DM841

Constraint Programming

Filtering algorithms for global constraints

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

Declarative and Operational Semantic

- Declarative Semantic: specify **what** the constraint means. Evaluation criteria is expressivity.
- Operational Semantic: specify **how** the constraint is computed, i.e., is kept *consistent* with its declarative semantic. Evaluation criteria are efficiency and effectiveness.

Example

So far, we have defined only the Declarative Semantic of the `alldifferent` constraint, not its Operational Semantic.

Domain Consistency

Definition

A constraint C on the variables $x_1, \dots, x_r$ with respective domains $D_1, \dots, D_r$ is called domain consistent (or generalized/hyper-arc consistent) if for each variable x_i and each value $d_i \in D_i$ there exist compatible values in the domains of all the other variables of C, that is, there exists a tuple $(d_1, \dots, d_i, \dots, d_r) \in C$.

In other terms: If value v is in the domain of variable x, then there exists a solution to the constraint with value v assigned to variable x.

Examples: `alldifferent` (`distinct`), `knapsack`, ...

Definition

Filtering algorithm $\equiv$ reduction rule: reduce $D(x_i)$ for $1 \leq i \leq r$ such that it still contains all values that the variable can assume in a solution of C.

$D(x_i) \leftarrow D(x_i) \cap \{d_i \in D(x_i) | D(x_1 \times D(x_{i-1}) \times \{v_i\} \times D(x_{i+1}) \times \dots, D(x_r)\} \cap C \neq \emptyset\}$ Generic arc consistency algorithms are in $O(erd^r)$.

Consistency and Filtering Algorithms

- Different filtering algorithms, which must be able to:
  1. Check consistency of C w.r.t. the current variable domains
  2. Remove inconsistent values from the variable domains

- The stronger is the level of consistency, the higher is the complexity of the filtering algorithm: Different level of consistency (domain, bound(Z), bound(D), range, value):
  - complete filtering, optimal pruning, domain completeness $\equiv$ domain/arc consistency
  - partial filtering, bound completeness $\equiv$ bound relaxed completeness

... again the `alldifferent` case

There exist in literature several filtering algorithms for the `alldifferent` constraints.

Decomposition Approach

A decomposition of a global constraint C is a polynomial time transformation $\delta_k(\mathcal{P})$ replacing C by some new bounded arity constraint (and possibly new variables) while preserving the set of tuples allowed on $X(C)$.

Global Constraint Decomposition

Given any $\mathcal{P} = \langle X(C), \mathcal{D}, \mathcal{C} = \{C\}\rangle$, $\delta_k(\mathcal{P})$ is such that

- $X(C) \subseteq X_{\delta_k(\mathcal{P})}$
- for all $x_i \in X(C)$, $D(x_i) = D_{\delta_k(\mathcal{P})}(x_i)$
- for all $C_j \in C_{\delta_k(\mathcal{P})}$, $|X(C_j)| \leq k$ and
- $sol(\mathcal{P}) = \pi_{X(C)}(sol(\delta_k(\mathcal{P}))$

Example

`atmost`$(x_1, \ldots, x_n, p, v)$ (at most p variables in $x_1, \ldots, x_n$ take value v).
Decomposition: $n+1$ additional variables $y_0, \ldots, y_n$ $(x_i = v \wedge y_i = y_{i-1} + 1) \vee (x_i \neq v \wedge y_i = y_{i-1})$ for all i, $1 \leq i \leq n$, and domains $D(y_0) = \{0\}$ and $D(y_i) = \{0, \ldots, p\}$ for $1 \leq i \leq n$.

These decompositions can be:

- preserving solutions
- preserving generalized arc consistency
- preserving the complexity of enforcing generalized arc consistency

The decomposition of `atmost` preserves solutions and generalized arc consistency
For the `alldifferent` only preserving solutions. Yet sometimes it is possible to construct a specialized algorithm that enforces GAC in polynomial time.

alldifferent

alldifferent constraint

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

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

Complete Filtering for alldifferent

1. build value graph $G = (X, D(X), E)$
2. compute maximum matching M in G
3. **if** $|M| < |X|$ **then return false**
4. mark all arcs in oriented graph G_M that are not in M as unused
5. compute SCCs in G_M and mark all arcs in a SCC as used
6. perform breadth-first search in G_M starting from M-free vertices, and mark all traversed arcs as used if they belong to an even path
7. **for all** arcs (x_i, d) in G_M marked as unused **do**
   $D(x_i) := D(x_i) \setminus d$
   if $D(x_i) = \emptyset$ **then return false**
8. **return true**

Overall complexity: $O(n\sqrt{m} + (n + m) + m)$
It can be updated incrementally if other constraints remove some values.

Example

$\text{alldiff}(x_1, \ldots, x_5)$

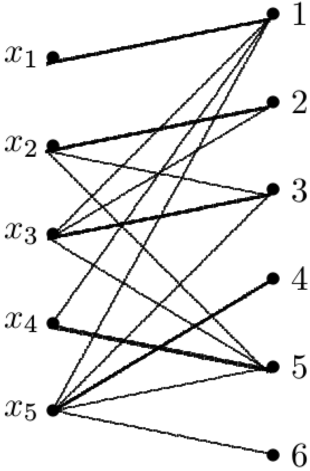
$D_{x_1} = \{1\}$

$D_{x_2} = \{2, 3, 5\}$

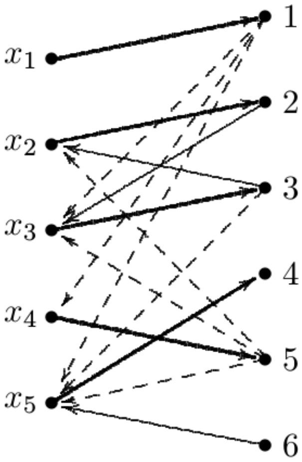
$D_{x_3} = \{1, 2, 3, 5\}$

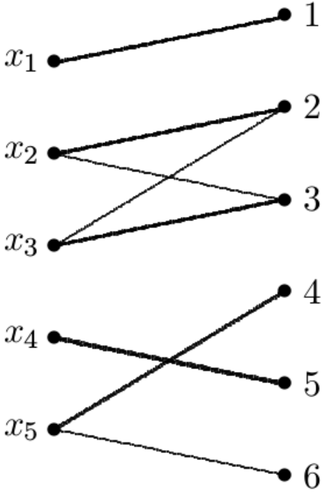
$D_{x_4} = \{1, 5\}$

$D_{x_5} = \{1, 3, 4, 5, 6\}$



Example





Relaxed Consistency

Definition

A constraint C on the variables $x_1, \ldots, x_m$ with respective domains $D_1, \ldots, D_m$ is called bound(Z) consistent if for each variable x_i and each value $d_i \in \{\min(D_i), \max(D_i)\}$ there exist compatible values between the min and max domain of all the other variables of C, that is, there exists a value $d_j \in [\min(D_j), \max(D_j)]$ for all $j \neq i$ such that $(d_1, \ldots, d_i, \ldots, d_k) \in C$.

Definition

A constraint C on the variables $x_1, \ldots, x_m$ with respective domains $D_1, \ldots, D_m$ is called range consistent if for each variable x_i and each value $d_i \in D_i$ there exist compatible values between the min and max domain of all the other variables of C, that is, there exists a value $d_j \in [\min(D_j), \max(D_j)]$ for all $j \neq i$ such that $(d_1, \ldots, d_i, \ldots, d_k) \in C$.

(bound(D) if its bounds belong to a support on C)

$GAC < (\text{bound(D)}, \text{range}) < \text{bound(Z)}$

Bound Consistency [Mehlorn&Thiel2000]

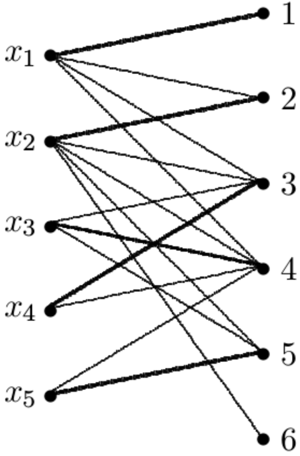
Definition (Convex Graph)

A bipartite graph $G = (X, Y, E)$ is convex if the vertices of Y can be assigned distinct integers from $[1, |Y|]$ such that for every vertex $x \in X$, the numbers assigned to its neighbors form a subinterval of $[1, |Y|]$.

In convex graph we can find a matching in linear time.

Example

$$\begin{aligned}
D_{x_1} &= \{1, 2, \quad 4 \quad\} & I_{x_1} &= \{1, 2, 3, 4 \quad\} \\
D_{x_2} &= \{\quad 2, 3, \quad 6\} & I_{x_2} &= \{\quad 2, 3, 4, 5, 6\} \\
D_{x_3} &= \{\quad 3, \quad 5 \quad\} & I_{x_3} &= \{\quad 3, 4, 5 \quad\} \\
D_{x_4} &= \{\quad 3, 4 \quad\} & I_{x_4} &= \{\quad 3, 4 \quad\} \\
D_{x_5} &= \{\quad 4, 5 \quad\} & I_{x_5} &= \{\quad 4, 5 \quad\}
\end{aligned}$$



Survey of complexity: effectiveness and efficiency

Consistency	Idea	Complexity	Amort.	Reference(s)
arc		$O(n^2)$		[VanHentenryck1989]
bound	Hall	$O(n \log n)$		[Puget1998]
	Flows			[Mehlhorn&Thiel2000]
	Hall			[Lopez&All2003]
		$O(n)$		[Mehlhorn&Thiel2000]
				[Lopez&All2003]
range	Hall	$O(n^2)$		[Leconte1996]
domain	Flows	$O(n\sqrt{m})$	$O(n\sqrt{k})$	[Régin1994],[Costa1994]

Where $n =$ number of variables, $m = \sum_{i \in 1 \dots n} |D_i|$, and $k =$ number of values removed.

Filtering `cardinality`

`cardinality` or `gcc` (global cardinality constraint)

Let $x_1, \ldots, x_n$ be assignment variables whose domains are contained in $\{v_1, \ldots, v_{n'}\}$ and let $\{c_{v_1}, \ldots, c_{v_{n'}}\}$ be count variables whose domains are sets of integers. Then

$$\texttt{cardinality}([x_1, ..., x_n],[c_{v_1}, ..., c_{v_{n'}}]) = \{(w_1, ..., w_n, o_1, ..., o_{n'}) \mid w_j \in D(x_j)\,\forall j, \texttt{occ}(v_i, (w_1, ..., w_n)) = o_i \in D(c_{v_i})\,\forall i\}.$$

(`occ` number of occurrences)

$\rightsquigarrow$ generalization of `alldifferent`
NP-hard to filter domain of all variables. But if constant intervals, then polynomial algorithm via network flows. (integral feasible (s, t)-flow)

Midterm Evaluation

https://padlet.com/marco43/dm841_2023

Reified Constraints

also called indicator constraints in mathematical programming

A reified constraint links a Boolean variable with the truth value of a constraint (0 is false and 1 is true).
For example, the reified constraint $\texttt{isEqual}(x, v, b)$ holds if $b \equiv (x = v)$.
In MiniZinc: `b <-> x=v` with `var bool: b`

Its implementation captures the following inference rules:

- when $D(b) = 1$ then add constraint $x = v$;
- when $D(b) = 0$ then add constraint $x \neq v$;
- when $D(x) = v$ then add constraint $b = 1$;
- when $v \notin D(x)$ then add constraint $b = 0$.

Note: the constraint can be deactivated whenever b or x is bound or whenever $v \notin D(x)$

Propagation

```
if  b = True then
    x.assign(v);
    dispose C;
else if  b = False then
    x.remove(v);
    dispose C;
else if  x is Fix then
    b = True;
    dispose C;
else if v ∉ D(x) then
    b = False;
    dispose C;
```

Propagation events:

- x propagate on domain change (`RemValue`)
- b propagate on bind (`Fix`)

In general:

Let $\overline{\gamma}(\ldots)$ denote the complement of $\gamma(\ldots)$ (not code for `not`$\gamma(\ldots)$, as CP solvers do not implement `not`):

- When b gets fixed to 1, post the constraint $\gamma(\ldots)$
- When b gets fixed to 0, post the constraint $\overline{\gamma}(\ldots)$.
- When $\gamma(\ldots)$ gets subsumed, post the constraint $b = 1$.
- When $\overline{\gamma}(\ldots)$ gets subsumed, post the constraint $b = 0$.

Propagation may be very poor! Reification may be hard for some predicates!

Constraint combination with reification

With reification, constraints can be arbitrarily combined with logical connectives: negation ($\neg$), disjunction ($\vee$), conjunction ($\&$), implication ($\implies$), and equivalence ($\iff$). However, propagation may be very poor!

Example

The composite constraint $(\gamma_1 \& \gamma_2) \vee \gamma_3$ is modelled as

$$(b_1 \iff \gamma_1) \; \& \; (b_2 \iff \gamma_2) \; \& \; (b_3 \iff \gamma_3) \; \& \; (b_1 \cdot b_2 = b) \; \& \; (b + b_3 \geq 1)$$

Hence, even the constraints γ_1 and γ_2 must be reified.

If γ_1 is $x = y + 1$ and γ_2 is $y = x + 1$, then $\gamma_1 \; \& \; \gamma_2$ is unsatisfiable;
however, b is then not fixed to 0 by propagation, as each propagator works individually and there is no communication through their shared variables x and y; hence $b_3 = 1$ is not propagated and γ_3 is not forced to hold.

The disjunction and negation of constraints (with
`\/`, **`xor`**, **`not`**, `<-`, `->`, `<->`, **`exists`**, **`xorall`**, **`if`** θ **`then`** ϕ_1 **`else`** ϕ_2 **`endif`**
in MiniZinc often makes the solving slow.

Example

The MiniZinc disjunctive constraint

```
constraint x = 0 \/ x = 9;
```

is modelled with reification:

$$(b0 \iff x = 0) \;\&\; (b9 \iff x = 9) \;\&\; (b0 + b9 \geq 1)$$

But it is logically equivalent to

```
constraint x in {0,9};
```

where no reification is involved and no further propagation is needed.

Conditionals **if** θ **then** ϕ_1 **else** ϕ_2 **endif**
or a comprehension, such as `[i | i in rho where theta]`, in MiniZinc with a test θ that depends on variables:

Example

Consider `var 1..9: x` and `var 1..9: y` for constraint

```
forall(i in 1..9 where i > x)(i > y)
```

This is a syntactic sugar for constraint

```
forall([i > y | i in 1..9 where i > x])
```

This is modelled with reification, as in constraint

```
forall(i in 1..9)(i > x -> i > y)
```

that is with a logical implication (->), hence with a hidden logical disjunction (\/): for each i, both sub-constraints are reified because both have variables

Filtering knapsack

Knapsack and Sum constraints (Linear constraints over integer variables)

Let $x_1, \ldots, x_n, z, c$ be integer variables:

$$\text{knapsack}([x_1, \ldots, x_n], z, c) =$$

$$\left\{ (d_1, \ldots, d_n, d) \mid d_i \in D(x_i) \, \forall i, d \in D(z), d \leq \sum_{i=1,\ldots,n} c_i d_i \right\} \cap$$

$$\left\{ (d_1, \ldots, d_n, d) \mid d_i \in D(x_i) \, \forall i, d \in D(z), d \geq \sum_{i=1,\ldots,n} c_i d_i \right\}.$$

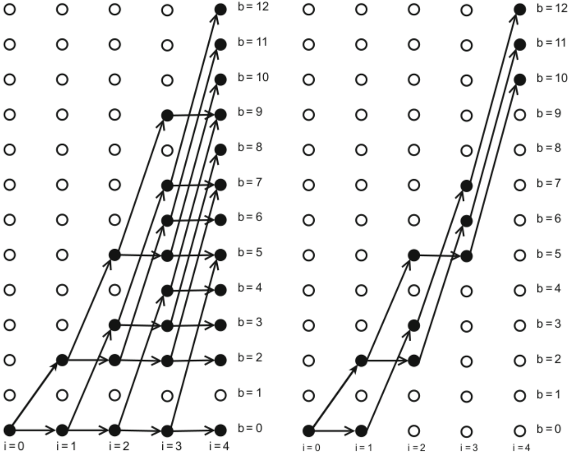
Binary Knapsack (Linear constraints over Boolean variables)

$\sum c_i x_i = z, x_i \in \{0, 1\} \rightsquigarrow l_z \leq \sum c_i x_i \leq u_z$

Variant of the subset sum problem: Given a set of numbers find a subset whose sum is 0.

Eg: $-7, -3, -2, 5, 8 \rightsquigarrow -3 - 2 + 5 = 0$

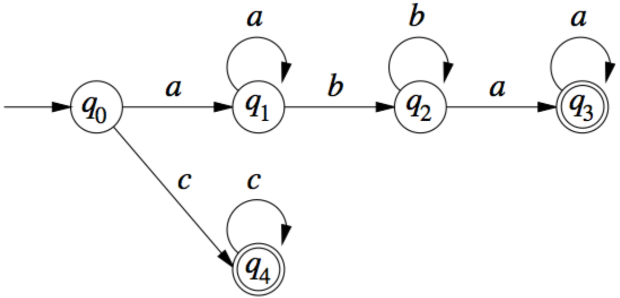
$10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$

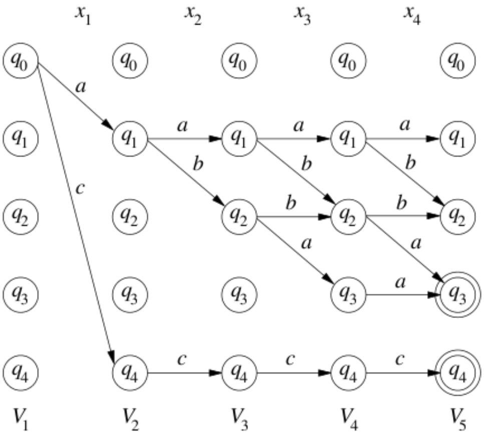


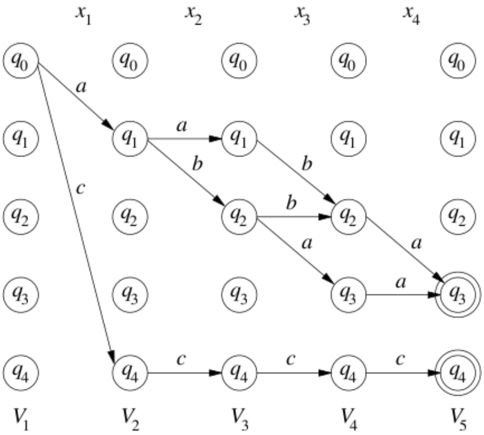
Filtering regular

"regular" constraint

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables with $D(x_i) \subseteq \Sigma$ for $1 \leq i \leq n$. Then $\texttt{regular}(X, M) = \{(d_1, ..., d_n) \mid \forall i, d_i \in D(x_i), [d_1, d_2, \ldots, d_n] \in L(M)\}$.




x1 x2 x3 x4
q0 q1 q2 q3 q4
a b c
V1 V2 V3 V4 V5


x1 x2 x3 x4
q0 q1 q2 q3 q4
a b c
V1 V2 V3 V4 V5

Other Filtering Algorithms

- `linear`
- `element`
- `disjunctive`
- `cumulative`

linear

$$\sum_{i=1}^{n} a_i x_i + b \begin{array}{c}\leq\\=\\\geq\end{array} 0 \qquad x_i \in [l_i, h_i]$$

Example

$$3x + 4y - 5z \leq 7$$

$$x \leq \frac{7 - 4y + 5z}{3} \quad \Longrightarrow \quad x \leq \left\lfloor \frac{7 - 4\ell_y + 5h_z}{3} \right\rfloor$$

$$[\ell_x, h_x] \longleftarrow \left[\ell_x, \min\left(h_x, \left\lfloor \frac{7 - 4\ell_y + 5h_z}{3} \right\rfloor\right)\right]$$

$$\sum_{i \in POS} a_i x_i - \sum_{i \in NEG} a_i x_i \leq b$$

$$x_{\leq} \frac{b - 4y + 5z}{3} \qquad \Longrightarrow \qquad x_j \leq \frac{b - \sum_{i \in POS \setminus \{j\}} a_i x_i + \sum_{i \in NEG} a_i x_i}{a_j}$$

$$\alpha_j = \frac{b - \sum_{i \in POS \setminus \{j\}} a_i l_i + \sum_{i \in NEG} a_i h_i}{a_j}$$

$$\beta_j = \frac{b - \sum_{i \in POS \setminus \{j\}} a_i h_i + \sum_{i \in NEG} a_i l_i}{a_j}$$

$$[l_j, h_j] \longleftarrow [\max(l_x, \lceil \beta_j \rceil), \min(h_j, \lfloor \alpha_j \rfloor)]$$

(domain consistency is NP-complete, this one is bound(Z))

element

- element$(y, z, \vec{a}) \equiv z = a_y$
- element$(y, z, \vec{x}) \equiv z = x_y$

Outline

Scheduling Constraints

One job at a time on a machine (disjunctive machines):

"disjunctive" scheduling

Let $(x_1, \ldots, x_n)$ be a tuple of (integer/real)-valued variables indicating the starting time of a job j.
Let $(p_1, \ldots, p_n)$ be the processing times of each job.

$$\texttt{disjunctive}([x_1, \ldots, x_n], [p_1, \ldots, p_n]) = \{[s_1, \ldots, s_n] \mid \forall i, j, i \neq j, \; (s_i + p_i \leq s_j) \vee (s_j + p_j \leq s_i)\}$$

In MiniZinc:

```
predicate disjunctive(array [int] of var int: s,
                      array [int] of var int: d);
```

In Gecode:

```
IntArgs p(4, 2,7,4,11);
unary(home, s, p);
```

Digression: Job Shop

General Shop Scheduling:

- $J = \{1, \ldots, N\}$ set of jobs; $M = \{1, 2, \ldots, m\}$ set of machines
- $J_j = \{O_{ij} \mid i = 1, \ldots, n_j\}$ set of operations for each job
- p_{ij} processing times of operations O_{ij}
- $\mu_{ij} \subseteq M$ machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time,
  one machine processing each job at a time
- C_j completion time of job j

$\Longrightarrow$ Find feasible schedule that minimize some regular function of C_j

Job shop

- $\mu_{ij} = \ell, \ell = 1, \ldots, n_j$ and $\mu_{ij} \neq \mu_{i+1,j}$ (one machine per operation)
- $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{n_j,j}$ precedences (without loss of generality)
- without repetition and with unlimited buffers

Task:

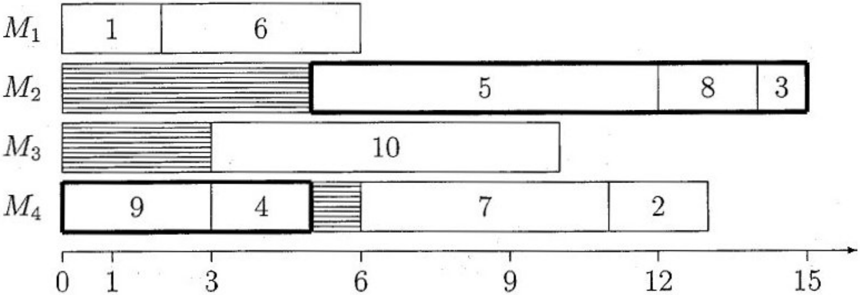
- Find a schedule $S = (x_{ij})$, indicating the starting times of O_{ij}, such that:

  it is feasible, that is,

  - $x_{ij} + p_{ij} \leq x_{i+1,j}$ for all $O_{ij} \to O_{i+1,j}$
  - $x_{ij} + p_{ij} \leq x_{uv}$ or $x_{uv} + p_{uv} \leq x_{ij}$ for all operations with $\mu_{ij} = \mu_{uv}$.

  and has minimum makespan: $\min\{\max_{j \in J}(x_{n_j,j} + p_{n_j,j})\}$.

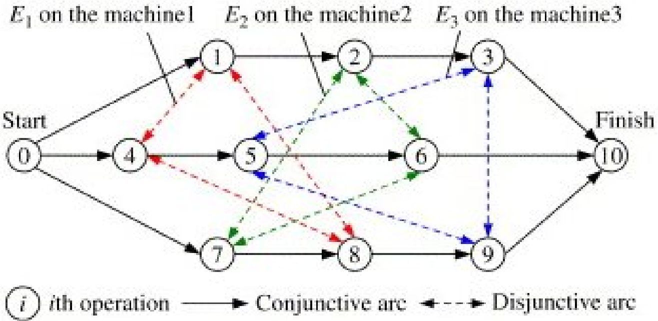
A schedule can also be represented by an m-tuple $\pi = (\pi^1, \pi^2, \ldots, \pi^m)$ where π^i defines the processing order on machine i.


M_1
1
6
M_2
5
8
3
M_3
10
M_4
9
4
7
2
0
1
3
6
9
12
15

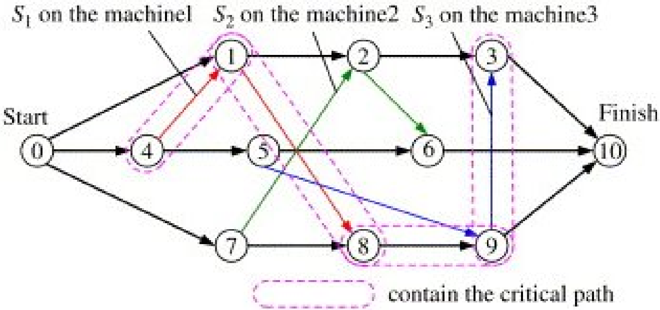
Representation via Disjunctive Graph

- Often simplified notation: $N = \{1, \ldots, n\}$ denotes the set of operations

- Disjunctive graph representation: $G = (N, A, E)$
  - vertices N: operations with two dummy operations 0 and $n + 1$ denoting "start" and "finish".
  - directed arcs A, conjunctions
  - undirected arcs E, disjunctions
  - length of (i, j) in A is p_i



- A complete selection corresponds to choosing one direction for each arc of E.
- A complete selection that makes D acyclic corresponds to a feasible schedule and is called consistent.
- Complete, consistent selection $\Leftrightarrow$ semi-active schedule (feasible earliest start schedule).
- Length of longest path 0–$(n+1)$ in D corresponds to the makespan



Longest path computation

In an acyclic digraph:

- construct topological ordering ($i < j$ for all $i \to j \in A$)
- recursion:

$$
\begin{aligned}
r_0 &= 0 \\
r_l &= \max_{\{j \mid j \to l \in A\}} \{r_j + p_j\} \qquad \text{for } l = 1, \ldots, n+1
\end{aligned}
$$

Back to Constraint Propagation: Edge Finding

Edge finding identifies jobs i that must occur first or last in a set $J \cup \{i\}$ of jobs.

$i \gg J$: indicates that i starts after every jobs in J has finished
$i \ll J$: indicates that i finishes before any jobs in J starts

$$\text{If } L_J - E_{J\cup\{i\}} < p_i + p_J, \text{ then } i \gg J \quad (a)$$
$$\text{If } L_{J\cup\{i\}} - E_J < p_i + p_J, \text{ then } i \ll J \quad (b)$$

$$\text{If } i \gg J, \text{ then update } E_i \text{ to } \max\left\{E_i, \max_{J'\subset J}\{E_{J'} + p_{J'}\}\right\}.$$

$$\text{If } i \ll J, \text{ then update } L_i \text{ to } \min\left\{L_i, \min_{J'\subset J}\{L_{J'} - p_{J'}\}\right\}.$$

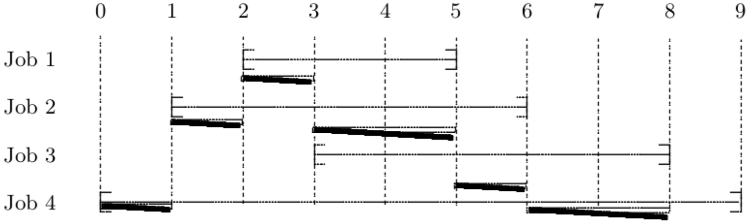
Example:

j	p_j	E_j	L_j
1	1	2	5
2	3	1	6
3	1	3	8
4	3	0	9

$4 \gg \{1, 2\}$

An $O(n^2)$ algorithm for bound propagation

j	p_j	E_j	L_j
1	1	2	5
2	3	1	6
3	1	3	8
4	3	0	9



i	J_i	$\bar{p}$	k	J_{ik}	$L_k - E_i$	$p_i + \bar{p}_{J_{ik}}$
1	$\{1,2,3,4\}$	$(1,2,1,2)$	4	$\{2,3,4\}$	$9-2$	$1+5$
			3	$\{2,3\}$	$8-2$	$1+3$
			2	$\{2\}$	$6-2$	$1+3$
2	$\{1,2,3,4\}$	$(1,3,1,2)$	4	$\{1,3,4\}$	$9-1$	$3+4$
			3	$\{1,3\}$	$8-1$	$3+2$
			1	$\{3\}$	$5-1$	$3+1$
3	$\{2,3,4\}$	$(0,2,1,2)$	4	$\{2,4\}$	$9-3$	$1+4$
			2	$\{2\}$	$6-3$	$1+2$
4	$\{1,2,3,4\}$	$(1,3,1,3)$	3	$\{1,2,3\}$	$8-0$	$3+5$
			2	$\{1,2\}$	$6-0$	$3+4$

Conclude that $4 \gg \{1,2\}$ and update E_4 from 0 to 5

- compute the Jackson preemptive schedule (JPS) for the given instance.
  Moving forward in time, the job in process at each time t is the job j that has the smallest L_j among the jobs available at t (ie, any job ℓ with $t \in [E_\ell, L_\ell]$ and not finished).

- for each job i, do the following.
  - J_i be the set of jobs that are not finished at time E_i in the JPS.
  - $\bar{p}_j$ be the processing time left for job j at time E_i in the JPS.
  - J_{ik} be the jobs in J_i, other than i

  that have deadlines at or before job k's deadline:

  $$J_{ik} = \{j \in J_i \setminus \{i\} \mid L_j \leq L_k\}$$

  Examine the jobs $k \in J_i (k \neq i)$ in decreasing order of deadline L_k, and select the first job for which

  $$L_k - E_i < p_i + \bar{p}_{J_{ik}}$$

  Then conclude that $i \gg J_{ik}$ and update E_i to $JPS(i, k)$, which is the latest completion time in the JPS of the jobs in J_{ik}

Not first, Not Last

A complementary type of rule identifies jobs i that cannot occur first or cannot occur last in $J \cup \{i\}$:

$$\text{If } L_J - E_i < p_i + p_J, \text{ then } \neg(i \ll J). \quad (a)$$
$$\text{If } L_i - E_J < p_i + p_J, \text{ then } \neg(i \gg J). \quad (b)$$

$$\text{If } \neg(i \ll J), \text{ then update } E_i \text{ to } \max\left\{E_i, \min_{j \in J}\{E_j + p_j\}\right\} \quad (a)$$
$$\text{If } \neg(i \gg J), \text{ then update } L_i \text{ to } \min\left\{L_i, \max_{j \in J}\{L_j - p_j\}\right\} \quad (b)$$

Exercise: verify the following deductions and compare the domain pruning wrt edge finding.

$\neg(4 \ll \{1, 2\})$
$\neg(3 \ll \{2\})$
$\neg(2 \gg \{3\})$

Scheduling Constraints: Cumulative

In Resource Constrained Project Scheduling each resource can be used at most up to its capacity:

`cumulative` constraints [Aggoun and Beldiceanu, 1993]

- p_j processing time
- $x_j \in [E_j, L_j - p_j]$, starting time of each job, E_j release time of job j, L_j deadline of job j
- c_j resource consumption
- C limit not to be exceeded at any point in time

$$\texttt{cumulative}([x_j], [p_j], [c_j], C) := \{([s_j], [p_j], [c_j], C) \mid \forall t \sum_{i \mid s_i \leq t \leq s_i + p_i} c_i \leq C\}$$

With $c_j = 1$ forall j and $C = 1 \rightsquigarrow$ `disjunctive`

Scheduling Constraints: Cumulative

The cumulative constraint is used in scheduling problems for describing cumulative resource usage.

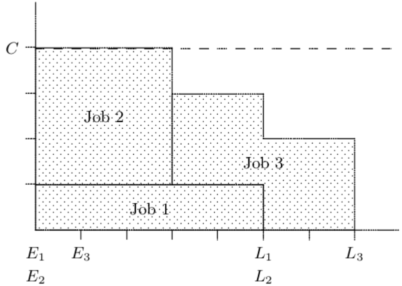
```
cumulative(array[int] of var int: s,
           array[int] of var int: d,
           array[int] of var int: r,
           var int: b)
```

A set of tasks with start times s, durations d, and resource requirements r,
must never require more than a global resource bound b at any one time.

Cumulative Scheduling: Example

j	p_j	c_j	E_j	L_j
1	5	1	0	5
2	3	3	0	5
3	4	2	1	7

$C = 4$



Verify that $3 > \{1, 2\}$ and update the domain of x_3

Cumulative Scheduling: Propagation

If all the problem data are integral, E_j should be rounded up and L_j rounded down.

The most important filtering methods are generalizations of those seen via the concept of energy of a job j: $e_j = c_j p_j$

- edge finding: checks whether the total energy demand exceeds the supply, which is the product of the time interval and the resource limit C. It can be done in $O(n^3)$
- extended edge finding
- not-first/not-last rules
- energetic reasoning

Edge Finding

It deduces that one job must finish after certain others finish (weaker than the previous: must start after others finish)

$i > J$: i must finish after all in J finished
$i < J$: i must start before any in J starts

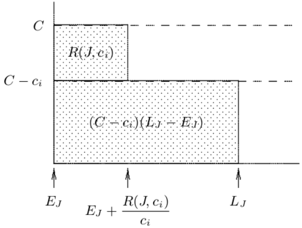
if J requires total energy e_J, then the time interval $[t_1, t_2]$ in which they are scheduled must have a length of at least e_J/C.

$$\text{If } e_i + e_J > C \cdot (L_J - E_{J\cup\{i\}}), \text{ then } i > J. \quad (a)$$
$$\text{If } e_i + e_J > C \cdot (L_{J\cup\{i\}} - E_J), \text{ then } i < J. \quad (b)$$

Rule (a): if i does not finish after all the jobs in J finished, then the time interval from $E_{J\cup\{i\}}$ to L_J must cover the energy demand $e_i + e_J$ of all the jobs.

Edge Finding: Domain Update

If $e_i + e_J > C \cdot (L_J - E_{J\cup\{i\}})$, then $i > J$. $\quad (a)$
If $e_i + e_J > C \cdot (L_{J\cup\{i\}} - E_J)$, then $i < J$. $\quad (b)$



If $i > J$ and $R(J, c_i) > 0$, update E_i to $\max\left\{E_i, E_J + \dfrac{R(J, c_i)}{c_i}\right\}$.

If $i < J$ and $R(J, c_i) > 0$, update L_i to $\min\left\{L_i, L_J - \dfrac{R(J, c_i)}{c_i}\right\}$.

If e_J exceeds the energy available between E_J and L_J within a resource limit of $C - c_i$, then at some time in the schedule the jobs in J must consume more resource than $C - c_i$.

Since the excess energy is

$$R(J, c_i) = e_J - (C - c_i)(L_J - E_J)$$

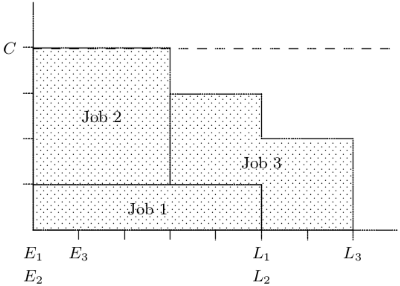
the jobs in J must consume more resource than $C - c_i$ for a period of at least $R(J, c_i)/c_i$.

None of this excess resource is consumed after job i finishes, because $i > J$, so it must be consumed before job i starts

Cumulative Scheduling: Exercise

j	p_j	c_j	E_j	L_j
1	5	1	0	5
2	3	3	0	5
3	4	2	1	7

$C = 4$



Verify that $3 > \{1, 2\}$ and update the domain of x_3

Filtering Algorithm Design

1. Filtering algorithms based on a generic algorithm
   Simple AC algorithms. Eg, `element`:
2. Filtering algorithms based on existing algorithms
   Reuse existing algorithms for filtering (e.g., flows algorithms, dynamic programming).
3. Filtering algorithms based on ad-hoc algorithms
   Pay particular attention to incrementality and amortized complexity
4. Filtering algorithms based on model reformulation
   See the Constraint Decomposition approach

Outline

Soft Constraints

Soft constraint

A *soft constraint* is a constraint that may be violated. We measure the violation of each constraint, and the goal is to minimize the total amount of violation of all soft-constraints.

Definition

A *violation measure* for a soft-constraint $C(x_1, \ldots, x_n)$ is a function

$$\mu : D(x_1) \times \cdots \times D(x_n) \to \mathbb{Q}.$$

This measure is represented by a *cost* variable z.

Violation measures

- The variable-based violation measure μ_{var} counts the minimum number of variables that need to change their value in order to satisfy the constraint.
- The decomposition-based violation measure μ_{dec} counts the number of constraints in the binary decomposition that are violated.

The soft-alldifferent

Definition

Let $x_1, x_2, ..., x_n, z$ be variables with respective finite domains $D(x_1), D(x_2), ..., D(x_n), D(z)$. Let μ be a violation measure for the alldifferent constraint. Then

$$\text{soft-alldifferent}(x_1, ..., x_n, z, \mu) = \\ \{(d_1, ..., d_n, d) \mid \forall i. d_i \in D(x_i), d \in D(z), \mu(d_1, ..., d_n) \leq d\}$$

is the soft alldifferent constraint with respect to μ.

The soft-alldifferent: an example

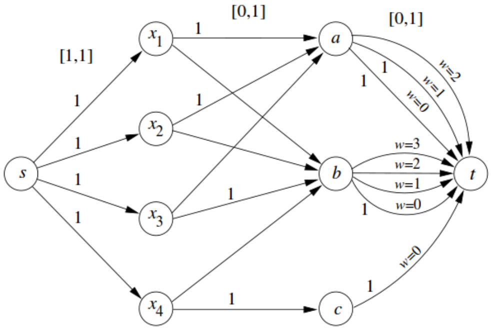
Example

Consider the following CSP

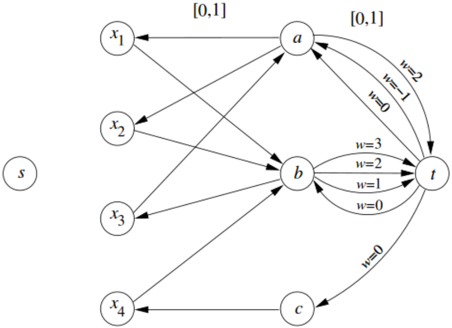
$$\begin{array}{l} x_1 \in \{a, b\}, x_2 \in \{a, b\}, x_3 \in \{a, b\}, x_4 \in \{a, b, c\}, z \in \mathbb{Z}^+ \\ \text{soft-alldifferent}(x_1, x_2, x_3, x_4, \mu, z) \\ \min z \end{array}$$

We have for instance $\mu_{var}(b, b, b, b) = 3$ and $\mu_{dec}(b, b, b, b) = 6$.

Filtering of `soft-alldiff`



Flow network and feasible flow



Residual graph

Outline

Optimization Constraints

Optimization Constraint bring the costs of variable-value pair into the declarative semantic of the constraints.

The filtering does take into account the cost, and a tuple may be inconsistent because it does not lead to a solution of “at least” a given cost.
Basic approach, solve a sequence of decision problems, allows one-way inference.
More powerful approach takes into account two-way inference.

gcc with costs

`cardinality` or `cost_gcc` (global cardinality constraint with costs)

Let $x_1, \ldots, x_n$ be assignment variables whose domains are contained in $\{v_1, \ldots, v_{n'}\}$ and let $\{c_{v_1}, \ldots, c_{v_{n'}}\}$ be count variables whose domains are sets of integers and $w(x, d) \in Q$ are costs. Then

$$\begin{aligned}\texttt{cost\_gcc}([x_1, ..., x_n], [c_{v_1}, ..., c_{v_{n'}}], z, w) = \\ &\{(d_1, ..., d_n, o_1, ..., o_{n'}) \mid \\ &\{(d_1, ..., d_n, o_1, ..., o_{n'}) \in \texttt{gcc}(([x_1, ..., x_n], [c_{v_1}, ..., c_{v_{n'}}]), \\ &\forall d_j \in D(x_j)\; d \in D(z) \sum_i w(x_i, d_i) \le d\}.\end{aligned}$$

Filtering for `cost_gcc`

(works on constant intervals)
Extend the (s,t)-network saw for `gcc` by weigths $w(x_i, v_i)\,\forall v_i$

1. compute initial min-cost feasible (s,t)-flow, f. $O(n(m + n\log n)$
2. For an arc uv with $f(a) = 0$ compute min cost directed path P from v to u in the residual graph. $P + a$ is a directed circuit.
3. since f is integer we can rerout one unit in the circuit and obtain: $cost(f') = cost(f) + cost(P)$.
4. if $cost(f') > max(D(z))$ remove v from $D(x_i)$

2.-4. in $O(\Delta(m + n\log n))$ where $\Delta = \min(n, |D(X)|)$.

Reduced-Cost Based Filtering [Focacci et al 1999]

Definition

Let $X = \{x_1, ..., x_n\}$ be a set of variables with corresponding finite domains $D(x_1), ..., D(x_n)$. We assume that each pair (x_i, j) with $j \in D(x_i)$ induces a cost c_{ij}.
We extend any global constraint C on X to an optimization constraint opt_C by introducing a cost variable z (that we wish to minimize) and defining

$$\text{opt\_C}(x_1, ..., x_n, z, c) = \{(d_1, ..., d_n, d) | (d_1, ..., d_n) \in C(x_1, ..., x_n),$$

$$\forall i . d_i \in D(x_i), d \in D(z), \sum_{i=1,...,n} c_{id_i} \leq d\}.$$

Linear Relaxation

We introduce binary variables y_{ij} for all $i \in \{1, ..., n\}$ and $j \in D(x_i)$, such that

$$
\begin{aligned}
& x_i = j \Leftrightarrow y_{ij} = 1, && \forall i = 1, \dots, n, \ \forall j \in D(x_i), \\
& x_i \neq j \Leftrightarrow y_{ij} = 0, && \forall i = 1, \dots, n, \ \forall j \in D(x_i) \\
& \sum_{j \in D(x_i)} y_{ij} = 1, && \forall i = 1, \dots, n.
\end{aligned}
$$

$+$ constraint dependent linear inequalities

The reduced-costs are given w.r.t. the objective:

$$\sum_{i=1,\dots,n} \sum_{j \in D(x_i)} c_{ij} y_{ij}$$

Example

`alldiff`

$$\begin{array}{ll} \min & \sum_{i,j} c_{i,j} y_{i,j} \\ & \sum_{j \in D(x_i)} y_{ij} = 1, \quad \forall i = 1, \ldots, n \\ & \sum_{i=1,\ldots,n} y_{ij} \leq 1, \quad \forall j \in D(x_i) \\ & y_{ij} \geq 0 \end{array}$$

Filtering by Reduced-Cost (aka “variable fixing”)

Recall that reduced-costs estimate the increase of the objective function when we force a variable into the solution.

Let $\bar{c}_{ij}$ be the reduced cost for the variable-value pair $x_i = j$, and let z^* be the optimal value of the current linear relaxation.

Reduced-costs estimate the increase of the objective function when we force a variable into the solution. Hence, $z^* + \bar{c}_{ij}$ is the increase of adding $y_{ij} = 1$ in the solution. We apply the following filtering rule:

$$\textbf{if } z^* + \bar{c}_{ij} > \max D(z) \textbf{ then } D(x_i) \leftarrow D(x_i) \setminus \{j\}.$$

References

Hooker J.N. (2007). **Integrated Methods for Optimization**, vol. 100 of **International Series in Operations Research & Management Science**. Springer.

van Hoeve W. and Katriel I. (2006). **Global constraints**. In *Handbook of Constraint Programming*, chap. 6. Elsevier.

Algorithms from the paper discussed at the blackboard