Outline

1. Global Constraints
   Scheduling

2. Soft Constraints

3. Optimization Constraints
Outline

1. Global Constraints
   Scheduling

2. Soft Constraints

3. Optimization Constraints
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**.
Definition

A constraint $C$ on the variables $x_1, \ldots, x_r$ with respective domains $D_1, \ldots, D_r$ 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 $(d_1, \ldots, d_i, \ldots, d_r) \in C$.

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 $C$.

$$D(x_i) \leftarrow D(x_i) \cap \{d_i \in D(x_i) | D(x_1 \times D(x_{i-1}) \times \{v_i\} \times D(x_{i+1}) \times \ldots, D(x_r) \} \cap C \neq \emptyset\}$$

Generic arc consistency algorithms are in $O(\text{erd}^r)$. 
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
- $\text{sol}(\mathcal{P}) = \pi_{X(C)}(\text{sol}(\delta_k(\mathcal{P})))$

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 $y_0, \ldots, y_n$

$(x_i = v \land y_i = y_{i-1} + 1) \lor (x_i \neq v \land y_i = y_{i-1})$ 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.
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.
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 $d_j \in [\min(D_i), \max(D_i)]$ 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 $d_j \in [\min(D_i), \max(D_i)]$ 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 $C$)

$GAC < (\text{bound}(D), \text{range}) < \text{bound}(Z)$
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.
Survey of complexity: effectiveness and efficiency

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Idea</th>
<th>Complexity</th>
<th>Amort.</th>
<th>Reference(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>arc</td>
<td>Hall</td>
<td>$O(n^2)$</td>
<td></td>
<td>[VanHentenryck1989]</td>
</tr>
<tr>
<td></td>
<td>Flows</td>
<td>$O(n \log n)$</td>
<td></td>
<td>[Puget1998]</td>
</tr>
<tr>
<td></td>
<td>Hall</td>
<td>$O(n)$</td>
<td></td>
<td>[Mehlhorn&amp;Thiel2000]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[Lopez&amp;All2003]</td>
</tr>
<tr>
<td>bound</td>
<td>Hall</td>
<td>$O(n^2)$</td>
<td></td>
<td>[Mehlhorn&amp;Thiel2000]</td>
</tr>
<tr>
<td></td>
<td>Flows</td>
<td>$O(n \sqrt{m})$</td>
<td>$O(n \sqrt{k})$</td>
<td>[Régin1994],[Costa1994]</td>
</tr>
<tr>
<td>range</td>
<td>Hall</td>
<td>$O(n^2)$</td>
<td></td>
<td>[Leconte1996]</td>
</tr>
<tr>
<td>domain</td>
<td>Flows</td>
<td>$O(n \sqrt{m})$</td>
<td>$O(n \sqrt{k})$</td>
<td>[Régin1994],[Costa1994]</td>
</tr>
</tbody>
</table>

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

$$\text{cardinality}([x_1, \ldots, x_n],[c_{v_1}, \ldots, c_{v_n'}]) =$$
$$\{(w_1, \ldots, w_n, o_1, \ldots, o_n') \mid w_j \in D(x_j) \forall j,$$
$$\text{occ}(v_i,(w_1, \ldots, w_n)) = o_i \in D(c_{v_i}) \forall i\}.$$ 

($\text{occ}$ number of occurrences)

~~ 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)
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)

$$\sum c_i x_i = z, x_i \in \{0, 1\} \; \Rightarrow \; l_z \leq \sum c_i x_i \leq u_z$$
Variant of the subset sum problem: Given a set of numbers find a subset whose sum is 0.

Eg: \(-7, -3, -2, 5, 8 \mapsto -3 - 2 + 5 = 0\)

\[10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12\]
"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

$\text{regular}(X, M) = \{(d_1, \ldots, d_n) \mid \forall i, d_i \in D(x_i), [d_1, d_2, \ldots, d_n] \in L(M)\}$. 
Other Filtering Algorithms

- linear
- element
- disjunctive
- cumulative
Global Constraints

Soft Constraints

Optimization Constraints

linear

\[ \sum_{i=1}^{n} a_i x_i + b \geq 0 \quad x_i \in [l_i, h_i] \]

Example

\[ 3x + 4y - 5z \leq 7 \]

\[ x \leq \frac{7 - 4y + 5z}{3} \quad \Rightarrow \quad x \leq \left\lfloor \frac{7 - 4l_y + 5h_z}{3} \right\rfloor \]

\[ [l_x, h_x] \leftarrow \left[ l_x, \min(h_x, \left\lfloor \frac{7 - 4l_y + 5h_z}{3} \right\rfloor ) \right] \]
\[
\sum_{i \in \text{POS}} a_i x_i - \sum_{i \in \text{NEG}} a_i x_i \leq b
\]

\[
x \leq \frac{b - 4y + 5z}{3} \implies x_j \leq \frac{b - \sum_{i \in \text{POS}\{j\}} a_i x_i + \sum_{i \in \text{NEG}} a_i x_i}{a_j}
\]

\[
\alpha_j = \frac{b - \sum_{i \in \text{POS}\{j\}} a_i l_i + \sum_{i \in \text{NEG}} a_i h_i}{a_j}
\]

\[
\beta_j = \frac{b - \sum_{i \in \text{POS}\{j\}} a_i h_i + \sum_{i \in \text{NEG}} a_i l_i}{a_j}
\]

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

(domain consistency is NP-complete, this one is bound(Z))
\[\text{element}(y, \vec{a}, z) \equiv z = a_y\]

\[D(z) \leftarrow D(z) \cap \{a_i \mid i \in D(y)\}\]
\[D(y) \leftarrow \{i \in D(y) \mid a_i \in D(z)\}\]

\[\text{element}(y, \vec{x}, z) \equiv z = x_y\]

\[D(z) \leftarrow D(z) \cap \bigcup_{i \in D(y)} D_{x_i}\]
\[D(y) \leftarrow \{i \in D(y) \mid D(z) \cap D_{x_i} = \emptyset\}\]
\[D(x_i) \leftarrow \begin{cases} D(z) & \text{if } D(y) = \{i\} \\ D(x_i) & \text{else} \end{cases}\]
Outline

1. Global Constraints
   Scheduling

2. Soft Constraints

3. Optimization Constraints
Edge Finding

If $L_J - E_{J \cup \{i\}} < p_i + p_J$, then $i \gg J$ (a)

If $L_{J \cup \{i\}} - E_J < p_i + p_J$, then $i \ll J$ (b)

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

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

<table>
<thead>
<tr>
<th>$j$</th>
<th>$p_j$</th>
<th>$E_j$</th>
<th>$L_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>
$O(n^2)$ algorithm

<table>
<thead>
<tr>
<th>$j$</th>
<th>$p_j$</th>
<th>$E_j$</th>
<th>$L_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

Job 1

Job 2

Job 3

Job 4

<table>
<thead>
<tr>
<th>$i$</th>
<th>$J_i$</th>
<th>$\bar{p}$</th>
<th>$k$</th>
<th>$J_{ik}$</th>
<th>$L_k - E_i$</th>
<th>$p_i + \bar{p}J_{ik}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2, 3, 4}</td>
<td>(1, 2, 1, 2)</td>
<td>4</td>
<td>{2, 3, 4}</td>
<td>9 - 2</td>
<td>1 + 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>{2, 3}</td>
<td>8 - 2</td>
<td>1 + 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>{2}</td>
<td>6 - 2</td>
<td>1 + 3</td>
</tr>
<tr>
<td>2</td>
<td>{1, 2, 3, 4}</td>
<td>(1, 3, 1, 2)</td>
<td>4</td>
<td>{1, 3, 4}</td>
<td>9 - 1</td>
<td>3 + 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>{1, 3}</td>
<td>8 - 1</td>
<td>3 + 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>{3}</td>
<td>5 - 1</td>
<td>3 + 1</td>
</tr>
<tr>
<td>3</td>
<td>{2, 3, 4}</td>
<td>(0, 2, 1, 2)</td>
<td>4</td>
<td>{2, 4}</td>
<td>9 - 3</td>
<td>1 + 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>{2}</td>
<td>6 - 3</td>
<td>1 + 2</td>
</tr>
<tr>
<td>4</td>
<td>{1, 2, 3, 4}</td>
<td>(1, 3, 1, 3)</td>
<td>3</td>
<td>{1, 2, 3}</td>
<td>8 - 0</td>
<td>3 + 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>{1, 2}</td>
<td>6 - 0</td>
<td>3 + 4</td>
</tr>
</tbody>
</table>

Conclude that $4 \gg \{1, 2\}$ and update $E_4$ from 0 to 5
Not first, Not Last

If $L_J - E_i < p_i + p_J$, then $\neg(i \ll J)$. (a)

If $L_i - E_J < p_i + p_J$, then $\neg(i \gg J)$. (b)

If $\neg(i \ll J)$, then update $E_i$ to max \[ E_i, \min_{j \in J} \{ E_j + p_j \} \] \quad (a)

If $\neg(i \gg J)$, then update $L_i$ to min \[ L_i, \max_{j \in J} \{ L_j - p_j \} \] \quad (b)
Cumulative Scheduling

<table>
<thead>
<tr>
<th>j</th>
<th>p_j</th>
<th>c_j</th>
<th>E_j</th>
<th>L_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
Edge Finding

If \( e_i + e_J > C \cdot (L_J - E_{J \cup \{i\}}) \), then \( i > J \). \( (a) \)

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

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

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

1. Filtering algorithms based on a generic algorithm
   Simple AC algorithms. Eg, element:
   \[ \text{element}(y, [2, 4, 8, 16, 32], x), x \in \{1, 2, 3, 4, 5\} \]

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

1. Global Constraints
   Scheduling

2. Soft Constraints

3. Optimization Constraints
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

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

This measure is represented by a *cost* variable $z$. 
Violation measures

- The **variable-based violation** measure $\mu_{\text{var}}$ counts the minimum number of variables that need to change their value in order to satisfy the constraint.

- The **decomposition-based violation** measure $\mu_{\text{dec}}$ 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 $\mu$ 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 $\mu$. 
Example
Consider the following CSP

\[ x_1 \in \{ a, b \}, \ x_2 \in \{ a, b \}, \ x_3 \in \{ a, b \}, \ x_4 \in \{ a, b, c \}, \ z \in \mathbb{Z}^+ \]
\[
\text{soft-alldifferent}(x_1, x_2, x_3, x_4, \mu, z) \\
\text{min } z
\]

We have for instance \( \mu_{\text{var}}(b, b, b, b) = 3 \) and \( \mu_{\text{dec}}(b, b, b, b) = 6 \).
Filtering of soft-alldiff

Flow network and feasible flow

Residual graph
Outline

1. Global Constraints
   Scheduling

2. Soft Constraints

3. Optimization Constraints
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.
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

$$\text{cost}_\text{gcc}([x_1, \ldots, x_n], [c_{v_1}, \ldots, c_{v_{n'}}], z, w) =$$
$$\{(d_1, \ldots, d_n, o_1, \ldots, o_{n'}) |$$
$$\{(d_1, \ldots, d_n, o_1, \ldots, o_{n'}) \in \text{gcc}(([x_1, \ldots, x_n], [c_{v_1}, \ldots, c_{v_{n'}}]),)$$
$$\forall d_j \in D(x_j) d \in D(z) \sum_i w(x_i, d_i) \leq d\}.$$
Filtering for \texttt{cost\_gcc}

(work on constant intervals)
Extend the \((s, t)\)-network saw for \tt{gcc} by weights \(w(x_i, v_i) \forall v_i\)

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:
   \[\text{cost}(f') = \text{cost}(f) + \text{cost}(P)\]

4. if \(\text{cost}(f') > \max(D(z))\) remove \(v\) from \(D(x_i)\)

2.-4. in \(O(\Delta(m + n \log n))\)
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 $\text{opt}_C$ by introducing a cost variable $z$ (that we wish to minimize) and defining

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

$$\forall i. d_i \in D(x_i), d \in D(z), \sum_{i=1, ..., n} c_{id_i} \leq d\}.$$
We introduce binary variables \( y_{ij} \) for all \( i \in \{1, \ldots, n\} \) and \( j \in D(x_i) \), such that
\[
\begin{align*}
  x_i = j &\iff y_{ij} = 1, \\
  x_i \neq j &\iff y_{ij} = 0,
\end{align*}
\]
\[\forall i = 1, \ldots, n, \forall j \in D(x_i),\]
\[\sum_{j \in D(x_i)} y_{ij} = 1, \quad \forall i = 1, \ldots, n.\]

+ constraint dependent linear inequalities

The reduced-costs are given w.r.t. the objective:
\[
\sum_{i=1, \ldots, n} \sum_{j \in D(x_i)} c_{ij} y_{ij}
\]
Example

**alldiff**

\[
\text{min } \sum_{i,j} c_{i,j} y_{i,j} \\
\sum_{j \in D(x_i)} y_{ij} = 1, \quad \forall i = 1, \ldots, n \\
\sum_{i=1,\ldots,n} y_{ij} \leq 1, \quad \forall j \in D(x_i) \\
y_{ij} \geq 0
\]
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.

We apply the following filtering rule:

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

Algorithms from the paper discussed at the blackboard