



Solving Soft Constraint Problems in MiniBrass

Alexander Schiendorfer et al.



We could keep adding more and more desirable constraints

- “Not only would I like to stand next to a friend, but Mike makes me look better than Joe.”
- “This robot should not do the drilling task as its driller is almost broken.”
- “This module should be deployed to server *A* as its required dependencies are already there.”

We could keep adding more and more desirable constraints

- “Not only would I like to stand next to a friend, but Mike makes me look better than Joe.”
- “This robot should not do the drilling task as its driller is almost broken.”
- “This module should be deployed to server *A* as its required dependencies are already there.”

but at some point, problems become **unsatisfiable**.

We could keep adding more and more desirable constraints

- “Not only would I like to stand next to a friend, but Mike makes me look better than Joe.”
- “This robot should not do the drilling task as its driller is almost broken.”
- “This module should be deployed to server *A* as its required dependencies are already there.”

but at some point, problems become **unsatisfiable**.

- → returning UNSAT is not an option in autonomic computing settings!

We could keep adding more and more desirable constraints

- “Not only would I like to stand next to a friend, but Mike makes me look better than Joe.”
- “This robot should not do the drilling task as its driller is almost broken.”
- “This module should be deployed to server *A* as its required dependencies are already there.”

but at some point, problems become **unsatisfiable**.

- → returning UNSAT is not an option in autonomic computing settings!
- We want *graceful* degradation: satisfy most constraints (maybe even distinguish them)

Consider:

$x, y, z \in \{1, 2, 3\}$ with

$$c_1 : x + 1 = y$$

$$c_2 : z = y + 2$$

$$c_3 : x + y \leq 3$$

- Not all constraints can be satisfied simultaneously

Consider:

$x, y, z \in \{1, 2, 3\}$ with

$$c_1 : x + 1 = y$$

$$c_2 : z = y + 2$$

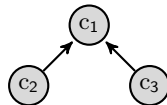
$$c_3 : x + y \leq 3$$

- Not all constraints can be satisfied simultaneously
 - e. g., c_2 forces $z = 3$ and $y = 1$, making c_1 impossible
- We **choose** between solutions that satisfy either $\{c_1, c_3\}$ or $\{c_2, c_3\}$.

Which solutions $v \in [X \rightarrow D]$ are **preferred**?

Approach (Schiendorfer et al., 2013)

- Define preference relation over constraints C to denote which constraints are **more important**, e. g.
 - c_1 is more important than c_2
 - c_1 is more important than c_3
- How **much** more important is c_1 ?
 - More important than only either of c_2 or c_3 ?
 - More important than c_2 and c_3 *combined*?

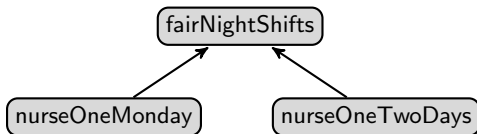


Example: Rostering



Consider again our rostersing example

worksShift	Monday	Tuesday	Wednesday	Thursday	Friday
Nurse 1	off	1	2	3	2
Nurse 2	1	off	1	1	1
Nurse 3	1	2	off	1	2



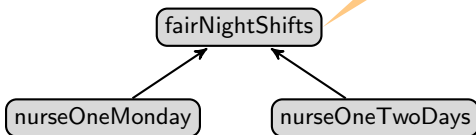
Example: Rostering



Consider again our rostering example

worksShift	Monday	Tuesday	Wednesday	Thursday	Friday
Nurse 1	off	1	2	3	2
Nurse 2	1	off	1	1	1
Nurse 3	1	2	off	1	2

$\forall n \text{ in NURSE } \exists d \text{ in DAYS : worksShift}[n, d] = 3$



Example: Rostering

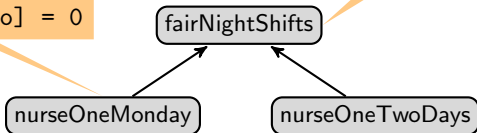


Consider again our rostering example

worksShift	Monday	Tuesday	Wednesday	Thursday	Friday
Nurse 1	off	1	2	3	2
Nurse 2	1	off	1	1	1
Nurse 3	1	2	off	1	2

$\forall n \text{ in NURSE } \exists d \text{ in DAYS : worksShift}[n, d] = 3$

$\text{worksShift}[n1, \text{mo}] = 0$



Example: Rostering

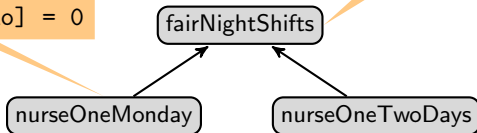


Consider again our rostering example

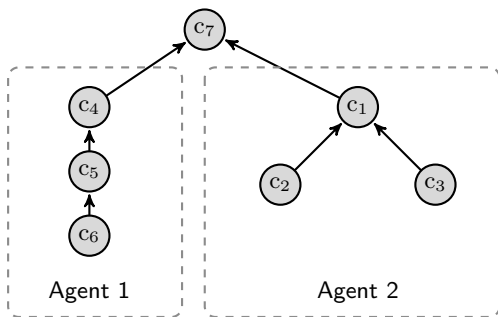
worksShift	Monday	Tuesday	Wednesday	Thursday	Friday
Nurse 1	off	1	2	3	2
Nurse 2	1	off	1	1	1
Nurse 3	1	2	off	1	2

$$\forall n \text{ in NURSE } \exists d \text{ in DAYS : worksShift}[n, d] = 3$$

$$\text{worksShift}[n1, \text{mo}] = 0$$



$$\exists d \text{ in DAYS : worksShift}[n1, d] = 0 \wedge \text{worksShift}[n1, d+1] = 0$$

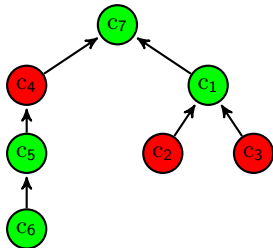


- “Organizational” constraint should be considered most important
- **No agent** (i.e., no agent’s constraint) should be preferred (unbiased combination).

Two rules for an **isWorseThan** ordering (called *single-predecessor-dominance*, SPD):

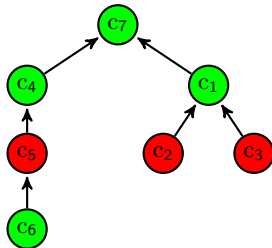
$$V \uplus \{c\} \supset_{\text{SPD}} V$$

$$V \uplus \{c_{\text{gold}}\} \supset_{\text{SPD}} V \uplus \{c_{\text{silver}}\} \quad \text{if } c_{\text{silver}} \text{ is less important than } c_{\text{gold}}$$




Solution A

\supset_{SPD}
"is worse than"






Solution B

[View on GitHub](#) 

MiniBrass*

A utility library for soft constraints with constraint relationships on top of the MiniZinc/MiniSearch toolchain.

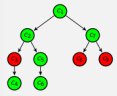
 tar.gz  .zip



Optimization with Preferences

Many combinatorial optimization problems are conveniently expressed using a constraint-based modeling language. They are then solved by powerful constraint programming or mathematical programming solvers.

We provide support for over-constrained problems or problems where desirable properties can be modeled as optional (soft) constraints. Importance is expressed only by



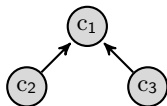
<http://isse-augsburg.github.io/minibrass/>

Base model (MiniZinc)

```
include "hello_o.mzn";  
include "soft_constraints/  
    pvs_gen_search.mzn";  
% the basic, "classic" CSP  
set of int: DOM = 1..3;  
  
var DOM: x; var DOM: y;  
var DOM: z;  
% add. *hard* constraints  
% e.g. constraint  $x < y$ ;  
  
solve search pvs_BAB();
```

Preference model (MiniBrass)

```
PVS: cr1 =  
    new ConstraintRelationships("cr1") {  
        soft-constraint c1: ' $x + 1 = y$ ';  
        soft-constraint c2: ' $z = y + 2$ ';  
        soft-constraint c3: ' $x + y \leq 3$ ';  
  
        crEdges : '[| mbr.c2, mbr.c1 |  
                    mbr.c3, mbr.c1 |]';  
        useSPD: 'true' ;  
    };  
  
solve cr1;
```



Base model (MiniZinc)

```
include "hello_o.mzn";
include "soft_constraints/
    pvs_gen_search.mzn";
% the basic, "classic" CSP
set of int: DOM = 1..3;

var DOM: x; var DOM: y;
var DOM: z;
% add. *hard* constraints
% e.g. constraint  $x < y$ ;

solve search pvs_BAB();
```

Solution: $x = 1; y = 2; z = 1$
Valuations: $\text{mbr_overall_cr1} = \{c2\}$

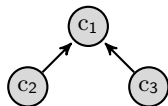
=====

Preference model (MiniBrass)

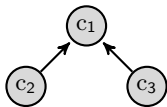
```
PVS: cr1 =
    new ConstraintRelationships("cr1") {
        soft-constraint c1: ' $x + 1 = y$ ';
        soft-constraint c2: ' $z = y + 2$ ';
        soft-constraint c3: ' $x + y \leq 3$ ';

        crEdges : '[| mbr.c2, mbr.c1 |
                    mbr.c3, mbr.c1 |]';
        useSPD: 'true' ;
    };

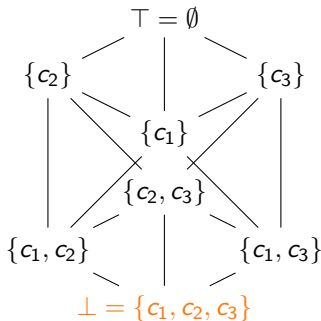
solve cr1;
```



The partially-ordered valuation space

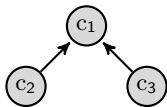


c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';

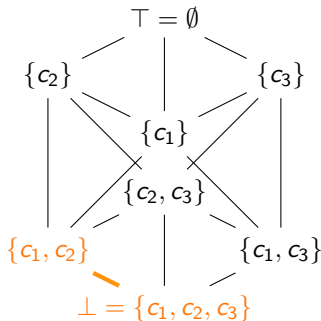


```
function ann: pvs_BAB() =  
  repeat(  
    if next() then  
      print("Intermediate solution:") /\ print() /\  
      commit() /\ postGetBetter()  
    else break endif  
  );
```

The partially-ordered valuation space



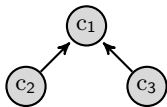
c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';



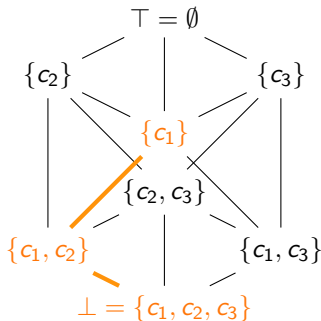
x = 1; y = 1; z = 1
Valuation = {c1, c2}

```
function ann: pvs_BAB() =  
  repeat(  
    if next() then  
      print("Intermediate solution:") /\ print() /\  
      commit() /\ postGetBetter()  
    else break endif  
  );
```

The partially-ordered valuation space



c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';

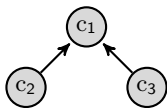


x = 1; y = 1; z = 1
Valuation = {c1, c2}

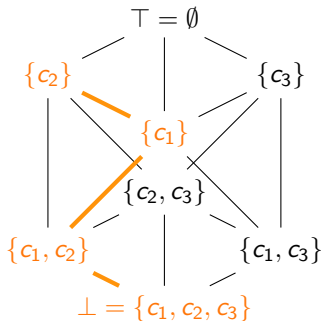
x = 1; y = 1; z = 3
Valuation = {c1}

```
function ann: pvs_BAB() =  
  repeat(  
    if next() then  
      print("Intermediate solution:") /\ print() /\  
      commit() /\ postGetBetter()  
    else break endif    );
```

The partially-ordered valuation space



c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';



x = 1; y = 1; z = 1
Valuation = {c1, c2}

x = 1; y = 1; z = 3
Valuation = {c1}

x = 1; y = 2; z = 1
Valuations = {c2}

=====

```
function ann: pvs_BAB() =  
  repeat(  
    if next() then  
      print("Intermediate solution:") /\ print() /\  
      commit() /\ postGetBetter()  
    else break endif    );
```

Is that all?

- Does MiniBrass only support Constraint Relationships?

Is that all?

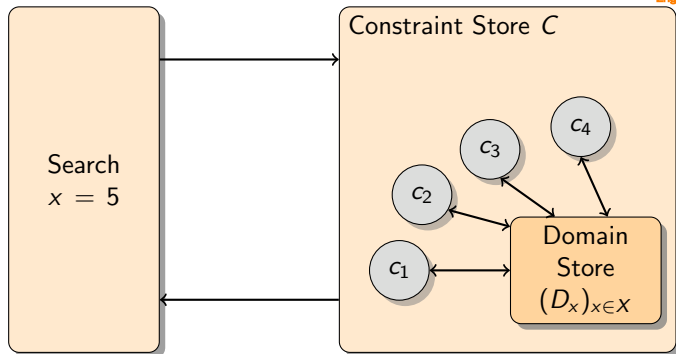
- Does MiniBrass only support Constraint Relationships?
- Of course not, there other ways to express preferences:

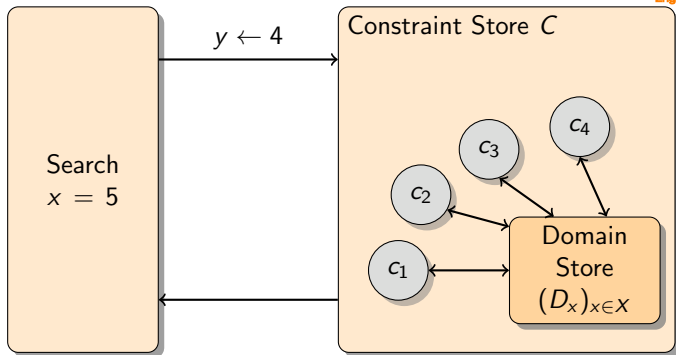
- Does MiniBrass only support Constraint Relationships?
- Of course not, there other ways to express preferences:
 - Weights/costs/penalties (violation of a constraint leads to 5 penalty points)

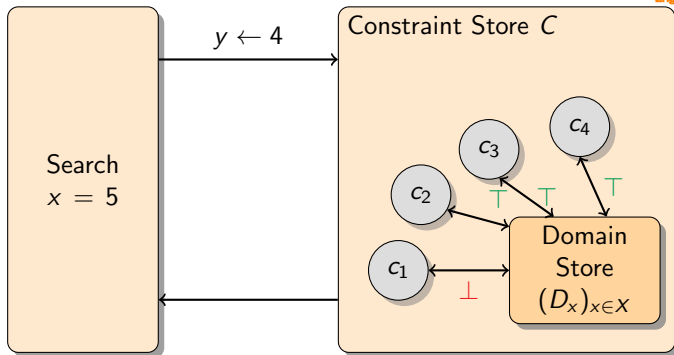
- Does MiniBrass only support Constraint Relationships?
- Of course not, there other ways to express preferences:
 - Weights/costs/penalties (violation of a constraint leads to 5 penalty points)
 - Fuzzy satisfaction degrees, ranging from 0 (not acceptable) to 1 (maximally satisfied)

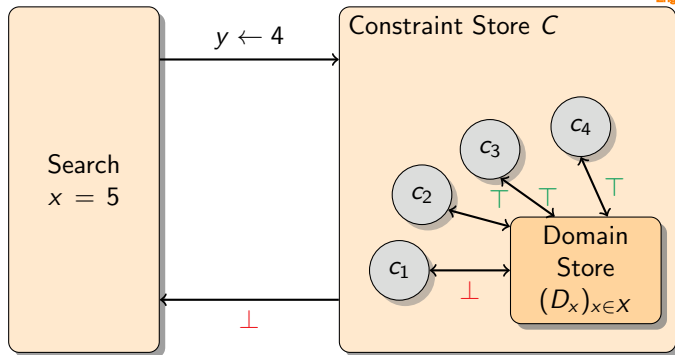
- Does MiniBrass only support Constraint Relationships?
- Of course not, there other ways to express preferences:
 - Weights/costs/penalties (violation of a constraint leads to 5 penalty points)
 - Fuzzy satisfaction degrees, ranging from 0 (not acceptable) to 1 (maximally satisfied)
 - Probabilities ("if I leave at 8am, the probability of actually meeting my deadline is 40%")

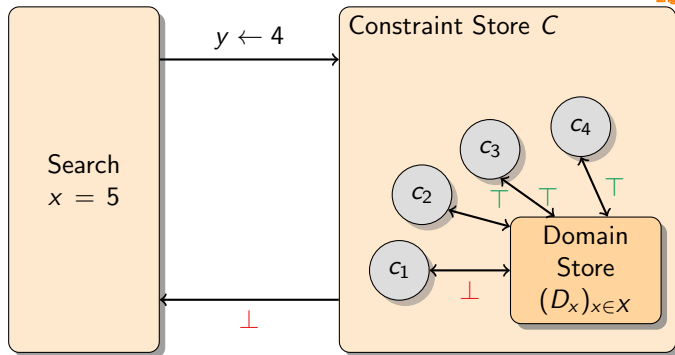
- Does MiniBrass only support Constraint Relationships?
- Of course not, there other ways to express preferences:
 - Weights/costs/penalties (violation of a constraint leads to 5 penalty points)
 - Fuzzy satisfaction degrees, ranging from 0 (not acceptable) to 1 (maximally satisfied)
 - Probabilities ("if I leave at 8am, the probability of actually meeting my deadline is 40%")
 - ...
- What is the common underlying abstract data type?
- What's up with PVS: `cr = new ConstraintRelationship`



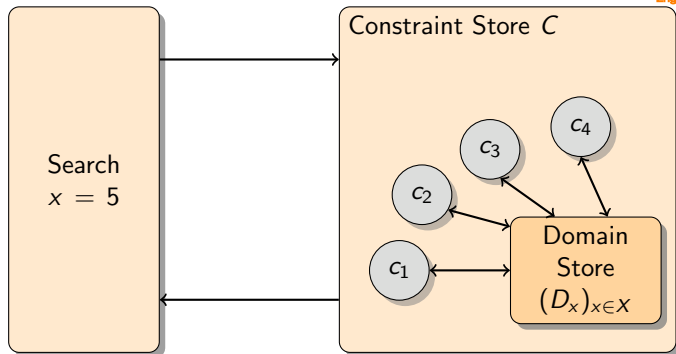


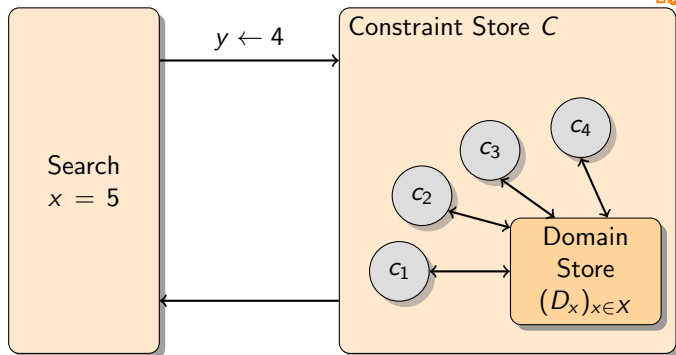


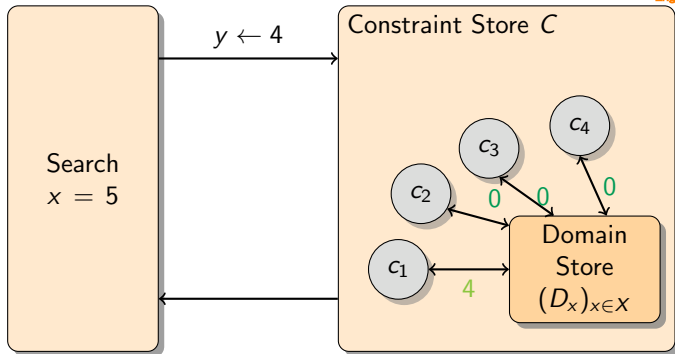


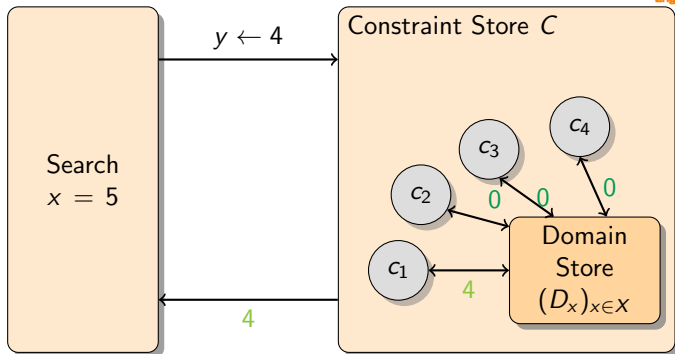


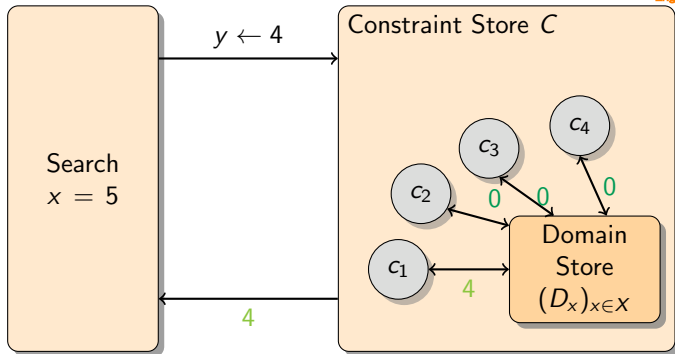
- A set of satisfaction degrees, $\mathbb{B} = \{\perp, \top\}$
- A combination operation \wedge
- A neutral element \top
- A partial ordering $(\mathbb{B}, \leq_{\mathbb{B}})$ with $\top <_{\mathbb{B}} \perp$











- A set of satisfaction degrees, e.g., $\{0, \dots, k\}$
- A combination operation $+$
- A neutral element 0
- A partial ordering (\mathbb{N}_0, \geq) with 0 as top

Underlying **abstract data type**: Partial valuation structure

- $(M, \cdot_M, \varepsilon_M, \leq_M)$
- $m \cdot_m \varepsilon_M = m$
- $m \leq_M \varepsilon_M$
- $m \leq_M n \rightarrow m \cdot_M o \leq_M n \cdot_M o$



Abstract

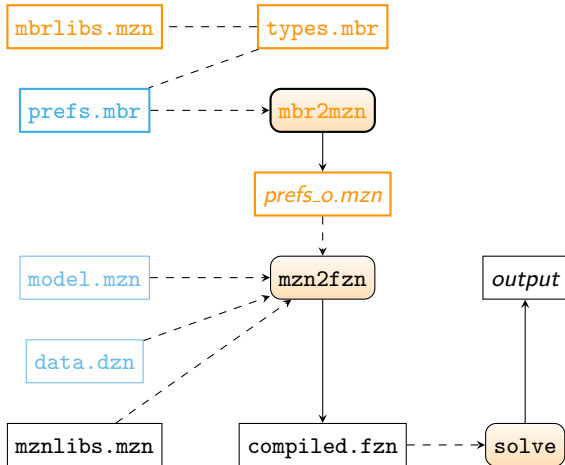
- M ... elements
- \cdot_M ... combination function
- ε_M ... neutral, “best” element
- \leq_M ... ordering, left “worse”

Concrete

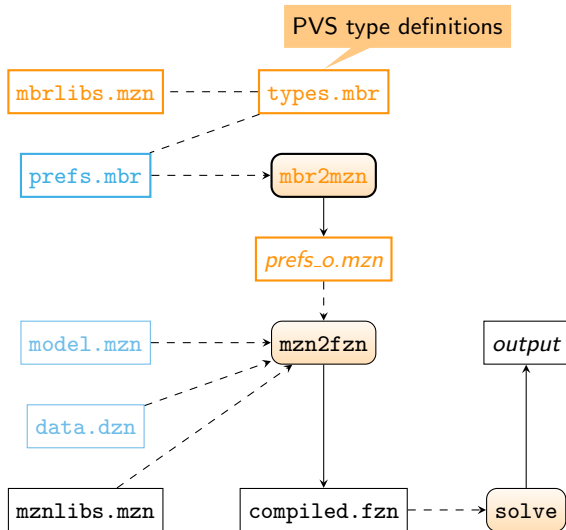
- $\{0, \dots, k\}$
- $+_k$
- 0
- \geq

(Gadducci et al., 2013; Schiendorfer et al., 2015)

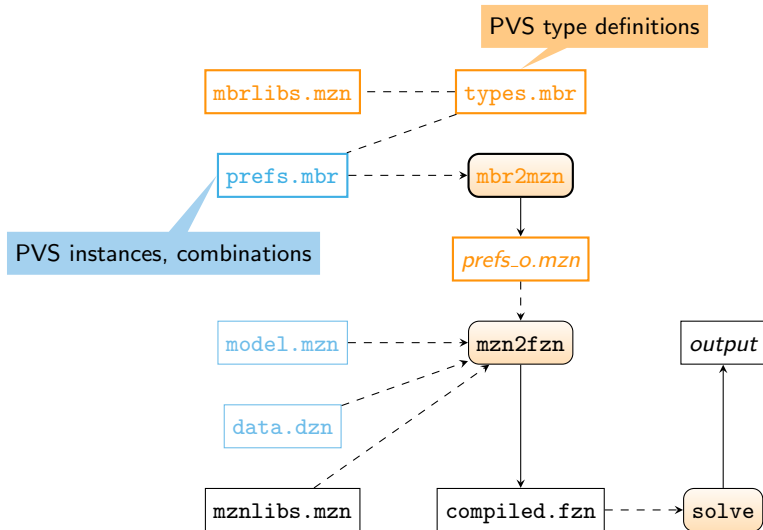
Architecture



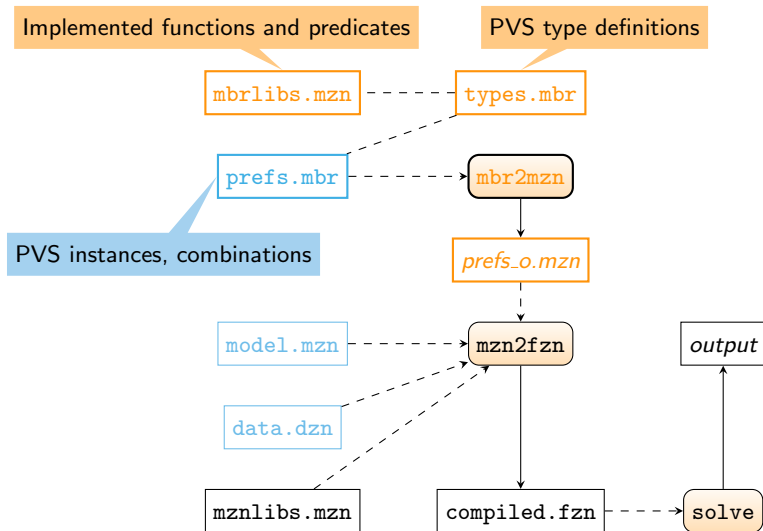
Architecture



Architecture



Architecture



Exercise 3

Now we would like to refine our group photo model with constraint relationships. Start by examining the “pure” MiniZinc model `groupphoto-pure.mzn` and test it with `groupphoto1.dzn`. We will augment this model with preferences, starting with person 3 (Carla). She has three preferences

- c1: She would like to be placed next to person 2. (Hint: Use the provided `isNextTo` function.)
- c2: She would like to be placed in the second row.
- c3: Carla doesn't particularly like person 5. Hence, the Manhattan distance (provided as `manhattanDist`) between them should be greater than 4.

Constraint c1 is most important to Carla, c2 and c3 are both less important than c1 but incomparable. Write a preference model `groupphoto.mbr` that incorporates these constraint relationships. Test the model (not in the IDE) using

```
mbr2mzn groupphoto.mbr  
minisearch groupphoto.mzn groupphoto1.dzn
```

What is the best solution you get? What happens if we add another constraint c4 that asks for person 5 not to be placed at either border (column 1 or *m*)?

```
type ConstraintRelationships = PVSType<bool, set of 1..nScs> =  
  params {  
    array[int, 1..2] of 1..nScs: crEdges; % adjacency matrix  
    bool: useSPD;  
  } in  
  instantiates with "../mbr_types/cr_type.mzn" {  
    times -> link_invert_booleans;  
    is_worse -> is_worse_cr;  
    top -> {};  
  };
```

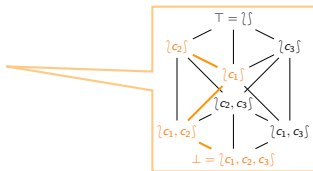
- $PVSType<S, E>$ distinguishes
 Specification type S
 Element type E
- Combination operation: $times : S^n \rightarrow E$
- Ordering relation: $isWorse \subseteq E \times E$

```
type WeightedCsp = PVSType<bool, int> =  
  params {  
    int: k;  
    array[1..nScs] of 1..k: weights :: default('1');  
  } in  
  instantiates with "../mbr_types/weighted_type.mzn" {  
    times -> weighted_sum;  
    is_worse -> is_worse_weighted;  
    top -> 0;  
  };  
  
type CostFunctionNetwork = PVSType<0..k> =  
  params {  
    int: k :: default('1000');  
  } in instantiates with "../mbr_types/cfn_type.mzn" {  
    times -> sum;  
    is_worse -> is_worse_weighted;  
    top -> 0;  
  };
```

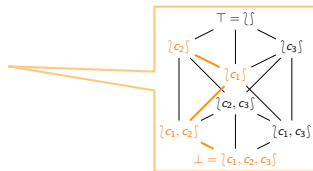
```
PVS: cr1 = new WeightedCsp("cr1") {  
    soft-constraint c1: 'x + 1 = y' :: weights('2');  
    soft-constraint c2: 'z = y + 2' :: weights('1');  
    soft-constraint c3: 'x + y <= 3' :: weights('1');  
  
    k : '20';  
};
```

- Weights can be annotated
- Or passed as array ([2,1,1])

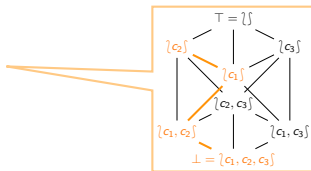
- There are good reasons for *translating* between PVS types
 - Ordering should be totalized (e.g. otherwise hard to understand)
 - Data type xy (e.g. sets) not supported by solver/algorithm (think LP/MIP)
 - \rightarrow user is **not** interested in a particular preference data structure but just cares about the solution ordering



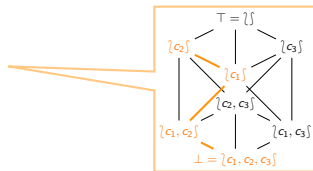
- There are good reasons for *translating* between PVS types
 - Ordering should be totalized (e.g. otherwise hard to understand)
 - Data type xy (e.g. sets) not supported by solver/algorithm (think LP/MIP)
 - \rightarrow user is **not** interested in a particular preference data structure but just cares about the solution ordering
- Hence, we need *structure-preserving* mappings



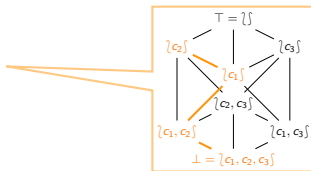
- There are good reasons for *translating* between PVS types
 - Ordering should be totalized (e.g. otherwise hard to understand)
 - Data type xy (e.g. sets) not supported by solver/algorithm (think LP/MIP)
 - \rightarrow user is **not** interested in a particular preference data structure but just cares about the solution ordering
- Hence, we need *structure-preserving* mappings
- PVS homomorphism $\varphi : \text{PVS}_{\text{cr}} \rightarrow \text{PVS}_{\text{weighted}}$



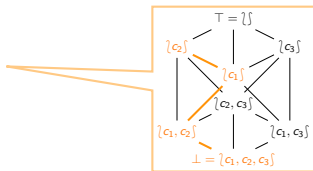
- There are good reasons for *translating* between PVS types
 - Ordering should be totalized (e.g. otherwise hard to understand)
 - Data type xy (e.g. sets) not supported by solver/algorithm (think LP/MIP)
 - \rightarrow user is **not** interested in a particular preference data structure but just cares about the solution ordering
- Hence, we need *structure-preserving* mappings
- PVS homomorphism $\varphi : \text{PVS}_{\text{cr}} \rightarrow \text{PVS}_{\text{weighted}}$
- $\text{PVS}_{\text{cr}} = \text{PVS}\langle P \rangle = \langle \mathcal{M}^{\text{fin}}(P), \cup, \supseteq_{\text{SPD}}, \perp \rangle$
 - $\varphi(T_{\text{cr}}) = T_{\text{weighted}}$



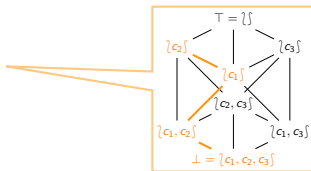
- There are good reasons for *translating* between PVS types
 - Ordering should be totalized (e.g. otherwise hard to understand)
 - Data type xy (e.g. sets) not supported by solver/algorithm (think LP/MIP)
 - \rightarrow user is **not** interested in a particular preference data structure but just cares about the solution ordering
- Hence, we need *structure-preserving* mappings
- PVS homomorphism $\varphi : \text{PVS}_{\text{cr}} \rightarrow \text{PVS}_{\text{weighted}}$
- $\text{PVS}_{\text{cr}} = \text{PVS}\langle P \rangle = \langle \mathcal{M}^{\text{fin}}(P), \cup, \supseteq_{\text{SPD}}, \perp \rangle$
 - $\varphi(\top_{\text{cr}}) = \top_{\text{weighted}}$
 - $\varphi(m \cdot_{\text{cr}} n) = \varphi(m) \cdot_{\text{weighted}} \varphi(n)$



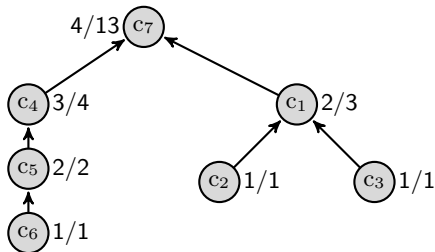
- There are good reasons for *translating* between PVS types
 - Ordering should be totalized (e.g. otherwise hard to understand)
 - Data type xy (e.g. sets) not supported by solver/algorithm (think LP/MIP)
 - \rightarrow user is **not** interested in a particular preference data structure but just cares about the solution ordering
- Hence, we need *structure-preserving* mappings
- PVS homomorphism $\varphi : \text{PVS}_{\text{cr}} \rightarrow \text{PVS}_{\text{weighted}}$
- $\text{PVS}_{\text{cr}} = \text{PVS}\langle P \rangle = \langle \mathcal{M}^{\text{fin}}(P), \cup, \supseteq_{\text{SPD}}, \perp \rangle$
 - $\varphi(\top_{\text{cr}}) = \top_{\text{weighted}}$
 - $\varphi(m \cdot_{\text{cr}} n) = \varphi(m) \cdot_{\text{weighted}} \varphi(n)$
 - $m \leq_{\text{cr}} n \rightarrow \varphi(m) \leq_{\text{weighted}} \varphi(n)$



- There are good reasons for *translating* between PVS types
 - Ordering should be totalized (e.g. otherwise hard to understand)
 - Data type xy (e.g. sets) not supported by solver/algorithm (think LP/MIP)
 - \rightarrow user is **not** interested in a particular preference data structure but just cares about the solution ordering
- Hence, we need *structure-preserving* mappings
- PVS homomorphism $\varphi : \text{PVS}_{\text{cr}} \rightarrow \text{PVS}_{\text{weighted}}$
- $\text{PVS}_{\text{cr}} = \text{PVS}\langle P \rangle = \langle \mathcal{M}^{\text{fin}}(P), \cup, \supseteq_{\text{SPD}}, \downarrow \rangle$
 - $\varphi(\top_{\text{cr}}) = \top_{\text{weighted}}$
 - $\varphi(m \cdot_{\text{cr}} n) = \varphi(m) \cdot_{\text{weighted}} \varphi(n)$
 - $m \leq_{\text{cr}} n \rightarrow \varphi(m) \leq_{\text{weighted}} \varphi(n)$



Example: Weights for Constraint Relationships



Calculated weights (SPD/TPD)

$$w^{\text{SPD}}(c) = 1 + \max_{c' \rightarrow c} w^{\text{SPD}}(c')$$

$$w^{\text{TPD}}(c) = 1 + \sum_{c' \rightarrow c} w^{\text{TPD}}(c')$$

$T = 0$

|

1

|

...

|

$k - 2$

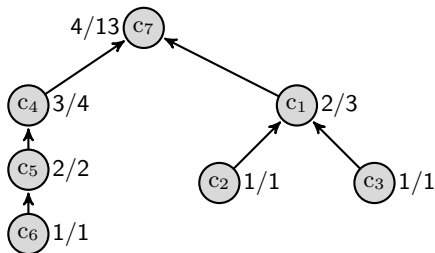
|

$k - 1$

|

$\perp = k$

Example: Weights for Constraint Relationships



Calculated weights (SPD/TPD)

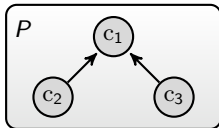
$$w^{\text{SPD}}(c) = 1 + \max_{c' \rightarrow c} w^{\text{SPD}}(c')$$

$$w^{\text{TPD}}(c) = 1 + \sum_{c' \rightarrow c} w^{\text{TPD}}(c')$$

- For ordering P over constraints: $\text{PVS Weighted}(P) = \langle \mathbb{N}, +, \geq, 0 \rangle$
- $\text{PVS}_{\text{cr}} = \text{PVS}(P) = \langle \mathcal{M}^{\text{fin}}(P), \cup, \supseteq_{\text{SPD}}, \cap \rangle$
- $\varphi(\cap) = 0$, $\varphi(\cap c \cup C) = w^{\text{SPD}}(c) + \varphi(C)$

$T = 0$
|
1
|
...
|
 $k-2$
|
 $k-1$
|
 $\perp = k$

(Schiendorfer et al., 2013)



Cat : POSet

Cat : PVS



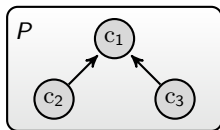
Cat : PVS

$$\mu(c) = w^{\text{SPD}}(c)$$


$$T = 0$$
 $k - 2$

1

$$\perp = k$$
 $\langle \mathbb{N}, +, \geq, 0 \rangle$



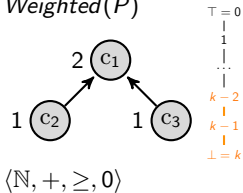
Cat : POSet

Cat : PVS

$$\mu(c) = w^{\text{SPD}}(c)$$

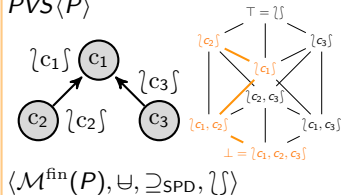
$$\eta(c) = \{c\}$$

Weighted(P)



$\langle \mathbb{N}, +, \geq, 0 \rangle$

PVS(P)

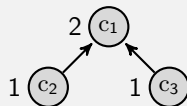


$\langle \mathcal{M}^{\text{fin}}(P), \cup, \supseteq_{\text{SPD}}, \{\} \rangle$

```
% import from a library
morph ConstraintRelationships -> WeightedCsp: ToWeighted =
  params {
    k = 'mbr.nScs * max(i in 1..mbr.nScs) (mbr.weights[i]) ';
    weights = calculate_cr_weights;
  } in id; % "in" denotes the function applied to each soft constraint
```

```
PVS: cr1 = new ConstraintRelationships("cr1") {
  soft-constraint c1: 'x + 1 = y';
  soft-constraint c2: 'z = y + 2';
  soft-constraint c3: 'x + y <= 3';

  crEdges : '[| mbr.c2, mbr.c1 | mbr.c3, mbr.c1 |]';
  useSPD: 'false' ;
};
solve ToWeighted(cr1);
```



Solution: $x = 1$; $y = 2$; $z = 1$
Valuations: overall = 1

c1: ' $x + 1 = y$ ';
c2: ' $z = y + 2$ ';
c3: ' $x + y \leq 3$ ';

With PVSs M and N we can build the *direct product* $M \times N$

$$(m, n) \leq_{M \times N} (m', n') \leftrightarrow m \leq_M m' \wedge n \leq_N n'$$

bilden. Entspricht der *Pareto*-Ordnung

```
% in MZN-file: var 1..10: x; var 1..10: y;
PVS: cfn1 = new CostFunctionNetwork("cfn1") {
    soft-constraint c1: 'y' ;
    k : '20';
};

PVS: cfn2 = new CostFunctionNetwork("cfn2") {
    soft-constraint c1: 'x' ;
    k : '20';
};

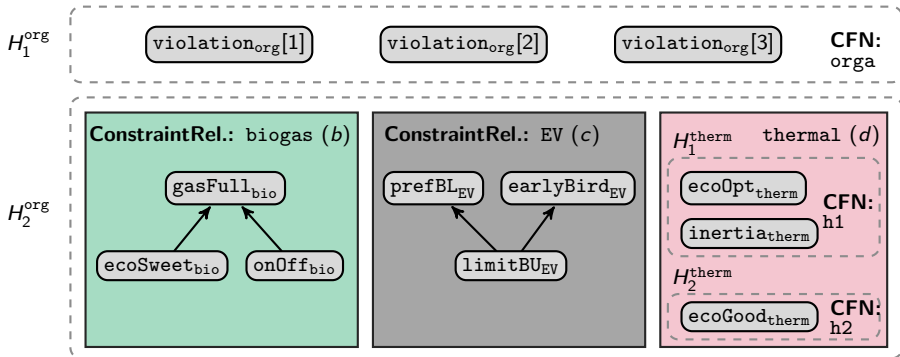
solve cfn1 pareto cfn2; % returns x = 1, y = 1
```

Additionally, we have the **lexicographic** product $M \ltimes N$

$$(m, n) \leq_{M \ltimes N} (m', n') \leftrightarrow (m <_M m') \vee (m = m' \wedge n \leq_N n')$$

Allows for strict hierarchies

```
% in MZN-file: var 1..3: x; var 1..3: y;
PVS: cfn1 = new CostFunctionNetwork("cfn1") {
    soft-constraint c1: 'x' ;
    soft-constraint c2: '3 - y' ;
    k : '20';
};
PVS: cfn2 = new CostFunctionNetwork("cfn2") {
    soft-constraint c1: 'y' ;
    soft-constraint c2: '3 - x' ;
    k : '20';
};
solve cfn1 lex cfn2; % returns x = 1, y = 3
% dually cfn2 lex cfn1 yields x = 3, y = 1
```



The valuation structure for this problem:

$$P_{org_1} \times (P_{biogas} \times P_{EV} \times (P_{thermal}^1 \times P_{thermal}^2))$$

(Schiendorfer et al., 2015)

MiniBrass has been used in several applications:

- Student-Mentor-Matching
- Exam Scheduling
- Distributed Power Systems
- Multi-User-Multi-Display (User Preferences)
- Reconfigurable robot teams

MiniBrass has been used in several applications:

- Student-Mentor-Matching
- Exam Scheduling
- Distributed Power Systems
- Multi-User-Multi-Display (User Preferences)
- Reconfigurable robot teams

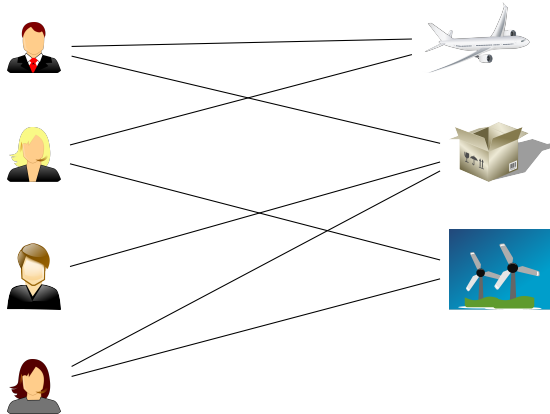
Goal: Assign mentees (e.g. students) to mentors (e.g. companies), such that

- Students are very content with their mentors
- Companies like their mentees as well
- Two-sided preferences

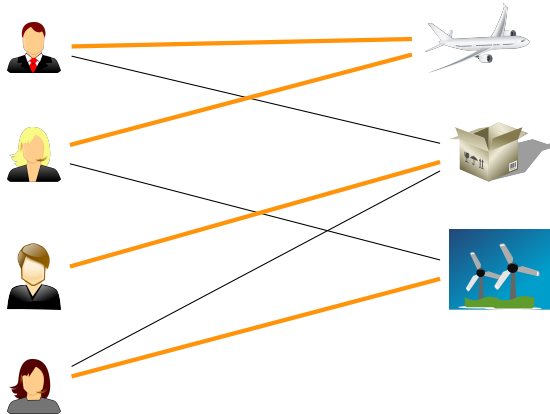
Sounds like a typical *stable matching*-problem, but:

- There is no 1:1 mapping (companies supervise several students)
- Additional side constraints are present:
 - Each company supervises at least l , at most u students
 - The number of supervised students *should* be roughly equal (Fairness)
 - Students that despise one company should not be forced to work with them

Mentor Matching: Example

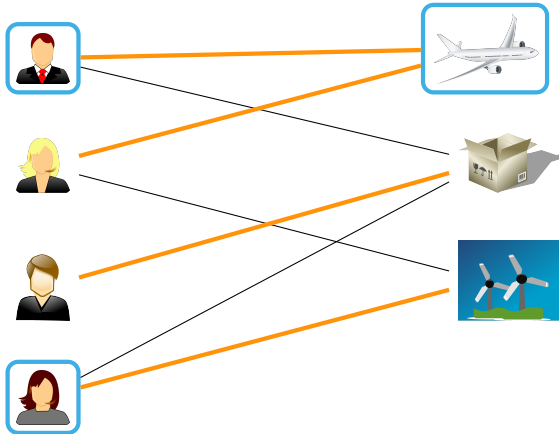


Mentor Matching: Example



This **assignment** respects students' preferences (edges)

Mentor Matching: Example



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

```
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;

int: minPerCompany = 1; int: maxPerCompany = 3;
constraint global_cardinality_low_up (
    worksAt, [c | c in COMPANY],
    [minPerCompany | c in COMPANY],
    [maxPerCompany | c in COMPANY]);

solve
search pvs_BAB();
```

```
% fas2016.mzn

n = 5; % students
m = 3; % companies

% student names for better readability
int: britney = 1;
int: christina = 2;
int: drdre = 3;
int: eminem = 4;
int: falco = 5;

% company names
int: disney = 1;
int: warner = 2;
int: uniaugsburg = 3;
```

```
PVS: students = new ConstraintRelationships("students") {
    soft-constraint britneyDisney : 'worksAt[britney] = disney';
    soft-constraint britneyWarner : 'worksAt[britney] = warner';
    soft-constraint eminemUnia : 'worksAt[eminem] = uniaugsburg';
    crEdges : '[| mbr.britneyDisney , mbr.britneyWarner | mbr.eminemUnia, m
    useSPD: 'true' ;
};

PVS: companies = new ConstraintRelationships("companies") {
    soft-constraint disneyChristina : 'worksAt[christina] = disney';
    soft-constraint disneyFalco : 'worksAt[falco] = disney';
    soft-constraint uniaugsburg : 'worksAt[britney] = uniaugsburg';

    crEdges : '[| mbr.disneyFalco, mbr.uniaugsburg |]';
    useSPD: 'true' ;
};
```

```
solve ToWeighted(students) lex ToWeighted(companies);
```

```
Intermediate solution:worksAt = [3, 2, 1, 1, 1]
```

```
Valuations: penCompanies = 1; penStudents = 6
```

```
-----
```

```
Intermediate solution:worksAt = [2, 3, 1, 1, 1]
```

```
Valuations: penCompanies = 3; penStudents = 3
```

```
-----
```

```
Intermediate solution:worksAt = [1, 1, 2, 3, 1]
```

```
Valuations: penCompanies = 2; penStudents = 3
```

```
-----
```

```
Intermediate solution:worksAt = [2, 1, 1, 3, 1]
```

```
Valuations: penCompanies = 2; penStudents = 2
```

```
-----
```

```
=====
```



```
solve ToWeighted(companies) lex ToWeighted(students);
```

```
Intermediate solution:worksAt = [3, 2, 1, 1, 1]  
Valuations: penCompanies = 1; penStudents = 6  
-----
```

```
Intermediate solution:worksAt = [3, 1, 2, 1, 1]  
Valuations: penCompanies = 0; penStudents = 6  
-----
```

```
Intermediate solution:worksAt = [3, 1, 2, 3, 1]  
Valuations: penCompanies = 0; penStudents = 5  
-----  
=====
```

- Collected preferences from emails

Example

"the favorites":

1. JuneDied-Lynx- HumanIT
2. Cupgainini

"I could live with that":

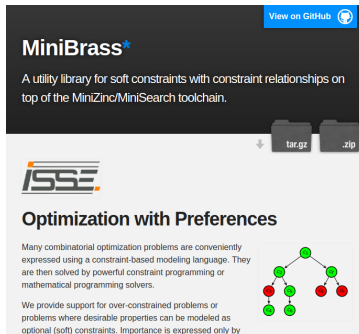
3. Seamless-German
4. gsm systems
5. Yiehlke

"I think, we won't be happy":

6. APS
7. Delphi Databases

- Priority for **students**
 - What are companies supposed to 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 students' preferences (soft constraints) of 16 students
 - in total 114 soft constraints (37 company preferences)
- *Proven* optimal solution
 - 6 minutes

- Soft constraints can be useful especially in autonomic computing settings
- Modeling problems independently from a solver can bring benefits
- Let us know if you can use MiniBrass



The screenshot shows the GitHub repository page for MiniBrass. At the top, there is a dark header with the text "MiniBrass*" and a description: "A utility library for soft constraints with constraint relationships on top of the MiniZinc/MiniSearch toolchain." To the right of the header is a "View on GitHub" button. Below the header, there are download buttons for "tar.gz" and ".zip". The main content area features the ISSE logo, the title "Optimization with Preferences", and two paragraphs of text. The first paragraph states: "Many combinatorial optimization problems are conveniently expressed using a constraint-based modeling language. They are then solved by powerful constraint programming or mathematical programming solvers." The second paragraph states: "We provide support for over-constrained problems or problems where desirable properties can be modeled as optional (soft) constraints. Importance is expressed only by". To the right of the text is a small tree diagram with green and red nodes.

<http://isse-augsburg.github.io/minibrass/>

Gadducci, F., Hölzl, M., Monreale, G., and Wirsing, M. (2013).

Soft constraints for lexicographic orders.

In Castro, F., Gelbukh, A., and González, M., editors, *Proc. 12th Mexican Int. Conf. Artificial Intelligence (MICAI'2013)*, Lect. Notes Comp. Sci. 8265, pages 68–79. Springer.

Schiendorfer, A., Knapp, A., Steghöfer, J.-P., Anders, G., Siefert, F., and Reif, W. (2015).

Partial Valuation Structures for Qualitative Soft Constraints.

In Nicola, R. D. and Hennicker, R., editors, *Software, Services and Systems - Essays Dedicated to Martin Wirsing on the Occasion of His Emeritation*, Lect. Notes Comp. Sci. 8950. Springer.

Schiendorfer, A., Steghöfer, J.-P., Knapp, A., Nafz, F., and Reif, W. (2013).

Constraint Relationships for Soft Constraints.

In Bramer, M. and Petridis, M., editors, *Proc. 33rd SGAI Int. Conf. Innovative Techniques and Applications of Artificial Intelligence (AI'13)*, pages 241–255. Springer.