Global Constraints in Constraint Programming

Willem-Jan van Hoeve

Tepper School of Business, Carnegie Mellon University

Pittsburgh, PA

Optimization Days 2010

Outline

- Constraint Programming
  - Central concepts, motivation, applications
  - Domain filtering algorithms
- Global Constraints
  - Classical (*alldifferent*)
  - Over-Constrained Problems (*soft-alldifferent*)
  - Sequencing and Scheduling (*sequence*)
- Recent Developments
  - Open constraints (*open-alldifferent*)
  - Constraint-based (local) search

(With apologies for the bias towards my own work...)

Constraint Programming

Comparison with Integer Programming

Integer Linear Programming (branch-and-bound/branch-and-cut)	Constraint Programming
• systematic search	• systematic search
• at each search state, solve continuous relaxation of problem (expensive)	• at each search state, reason on individual constraints (cheap)
• add cuts to reduce search space	• filter variable domains to reduce search space
• domains are intervals	• domains may contain holes
very suitable for optimization problems	very suitable for highly combinatorial problems, e.g., scheduling, timetabling

Constraint Programming History

1970s: Artificial Intelligence

- image processing applications
- search + qualitative inference

1980s: Logic Programming

- logic programming languages (e.g., Prolog)
- search + logical inference

1989: CHIP system (Constraint Handling In Prolog)

- constraint logic programming

1990s: Constraint Programming

- combines artificial intelligence, logic programming, and operations research
- industrial solvers (e.g., ILOG, Eclipse, Xpress-Kalis) and industrial applications

1994: filtering for *alldifferent* and resource scheduling (edge finding)

2000s: Various developments

- efficient algorithms for special constraints
- integrated methods (with OR techniques)
- modeling languages (e.g., OPL, Comet, Zinc)

Successful applications

Sport Scheduling

An 8 Team Round Robin Timetable

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

Schedule of 1997/1998 ACC basketball league (9 teams)

- various complicated side constraints
- all 179 solutions were found in 24h using enumeration and integer linear programming [Nemhauser & Trick, 1998]
- all 179 solutions were found in less than a minute using constraint programming [Henz, 1999, 2001]

Hong Kong Airport

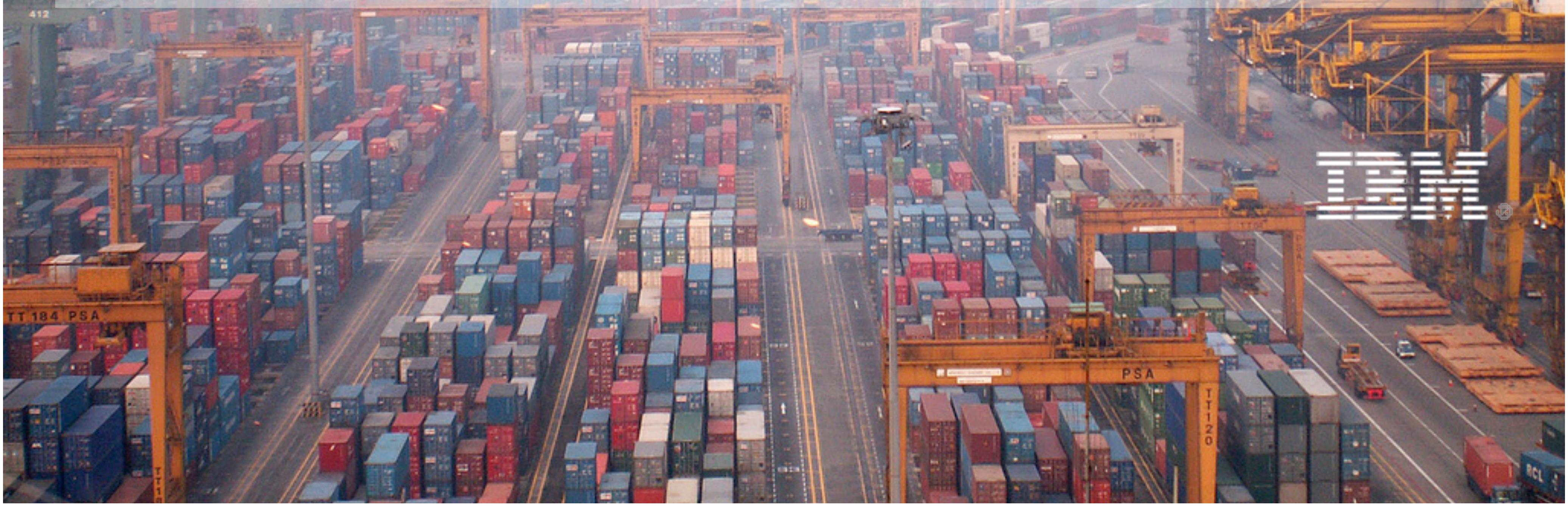
- Gate allocation at the new (1998) Hong Kong airport
- System was implemented in only four months, including constraint programming technology (ILOG)
- Schedules ~800 flights a day

  (47 million passengers in 2007)

G. Freuder and M. Wallace. Constraint Technology and the Commercial World. *IEEE Intelligent Systems* 15(1): 20-23, 2000.

Port of Singapore

- One of the world's largest container transshipment hubs
- Links shippers to a network of 200 shipping lines with connections to 600 ports in 123 countries
- Problem: Assign yard locations and loading plans under various operational and safety requirements
- Solution: Yard planning system, based on constraint programming



Railroad Optimization

Carnegie Mellon Tepper School of Business

- Netherlands Railways has among the densest rail networks in the world, with 5,500 trains per day
- Constraint programming is one of the components in their railway planning software, which was used to design a new timetable from scratch (2009)
- Much more robust and effective schedule, and $75M additional annual profit
- INFORMS Edelman Award winner (2009)

CP Modeling

A Constraint Satisfaction Problem, or CSP, consists of

- a set of variables X,
- variable domains D(x) (for all $x \in X$),
- and a set of constraints on subsets of the variables

A solution to a CSP is:

assign to each variable a single element from its domain

such that all constraints are satisfied

Example:

variables x_1, x_2, x_3

domains $D(x_1) = \{1,2\}$, $D(x_2) = \{0,1,2,3\}$, $D(x_3) = \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)

solution: $x_1 = 2$, $x_2 = 1$, $x_3 = 3$

A Constraint *Optimization* Problem, or COP, consists of

- a set of variables X,
- variable domains D(x) (for all $x \in X$),
- a set of constraints on subsets of the variables,
- and an objective function $f(X) \rightarrow \mathbb{R}$ to be optimized

A solution to a COP is:

assign to each variable a single element from its domain

such that all constraints are satisfied, and the objective function is a global optimum

Example:

variables/domains	$x_1 \in \{1,2\}$, $x_2 \in \{0,1,2,3\}$, $x_3 \in \{2,3\}$
constraints	$x_1 > x_2$
	$x_1 + x_2 = x_3$
	alldifferent(x_1,x_2,x_3)
objective function	maximize $x_2 + x_3$

solution: $x_1 = 2$, $x_2 = 1$, $x_3 = 3$

More modeling examples

- variables range over finite or continuous domain:
  $v \in \{a,b,c,d\}$, $start \in \{0,1,2,3,4,5\}$, $z \in [2.18, 4.33]$, $S \in [\ \{b,c\}, \{a,b,c,d,e\}\ ]$

- algebraic expressions:
  $x^3(y^2 - z) \geq 25 + x^2 \cdot \max(x,y,z)$

- variables as subscripts:
  $y = cost[x]$ (here y and x are variables, 'cost' is an array of parameters)

- logical relations in which constraints can be mixed:
  $((x < y) \text{ OR } (y < z)) \Rightarrow (c = \min(x,y))$

- 'global' constraints (a.k.a. symbolic constraints):
  alldifferent$(x_1,x_2, \ldots,x_n)$
  UnaryResource$(\ [start_1,\ldots, start_n], [duration_1,\ldots,duration_n]\ )$

Solving

Example:

variables/domains $x_1 \in \{1,2\}$, $x_2 \in \{0,1,2,3\}$, $x_3 \in \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)

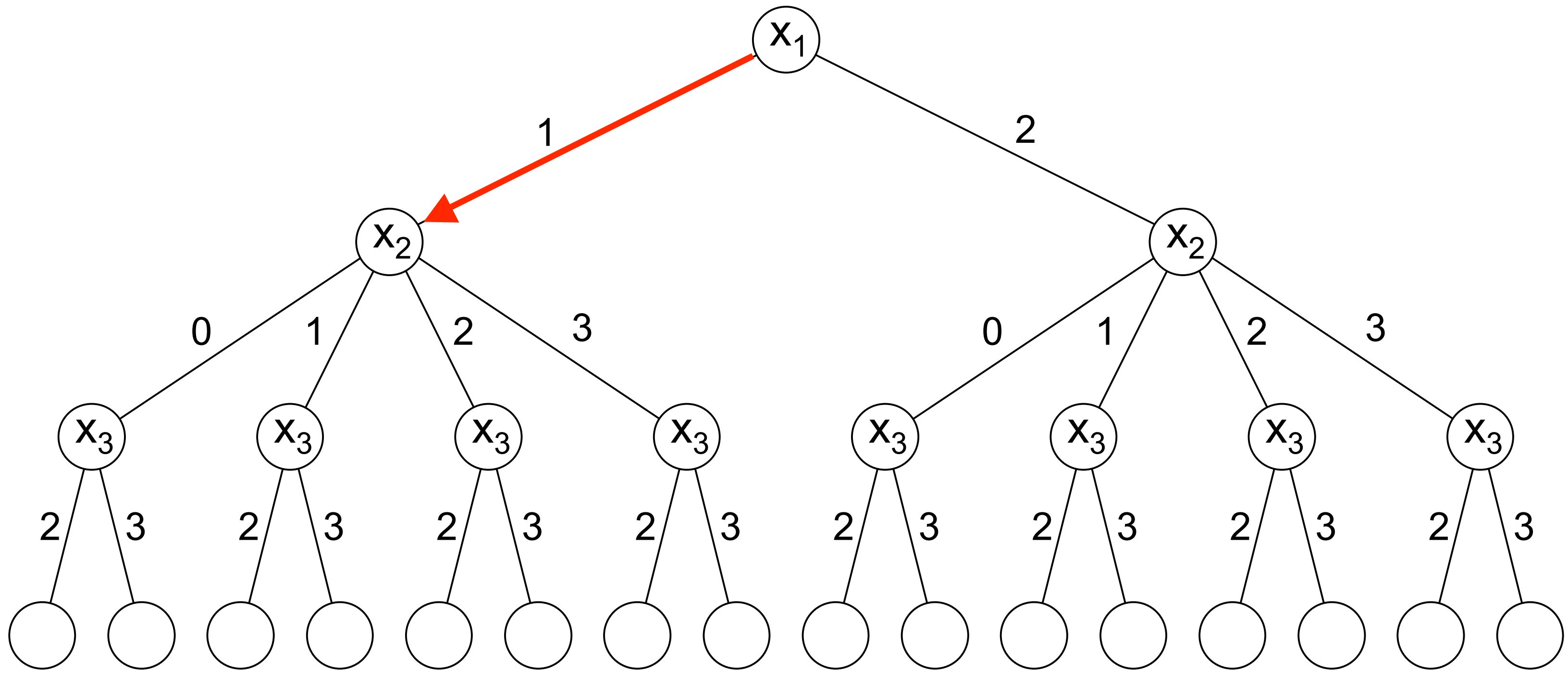
Example:

variables/domains $x_1 \in \{1,2\}$, $x_2 \in \{0,1,\not{2},\not{3}\}$, $x_3 \in \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)



Solving

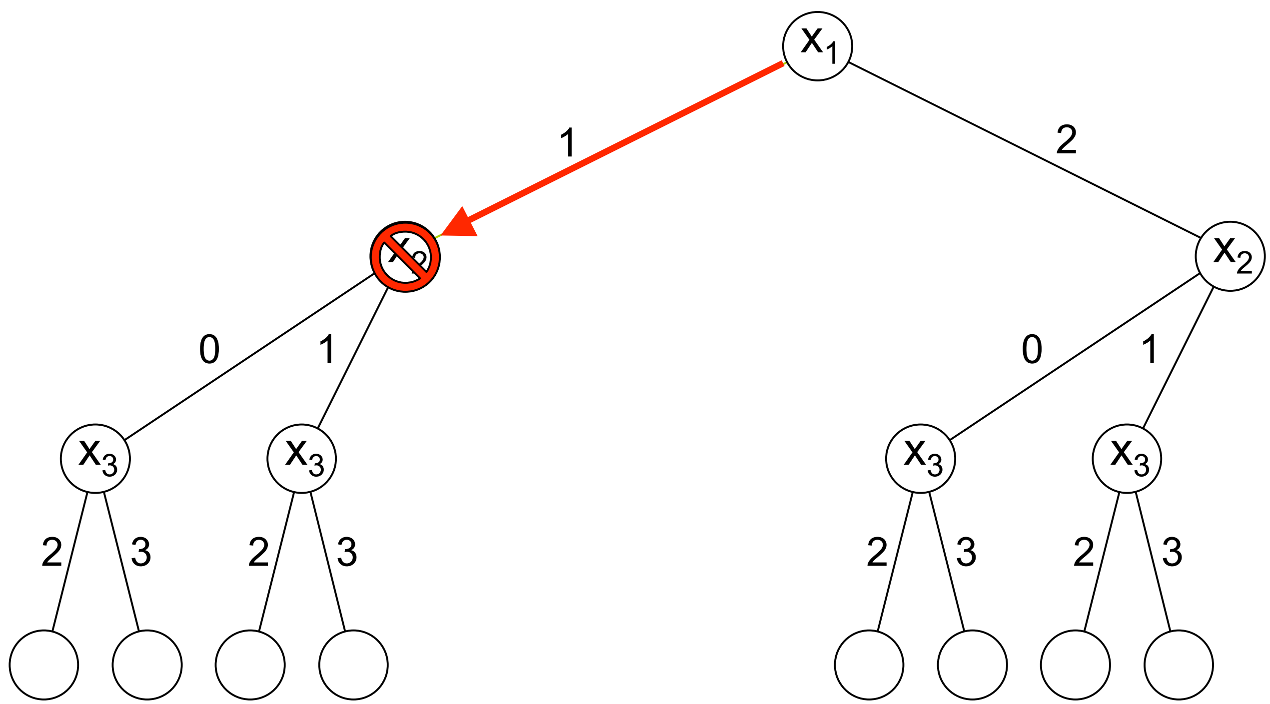
Example:

variables/domains $x_1 \in \{1\}$, $x_2 \in \{0, 1\}$, $x_3 \in \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)



Solving

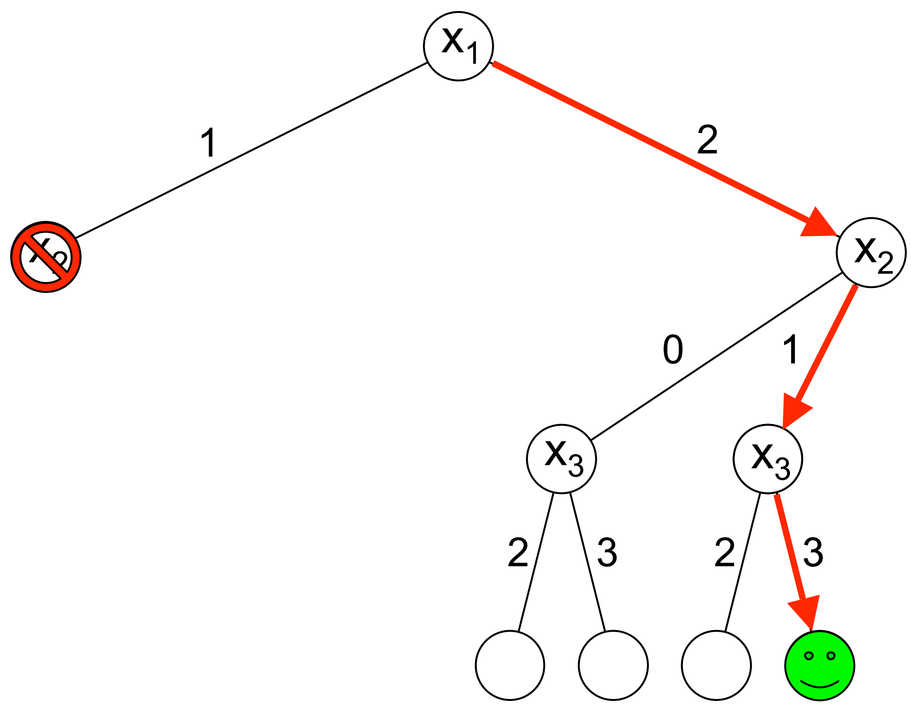
Example:

variables/domains $x_1 \in \{2\}$, $x_2 \in \{0,1\}$, $x_3 \in \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)



Solving - summary

The solution process of CP interleaves

- domain filtering
  - remove inconsistent values from the domains of the variables, based on individual constraints
- constraint propagation
  - propagate the filtered domains through the constraints, by re-evaluating them until there are no more changes in the domains
- search
  - implicitly all possible variable-value combinations are enumerated, but the search tree is kept small due to the domain filtering and constraint propagation

Because all variable-value combinations are (implicitly) enumerated, this solution method is complete

Domain Filtering Algorithms

Domain Filtering

Example:

alldifferent$(x_1,x_2,...,x_n)$ semantically equivalent to $\{ x_i \neq x_j$ for all $i \neq j \}$

$x_1 \in \{1,2\}$, $x_2 \in \{1,2\}$, $x_3 \in \{1,2\}$

$x_1 \neq x_2$, $x_1 \neq x_3$, $x_2 \neq x_3$

→ no filtering for <u>individual</u> not-equal constraints

$x_1 \in \{1,2\}$, $x_2 \in \{1,2\}$, $x_3 \in \{1,2\}$

alldifferent(x_1,x_2,x_2)

→ global view of *alldifferent*: no solution

Observation: conjunction of constraints allows more filtering!

Domain Filtering (cont'd)

More filtering: just group constraints together?

Problem: solving arbitrary conjunction of constraints is NP-hard

Solution:

- group constraints together that *occur frequently in applications*, and capture *tractable* structure
- result is called a *global* constraint

  (e.g., *alldifferent*)

(Alternative: keep NP-hard subproblem, but don't require to filter *all* inconsistent values)

Overview of global constraints

Constraint	Structure/technique
alldifferent	bipartite matching [Régin, 1994]
symmetric-alldifferent	general matching [Régin, 1999]
soft-alldifferent	matching [Petit, Régin & Bessière, 2001], minimum-cost flow [v.H., 2004]
open-alldifferent	network flow [v.H. & Régin, 2006]
cardinality	network flow [Régin, 1999, 2002]
soft-cardinality	minimum-cost flow [v.H., Pesant & Rousseau, 2006], [Milano & Zanarini, 2006]
open-cardinality	network flow [v.H. & Régin, 2006]
knapsack/sum	dynamic programming [Trick, 2003]
regular	directed acyclic graph [Pesant, 2004]
soft-regular	shortest paths [v.H., Pesant & Rousseau, 2006]
circuit	network flow [Genc Kaya & Hooker, 2006]
sequence	dedicated algorithm [v.H., Pesant, Rousseau & Sabharwal, 2006, 2009]
disjunctive/cumulative	dedicated algorithm [Carlier & Pinson, 1994] [Vilim, 2009]
inter-distance	dedicated algorithm [Quimper, Lopez-Ortiz & Pesant, 2006]
. . .	. . .

Filtering algorithm for *alldifferent*

J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 362-367, 1994.

Filtering for *alldifferent*

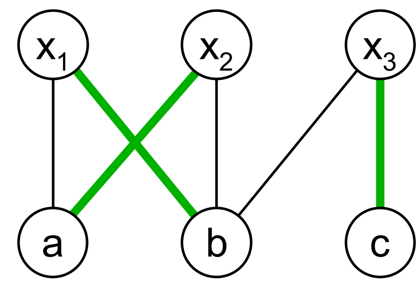
Observation [Régin, 1994]:

solution to *alldifferent* ⇔ matching in bipartite graph covering all variables

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$, $x_3 \in \{\cancel{b},c\}$

alldifferent(x_1,x_2,x_3)



Filtering: remove all edges (and corresponding domain values) that are not in any matching covering the variables

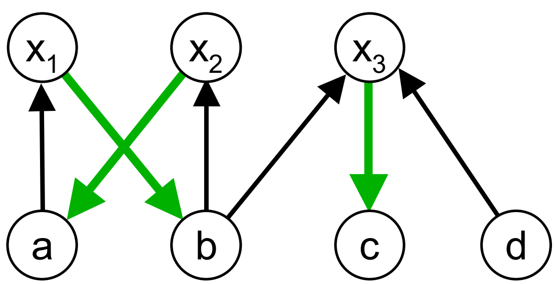
Find initial matching: $O(m\sqrt{n})$ time[1] [Hopcroft and Karp, 1973]

How to filter all inconsistent edges?

1 for *n* variables and *m* edges

Filtering Edges

- Naive approach is to fix each edge and test for consistency
  - Time complexity too high: $O(m^2\sqrt{n})$
- Instead, the following can be done
  - compute one maximum matching M: is it covering all variables X ?
  - orient the edges in M 'forward', and edges not in M 'backward'
  - compute the strongly connected components (SCCs) ← $O(m)$ [Tarjan `72]
  - edges in M, and edges on even M-alternating path are consistent (i.e., edges within SCC and edges on path starting from M-free vertex)
  - all other edges are not consistent and can be removed



Filtering in $O(m)$ time

Important aspects

- Separation of consistency check ($O(m\sqrt{n})$) and domain filtering ($O(m)$)
- Incremental algorithm
  - When k domain values have been removed, we can repair the matching in $O(km)$ time

Note that these algorithms are typically invoked many times during constraint propagation

- We can apply/embed efficient algorithms from graph theory, computer science, and operations research in global constraints

Soft Global Constraints

Over-constrained problems

- Assign seats for overbooked airplane; no solution that carries all passengers
- Create roster for employees with conflicting preferences
- Factory wants to satisfy demands of all customers, but has limited resources

  (Many industrial problems are essentially over-constrained)

A CP solver will report that no solution exists. How to find *acceptable* 'solution'?

- Soften (some of) the constraints of the problem
- Compute solution that minimizes conflicts or maximizes satisfaction

From soft constraints to hard optimization constraints

Carnegie Mellon Tepper SCHOOL OF BUSINESS

Cost-based approach [Petit, Régin, and Bessiere, 2000] (see also [Baptiste et al., 1998]):

- Introduce a cost variable for each soft constraint
- This variable represents some violation measure of the constraint
- Optimize aggregation of all cost variables (e.g., take their sum, or max)
- Use upper bound on cost variable to apply cost-based filtering (with back-propagation)

In this way

- soft global constraints become hard optimization constraints
- soft CSPs become hard COPs
- the cost variables can be used in other (meta-)constraints!

  if $(z_1 > 0)$ then $(z_2 = 0)$
- we can apply classical constraint programming solvers
- we can apply (cost-based) domain filtering algorithms!

Soft-alldifferent

Example: $x_1 \in \{1,2\}$, $x_2 \in \{1,2\}$, $x_3 \in \{1,2\}$

alldifferent(x_1,x_2,x_3)

↓

$x_1 \in \{1,2\}$, $x_2 \in \{1,2\}$, $x_3 \in \{1,2\}$, $z \in \{0,1,2,3\}$

soft-alldifferent(x_1,x_2,x_3,z)

minimize z

Let z represent the total number of violated not-equal constraints

Solution: $x_1=1$, $x_2=2$, $x_3=1$, $z=1$ with only $x_1 \neq x_3$ violated

Filter *soft-alldifferent*:

remove domain values for which minimum violation > max(z)

Note: Typically we have many more constraints in our model

Filtering algorithm for *soft-alldifferent*

v.H. A Hyper-Arc Consistency Algorithm for the Soft Alldifferent Constraint. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, LNCS 3258, pp. 679-689. Springer, 2004.

v.H., Pesant, and Rousseau. On Global Warming: Flow-Based Soft Global Constraints. *Journal of Heuristics* 12(4-5), pp. 347-373, 2006.

Filtering soft-alldifferent

Observation: solution to *soft-alldifferent* with minimum violation

$\Leftrightarrow$ integer minimum-cost flow

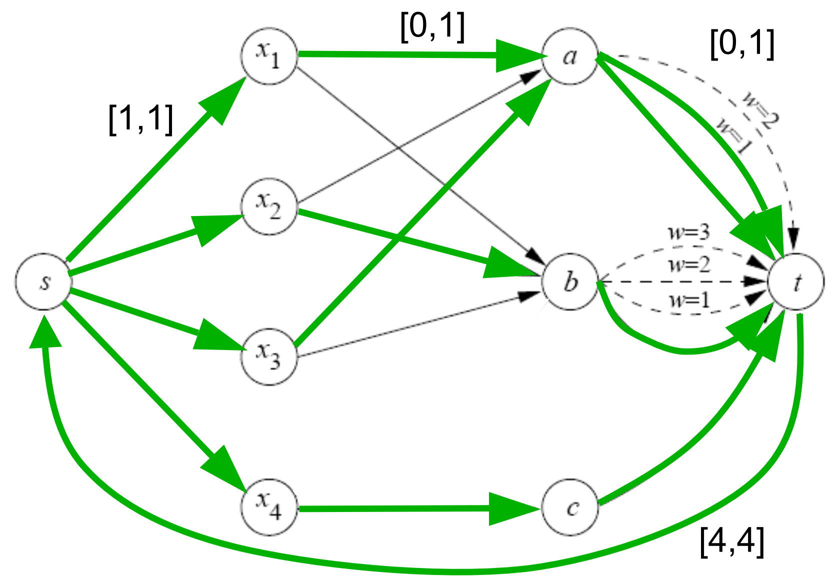
Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,

$x_3 \in \{a,b\}$, $x_4 \in \{$~~b~~$,c\}$, $z \in \{$~~0~~$,1\}$

soft-alldifferent(x_1,x_2,x_3,x_4,z)

minimize z



Filtering: remove all edges (and corresponding domain values) that are not in any flow f with cost(f) $\leq$ max(z)

Filtering soft-alldifferent

- Naive approach: fix each edge and test for consistency by computing a minimum-cost network flow
  - Time complexity: $O(m^2 n)$

Drawbacks:

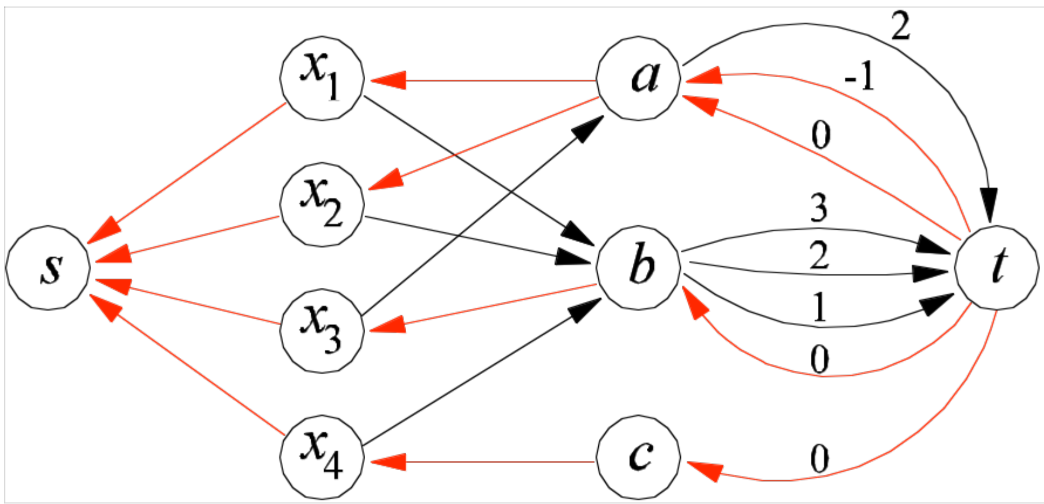
- no separation between consistency check and filtering
- time complexity too high
- algorithm not incremental: start from scratch every time

Improved Algorithm

To improve algorithm: use residual graph G_f

for all arcs a

- if f(a) = 1: reverse a and weight(a)
- if f(a) = 0: leave unchanged



Helpful Flow Theory

Theorem (e.g., Ahuja et al. 1993)

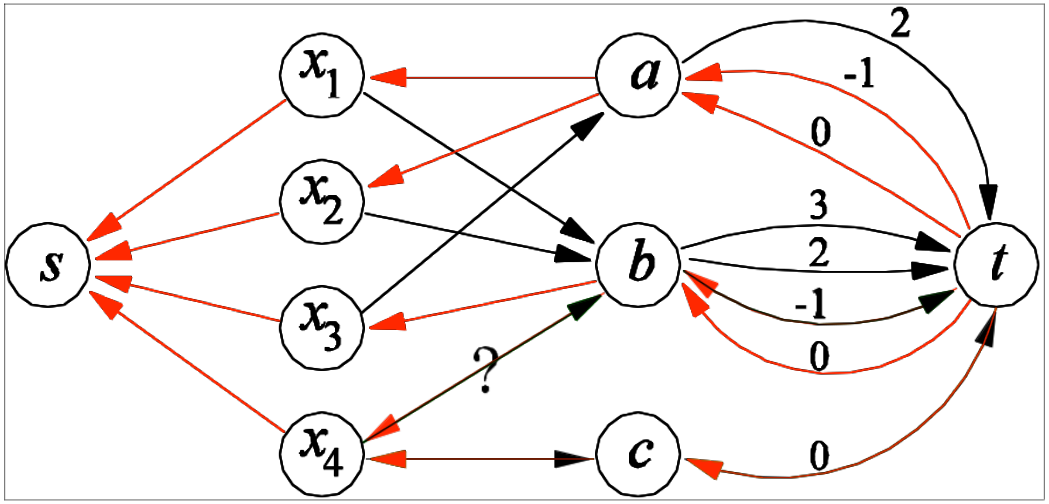
f minimum-cost flow in G

P shortest d-x_i path in G_f

$\Leftrightarrow$

minimum-cost flow f' in G with f'(x_i,d) = 1 has

cost(f') = cost(f) + cost(P)



Improved Algorithm

compute minimum-cost flow f in G

if cost(f) > max(D(z)) **return** inconsistent

for all arcs (x_i,d) {

  compute minimum-cost d-x_i path P in G_f

  if cost(f) + cost (P) > max(D(z)) remove d from D(x_i)

  if D(x_i) is empty **return** inconsistent

}

update min(D(z)) ≥ min(cost(f))

if D(z) is empty **return** inconsistent

else return consistent

Minimum-cost paths

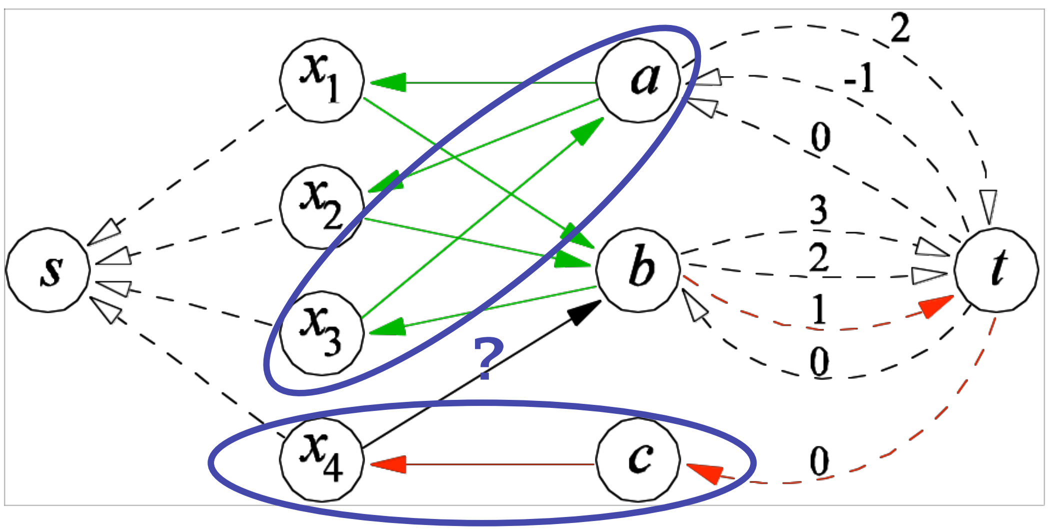
how to compute minimum-cost d-x_i path P in G_f:

if $f(x_i,d)=1$ then P = d,x_i and cost(P) = 0

else if x_i and d in same SCC[1] then cost(P) = 0

else P must visit t (once) and

cost(P) = min-cost SCC(d)-t path + min-cost t-SCC(x_i) path



[1] strongly connected component in G_f - {s,t}.

Improved algorithm in detail

compute minimum-cost flow f in G ← O(mn)

if cost(f) > max(z) **return** inconsistent

compute SCCs in G_f-{s,t} ← O(m)

compute minimum-cost paths from all nodes to t and reverse ← O(m)

for all arcs (x_i,d) {

compute minimum-cost d-x_i path P in G_f ← O(1)

if cost(f) + cost(P) > max(z) remove d from D(x_i)

if D(x_i) is empty **return** inconsistent

}

update min(D(z)) ≥ min(cost(f))

if D(z) is empty **return** inconsistent

else return consistent

- Consistency check: O(*mn*)
- Filtering all inconsistent values: O(*m*)
- Incremental: after *k* changes initial flow can be repaired in O(*km*) time

Many other soft global constraints

- Soft *cumulative* constraint [Baptiste et al., 1998], [Petit and Poder 2008]
- Soft global cardinality constraint [v.H. et al., 2006] [Zanarini et al. 2006, 2010]
- Soft *regular* constraint, soft *same* constraint [v.H. et al., 2006]
- Soft *slide* constraint [Bessiere et al., 2007]
- Sigma-*alldifferent*, Sigma-*Gcc*, Sigma-*regular* [Métivier et al., 2007, 2009]
- Soft *sequence* constraint [Maher et al., 2008]
- Soft *context-free grammar* constraint [Katsirelos et al., 2008]
- Soft constraints for timetabling application [Cambazard et al., 2008]
- Soft *all-equal* constraint [Hebrard et al., 2008], [Hebrard et al., 2009]
- Soft *precedence* constraint [Lesaint et al., 2009]
- Soft open global constraints [Maher, 2009]
- Soft global constraints for Weighted CSPs [Lee and Leung, 2009]

v.H. Over-Constrained Problems. In M. Milano and P. Van Hentenryck (eds.), *Hybrid Optimization: the 10 years of CPAIOR*, chapter 6. Springer, to appear.

Filtering algorithm for *sequence*

v.H., Pesant, Rousseau and Sabharwal. Revisiting the Sequence Constraint. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP 2006)*, pp. 620-634, LNCS 4204, 2006.

v.H., Pesant, Rousseau, and Sabharwal. New Filtering Algorithms for Combinations of Among Constraints. *Constraints* 14: 273-292, 2009.

Nurse rostering

- find feasible working pattern for each employee
- restrictions:
  - every calendar-week 4 or 5 working days
  - every 9 consecutive days at most 7 working days
  - every 30 consecutive days at least 20 working days

week	1	2	3	4	5
Sun	1	8	15	22	29
Mon	2	9	16	23	30
Tue	3	10	17	24	1
Wed	4	11	18	25	2
Thu	5	12	19	26	3
Fri	6	13	20	27	4
Sat	7	14	21	28	5

...

- additional constraints
  - demand, union requirements, night shift restrictions, etcetera

Sequence *constraint*

Example: every 9 consecutive days at most 7 working days

variable $x_i \in \{0,1\}$ for each day i

sun	mon	tue	wed	thu	fri	sat	sun	mon	tue	wed	thu
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}

$$\left.\begin{array}{l} 0 \le x_1+x_2+ \ldots +x_9 \le 7 \\ 0 \le x_2+x_3+ \ldots +x_{10} \le 7 \\ 0 \le x_3+x_4+ \ldots +x_{11} \le 7 \\ 0 \le x_4+x_5+ \ldots +x_{12} \le 7 \end{array}\right\} =: \textit{sequence}(x_1,x_2,\ldots,x_{12}, q=9, \textit{min}=0, \textit{max}=7)$$

sequence($x_1,x_2,\ldots,x_n$, *q*, *min*, *max*):

the sum of every *q* consecutive variables is between *min* and *max*

A *sequence* constraint groups together the individual constraints

Sequence *constraint*

sequence is more powerful than individual constraints filtered separately

Example: *sequence*($x_1, x_2, \ldots, x_7$, *q*=5, *min*=2, *max*=3)

$x_1=1,\ x_2=1,\ x_6=0$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	0/1	0/1	0/1	0	0/1

≤ 1 (x_3, x_4, x_5)

$=1$ (x_7)

$$2 \leq x_1+x_2+x_3+x_4+x_5 \leq 3$$

$$2 \leq x_2+x_3+x_4+x_5+x_6 \leq 3$$

$$2 \leq x_3+x_4+x_5+x_6+x_7 \leq 3$$

Sequence *constraint*

History:

1988: car sequencing (Dincbas, Simonis & Van Hentenryck, 1988)

1994: *sequence* introduced (Beldiceanu & Contejean, 1994)
as conjunction of overlapping cardinality constraints

1997: filtering algorithm (Régin & Puget, 1997)
tailored to car sequencing, no complete filtering

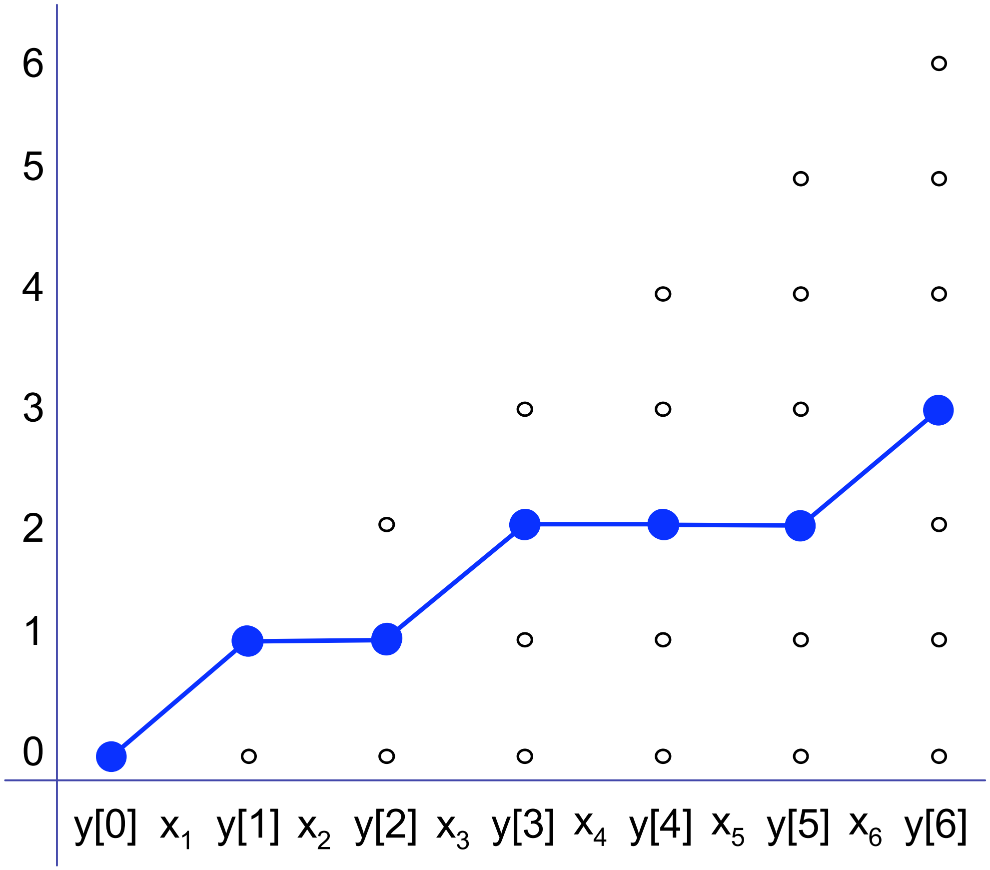
2001: filtering algorithm (Beldiceanu & Carlsson, 2001)
instance of generic class of *cardinality-path* constraints, no complete filtering

Goal: efficient (polynomial-time) complete filtering for sequence

Accumulative solutions

Accumulate variables: $y[i] = x_1 + x_2 + \ldots + x_i$

Example: *sequence*(x_1,x_2,x_3,x_4,x_5,x_6, q=3, min=1, max=2)



$1 \le y[3] - y[0] \le 2$

$1 \le y[4] - y[1] \le 2$

$1 \le y[5] - y[2] \le 2$

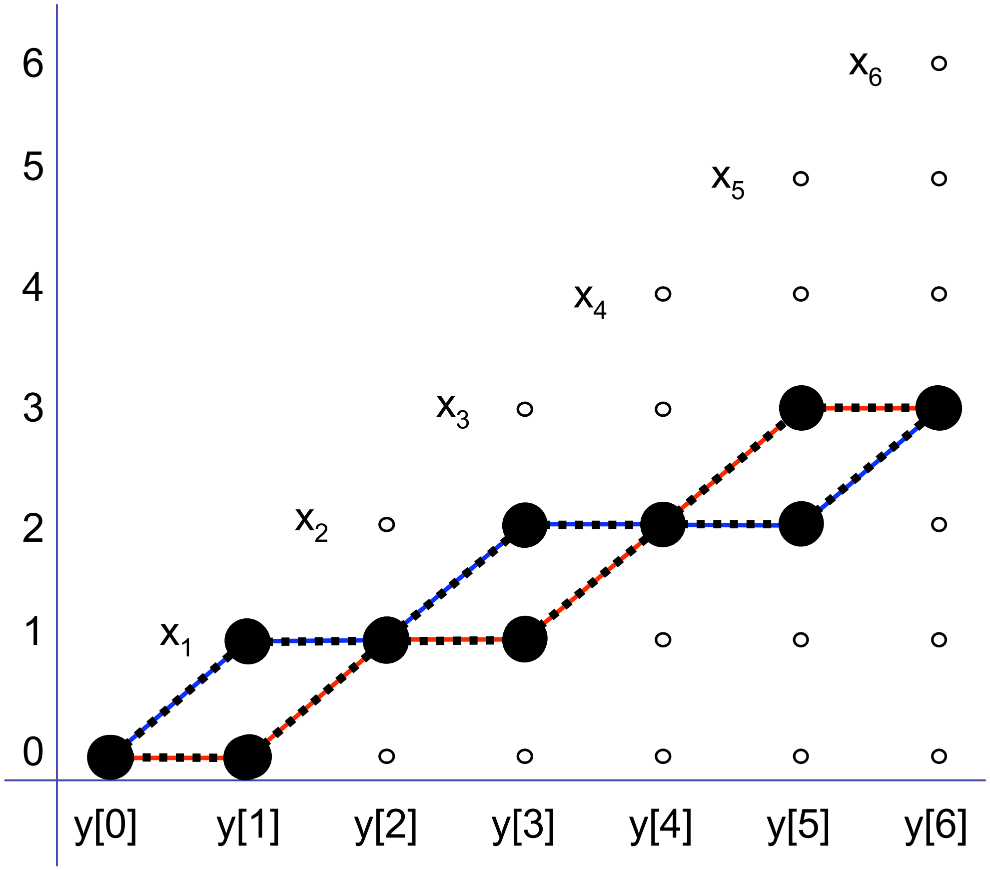
$1 \le y[6] - y[3] \le 2$

y_{blue} = 0 1 1 2 2 2 3

Accumulative solutions

Accumulate variables: $y[i] = x_1 + x_2 + \ldots + x_i$

Example: *sequence*(x_1,x_2,x_3,x_4,x_5,x_6, q=3, *min*=1, *max*=2)



Observation: for any two accumulate solutions, their pointwise minimum and maximum are also solutions

y_{blue} = 0 1 1 2 2 2 3

y_{red} = 0 0 1 1 2 3 3

This is not true for binary x representation!

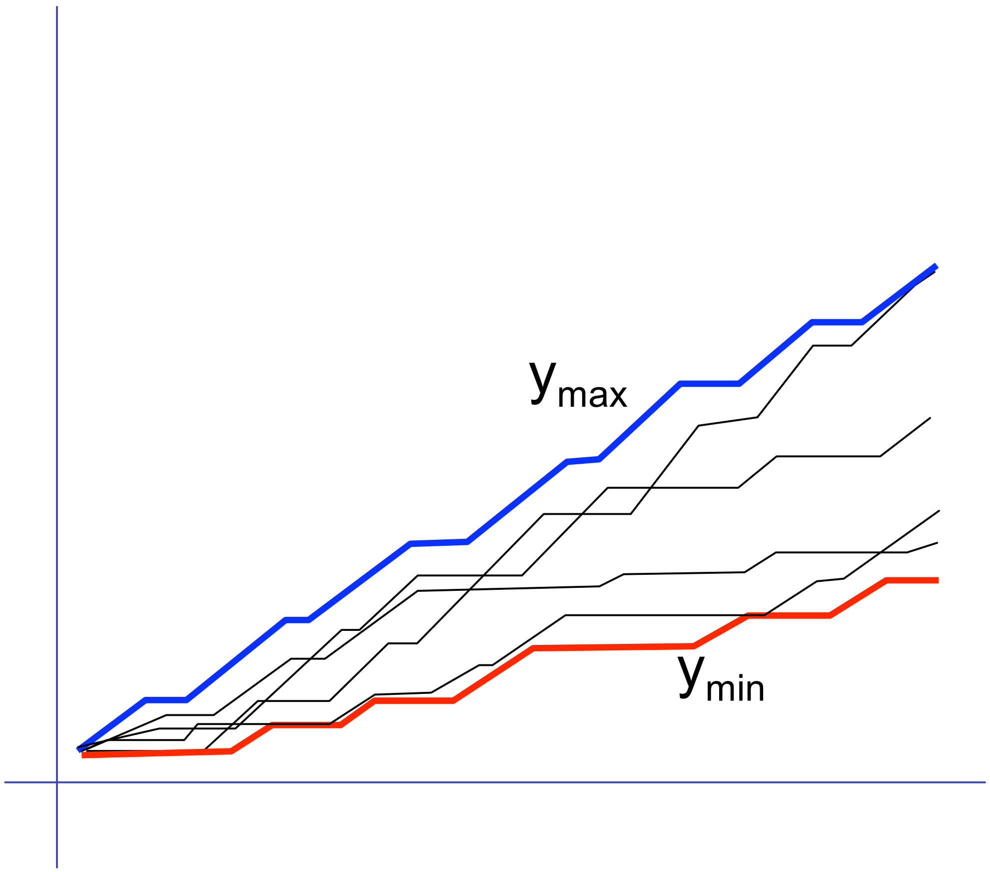
x_{blue} = 1 0 1 0 0 1

x_{red} = 0 1 0 1 1 0

Accumulative solutions

Accumulate variables: $y[i] = x_1 + x_2 + \ldots + x_i$

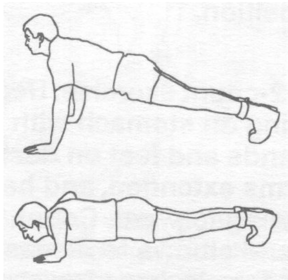
Example: *sequence*$(x_1,x_2,x_3,x_4,x_5,x_6,$ q=3, *min*=1, *max*=2)



Observation: for any two accumulate solutions, their pointwise minimum and maximum are also solutions

Corollary: absolute *minimum* and *maximum* solutions envelope all solutions

Find minimum solution



Algorithm:

initialize y

while some subsequence violated

push-up endpoint minimally

repair on left and right (using push-ups)

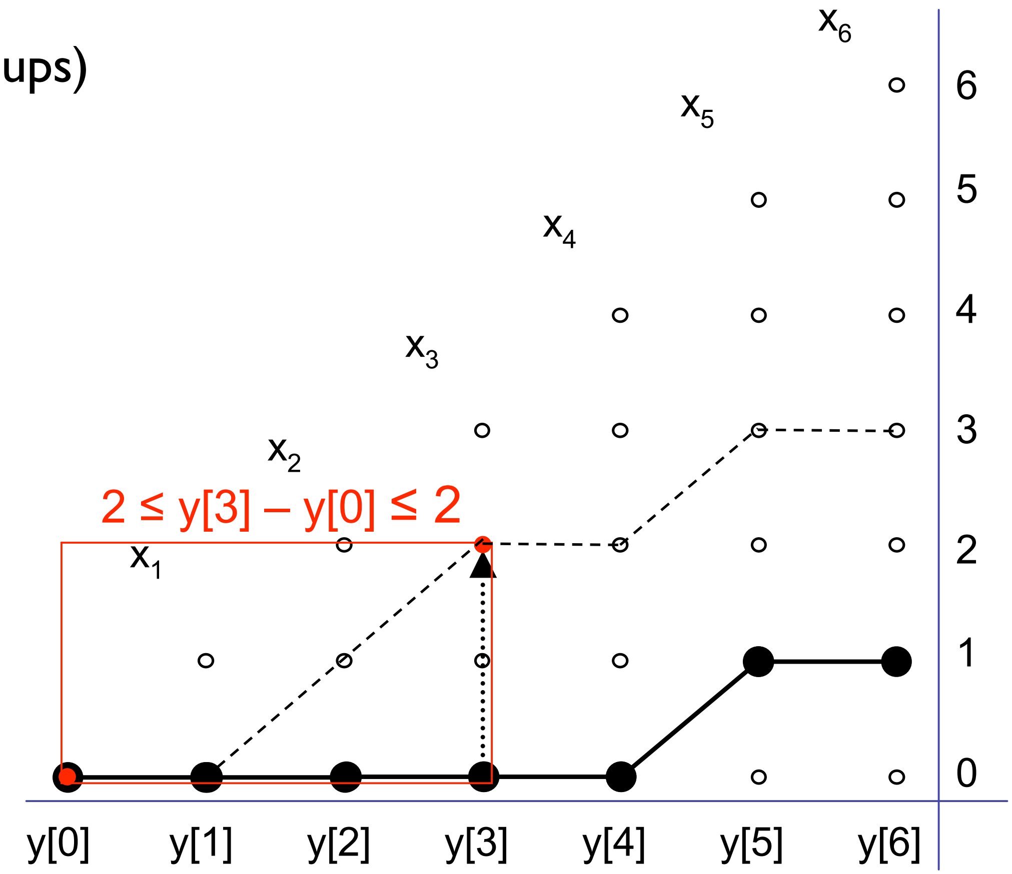
invariant: y[i+1] - y[i] is 0 or 1

Example:

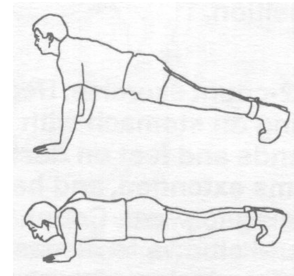
sequence($x_1,x_2,\ldots,x_6$, *q*=3, *min*=2, *max*=2)

$D(x_i) = \{0,1\}$ for all $i \neq 5$

$D(x_5) = \{1\}$



Find minimum solution



Algorithm:

initialize y

while some subsequence violated

push-up endpoint minimally

repair on left and right (using push-ups)

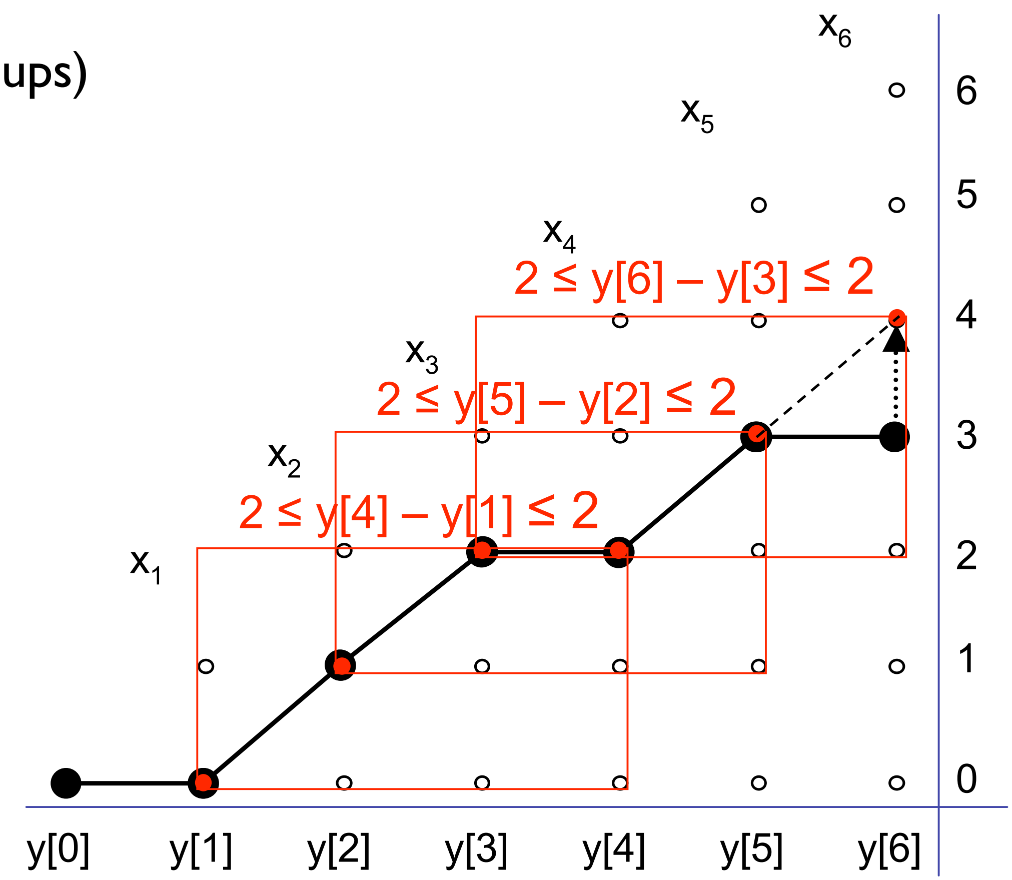
invariant: y[i+1] - y[i] is 0 or 1

Example:

sequence(x_1,x_2,...,x_6, *q*=3, *min*=2, *max*=2)

$D(x_i) = \{0,1\}$ for all $i \neq 5$

$D(x_5) = \{1\}$



Find minimum solution

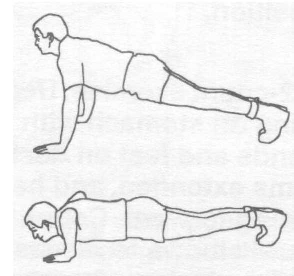
Algorithm:

initialize y

while some subsequence violated

push-up endpoint minimally

repair on left and right (using push-ups)



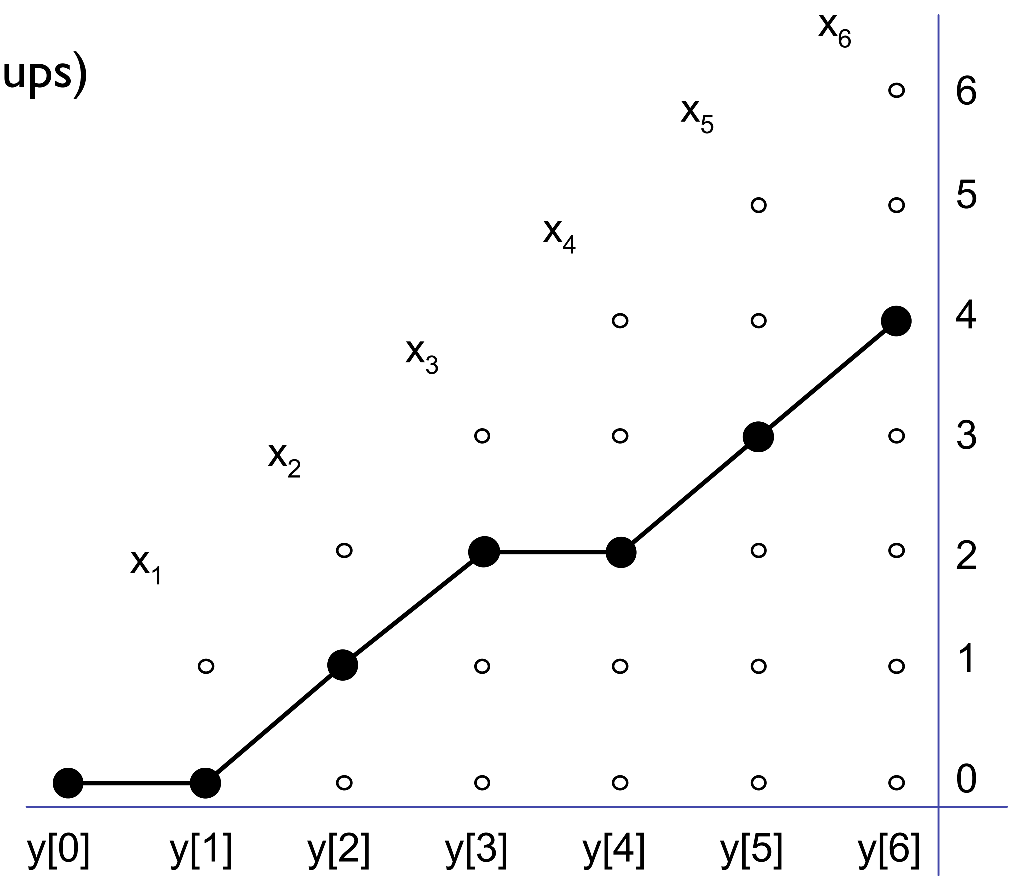
invariant: y[i+1] - y[i] is 0 or 1

Example:

sequence($x_1, x_2, \ldots, x_6$, *q*=3, *min*=2, *max*=2)

$D(x_i) = \{0,1\}$ for all $i \neq 5$

$D(x_5) = \{1\}$



Find minimum solution

Properties:

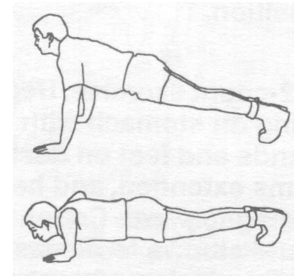
- repair keeps $y[i] \leq y_{min}[i]$ for all i (by induction)

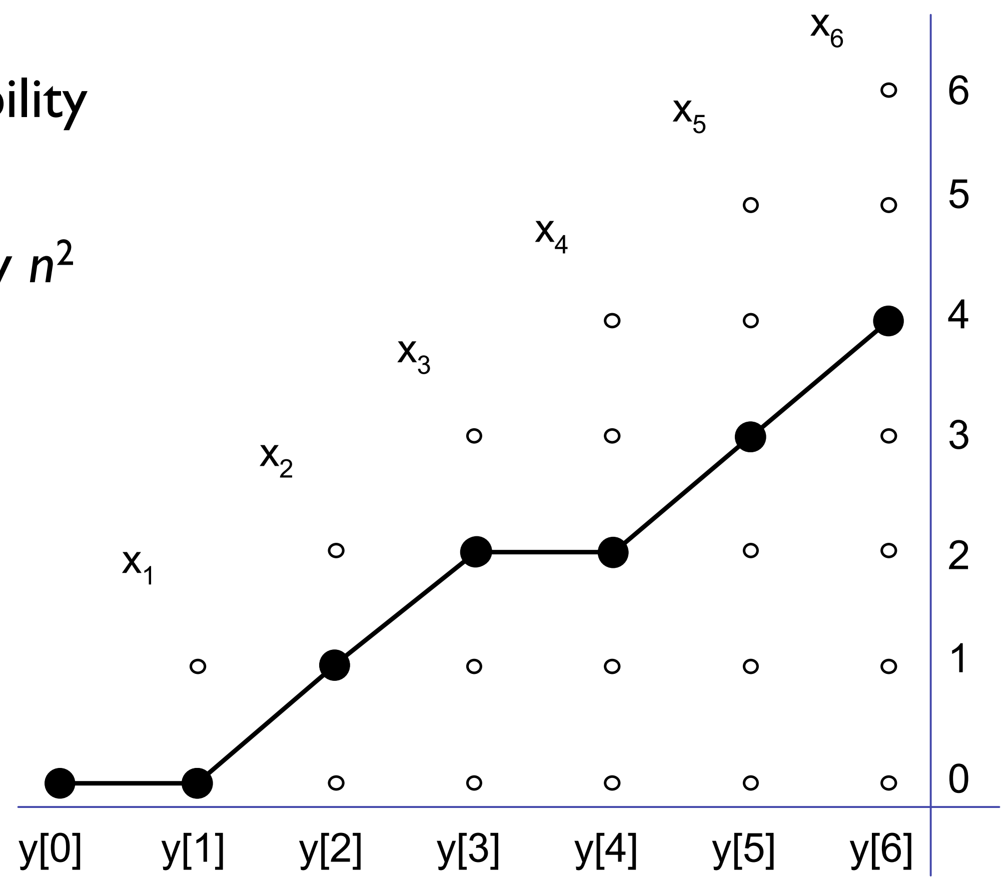
  hence, if minimum solution exists, algorithm finds it

  otherwise, $y[i] > i$ leads to unsatisfiability

- total number of push-ups bounded by n^2

  algorithm runs in $O(n^2)$ time





Filtering algorithm

Basic algorithm:

for every domain value:

compute minimum solution (using this value)

if no solution, remove value

Filtering: we remove *all* inconsistent values

Time complexity: $O(n^3)$

Improvements:

- maintain supports for domain values
  - each solution provides support for *n* values
- for each value, restart from y_{min}
- also compute maximum solution y_{max}
  - detect violation if $y[i] > y_{max}[i]$
- maintain y_{min} and y_{max} during search (both are monotone)

amortize complexity: $O(n^3)$ on any path from root to a leaf

Additional remarks

- filtering algorithm also applies to *generalized sequence*:

  q, *min*, and *max* vary per subsequence

  Example: nurse rostering problem

  – every calendar-week 4 or 5 working days
  – every 9 consecutive days at most 7 working days
  – every 30 consecutive days at least 20 working days

- '*sequence*' on non-consecutive subsequences is NP-hard
  [Régin, 2005]

Single sequence *constraint*

Carnegie Mellon Tepper School of Business

n = 100		ILOG Basic		ILOG Extended		our algorithm	
q	(max – min)	back-tracks	CPU	back-tracks	CPU	back-tracks	CPU
5	1	limit	limit	34K	18	0	0.01
6	2	362K	54	19K	6	0	0.01
7	1	381K	55	113K	48	0	0.01
7	2	265K	54	7K	4	0	0.02
7	3	287K	48	0	0.5	0	0.02
9	1	limit	limit	61K	42	0	0.01
9	3	195K	43	0	0.7	0	0.02

Single sequence *constraint*

max – min=1		ILOG Basic		ILOG Extended		our algorithm	
q	*n*	back-tracks	CPU	back-tracks	CPU	back-tracks	CPU
5	50	459K	18	23K	18	0	0.001
5	100	192K	12	12K	12	0	0.01
5	500	48K	12	1K	42	0	0.47
5	1000	1K	1	2.3	160	0	4.2
7	50	210K	12	68K	12	0	0.001
7	100	221K	18	45K	19	0	0.01
7	500	80K	21	624	49	0	0.50
7	1000	30K	28	46	139	0	3.3
9	50	18K	1	18K	8	0	0.001
9	100	3K	0.3	2K	11	0	0.01
9	500	49K	18	1K	66	0	0.49
9	1000	17K	20	19	169	0	3.3

Generalized sequence constraint

Instances:

- inspired by nurse rostering problems
- two *sequence* constraints
- find *all* solutions

			our individual *sequence* constraints		*our generalized sequence* constraint	
instance type	horizon	#solutions	backtracks	time	backtracks	time
max6/8-min22/30	40	2248	185k	4 min	0	0.77 s
	80	730	198k	18 min	0	0.61 s
max6/9-min20/30	40	3	394k	7 min	0	0.01 s
	80	3	394k	30 min	0	0.05 s
max7/9-min22/30	40	138k	328k	7 min	0	34 s
	80	23k	1847k	2 hours	0	15 s

Further Improvements

- Brand et al. [2007] have shown that our algorithm can be interpreted as a 'Singleton Bounds Consistency' algorithm on the cumulative decomposition:

  $$y_{i+1} = y_i + x_i$$

  $$y_{i+q} - y_i \geq l$$

  $$y_{i+q} - y_i \leq u$$

  This decomposition has the same filtering power and the same complexity, but runs faster in practice
- Using a different decomposition Brand et al. show that complete filtering can be done in O(n^2 log n) time
- Maher et al. [2008] present an O(n^2) algorithm, by representing the problem as an integer program and then converting it into a network flow

Other Recent Developments

Open constraints

Traditional CSPs:

- all variables and constraints are fixed from the beginning
- "closed-world scenarios"

Open CSPs:

- variables and constraints are revealed over time

Example:

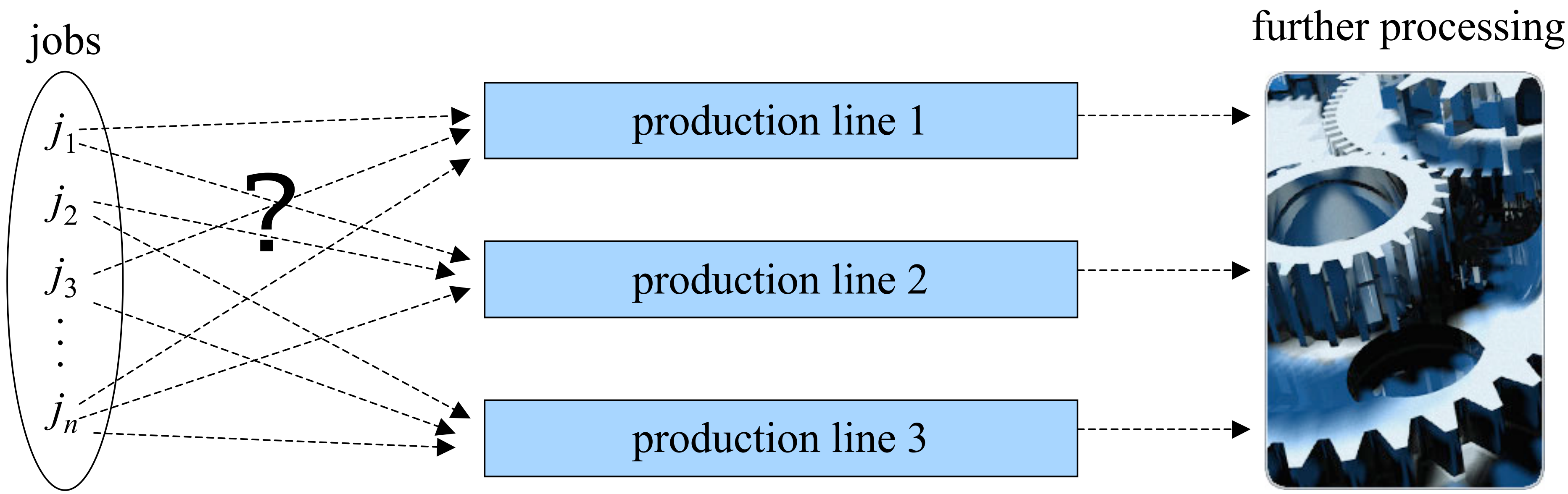
- process set of activities over different factory lines: each factory line has predefined set of constraints, but paths of the activities are unknown initially

Open constraints: defined on a-priori unknown set of variables

Open constraints in a closed world: all *potential* variables and domains are known

[v.H. and Regin, 2006]

Example



variables:

- start(j_1),...,start(j_n)

constraints:

- each task must be processed on one production line
- on each production line, the start times of the tasks are different:

open-alldifferent(S_i) for production line i=1,2,3

where S_i is a set-variable representing the start time variables of the jobs on line i

Open constraints

Traditional CSP:

alldifferent$(x_1,x_2,\ldots,x_n)$

$x_i \in D(x_i)$ for $i=1,\ldots,n$

Open CSP (in a closed world):

alldifferent(S)

$S \in [\emptyset,\{x_1,x_2,\ldots,x_n\}]$ (S is a *set variable*)

$x_i \in D(x_i)$ for $i=1,\ldots,n$

Goals: given an open constraint, we want to filter all inconsistent values from

- $D(x_i)$ for all i,
- and $D(S)$
  - add mandatory elements to lower bound,
  - remove impossible elements from upper bound
  - compute tight lower and upper bound on cardinality of S

Efficient filtering algorithms for open *alldifferent* (and *gcc*), and *conjunctions* of them, can be designed using specific network flow representation [v.H. and Regin, 2006]

Constraint-Based Search

- The combinatorial structure embedded by global constraints can also be used for other purposes than only filtering, for example to guide the search
- Examples
  - Constraint Based Local Search
  - Counting Based Search

Constraint-Based Local Search

Aim: Model the problem using variables and constraints (as in CP), and apply an automatically-derived Local Search method to solve the model

[Van Hentenryck and Michel, 2002, 2005], [Galinier and Hao, 2000,2004], [Bohlin 2004, 2005]

Essential to CBLS is that the solution method can be derived from the constraints

- Local Search evaluates current assignment and then moves to an (improving) assignment in its neighborhood
- Neighborhoods as well as evaluation functions can be based on combinatorial properties of the constraints
- Global constraints can be particularly useful for this purpose [Nareyek, 2001]

Constraint-Based Local Search (cont'd)

Soft global constraints for CBLS [Van Hentenryck and Michel 2005]

- Instead of domain filtering, the task is to measure the additional amount of violation (gradient) if we were to assign a variable to a certain value
- Violation measures are given for *alldifferent*, *atmost*, *atleast*, *multi-knapsack*, *sequence*, systems of not-equal constraints, and weighted constraint systems

Counting-Based Search

- Aim: Guide the search to ‘promising’ search space containing many solutions
- Branching decision defined by selecting a variable-value pair
- So we need to associate to each variable-value pair a measure indicating to how many solutions it belongs
- Counting number of solution is #P-complete in general
- However, we can efficiently find approximations for individual *global constraints* and then aggregate the results [Pesant 2005], e.g.,
  - *alldifferent* and *regular* constraints [Zanarini and Pesant, 2007, 2009]
  - *knapsack* constraints [Pesant and Quimper, 2008]

Summary

- Global constraints are driving force of successful application of constraint programming
- This talk: efficient domain filtering algorithms for
  - *alldifferent* (matchings)
  - *soft-alldifferent* (minimum-cost network flow)
  - *sequence* (dedicated algorithm)

  They provide of flavor of what can be done with global constraints

- Many more research opportunities