Constraint Relationships in MiniZinc

Case Studies

Universität Augsburg
University

Preferences in Constraint Solving

Constraint problem (X, D, C)

- Variables X, Domains $D = (D_x)_{x \in X}$, Constraints C

How to deal with over-constrained problems?

$((\{x, y, z\}, D_x = D_y = D_z = \{1, 2, 3\}), \{c_1, c_2, c_3\})$ mit

$$c_1 : x + 1 = y$$
$$c_2 : z = y + 2$$
$$c_3 : x + y \leq 3$$

- Not all constraints can be satisfied simultaneously

Preferences in Constraint Solving

Institute for Software & Systems Engineering

Constraint problem (X, D, C)

- Variables X, Domains $D = (D_x)_{x \in X}$, Constraints C

How to deal with over-constrained problems?

$((\{\mathrm{x}, \mathrm{y}, \mathrm{z}\}, \mathrm{D_x} = \mathrm{D_y} = \mathrm{D_z} = \{1, 2, 3\}), \{\mathrm{c_1}, \mathrm{c_2}, \mathrm{c_3}\})$ mit

$$\mathrm{c_1} : \mathrm{x} + 1 = \mathrm{y}$$
$$\mathrm{c_2} : \mathrm{z} = \mathrm{y} + 2$$
$$\mathrm{c_3} : \mathrm{x} + \mathrm{y} \leq 3$$

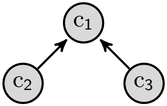
- Not all constraints can be satisfied simultaneously
  - e. g., $\mathrm{c_2}$ forces $\mathrm{z} = 3$ and $\mathrm{y} = 1$, conflicting $\mathrm{c_1}$
- We can choose between assignments satisfying $\{\mathrm{c_1}, \mathrm{c_3}\}$ or $\{\mathrm{c_2}, \mathrm{c_3}\}$.

Which assignments $v \in [X \to D]$ should be preferred by an agent/several agents?

Constraint Relationships

Approach (**?**)

- Define relation R over constraints C to denote which constraints are more important than others, e. g.
  - c_1 is more important than c_2
  - c_1 is more important than c_3



Benefits

- Qualitative formalism — easy to specify
- Graphical interpretation
  - Semantics (how much more important is a constraint) regulated by
  - dominance properties that are either “hierarchical” or “egalitarian”
  - Single-Predecessors-Dominance (SPD) vs. Transitive-Predecessors-Dominance (TPD)

SoftConstraints in MiniZinc

```
% X: {x,y,z} D_i = {1,2,3}, i in X
%    * c1: x + 1 = y * c2: z = y + 2 * c3: x + y <= 3
% (c) ISSE
% isse.uni-augsburg.de/en/software/constraint-relationships/
include "soft_constraints/minizinc_bundle.mzn";

var 1..3: x; var 1..3: y; var 1..3: z;

% read as "soft constraint c1 is satisfied iff x + 1 = y"
constraint x + 1 = y <-> satisfied[1];
constraint z = y + 2 <-> satisfied[2];
constraint x + y <= 3 <-> satisfied[3];

% soft constraint specific for this model
nScs = 3; nCrEdges = 2;
crEdges = [| 2, 1 | 3, 1 |]; % read c2 is less important than c1

solve minimize penSum; % minimize the sum of penalties
```

Case Studies

Institute for
Software & Systems
Engineering

Applied to domains where

- Certain properties should really capture preferences, not constraints
- at design time, it is unclear whether an instance is actually solvable
- Solution space is *combinatorial*
  - Discrete choices
  - Additional hard constraints

Illustrative case studies (found in `example-problems`)

- Mentor Matching
- Exam Scheduling
- Power Plant Scheduling

Mentor Matching

Institute for Software & Systems Engineering

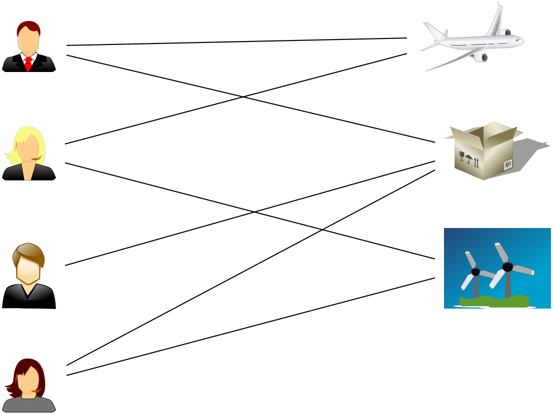
Goal: Assign mentees (e.g. students) to mentors (e.g. companies) such that

- Students are most satisfied with their mentors
- Companies are satisfied with their mentees
- Two-sided preferences

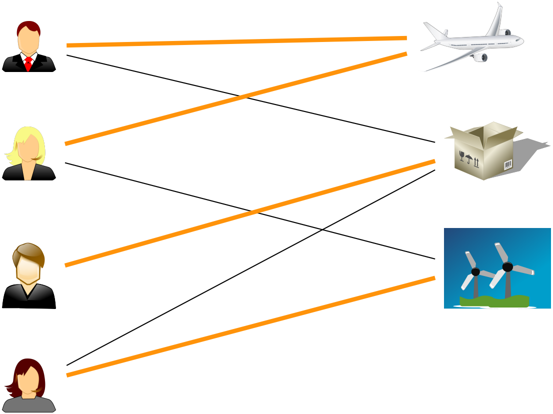
So far, sounds like a typical *stable matching* problem, but:

- We do not have a 1:1 mapping (companies advise several students)
- Additional constraints are present
  - Each company has to advise at least l, at most u students
  - The number of advised students *should* be roughly equal per company (fairness)
  - Students actually despising a company should not be forced to go there (*hard exclusion* of solutions)

Mentor Matching: Example

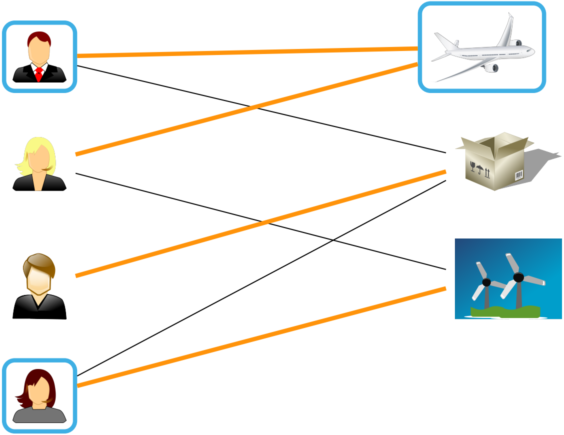


Mentor Matching: Example



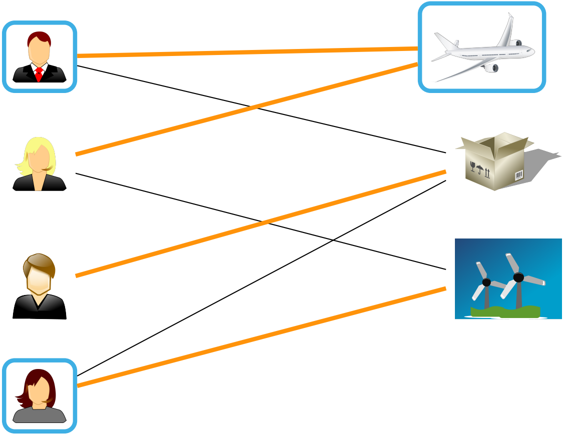
This assignment respects the students' preferences (edges)

Mentor Matching: Example



This assignment respects the students' preferences (edges) but ignores the companies' preferences.

Mentor Matching: Example



This assignment respects the students' preferences (edges) but ignores the companies' preferences. ok, it's not really a *matching* since companies supervise more than one student ...

Mentor Matching: Model

```
int: n; set of int: STUDENT = 1..n;
int: m; set of int: COMPANY = 1..m;

% assign students to companies
array[STUDENT] of var COMPANY: worksAt;

% insert relationships of students and companies here

int: minPerCompany = 2; int: maxPerCompany = 3;
constraint global_cardinality_low_up (
          worksAt, [c | c in COMPANY],
          [minPerCompany | c in COMPANY],
          [maxPerCompany | c in COMPANY]);

solve
:: int_search([ satisfied[mostImpFirst[i]] | i in SOFTCONSTRAINTS],
  input_order, indomain_max, complete)
minimize penSum;
```

Mentor Matching: Preferences

```
n = 3; m = 3;
int: brenner = 1;
int: teufel = 2;
int: fennek = 3;

int: cupgainini = 1;
int: gsm = 2;
int: junedied = 3;

% specify soft constraints, order by relationship
constraint worksAt[teufel] = junedied <-> satisfied[teufJune];
constraint worksAt[teufel] = cupgainini <-> satisfied[teufCap];
constraint worksAt[teufel] = gsm <-> satisfied[teufGsm];

constraint worksAt[fennek ] in {cupgainini, gsm} <-> satisfied[fenFavs];
constraint worksAt[fennek ] in {junedied} <-> satisfied[fenOK];

crEdges = [| teufGsm, teufCap | teufGsm, teufJune
           | fenOK, fenFavs |];
```

Mentor Matching: Refinements

Institute for Software & Systems Engineering

Split company and student preferences:

```
% first, our students' preferences
var int: penStud = sum(sc in 1..lastStudentPref)
     (bool2int(not satisfied[sc]) * penalties[sc]);
% now companies' preferences
var int: penComp = sum(sc in lastStudentPref+1..nScs)
     (bool2int(not satisfied[sc]) * penalties[sc]);
```

Optimize lexicographically

```
solve
:: int_search([ satisfied[mostImpFirst[i]] | i in SOFTCONSTRAINTS],%...
%search minimize_lex([penStud, penComp]) /\ if % ...
search minimize_lex([penComp, penStud]) /\ if % ...
```

Mentor Matching: Priority Example

Taken from example: `student-company-matching.mzn`

```
solve
:: int_search([ satisfied[mostImpFirst[i]] | i in SOFTCONSTRAINTS],%...
search minimize_lex([penStud, penComp]) /\ if %...
```

```
worksAt = [1, 3, 2, 3], penalty Students: 8, penalty Companies: 6
----------
==========
```

```
solve
:: int_search([ satisfied[mostImpFirst[i]] | i in SOFTCONSTRAINTS],%...
search minimize_lex([penStud, penComp]) /\ if %...
```

```
worksAt = [1, 3, 1, 2], penalty Students: 10, penalty Companies: 4
----------
==========
```

Here, company 1 (cupgainini) wanted to have student 3, and company 2 (APS) did not have any preferences whatsoever (so accepted student 4 instead of 3). Student 4 would have liked company 3 (junedied) better, though.

Mentor Matching: Real Instance

Institute for Software & Systems Engineering

- Collected data from winter term

Example

```
"the favorites":
1. JuneDied-Lynx- HumanIT
2. Cupgainini

"I could live with that":
3. Seamless-German
4. gsm systems
5. Yiehlke

"I think, we won't be happy":
6. APS
7. Delphi Databases
```

Mentor Matching: Real Instance

Institute for Software & Systems Engineering

- Gave precedence to students
  - After all, what should companies do with unhappy students?
- Search space: 7 companies for 16 students $\rightarrow 7^{16} = 3.3233 \cdot 10^{13}$
- Led to a constraint problem with
  - 77 student preferences (soft constraints) from 16 students
  - of a total of 114 soft constraints (37 company preferences)
- *Proved* optimal solution
  - 4 minutes compilation
  - another 2m 12s solving time

Exam Scheduling

Goal: Assign exam dates to students such that

- Each student likes their appoints (*approves of it*)
- The number of distinct dates is minimized (to reduce time investment of teachers)

At least 3 options have to be selected

		Approve	Absolutely not
12 February 2016	Morning	○	○
12 February 2016	Afternoon	○	○
18 February 2016	Morning	○	○
18 February 2016	Afternoon	○	○
...	...	○	○
	Name		

- No preference of any student should be weighted higher than another one's
- Solution (exam schedule) is a shared decision

Exam Scheduling: Core Model

Institute for Software & Systems Engineering

See exam-scheduling-approval.mzn:

```
% Exam scheduling example with just a set of
% approved dates and *impossible* ones
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";

int: n; set of int: STUDENT = 1..n;
int: m; set of int: DATE = 1..m;
array[STUDENT] of set of DATE: possibles;
array[STUDENT] of set of DATE: impossibles;

% the actual decisions
array[STUDENT] of var DATE: scheduled;

int: minPerSlot = 0; int: maxPerSlot = 4;
constraint global_cardinality_low_up(scheduled % minPerSlot, maxPerSlot
constraint forall(s in STUDENT) (not (scheduled[s] in impossibles[s]));
```

Exam Scheduling: Preferences

Institute for Software & Systems Engineering

See exam-scheduling-approval.mzn:

```
% have a soft constraint for every student
nScs = n;
penalties = [ 1 | n in STUDENT]; % equally important in this case

constraint forall(s in STUDENT) (
     (scheduled[s] in possibles[s]) <-> satisfied[s] ) ;
var DATE: scheduledDates;
% constrains that "scheduledDates" different
% values (appointments) appear in "scheduled"
constraint nvalue(scheduledDates, scheduled);

% search variants
solve
:: int_search(satisfied, input_order, indomain_max, complete)
search minimize_lex([scheduledDates, violateds]); % pro teachers
%search minimize_lex([violateds, scheduledDates]); % pro students
```

Exam Scheduling: Real Instance

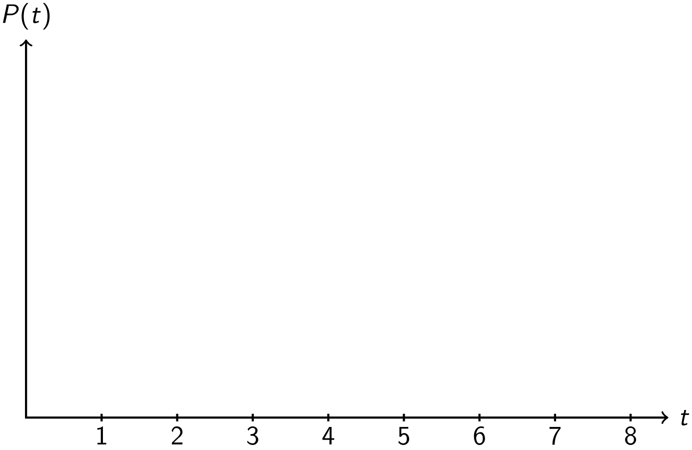
- Collected preferences of 33 students
- over 12 possible dates (6 days, morning and afternoon)
  - *Approval* set
  - *Impossible* set

- Aggregated via approval voting (has nice voting-theoretical properties!)
- At most 4 per appointment
- Immediately (61 msec) found an optimal solution that
  - Is approved by *every* student
  - Is achieved with the minimal number of 9 dates
- Used Strategy:

```
search minimize_lex([violateds, scheduledDates]); % pro students
```

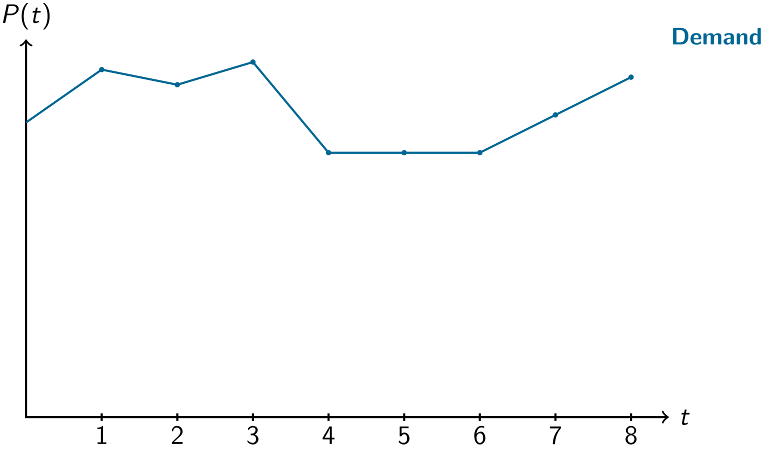
Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



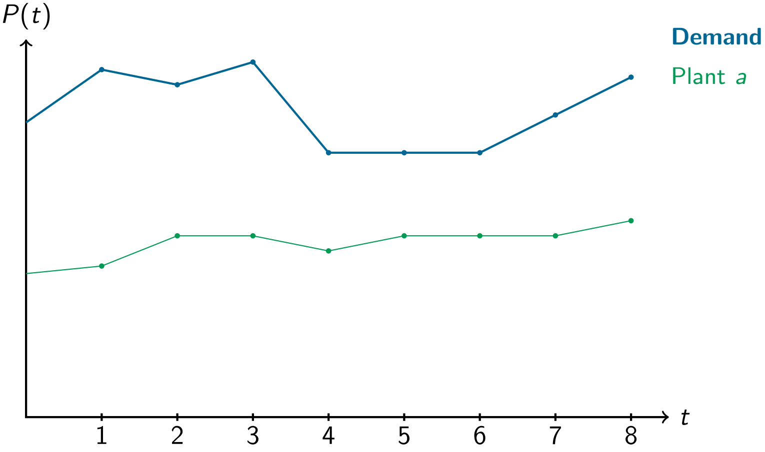
Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



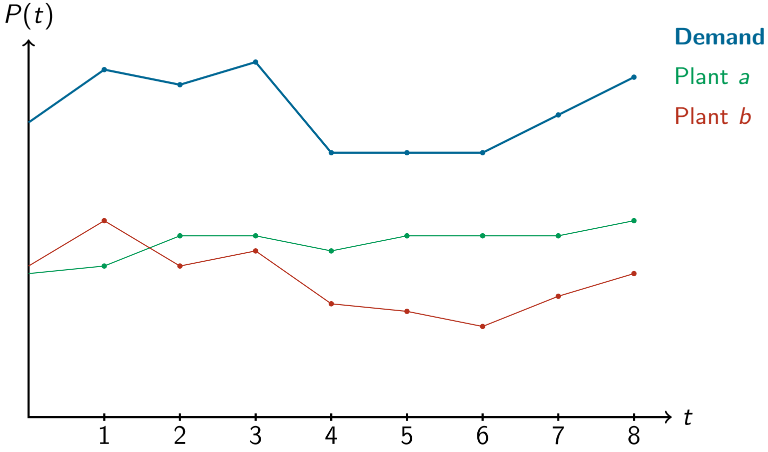
Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



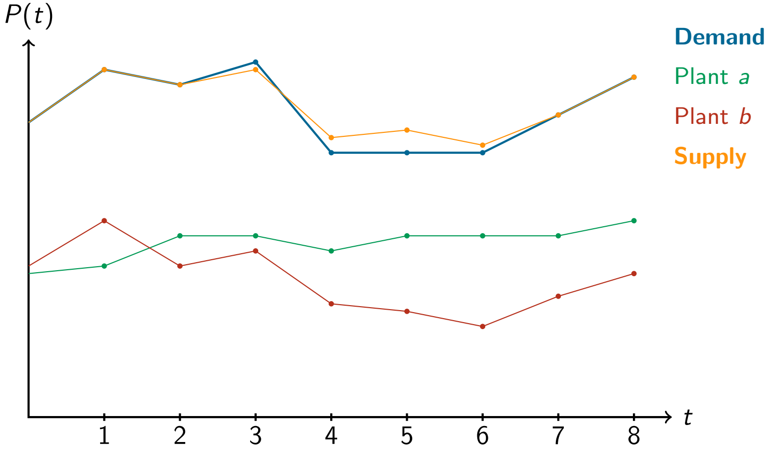
Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



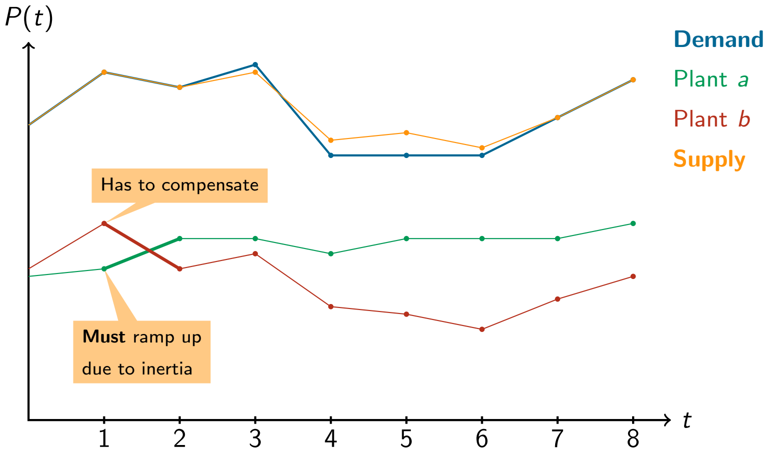
Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



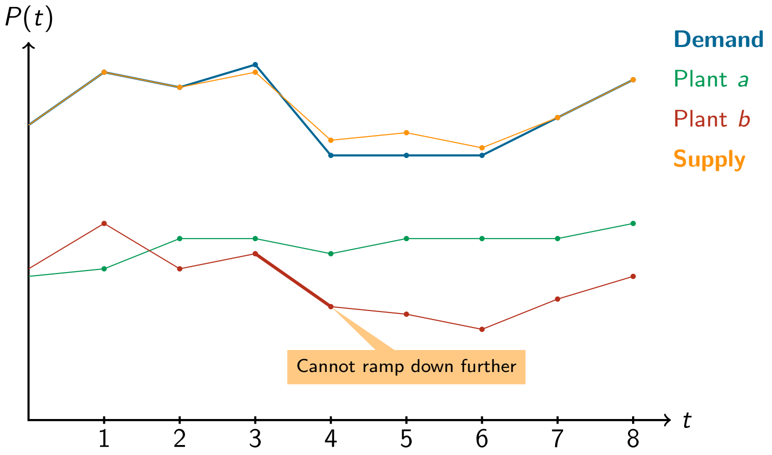
Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



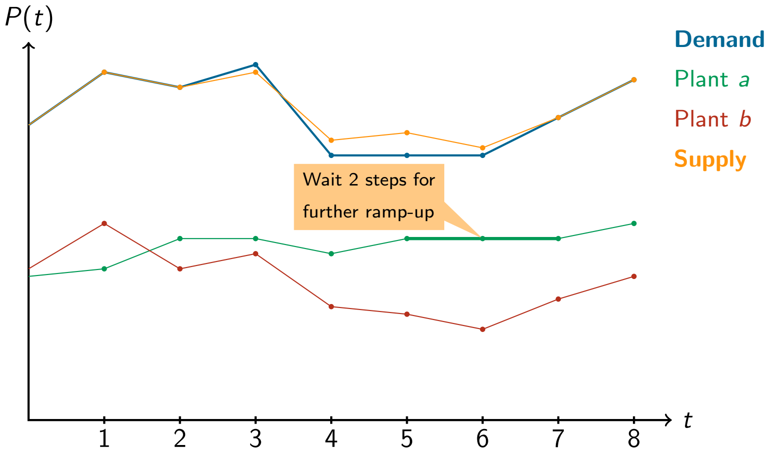
Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



Power Plant Scheduling

Goal: Schedule plants such that they meet the demand; See `unitCommitment.mzn`



Power Plant Scheduling: Core Model

Institute for

```
include "soft_constraints/soft_constraints_noset.mzn";
include "soft_constraints/cr_types.mzn";
include "soft_constraints/cr_weighting.mzn";
% ground penalties using the appropriate weighting
penalties = [weighting(s, SOFTCONSTRAINTS, crEdges, true)
               | s in SOFTCONSTRAINTS];

int: T = 5; set of int: WINDOW = 1..T;
array[WINDOW] of float: demand = [10.0, 11.3, 15.2, 20.7, 19.2];

int: P = 3; set of int: PLANTS = 1..P;
array[PLANTS] of float: pMin = [12.0, 5.0, 7.3];
array[PLANTS] of float: pMax = [15.0, 11.3, 9.7];

array[WINDOW, PLANTS] of var 0.0..15.0: supply;
var float: obj;

constraint obj = sum(w in WINDOW) ( abs( sum(p in PLANTS)
          (supply[w, p]) - demand[w] ) );
```

Power Plant Scheduling: Soft Constraints

Institute for Software & Systems Engineering

```
% ground penalties using the appropriate weighting
penalties = [weighting(s, SOFTCONSTRAINTS, crEdges, true)
               | s in SOFTCONSTRAINTS];
[...]

% some soft constraints
constraint supply[1, 2] >= 6.0 <-> satisfied[1];
constraint supply[2, 2] >= 6.0 <-> satisfied[2];


% constraint time step 1 seems more urgent
nCrEdges = 1;
crEdges = [| 2, 1 |];

% could do something more sophisticated here
solve minimize obj + penSum;
```

→ Library works with MIP (*Mixed Integer Programming*) as well!

Quellen I