Constraint Programming and Graph Algorithms

Kurt Mehlhorn
Max-Planck-Institute for Computer Science
(with significant help from Sven Thiel)

- I am not an expert on the subject (four publications),
  but I consider the subject an important one.
- I want to get some of you interested.
- constraint programming is a rich source of algorithmic problems
- efficient algorithms make a difference
- impact is multiplied through CP systems

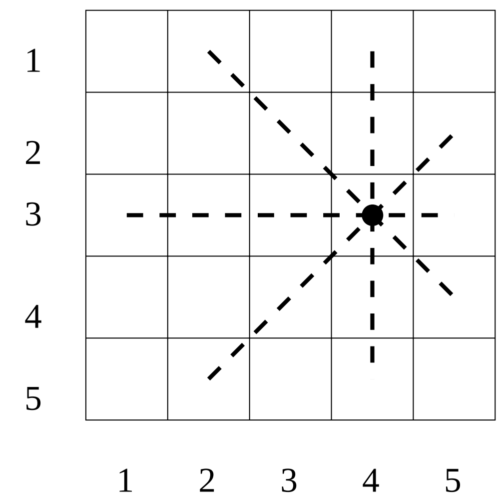
Constraint Programming

- What is it about?
  - specify problems by systems of constraints
    - **Variables:** $x_1, x_2, \ldots, x_n$, values in $I\!N$
    - **Constraints:** $C_1(x_1, \ldots, x_n), \ldots, C_k(x_1, \ldots, x_n)$
  - solve problems (= find satisfying assignment) by pressing **solve**.
- What are its benifits?
  - it is general: N-queens, logical satisfiability, scheduling
  - it is convenient: ILOG-solver, Oz, Claire, Eclipse, Chip, Sictus-Prolog, … offer powerful constraint languages

N-Queens-Problem

- Place n queens on an $n \times n$ chessboard,
  no two in a row, column, diagonal, antidiagonal
- Variables: $x_1, \ldots, x_n$ — i-th queen is in row i and column x_i
- Constraints:

$$x_i \in \{1, 2, \ldots, n\}$$
$$\mathit{Alldiff}\,(x_1, \ldots, x_n)$$
$$\mathit{Alldiff}\,(x_1 + 1, \ldots, x_n + n)$$
$$\mathit{Alldiff}\,(x_1 - 1, \ldots, x_n - n)$$



- that's it (the Oz-program has 10 lines), **isn't this great?**

Solution Method: Enumeration and Narrowing

- assume variable x_i takes values in S_i, $1 \leq i \leq n$.
- S_i is called the domain of x_i
- **enumeration** tries all possibilities

```
forall x1 in S1
  forall x2 in S2
    forall x3 in S3
      ....
```

- **narrowing (pruning)** attempts to eliminate values from the domains and to close branches
- For example,
  - if x_1 is already restricted to a single value, say z,
  - the constraint $x_2 \neq x_1$ may remove z from the domain of x_2.

Solution Method: Enumeration and Narrowing

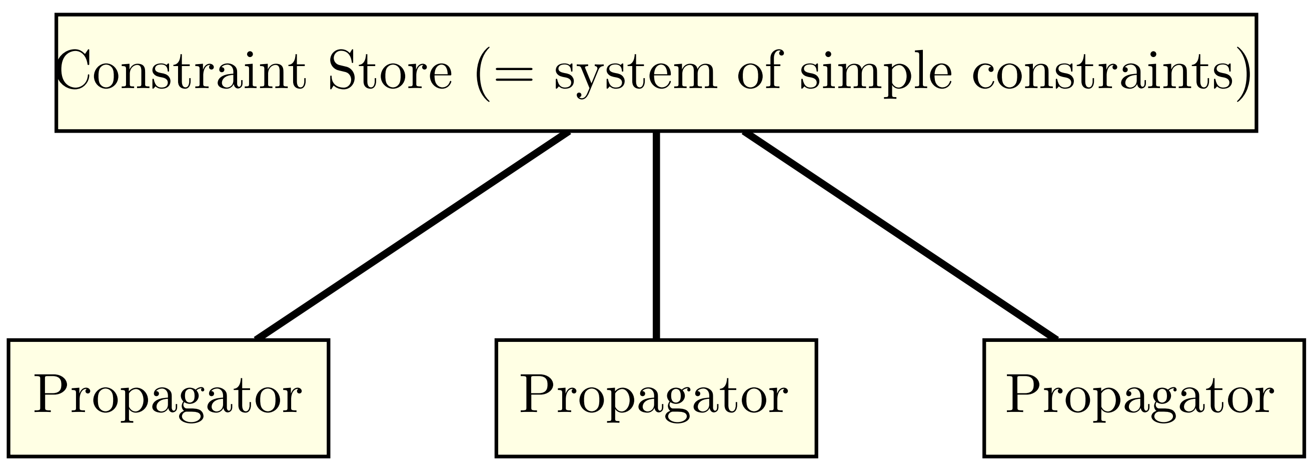
- distinguish between *simple* and *difficult* constraints

  distinction is a pragmatic one

- satisfaction problem must be trivially solvable for systems of simple constraints

  Example: $x_1 \in S_1 \wedge x_2 \in S_2 \wedge \ldots \wedge x_n \in S_n$

- difficult constraints are implemented as algorithms (called *propagators*): propagators strengthen (narrow) the constraint store



Propagators

Constraint store = set A of assignments.

A propagator for a constraint C (Ex: $x < y$) may conclude that:

- no $a \in A$ satisfies $C \Longrightarrow$ state becomes empty, Ex: $x \in \{5, 6\}, y \in \{1, 2, 3\}$
- all $a \in A$ satisfy $C \Longrightarrow$ propagator dies, Ex: $x \in \{1, 2, 3\}, y \in \{5, 6\}$
- additional simple constraints hold $\Longrightarrow$ state is narrowed,
  Ex: $x \in \{2, 3, 4, 5\}, y \in \{2, 3, 4\} \Longrightarrow x \neq 5, x \neq 4, y \neq 2$
- it is too weak to make any conclusions

Propagators

Constraint store = set A of assignments.

A propagator for a constraint C (Ex: $x < y$) may conclude that:

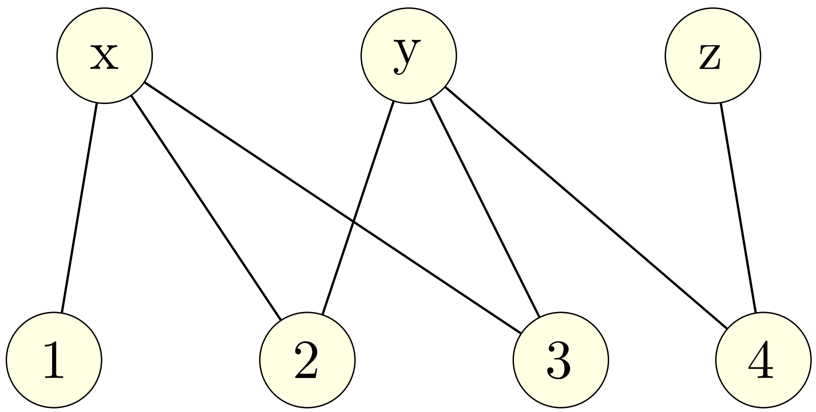
- no $a \in A$ satisfies $C \Longrightarrow$ state becomes empty, Ex: $x \in \{5,6\}, y \in \{1,2,3\}$
- all $a \in A$ satisfy $C \Longrightarrow$ propagator dies, Ex: $x \in \{1,2,3\}, y \in \{5,6\}$
- additional simple constraints hold $\Longrightarrow$ state is narrowed,
  Ex: $x \in \{2,3,4,5\}, y \in \{2,3,4\} \Longrightarrow x \neq 5, x \neq 4, y \neq 2$
- it is too weak to make any conclusions

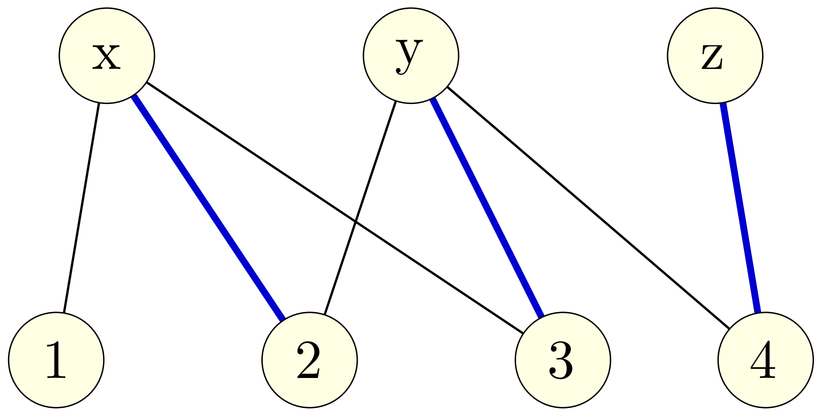
The system applies propagators to the constraint store until either

state becomes empty	failure, no satisfying assignment
all propagators die	success, all assignments satisfy
neither of the above	branch and recur on the resulting stores

Alldiff and Matchings

- Ex: $x \in \{1, 2, 3\}$, $y \in \{2, 3, 4\}$, $z \in \{4\}$ and $Alldiff(x, y, z)$
- Régin (94): A filtering algorithm for constraints of difference in CSPs
- bipartite graph: vars on one side, values on other side
- $(x, val) \in E$ iff val is a possible value for x

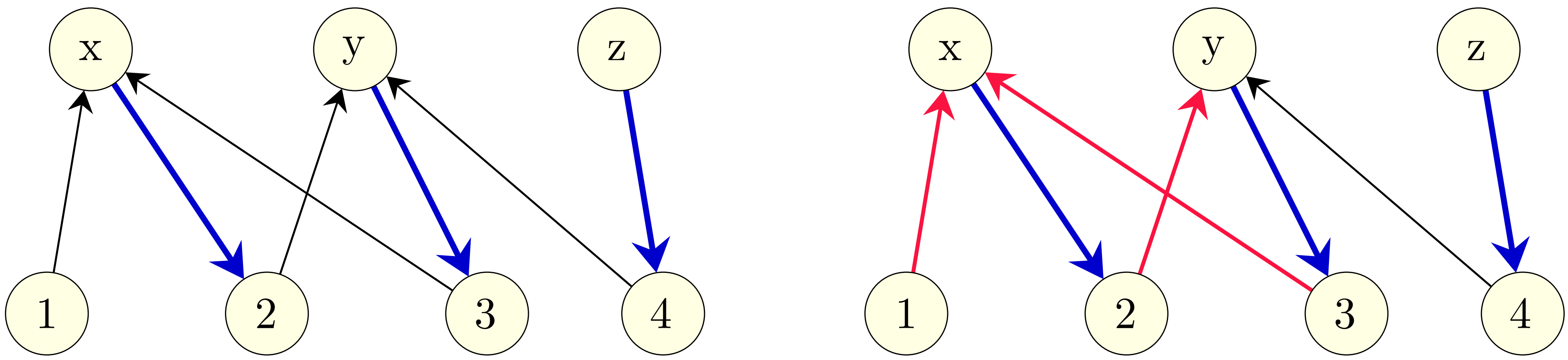




- satisfying assignment = var-perfect matching
- narrowing: delete edges that belong to **no** var-perfect matching

Alldiff and Matchings, Part II

- a var-perfect matching can be found in time $O(\sqrt{n}m)$ time, where $n =$ number of nodes and $m =$ number of edges.
- narrowing: orient matching edges from vars to vals, free edges from vals to vars



- an edge belongs to some var-perfect matching iff
  - it lies in a strongly connected component or
  - on a path starting in a free value
- narrowing is a simple $O(m)$ computation (given a perfect matching)

Alldiff and Matchings, Part III

- branching and decremental algorithms
  - a branch step on a variable splits the domain of a variable into two
  - gives us near-perfect matchings in both subgraphs
  - recomputation of a perfect matching in time $O(m)$ by a single search for an augmenting path
  - we need decremental dynamic algorithms

- graphs are frequently dense, $m = \Theta(n^2)$
  - $O(m)$ might be too slow
  - we need sublinear time algorithms

High-Level Constraints Make A Difference

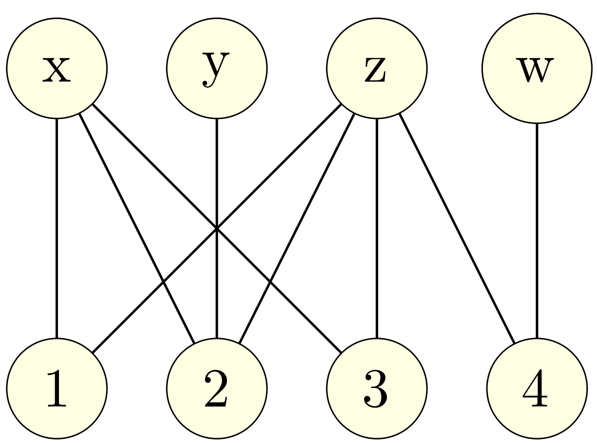
- *Alldiff* $(x_1, \ldots, x_n)$ can also be modelled as $x_i \neq x_j$ for $i \neq j$.
- narrowing algorithm for low-level formulation is trivial: if domain of some variable is a singleton, remove the value from the other domains.
- but propagation strength is much lower
  - assume: domain of each x_i is a random subset S_i of size 3 of $\{1, \ldots, n\}$
  - if no var-perfect matching exists, *Alldiff* will terminate immediately
  - on the other hand: as long as $k \ll \sqrt{n}$ variables are fixed
    * it is unlikely that there is a variable v whose domain is pruned to a singleton or even less. For fixed v

$$\text{prob}(|S_v \cap \text{the } 3k \text{ values already used up}| \geq 2) \leq 3(3k/n)^2 \ll 1/n$$

    * at least two values have to be tried for the first $\approx \sqrt{n}$ vars
    * running time $\Omega(2^{\sqrt{n}})$

The Alldiff Constraint: Bound Narrowing

- ranges are intervals; $x_i \in [l_i \,..\, r_i]$
- goal: narrow the intervals by increasing l_i's and/or decreasing r_i's.
- Puget (98) $O(n \log n)$, M/Thiel (2000) $O(n)$ + time to sort endpoints
- bipartite graph has $\sum_i (r_i - l_i + 1)$ edges (that's a lot),
  but a simple structure (that's good) already known to Glover (67)



  - 1 can be matched with x or with z
  - it would be stupid to match it with z, since z has more possibilities than x
  - scan through the vals and match with the var that ends first.
  - $insert(r_x = 3)$, $insert(r_z = 4)$, $delmin$, $insert(r_y = 2)$, $delmin$, $delmin$, $insert(r_w = 4)$, $delmin$

- matching and sccs in time $O(n)$ after sorting

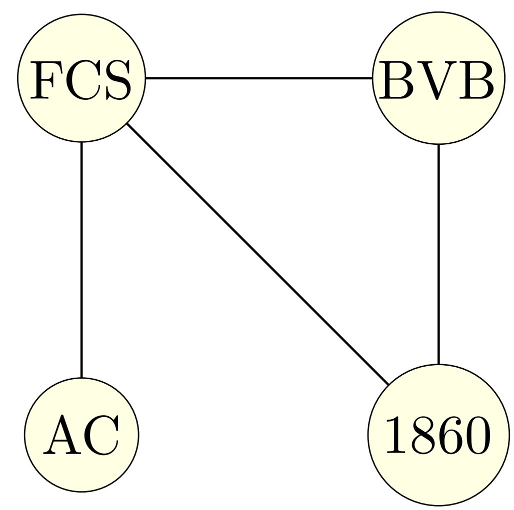
Linear Time Matching in Convex Graph

$insert(r_x = 3)$, $insert(r_z = 4)$, $delmin$, $insert(r_y = 2)$, $delmin$, $delmin$, $insert(r_w = 4)$, $delmin$

- can be answered in time $O(n \log n)$ using a priority queue
- but this is an off-line extract min problem
- full sequence of inserts and delmins is known before ...
- can be answered by union-find
  - 2,4, *delmin*, 1, *delmin*, *delmin*, 3, *delmin*
  - which delmin is going to return 1? The first one following it.
  - 2, 4, *delmin*, *delmin*, 3, *delmin*
  - which delmin is going to return 2? The first one following it.
  - 4, *delmin*, 3, *delmin*,
- union-find on partition of a line is $O(n)$ (Asano/Asano)

Matchings in Sports Scheduling

- schedule a round of matches for n teams
- $(x, y) \in E$ iff x may be matched with y
- satisfaction: find a matching in a general graph
- narrowing: delete edges that do not belong to a perfect matching
- satisfaction: $O(\sqrt{n}m\alpha(n, m))$
  narrowing: $O(nm\alpha(n, m))$
- theory: Régin
- experiments: Henz/Müller/Tan/Thiel
- open problem: find a faster narrowing algorithm



Clever Narrowing Algorithms Help, I

- Henz/Müller/Tan/Thiel
- n teams, schedule $n-1$ rounds of play so that ...
- in each round a subset of the matches are forbidden
- $x_{t,i}$ opponent of team t in round i

$$\begin{array}{rll} \mathit{Alldiff}(x_{t,1},\ldots,x_{t,n-1}) & \text{for all} & t,\ 1 \le t \le n \\ \mathit{Pairing}(x_{1,i},\ldots,x_{n,i}) & \text{for all} & i,\ 1 \le i \le n-1 \end{array}$$

Clever Narrowing Algorithms Help, II

problem	*neq* / *eq*	*Alldiff*	*Pairing*
`s_14_yes`	242.	75.3	20.4
`s_14_no`	16.7	10.9	2.54
`s_16_no`	64.5	18.0	5.37

- *Alldiff* and *Pairing* are powerful constraints
  - can also be expressed as collections of simpler constraints:

$$\mathit{Alldiff}(z_1, \ldots, z_n) \iff z_i \neq z_j \text{ for } i \neq j$$

$$\mathit{Pairing}(z_1, \ldots, z_n) \iff z_i \neq i \text{ and } z_i = j \text{ iff } z_j = i$$

  - progragation algorithms for the simpler constraints are trivial, but narrowing is much less effective.

Further Examples

constraint	algorithm	authors
sorting	bipartite and sccs in convex graphs	GC, MT
global cardinality	flow	RP
global cardinality with costs	min cost flow	R
aggregation of constraints	sweep + trees	B
pairing	general matching	R, HMTT
domincance of trees	weighted matching	DKMNT
tour	bipartite weighted matching	FLM

Scheduling and Sortedness

- another example for the power of constraints
- schedule jobs of duration d_1,d_2, ... ,d_n on k machines
- variables and constraints (Older, Swinkels, van Emden)
  - s_i and $t_i = s_i + d_i$ starting and finishing times of job i
  - $s_i, t_i \in \{0, \ldots, D-1\}$
  - $\sigma_j = j$-th largest starting time: $(\sigma_1, \ldots, \sigma_n) = sort(s_1, \ldots, s_n)$
  - $\tau_j = j$-th largest finishing time: $(\tau_1, \ldots, \tau_n) = sort(t_1, \ldots, t_n)$
  - $\sigma_1 = \sigma_2 = \ldots = \sigma_k = 0$, $\sigma_{k+1} = \tau_1$, $\sigma_{k+2} = \tau_2$, ... , $\sigma_n = \tau_{n-k}$
- bound narrowing for sort-constraint: Guernalec/Colmerauer, M/Thiel

Research Strategies

- theoretical research is not enough
- cooperation with constraint programmers is vital
  - they have the problems and
  - they control the systems
  - we cooperate with
    * Oz/Mozart group in Saarbrücken: D. Duchier, A. Koller, T. Müller, J. Niehren, G. Smolka
    * N. Beldiceanu (SICS)
- you must provide implementations
  - we base our implementations on LEDA
  - experiment with them in Oz
  - make them general enough to be used in other systems

Summary and Further Work

- constraints are a rich and powerful specification language
- in comparison, integer linear programming is assembly language
- constraint programming is a rich source of algorithmic problems
- solutions have wide impact through constraint programming systems

- find solutions to the problems in Beldiceanu's list
- develop library of narrowing algorithms
- integrate CP and ILP

Thanks for your attention.