
    while Q non empty and no domain is empty do
        select and delete c from Q
        Q ← Q ∪ REVISE(c,C)
    end while
end AC-3
```



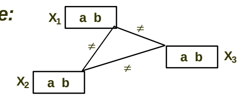
13

Is arc consistency enough?

By using AC we can remove many incompatible values
- Do we get a solution?
- Do we know that there exists a solution?

Unfortunately, the answer to both above questions is NO!

Example:



CSP is arc consistent but there is no solution

So what is the benefit of AC?

Sometimes we have a solution after AC
- any domain is empty ⇒ no solution exists
- all the domains are singleton ⇒ we have a solution

In general, AC prunes the search space.

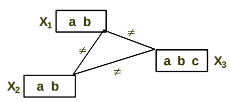
14

Global constraints

a set of binary inequality constraints among all variables

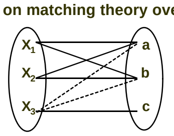
$X_1 \neq X_2, X_1 \neq X_3, \ldots, X_{k-1} \neq X_k$

weak pruning using local consistency algorithms (AC)



$\text{all\_different}(X_1,\ldots, X_k) = \{(d_1,\ldots, d_k) \mid \forall i\ d_i \in D_i\ \&\ \forall i \neq j\ d_i \neq d_j\}$

better pruning based on matching theory over bipartite graphs



15

Combining search and consistency

Backtracking uses constraints passively only!
- wasting information (visibly wrong instantiations are explored)

Domain filtering is (usually) not complete!

We can combine backtracking search with domain filtering
- process constraints after variable instantiation

Look Back methods
- constraints among already instantiated variables are checked to analyse the conflicts
- backjumping, backchecking, backmarking

Look Ahead methods
- domain filtering among not yet instantiated variables to prevent future conflicts
- forward checking, partial look ahead, (full) look ahead

16

Look Ahead methods

Domain filtering among not yet instantiated variables
- removes incompatible values from domains
- detects conflicts earlier (when any domain become empty)

How strong consistency should be achieved and what variables should be involved?

Forward checking (FC)

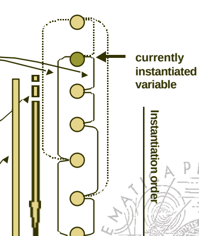
AC between the current variable and neighbouring variables

Partial look ahead (PLA)

AC between future variables in one direction only (DAC)

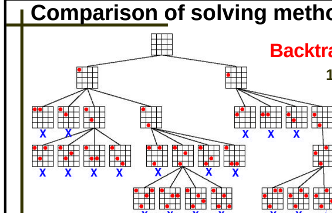
(Full) look ahead (LA)

AC between all future variables



17

Comparison of solving methods (4 queens)

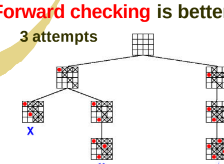


Backtracking is not very good

19 attempts

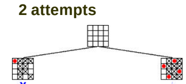
Forward checking is better

3 attempts



And the winner is Look Ahead

2 attempts



18

Search strategies

Can we further influence efficiency of the solver?

Variable ordering

defines the structure of the search tree

FIRST FAIL principle

prefer variable whose instantiation will lead to failure with the highest probability (solve the hardest case first)

- **prefer the variables with the smallest domain**
- **prefer the most constrained variables**

Value ordering

defines the search order (how the explore the search tree)

SUCCEED FIRST principle

- **prefer the values with higher number of supporters**
- **usually problem dependent**

19

Tree search and heuristics

Observation 1:

The search space for real-life problems is so huge that it cannot be fully explored.

Heuristics **- a guide of search**

- **they recommend a value for assignment**
- **quite often leads to solution**

What to do upon a failure of the heuristics?

BT cares about the end of search (a bottom part of the search tree)

- **so it rather repairs later assignments than the earliest ones**
- **it assumes that the heuristic guides it well in the top part**

Observation 2:

The heuristics are less reliable in the earlier parts of the search (as search proceeds, more information for better decision is available).

Observation 3:

The number of heuristic violations is usually small.

20

Limited discrepancy search

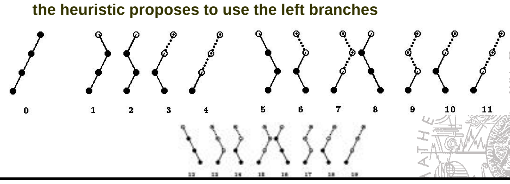
Discrepancy **= heuristic is not followed (a value different from the heuristic is chosen)**

Idea of *Limited Discrepancy Search* (LDS):

- **first, follow the heuristic**
- **when a failure occurs then explore the paths when the heuristic is not followed maximally once (start with earlier violations)**
- **after next failure occurs then explore the paths when the heuristic is not followed maximally twice...**

Example:

the heuristic proposes to use the left branches



21

A motivation - robot dressing problem

Dress a robot using minimal wardrobe and fashion rules.

Variables and domains:

shirt: {red, white}

footwear: {cordovans, sneakers}

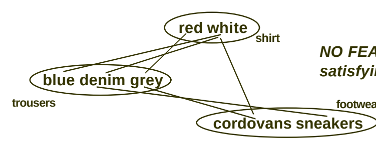
trousers: {blue, denim, grey}

Constraints:

shirt x trousers: **red-grey, white-blue, white-denim**

footwear x trousers: **sneakers-denim, cordovans-grey**

shirt x footwear: **white-cordovans**



NO FEASIBLE SOLUTION satisfying all the constraints

We call the problems where no feasible solution exists over-constrained problems.

22

First solution to the robot dressing problem

There is no feasible valuation but we need to dress robot!

1) buy new wardrobe

enlarge the domain of some variable — *Domain is defined by a unary constraint*

2) less elegant wardrobe

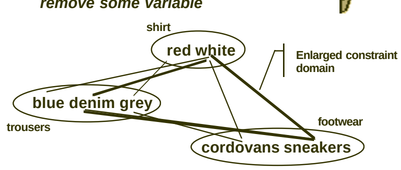
enlarge the domain of some constraint — *All combinations are assumed feasible*

3) no matching of shoes and shirt

remove some constraint

4) do not wear shoes

remove some variable — *Delete the constraint bounding the variable*



23

Second solution of the robot dressing problem

It is possible to assign a preference to each constraint to describe priorities of satisfaction of the constraints.

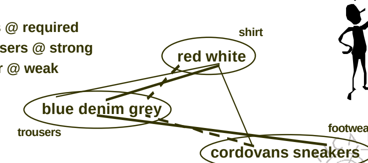
The preference describes a strict priority.

a stronger constraint is preferred to arbitrary number of weaker constraints

shirt x trousers @ required

footwear x trousers @ strong

shirt x footwear @ weak



Constraints marked by a preference make a hierarchy, thus we are speaking about constraint hierarchies.

24

Conclusions

The benefits of constraint programming

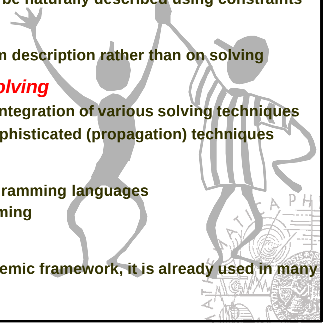
Close to real-life (combinatorial) problems
- everyone uses constraints to specify problem properties
- real-life restriction can be naturally described using constraints

A declarative character
- concentrate on problem description rather than on solving

Co-operative problem solving
- unified framework for integration of various solving techniques
- simple (search) and sophisticated (propagation) techniques

Semantically pure
- clean and elegant programming languages
- roots in logic programming

Applications
- CP is not another academic framework, it is already used in many applications



26

Systems

Prolog
CHIP, ECLiPSe, SICStus Prolog, Prolog IV, GNU Prolog, IF/Prolog

C/C++
CHIP++, ILOG Solver

Java
JCK, JCL, Koalog

LISP
Screamer

others
Python Constraints, Mozart

More at
http://kti.mff.cuni.cz/~bartak/constraints/systems.html

27

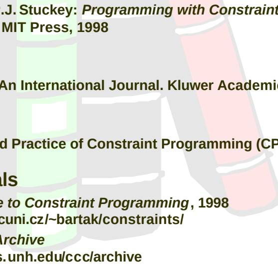
Resources

Books
- P. Van Hentenryck: *Constraint Satisfaction in Logic Programming*, MIT Press, 1989
- E. Tsang: *Foundations of Constraint Satisfaction*, Academic Press, 1993
- K. Marriott, P.J. Stuckey: *Programming with Constraints: An Introduction*, MIT Press, 1998

Journal
- Constraints, An International Journal. Kluwer Academic Publishers

Conference
- Principles and Practice of Constraint Programming (CP)

On-line materials
- *On-line Guide to Constraint Programming*, 1998
  http://kti.mff.cuni.cz/~bartak/constraints/
- *Constraints Archive*
  http://www.cs.unh.edu/ccc/archive



28

"Were you to ask me which programming paradigm is likely to gain most in commercial significance over the next 5 years I'd have to pick Constraint Logic Programming, even though it's perhaps currently one of the least known and understood."

Dick Pountain, BYTE, February 1995

Constraint Programming

In Pursuit of The Holly Grail

THE END!

Roman Barták
Charles University in Prague

bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak