

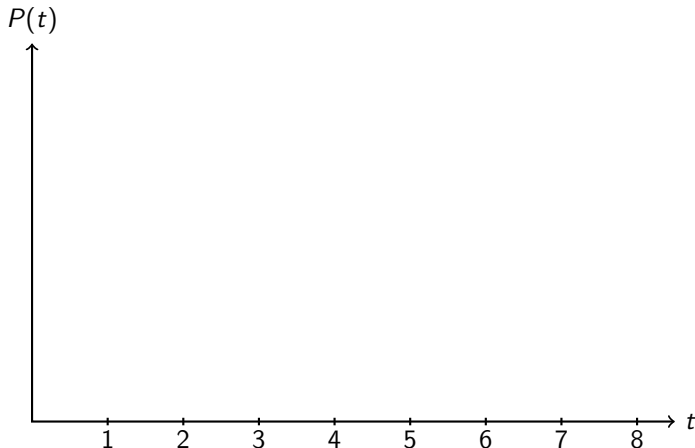


MiniBrass: Soft Constraint Programming

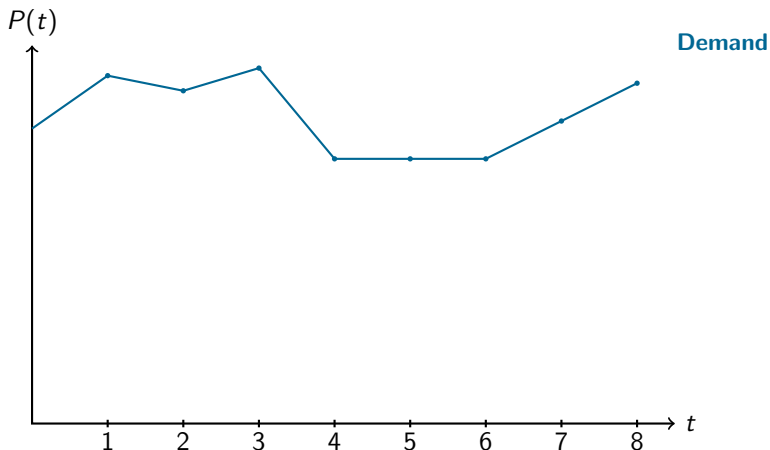
Alexander Schiendorfer et al.



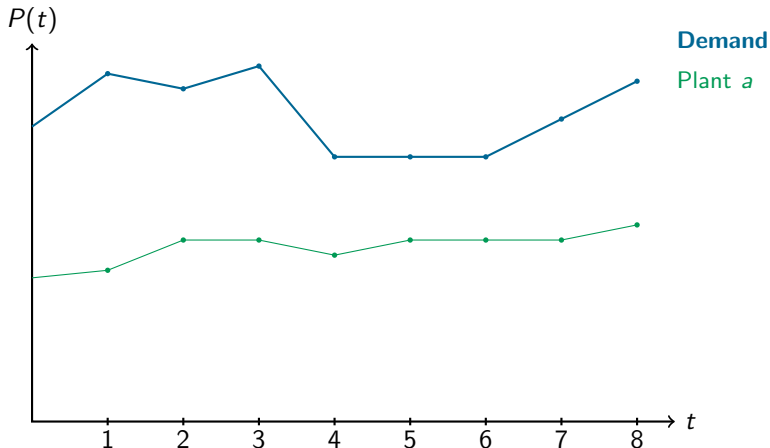
Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen



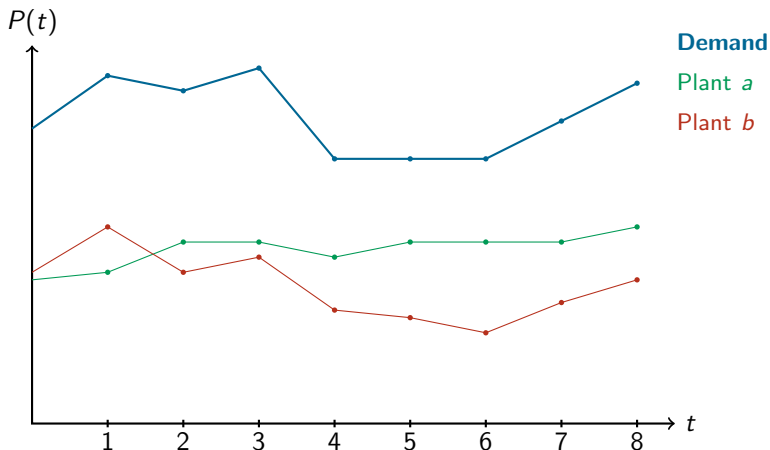
Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen



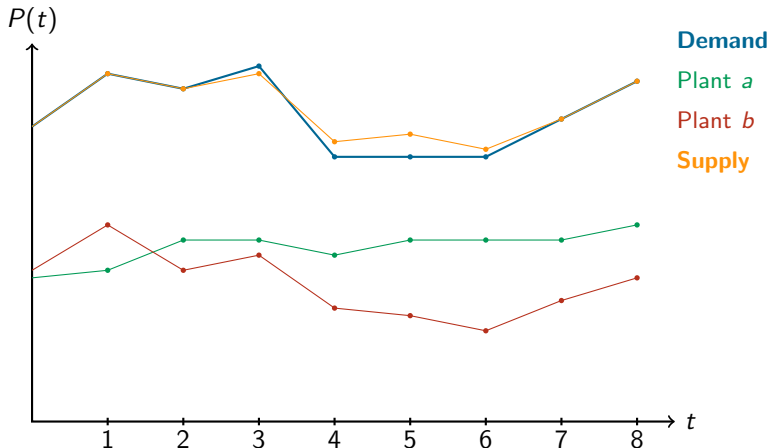
Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen



Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen

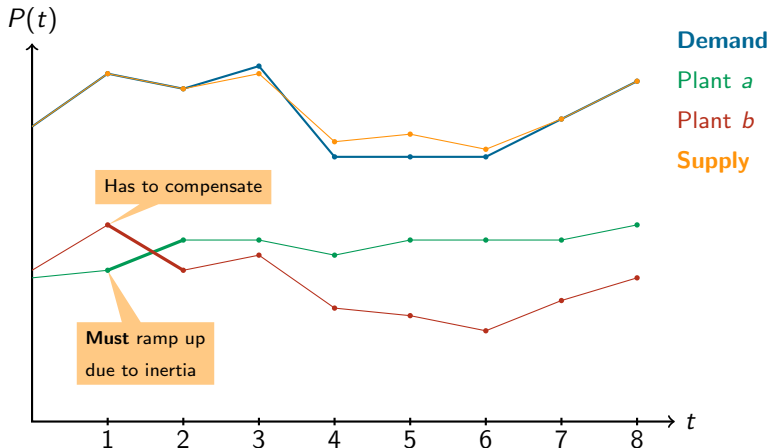


Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen



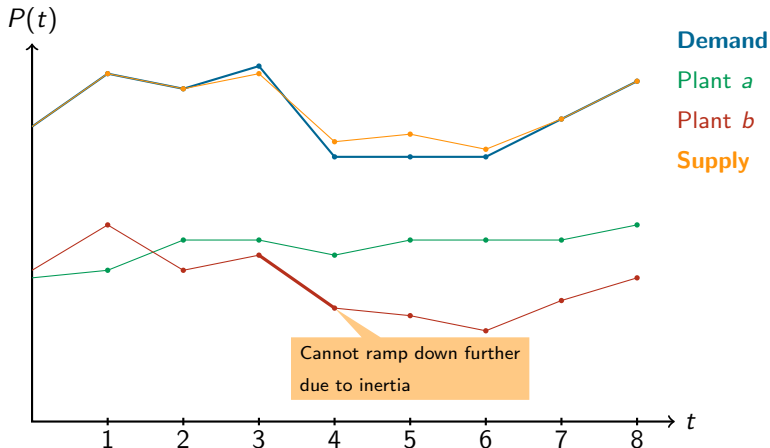
Fahrplanerstellungproblem

Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen

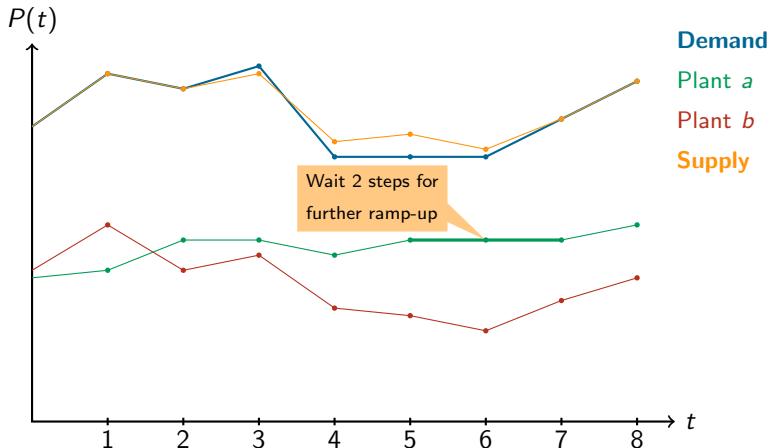


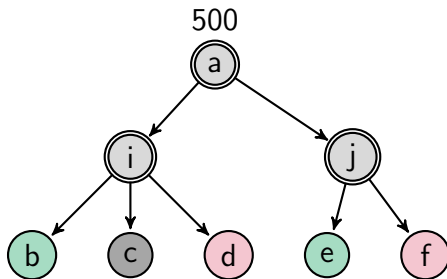
Fahrplanerstellungproblem

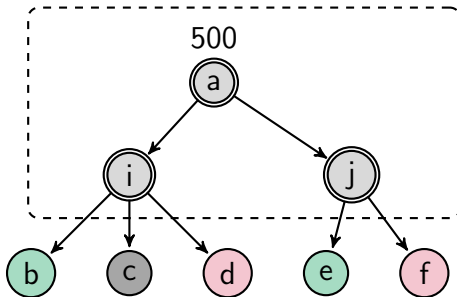
Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen



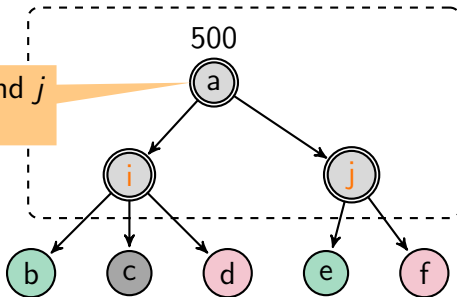
Ziel: Plane Kraftwerke so ein, dass sie die Last gemeinsam erfüllen





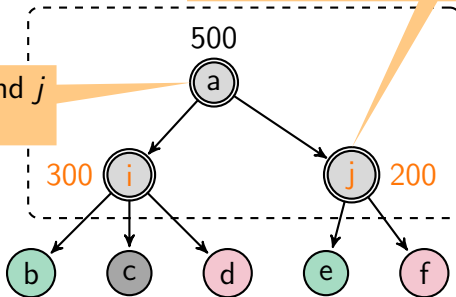


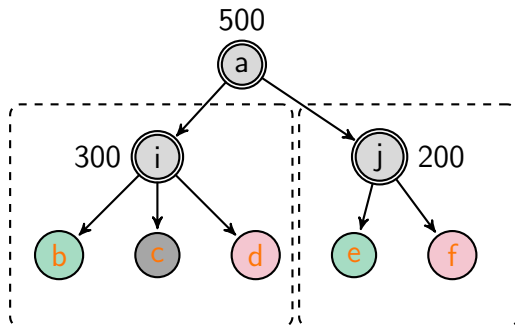
Was sollen i und j
beisteuern?

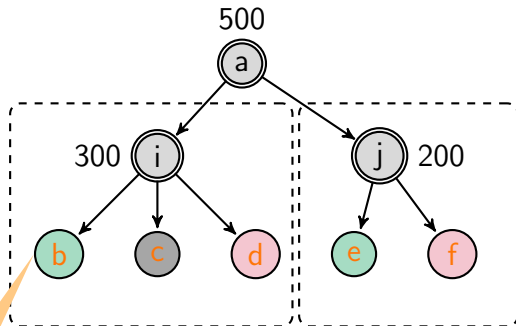


Wie kann ich e und f repräsentieren?

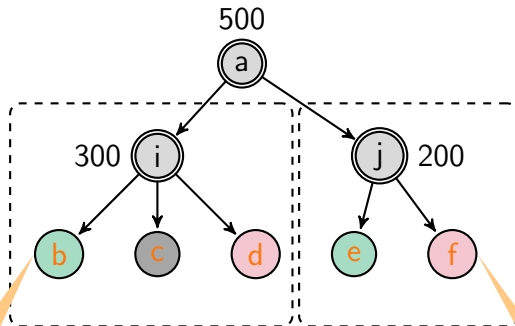
Was sollen i und j beisteuern?





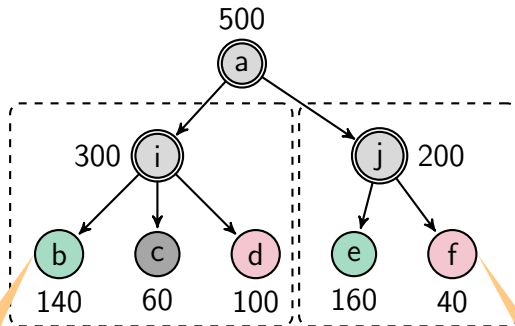


Wie vermeide ich
meinen Speicher
über 90% zu füllen?



Wie vermeide ich
meinen Speicher
über 90% zu füllen?

Wie beschreibe ich
gültige Abläufe?

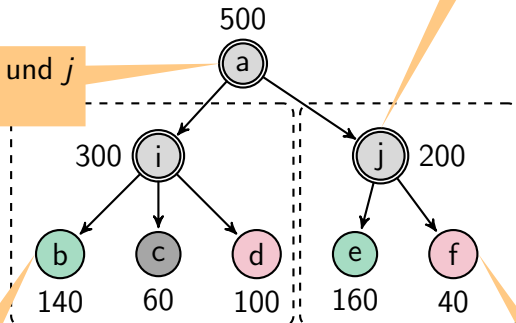


Wie vermeide ich
meinen Speicher
über 90% zu füllen?

Wie beschreibe ich
gültige Abläufe?

Wie kann ich e und f repräsentieren?

Was sollen i und j beisteuern?

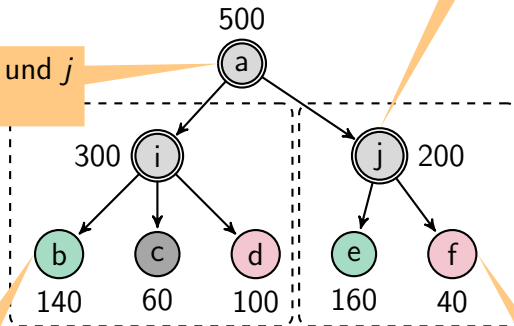


Wie vermeide ich meinen Speicher über 90% zu füllen?

Wie beschreibe ich gültige Abläufe?

Wie kann ich e und f repräsentieren?

Was sollen i und j beisteuern?



Wie vermeide ich
meinen Speicher
über 90% zu füllen?

Constraint Relationships / PVS
SGAI'13, ICTAI'14
Wirsing'15, Constraints'16

Wie beschreibe ich
gültige Abläufe?

Regio-zentrale Fahrpläne

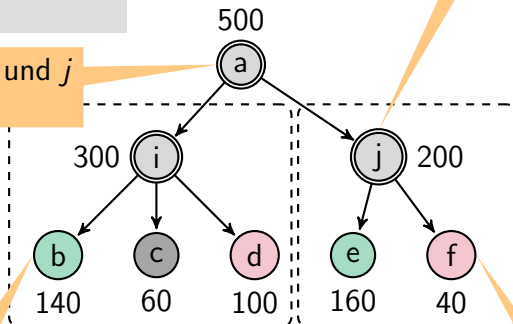
ICAART'14, SAOS'14

Marktbasiert

TAAS'15

Wie kann ich e und f repräsentieren?

Was sollen i und j beisteuern?



Wie vermeide ich meinen Speicher über 90% zu füllen?

Constraint Relationships / PVS

SGAI'13, ICTAI'14

Wirsing'15, Constraints'16

Wie beschreibe ich gültige Abläufe?

Regio-zentrale Fahrpläne

ICAART'14, SAOS'14

Marktbasiert

TAAS'15

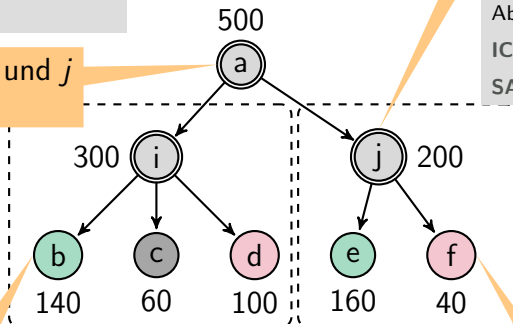
Wie kann ich e und f repräsentieren?

Abstraktion

ICAART'14, TCCI'15

SASO'15

Was sollen i und j
beisteuern?



Wie vermeide ich
meinen Speicher
über 90% zu füllen?

Constraint Relationships / PVS

SGAI'13, ICTAI'14

Wirsing'15, Constraints'16

Wie beschreibe ich
gültige Abläufe?

Regio-zentrale Fahrpläne

ICAART'14, SAOS'14

Marktbasiert

TAAS'15

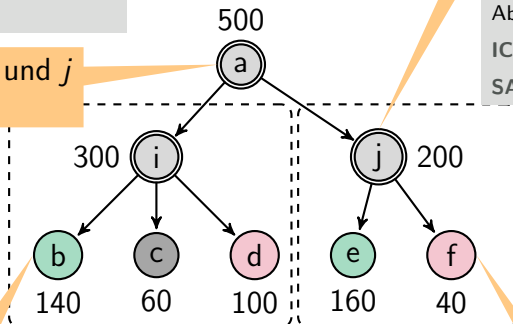
Wie kann ich e und f repräsentieren?

Abstraktion

ICAART'14, TCCI'15

SASO'15

Was sollen i und j
beisteuern?



Supply Automata

SEN-MAS'14

TCCI'15

Wie vermeide ich
meinen Speicher
über 90% zu füllen?

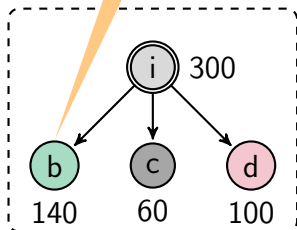
Constraint Relationships / PVS

SGAI'13, ICTAI'14

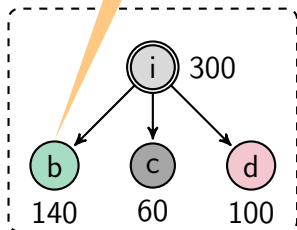
Wirsing'15, Constraints'16

Wie beschreibe ich
gültige Abläufe?

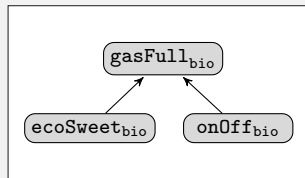
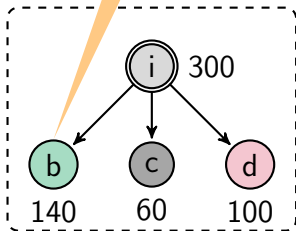
Wie vermeide ich
meinen Speicher
über 90% zu füllen?



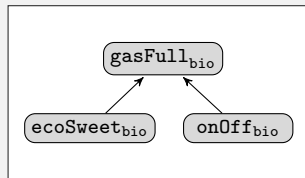
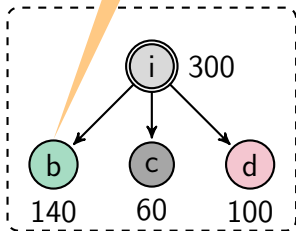
Wie vermeide ich
meinen Speicher
über 90% zu füllen?



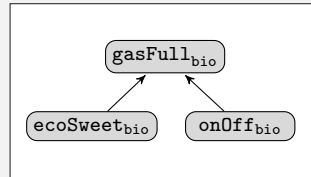
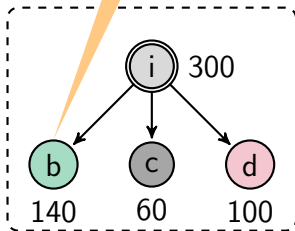
Wie vermeide ich
meinen Speicher
über 90% zu füllen?



Wie vermeide ich
meinen Speicher
über 90% zu füllen?



Wie vermeide ich
meinen Speicher
über 90% zu füllen?



Ziel: Integration von Individualpräferenzen.

Constraint-Problem $((X, D), C)$

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

In der Praxis: unerfüllbare Probleme

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

$$c_1 : x + 1 = y$$

$$c_2 : z = y + 2$$

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

- Nicht alle Constraints können gleichzeitig erfüllt werden
 - z. B., c_2 erzwingt $z = 3$ und $y = 1$, im Konflikt zu c_1
- Ein Agent wählt also zwischen Belegungen, die $\{c_1, c_3\}$ oder $\{c_2, c_3\}$ erfüllen.

Welche Belegungen $v \in [X \rightarrow D]$ sollen bevorzugt werden?

Harte Constraints aus Supply Automata:

$$\text{hardBounds} : \forall t \in T, a \in A : m[a][t] = \text{on} \rightarrow P_{\min} \leq S[a][t] \leq P_{\max}$$

Harte Constraints aus Supply Automata:

$$\text{hardBounds} : \forall t \in T, a \in A : m[a][t] = \text{on} \rightarrow P_{\min} \leq S[a][t] \leq P_{\max}$$

Weiche Constraints anlagenspezifisch (z.B. Präferenz für 350 bis 390 KW):

$$\text{ecoSweet}_{\text{bio}} : \forall t \in T : m[\text{biogas}][t] = \text{on} \rightarrow 350 \leq S[\text{biogas}][t] \leq 390$$

Harte Constraints aus Supply Automata:

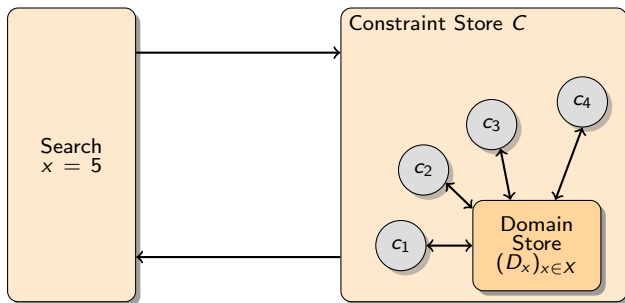
$$\text{hardBounds} : \forall t \in T, a \in A : m[a][t] = \text{on} \rightarrow P_{\min} \leq S[a][t] \leq P_{\max}$$

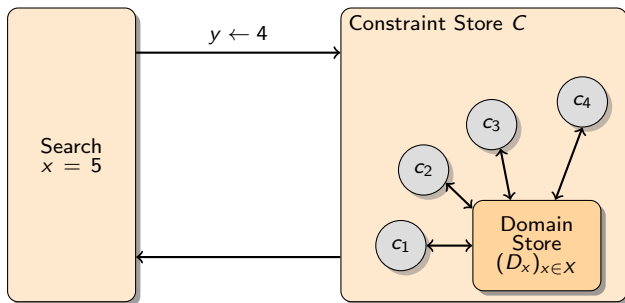
Weiche Constraints anlagenspezifisch (z.B. Präferenz für 350 bis 390 KW):

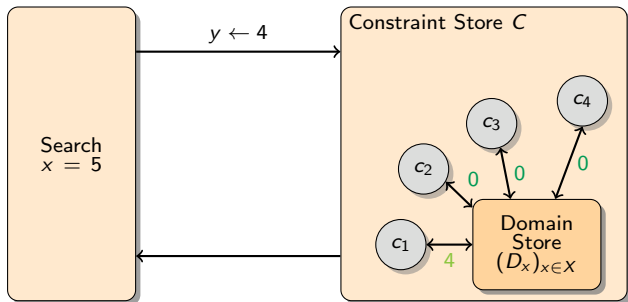
$$\text{ecoSweet}_{\text{bio}} : \forall t \in T : m[\text{biogas}][t] = \text{on} \rightarrow 350 \leq S[\text{biogas}][t] \leq 390$$

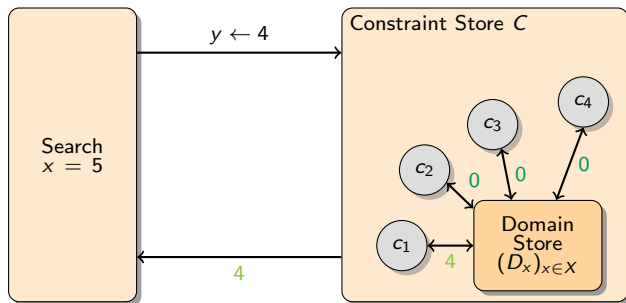
oder Änderungsgeschwindigkeit

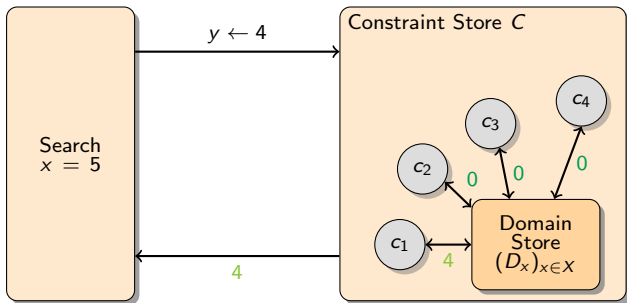
$$\text{inertia}_{\text{therm}} : \forall t \in T : |S[\text{biogas}][t] - S[\text{biogas}][t + 1]| \leq 10$$











- Eine Menge von Bewertungen, z.B., $\{0, \dots, k\}$
- Eine Kombination +
- Ein neutrales Element 0
- Eine partielle Ordnung (\mathbb{N}, \geq) mit 0 als Top

Genannt **valuation structure** (Schiex et al., 1995), bei totaler Ordnung, ansonsten **partial valuation structure** (Gadducci et al., 2013). Ähnlich: (Bistarelli et al., 1999): c-Semiringe

Zugrundeliegende **algebraische Struktur**: Partielle Bewertungsstruktur (*partial valuation structure, partiell geordnetes, kommutatives Monoid*)

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

Abstrakt

- M ... Elemente
- \cdot_M ... Kombination von Bewertungen
- ε_M ... neutrales, "bestes" Element
- \leq_M ... Ordnung, links "schlechter"

Konkret

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

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

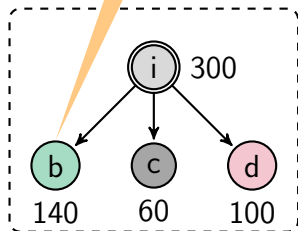
Konkrete PVS-Typen	M	\cdot_M	\leq_M	ε_M
Weighted CSP (WCSP)	\mathbb{N}	$+$	\geq	0
Cost Function Network (CFN)	$\{0, \dots, k\}$	$+/max$	\geq	0
Fuzzy CSP	$[0, 1]$	min	\leq	1
Inclusion Max CSP	2^{C_s}	\cup	\supseteq	\emptyset
Constraint Relationships (CR) ¹	$\mathcal{M}^{fin}(C_s)$	\sqcup	\supseteq_{SPD}	$\{\}$

Hauptidee

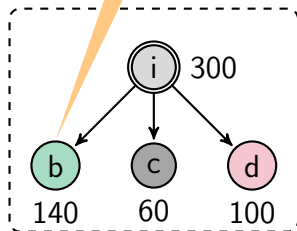
Implementiere Lösungsverfahren für Constraint-Probleme, die durch Bewertungsstrukturen geordnet sind. Instantiiere für konkrete Probleme.

¹ C_s is the set of soft constraints, \supseteq_{SPD} is the SPD-ordering on sets.

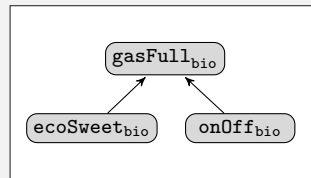
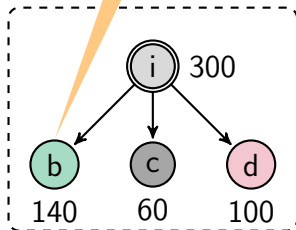
Wie vermeide ich
meinen Speicher
über 90% zu füllen?



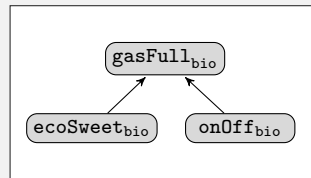
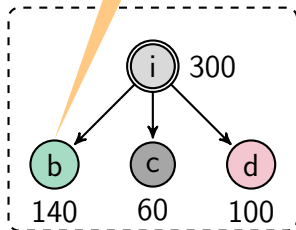
Wie vermeide ich
meinen Speicher
über 90% zu füllen?



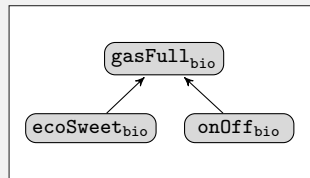
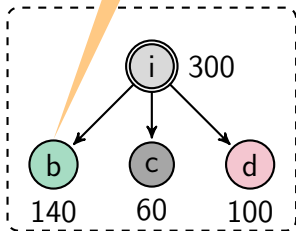
Wie vermeide ich
meinen Speicher
über 90% zu füllen?



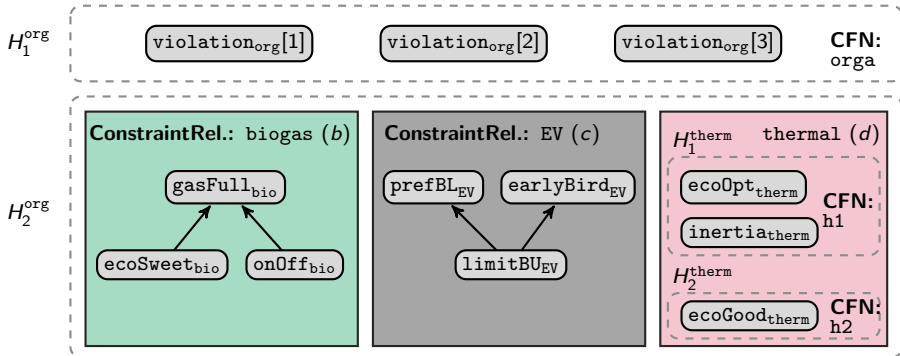
Wie vermeide ich
meinen Speicher
über 90% zu füllen?

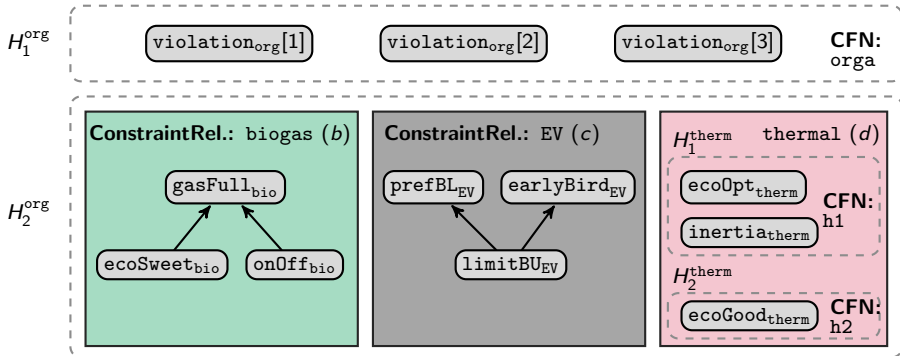


Wie vermeide ich
meinen Speicher
über 90% zu füllen?



Ziel: Integration von Individualpräferenzen.





Die Bewertungsstruktur dieses Problems:

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

Third International CSP Competition (CPAI'08)

Max-CSP and WCSP solvers submitted

Max-CSP: 4 submitted solvers (and a few more versions)

AbsconMax a CSP solver in Java

CSP4J a CSP library in Java

Sugar a SAT based solver

toulbar2 a WCSP solver

WCSP: only one solver submitted (Toulbar)

- the competition has been postponed

toulbar2 ist der einzige dedizierte Weighted-CSP-Solver

Fazit: Nur *ein* Solver für eine Teilklasse der PVS-Probleme!

Eingabeformat: wcsp

SAMPLE-PROB 3 3 3 4

3 3 3

2 0 1 2 2

0 1 0

1 2 0

2 1 2 1 1

0 2 0

2 0 1 1 6

0 0 0

0 1 0

0 2 0

1 0 0

1 1 0

2 0 0

Hoher Gap zwischen Theorie und Praxis (Anwendbarkeit, Modellierungsstärke);
häufig nur Fokus auf Performanz!

Im Constraint Programming:

- Fokus auf *klassischen* Constraint-Lösern
- Erweiterung auf einfache Optimierung (Branch & Bound)
- Zielfunktion kann skalare Variable (`int` oder `float`) sein

In der mathematischen Programmierung:

- Probleme müssen gewisse Struktur aufweisen (lineare Constraints, quadratische Constraints, etc.)
- Schlecht geeignet für beliebige Ordnungen nach denen optimiert werden soll
- Wir planen allerdings mit heterogenen PVS!

Im Constraint Programming:

- Fokus auf *klassischen* Constraint-Lösern
- Erweiterung auf einfache Optimierung (Branch & Bound)
- Zielfunktion kann skalare Variable (`int` oder `float`) sein

In der mathematischen Programmierung:

- Probleme müssen gewisse Struktur aufweisen (lineare Constraints, quadratische Constraints, etc.)
- Schlecht geeignet für beliebige Ordnungen nach denen optimiert werden soll
- Wir planen allerdings mit heterogenen PVS!

Fazit: Meist ad-hoc Kodierungen und geringer Support für die vielfältigen Präferenzformalismen

Rationale

Eine Modellierungssprache – viele Solver

Reduziere Soft-Constraint-Probleme auf konventionelle Constraint-Probleme

- Gecode (CP)
- JaCoP (CP)
- Google Optimization Tools (CP)
- CPLEX (CP/LP/MIP)
- G12 (CP/LP/MIP)
- ...



Optimisation Research Group NICTA


Findet jährlich seit 2008 statt. Klassische Constraint-Probleme in MiniZinc werden an teilnehmende Solver geschickt.

Gewinner 2015

Category	Gold	Silver	Bronze
Fixed	Opturion CPX	OR-Tools	JaCoP
Free	Opturion CPX	IZplus	OR-Tools
Parallel	OR-Tools	Opturion CPX	Choco
Open	sunny-cp	OR-Tools	Opturion CPX




Jahr	Teilnehmende Solver
2015	20
2014	18
2013	10
2012	9




[View on GitHub](#) 

MiniBrass*

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

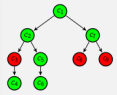
  



Optimization with Preferences

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



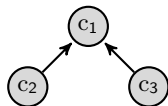
<http://isse-augsburg.github.io/minibrass/>

Basismodell (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();
```

Präferenzmodell (MiniBrass)

```
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: 'true' ;  
    };  
  
solve cr1;
```



Basismodell (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: $\text{mbr_overall_cr1} = \{c2\}$

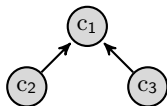
=====

Präferenzmodell (MiniBrass)

```
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: 'true' ;
  };

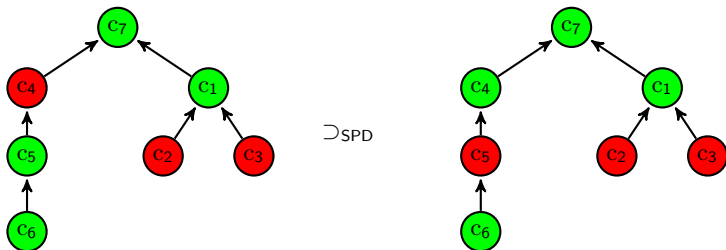
solve cr1;
```



isWorseThan-Relation für Mengen verletztter Constraints (Schiendorfer et al., 2013)

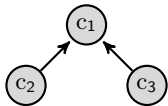
$$V \uplus \{c\} \supset_{\text{SPD}} V$$

$$V \uplus \{c_{\text{gold}}\} \supset_{\text{SPD}} V \uplus \{c_{\text{silber}}\} \quad \text{wenn } c_{\text{silber}} \text{ weniger wichtig als } c_{\text{gold}}$$

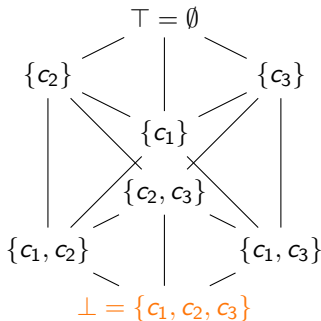


- Bekannt als *Smyth-Ordnung* (Powerdomains) (Amadio and Curien, 1998, Ch. 9)
- Entsteht aus *freier Konstruktion* über Constraint-Relationship.(Knapp et al., 2014)

Der partiell geordnete Bewertungsraum

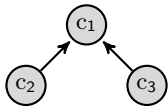


c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';

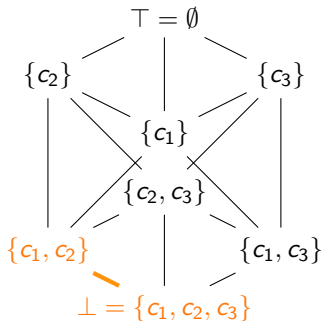


```
function ann: pvs_BAB() =  
  repeat(  
    if next() then  
      print("Intermediate solution:") /\ print() /\  
      commit() /\ postGetBetter()  
    else break endif  
  );
```


Der partiell geordnete Bewertungsraum



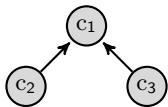
c1: 'x + 1 = y';
c2: 'z = y + 2';
c3: 'x + y <= 3';



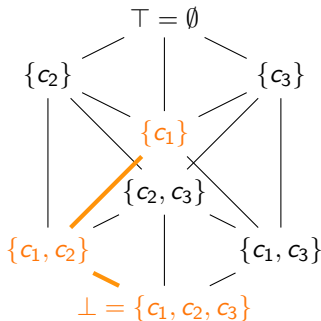
x = 1; y = 1; z = 1
Valuation = {c1, c2}

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

Der partiell geordnete Bewertungsraum



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

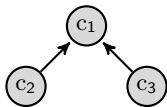


x = 1; y = 1; z = 1
Valuation = {c1, c2}

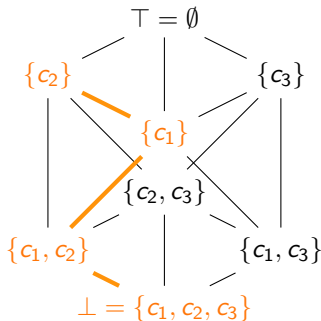
x = 1; y = 1; z = 3
Valuation = {c1}

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

Der partiell geordnete Bewertungsraum



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



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

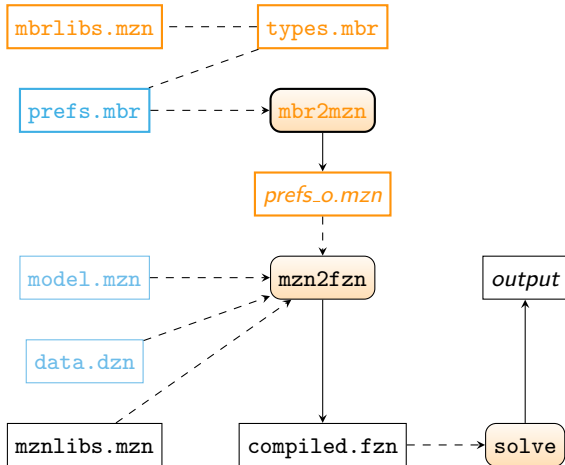
x = 1; y = 1; z = 3
Valuation = {c1}

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

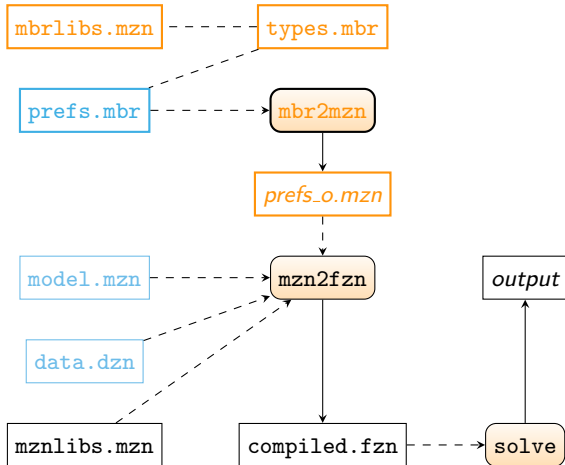
=====

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

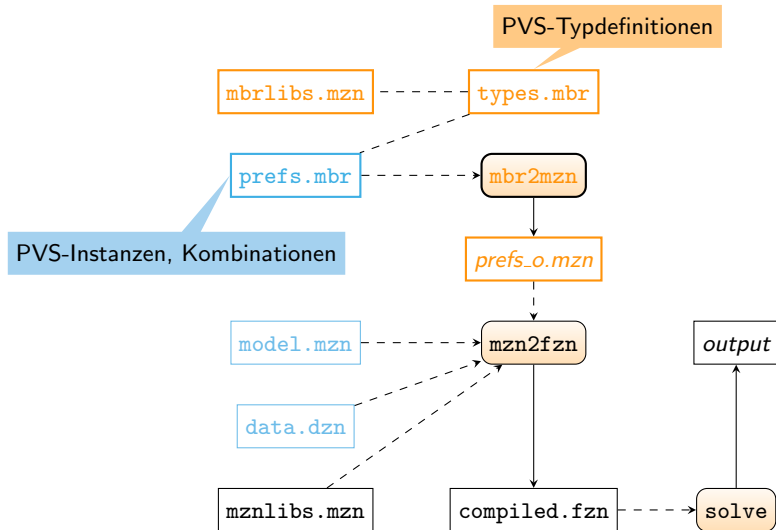
Architektur



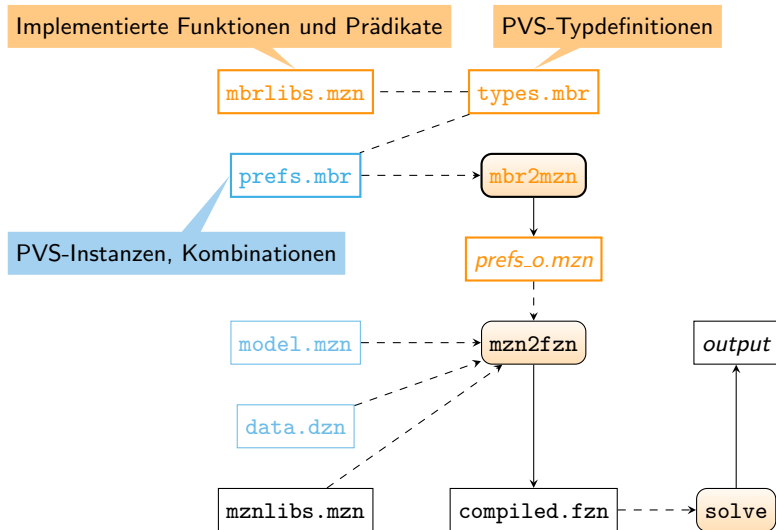
Architektur



Architektur



Architektur



```
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 -> {};  
  };
```

- $PVSType<S, E>$ Unterscheidet zur einfacheren Verwendung zwischen Spezifikationstyp S Elementtyp E
- Kombinationsoperation: $times : S^n \rightarrow E$
- Ordnungsrelation: $isWorse \subseteq E \times E$


```
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' ;  
};
```

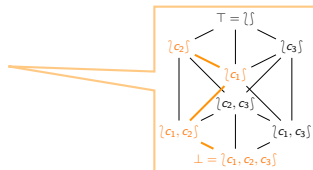
- Jeder Soft-Constraint ein *S*-Ausdruck (hier z.B. bool)
- Mittels der Funktion *times* auf einen *E*-Wert abgebildet
- Ausdrücke in einfachen Anführungszeichen: **MiniZinc**-Code (nicht geparkt, bis auf *mbr.*-Präfixe)
- Parameter aus PVSType müssen Wert erhalten

```
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;  
  };
```

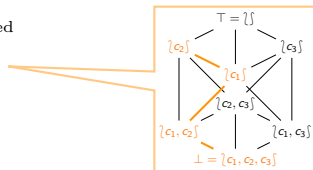
```
PVS: cr1 = new WeightedCsp("cr1") {  
    soft-constraint c1: 'x + 1 = y' :: weights('2');  
    soft-constraint c2: 'z = y + 2' :: weights('1');  
    soft-constraint c3: 'x + y <= 3' :: weights('1');  
  
    k : '20';  
};
```

- Gewichte können direkt an Soft Constraints annotiert werden
- Oder direkt als Feld übergeben werden ([2,1,1])

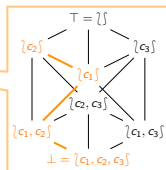
- Unter Umständen wollen wir zwischen PVS-Typen *übersetzen* können
 - Ordnung soll bewusst totalisiert werden (z.B. sonst unübersichtlich)
 - Datentyp xy (z.B. Mengen) wird von Solver/Algorithmus nicht unterstützt (häufig in math. Prog.)
 - \rightarrow Benutzer interessiert **nicht** konkrete Datenstruktur sondern nur die Erhaltung der gewünschten Ordnung
 - Beispiel: Gewichte aus Constraint Relationships für CPLEX
- Daher suchen wir nach *strukturerohaltenden* Abbildungen



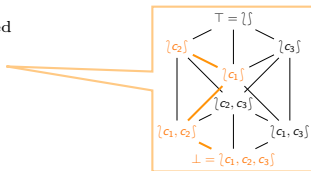
- Unter Umständen wollen wir zwischen PVS-Typen *übersetzen* können
 - Ordnung soll bewusst totalisiert werden (z.B. sonst unübersichtlich)
 - Datentyp xy (z.B. Mengen) wird von Solver/Algorithmus nicht unterstützt (häufig in math. Prog.)
 - \rightarrow Benutzer interessiert **nicht** konkrete Datenstruktur sondern nur die Erhaltung der gewünschten Ordnung
 - Beispiel: Gewichte aus Constraint Relationships für CPLEX
- Daher suchen wir nach *strukturerohaltenden* Abbildungen
- PVS-Homomorphismus $\varphi : \text{PVS}_{\text{cr}} \rightarrow \text{PVS}_{\text{weighted}}$



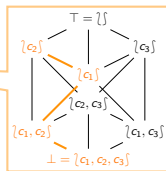
- Unter Umständen wollen wir zwischen PVS-Typen *übersetzen* können
 - Ordnung soll bewusst totalisiert werden (z.B. sonst unübersichtlich)
 - Datentyp xy (z.B. Mengen) wird von Solver/Algorithmus nicht unterstützt (häufig in math. Prog.)
 - \rightarrow Benutzer interessiert **nicht** konkrete Datenstruktur sondern nur die Erhaltung der gewünschten Ordnung
 - Beispiel: Gewichte aus Constraint Relationships für CPLEX
- Daher suchen wir nach *strukturerohaltenden* Abbildungen
- PVS-Homomorphismus $\varphi : PVS_{cr} \rightarrow PVS_{weighted}$
- $PVS_{cr} = PVS\langle P \rangle = \langle \mathcal{M}^{fin}(P), \cup, \supseteq_{SPD}, \{ \} \rangle$
 - $\varphi(T_{cr}) = T_{weighted}$



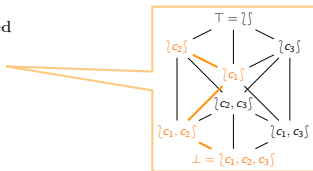
- Unter Umständen wollen wir zwischen PVS-Typen *übersetzen* können
 - Ordnung soll bewusst totalisiert werden (z.B. sonst unübersichtlich)
 - Datentyp xy (z.B. Mengen) wird von Solver/Algorithmus nicht unterstützt (häufig in math. Prog.)
 - \rightarrow Benutzer interessiert **nicht** konkrete Datenstruktur sondern nur die Erhaltung der gewünschten Ordnung
 - Beispiel: Gewichte aus Constraint Relationships für CPLEX
- Daher suchen wir nach *strukturerohaltenden* Abbildungen
- PVS-Homomorphismus $\varphi : PVS_{cr} \rightarrow PVS_{weighted}$
- $PVS_{cr} = PVS\langle P \rangle = \langle \mathcal{M}^{fin}(P), \cup, \supseteq_{SPD}, \perp \rangle$
 - $\varphi(T_{cr}) = T_{weighted}$
 - $\varphi(m \cdot_{cr} n) = \varphi(m) \cdot_{weighted} \varphi(n)$

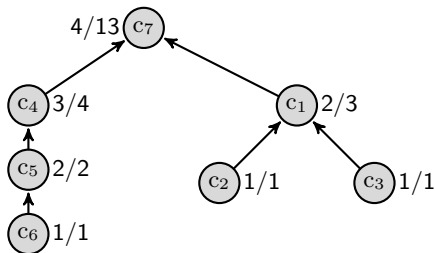


- Unter Umständen wollen wir zwischen PVS-Typen *übersetzen* können
 - Ordnung soll bewusst totalisiert werden (z.B. sonst unübersichtlich)
 - Datentyp xy (z.B. Mengen) wird von Solver/Algorithmus nicht unterstützt (häufig in math. Prog.)
 - \rightarrow Benutzer interessiert **nicht** konkrete Datenstruktur sondern nur die Erhaltung der gewünschten Ordnung
 - Beispiel: Gewichte aus Constraint Relationships für CPLEX
- Daher suchen wir nach *strukturerohaltenden* Abbildungen
- PVS-Homomorphismus $\varphi : PVS_{cr} \rightarrow PVS_{weighted}$
- $PVS_{cr} = PVS\langle P \rangle = \langle \mathcal{M}^{fin}(P), \cup, \supseteq_{SPD}, \perp \rangle$
 - $\varphi(T_{cr}) = T_{weighted}$
 - $\varphi(m \cdot_{cr} n) = \varphi(m) \cdot_{weighted} \varphi(n)$
 - $m \leq_{cr} n \rightarrow \varphi(m) \leq_{weighted} \varphi(n)$



- Unter Umständen wollen wir zwischen PVS-Typen *übersetzen* können
 - Ordnung soll bewusst totalisiert werden (z.B. sonst unübersichtlich)
 - Datentyp xy (z.B. Mengen) wird von Solver/Algorithmus nicht unterstützt (häufig in math. Prog.)
 - \rightarrow Benutzer interessiert **nicht** konkrete Datenstruktur sondern nur die Erhaltung der gewünschten Ordnung
 - Beispiel: Gewichte aus Constraint Relationships für CPLEX
- Daher suchen wir nach *strukturerohaltenden* Abbildungen
- PVS-Homomorphismus $\varphi : PVS_{cr} \rightarrow PVS_{weighted}$
- $PVS_{cr} = PVS\langle P \rangle = \langle \mathcal{M}^{fin}(P), \cup, \supseteq_{SPD}, \perp \rangle$
 - $\varphi(T_{cr}) = T_{weighted}$
 - $\varphi(m \cdot_{cr} n) = \varphi(m) \cdot_{weighted} \varphi(n)$
 - $m \leq_{cr} n \rightarrow \varphi(m) \leq_{weighted} \varphi(n)$





Beispiel mit errechneten Gewichten (SPD/TPD)

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

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

$T = 0$

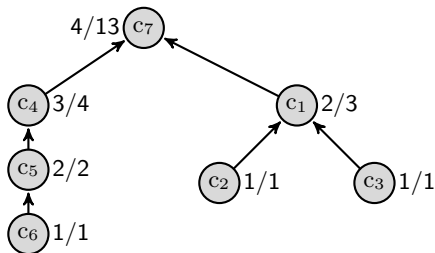
1

...

$k - 2$

$k - 1$

$\perp = k$



Beispiel mit errechneten Gewichten (SPD/TPD)

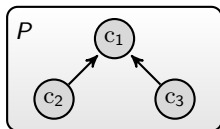
$$w^{\text{SPD}}(c) = 1 + \max_{c' \rightarrow c} w^{\text{SPD}}(c')$$

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

- Für Ordnung P über Constraints: $\text{PVS Weighted}(P) = \langle \mathbb{N}, +, \geq, 0 \rangle$
- $\text{PVS}_{\text{cr}} = \text{PVS}\langle P \rangle = \langle \mathcal{M}^{\text{fin}}(P), \cup, \supseteq_{\text{SPD}}, \downarrow \rangle$
- $\varphi(\downarrow) = 0$, $\varphi(\downarrow c \downarrow \cup C) = w^{\text{SPD}}(c) + \varphi(C)$

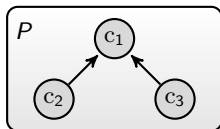
$T = 0$
|
1
|
⋮
|
 $k-2$
|
 $k-1$
|
 $\perp = k$

(Schiendorfer et al., 2013)



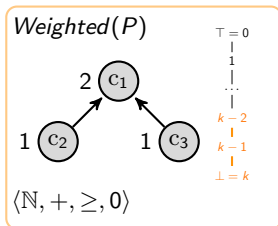
Cat : POSet

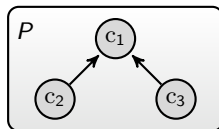
Cat : PVS



Cat : POSet

Cat : PVS



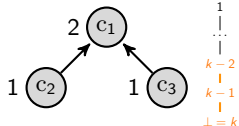


Cat : POSet

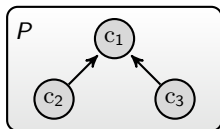
Cat : PVS

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

Weighted(P)



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

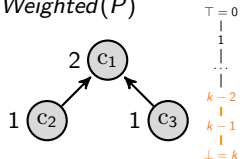


Cat : POSet

Cat : PVS

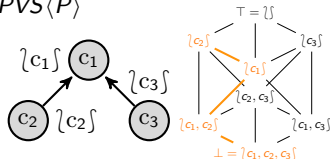
$$\mu(c) = w^{\text{SPD}}(c)$$

Weighted(P)

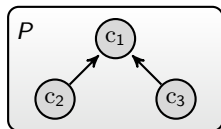


$\langle \mathbb{N}, +, \geq, 0 \rangle$

PVS(P)



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



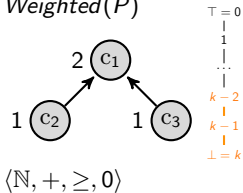
Cat : POSet

Cat : PVS

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

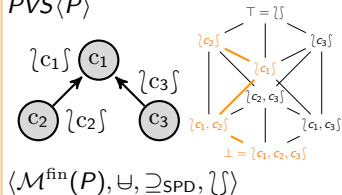
$$\eta(c) = \{c\}$$

Weighted(P)

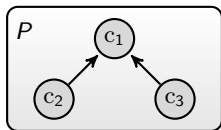


$\langle \mathbb{N}, +, \geq, 0 \rangle$

PVS(P)



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



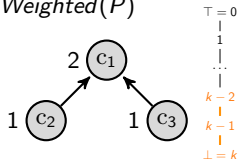
Cat : POSet

Cat : PVS

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

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

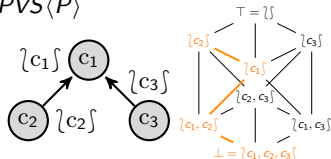
$Weighted(P)$



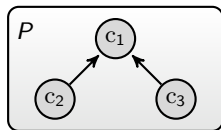
$\langle \mathbb{N}, +, \geq, 0 \rangle$

$$(w^{\text{SPD}})^{\#}$$

$PVS\langle P \rangle$



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



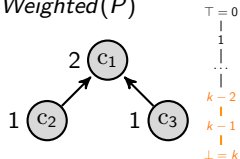
Cat : POSet

Cat : PVS

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

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

Weighted(P)

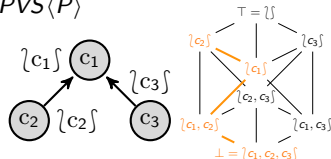


$\langle \mathbb{N}, +, \geq, 0 \rangle$

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



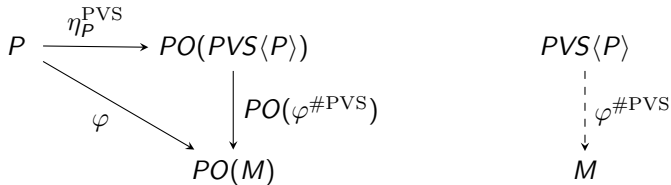
PVS(P)



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

Lemma (PVS-Freiheit (Knapp and Schiendorfer, 2014))

PVS⟨P⟩ is the free partial valuation structure over the partial order P.



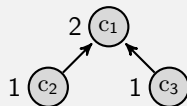
Freie Konstruktionen

- no junk
- no confusion

```
% aus Bibliothek
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: ' $x + y \leq 3$ ';

Mit PVSs M und N können wir das direkte Produkt $M \times N$

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

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
```

Außerdem das **lexikographische** Produkt $M \ltimes N$

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

Ermöglicht strikte Hierarchien

```
% 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
```

MiniBrass wurde für verschiedene Anwendungen eingesetzt:

- Studenten-Mentoren-Matching
- Prüfungsterminfindung
- Energiefallstudie
- Multi-User-Multi-Display
- Rekonfigurierbare Roboterteams
- (Rollenallokation im ODP, dzt. nur harte Constraints)

MiniBrass wurde für verschiedene Anwendungen eingesetzt:

- Studenten-Mentoren-Matching
- Prüfungsterminfindung
- Energiefallstudie
- Multi-User-Multi-Display
- Rekonfigurierbare Roboterteams
- (Rollenallokation im ODP, dzt. nur harte Constraints)

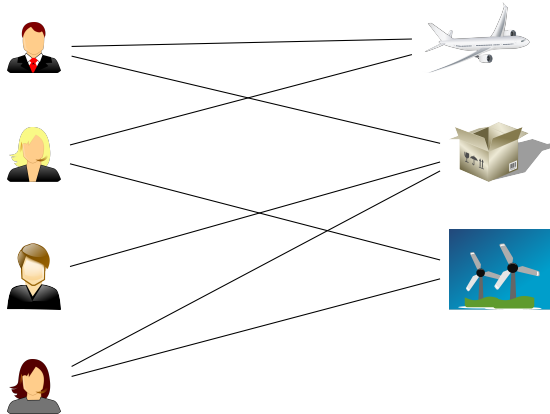
Ziel: Teile Mentees (z.B. Studenten) Mentoren zu (z.B. Firmen), sodass

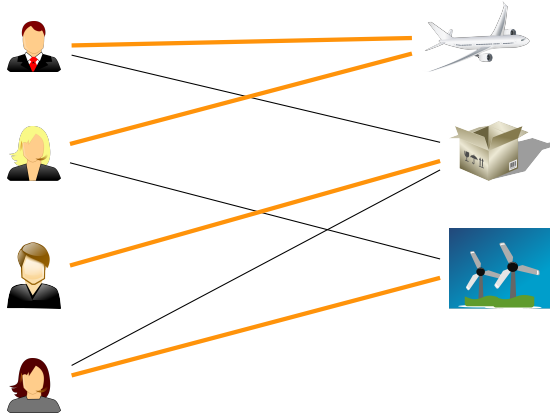
- Studenten sind sehr zufrieden mit ihren Mentoren
- Firmen sind mit ihren Mentees ebenfalls zufrieden
- Zweiseitige Präferenzen

Bisher klingt das wie ein typisches *Stable Matching*-Problem, aber:

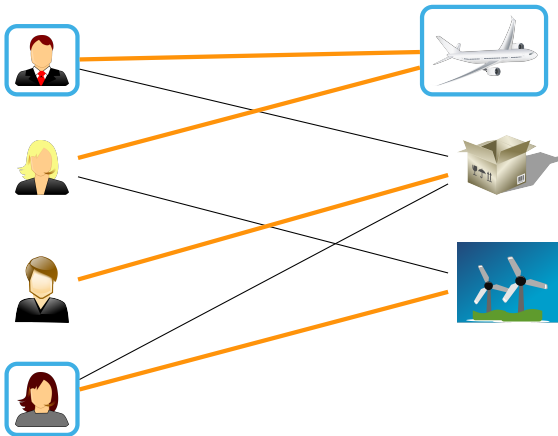
- Es gibt keine 1:1 Abbildung (Firmen betreuen mehrere Studenten)
- Zusätzliche Constraints sind vorhanden:
 - Jede Firma betreut zumindest l , höchstens aber u Studenten
 - Die Anzahl betreuter Studenten *sollten* ungefähr gleich sein pro Firma (Fairness)
 - Studenten, die eine Firma "verachten", sollen nicht gezwungen werden (*harder Ausschluss* von Lösungen)

Mentor Matching: Beispiel

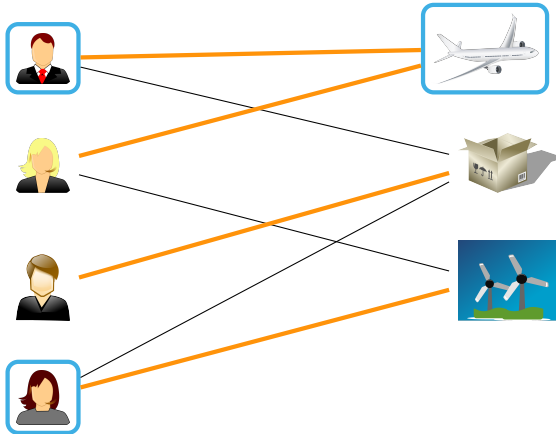




Diese **Zuweisung** respektiert die studentischen Präferenzen (Kanten)



Diese **Zuweisung** respektiert die studentischen Präferenzen (Kanten) ignoriert aber die **Firmenpräferenzen**.



Diese **Zuweisung** respektiert die studentischen Präferenzen (Kanten) ignoriert aber die **Firmenpräferenzen**. OK, es ist nicht wirklich ein *Matching* da Firmen mehr als einen Studenten betreuen ...

```
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();
```

```
% fmsoft2016.mzn

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

% student names for better readability
int: raubholz = 1;
int: schraubale = 2;
int: meerfluss = 3;
int: gleich = 4;
int: lustig = 5;

% company names
int: delphi = 1;
int: cupgainini = 2;
int: youthlab = 3;
```

```
PVS: students = new ConstraintRelationships("students") {
    soft-constraint raubholzdelphi: 'worksAt[raubholz] = delphi';
    soft-constraint raubholzyouthlab: 'worksAt[raubholz] = youthlab';
    soft-constraint gleichcupg: 'worksAt[gleich] = cupgainini';

    crEdges : '[| mbr.raubholzyouthlab, mbr.raubholzdelphi |
                mbr.gleichcupg, mbr.raubholzdelphi |]';
    useSPD: 'true' ;
};

PVS: companies = new ConstraintRelationships("companies") {
    soft-constraint delphi_meer: 'worksAt[meerfluss] = delphi';
    soft-constraint delphi_gleich: 'worksAt[gleich] = delphi';
    soft-constraint youthlab: 'worksAt[lustig] = youthlab';

    crEdges : '[| mbr.delphi_meer, mbr.delphi_gleich |]';
    useSPD: 'true' ;
};
```



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

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

```
Valuations: pen_companies = 1; pen_students = 3
```

```
-----
```

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

```
Valuations: pen_companies = 2; pen_students = 2
```

```
-----
```

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

```
Valuations: pen_companies = 3; pen_students = 1
```

```
-----
```

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

```
Valuations: pen_companies = 2; pen_students = 1
```

```
-----
```

```
=====
```

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

```
Intermediate solution:worksAt = [3, 2, 1, 1, 1]  
Valuations: pen_companies = 1; pen_students = 3  
-----
```

```
Intermediate solution:worksAt = [2, 1, 1, 1, 3]  
Valuations: pen_companies = 0; pen_students = 4  
-----
```

```
Intermediate solution:worksAt = [1, 2, 1, 1, 3]  
Valuations: pen_companies = 0; pen_students = 2  
-----  
=====
```

- Präferenzen aus E-Mails vom WS 15/16 gesammelt

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

- Priorität zu **Studenten**
 - Was sollen Firmen schon mit unzufriedenen Mentees anfangen?
- Suchraum: 7 Firmen für 16 Studenten $\rightarrow 7^{16} = 3.3233 \cdot 10^{13}$
- Führte zu einem Constraint-Problem mit
 - 77 student. Präferenzen (Soft Constraints) von 16 Studenten
 - insgesamt 114 Soft Constraints (37 Firmenpräferenzen)
- *Bewiesen* optimale Lösung
 - 6 Minuten Lösungszeit

Ziel: Weise Prüfungstermine an Studenten zu, sodass

- Jeder Student stimmt seinem Termin zu
- Die Anzahl verschiedener Termine wird minimiert (um das Zeitinvestment der Dozenten zu schonen)



At least 3 options have to be selected

		Approve	Absolutely not
12 February 2016	Morning	<input type="radio"/>	<input type="radio"/>
12 February 2016	Afternoon	<input type="radio"/>	<input type="radio"/>
18 February 2016	Morning	<input type="radio"/>	<input type="radio"/>
18 February 2016	Afternoon	<input type="radio"/>	<input type="radio"/>
...	...	<input type="radio"/>	<input type="radio"/>
Name			

- Kein studentischer Wunsch sollte höher gewichtet werden
- Prüfungsplan ist eine gemeinsame Entscheidung

```
% 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: sd;

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

```
include "../defs.mbr";
PVS: students = new WeightedCsp("students") {
  k: '100';
  soft-constraint raubholz: 'sd[raubholz] in {monday, tuesday}';
  soft-constraint schraubale: 'sd[schraubale] in {tuesday, wednesday}';
  soft-constraint meerfluss: 'sd[meerfluss] in {tuesday}';
  soft-constraint gleich: 'sd[gleich] in {monday, tuesday}';
  soft-constraint lustig: 'sd[lustig] in {monday, wednesday}';
  % hard by weight (less than bottom)
  soft-constraint lustig-urlaub: 'sd[lustig] != tuesday'
    :: weights('101');
};
PVS: teachers = new CostFunctionNetwork("teachers") {
  soft-constraint scheduledDates: 'scheduledDates';
};
solve students lex teachers;
```

Scheduled: [1, 2, 2, 1, 1], Distinct dates: 2

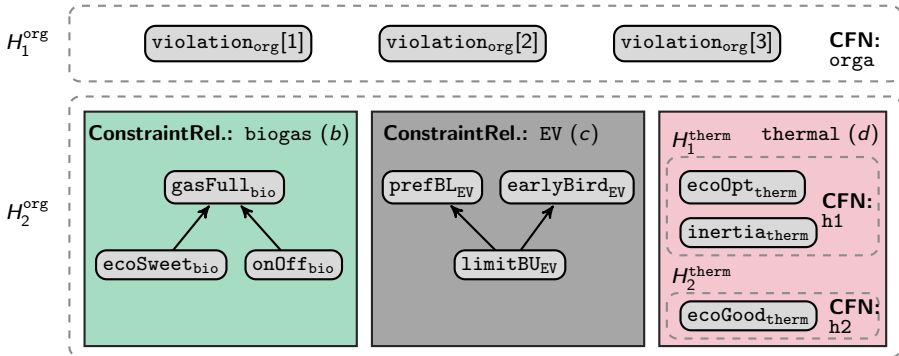
Valuations: mbr_overall_students = 0; mbr_overall_teachers = 2

- Gesammelte Präferenzen von 33 Studenten
- 12 mögliche Termine (6 Tage, Vormittag und Nachmittag)
 - *Approval*-Menge
 - *Impossible*-Menge
- Aggregiert via Wahl durch Zustimmung (**Approval voting**), hat ansprechende wahltheoretische Eigenschaften (Arrow)!
- Höchstens 4 pro Termin
- Sofort (61 msec) wurde eine optimale Lösung gefunden, die
 - von *jedem* Student Zustimmung erhält
 - Mit der Minimalanzahl von 9 Terminen auskommt
- Verwendete Strategie (natürlich, ...):

```
solve students lex teachers;% pro students
```


Ziel: Weise Kraftwerken Produktion zu, sodass

- Der Bedarf gedeckt wird
- Steuerungsvorlieben (ökonomische Leistungsbereiche, ...) eingehalten werden



```
int: T = 5; set of int: WINDOW = 1..T;
array[WINDOW] of int: demand = [20, 21, 25, 30, 29];

int: P = 3; set of int: PLANTS = 1..P;

array[PLANTS] of int: pMin = [12, 5, 7];
array[PLANTS] of int: pMax = [15, 11, 9];

array[WINDOW, PLANTS] of var 0..15: supply;
constraint forall(p in PLANTS, w in WINDOW)
    (supply[w,p] in pMin[p]..pMax[p]);

array[WINDOW] of var int: violation =
    [ abs( sum(p in PLANTS) (supply[w, p]) - demand[w] ) | w in WINDOW];

solve search pvs_BAB();
```

```
PVS: orga = new CostFunctionNetwork("Orga") {
  soft-constraint vio_1: 'violation[1]';
  soft-constraint vio_2: 'violation[2]';
  soft-constraint vio_3: 'violation[3]';
  isWorstCase: 'true';
};

PVS: biogas = new ConstraintRelationships("biogas") {
  soft-constraint gasFull:
    'forall(w in WINDOW) (supply[w,biogas] >= 13)';
  soft-constraint ecoSweet:
    'forall(w in WINDOW) (supply[w,biogas] >= 14)';
  soft-constraint onOff:
    'forall(w in 1..T-1) (
      abs(supply[w,biogas] - supply[w+1,biogas]) <= 1)';

  crEdges : '[| mbr.ecoSweet, mbr.gasFull | mbr.onOff, mbr.gasFull |]';
  useSPD: 'true' ;
};
```

```
PVS: ev = new ConstraintRelationships("ev") {...};

PVS: therm1 = new CostFunctionNetwork("therm1") {
  soft-constraint ecoOpt:
    'sum(w in WINDOW) ( abs(supply[w,thermal] - 8) )';
  soft-constraint inertia:
    'sum(w in 1..T-1) ( abs(supply[w,thermal] - supply[w+1,thermal]))';
};

PVS: therm2 = new CostFunctionNetwork("therm2") {
  soft-constraint ecoGood:
    'sum(w in WINDOW) ( abs(supply[w,3] - 9) )';
};

solve orga lex ( biogas pareto ev pareto (therm1 lex therm2) );
```

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

- Direkter Vergleich schwierig
- Kein anderes System implementiert PVS (oder c-Semiringe)
- → Dafür unterstützt `toulbar2` Weighted CSP und *Cost Function Networks* (der einzig frei verfügbare *Soft-Constraint-Solver*)

²<https://github.com/MiniZinc/minizinc-benchmarks>

- Direkter Vergleich schwierig
- Kein anderes System implementiert PVS (oder c-Semiringe)
- → Dafür unterstützt toulbar2 Weighted CSP und *Cost Function Networks* (der einzig frei verfügbare *Soft-Constraint-Solver*)
- Daher Beschränkung auf *Weighted CSP* für die Evaluierung
- Abbildung aus Constraint-Relationships
- Probleme sind Variationen von 5 Problemen aus den MiniZinc-Benchmarks² (erweitert um Soft Constraints in PVS)

²<https://github.com/MiniZinc/minizinc-benchmarks>

- Eingesetzte Solver
 - JaCoP
 - Gecode
 - Google OR-Tools
 - G12
 - Choco
 - toulbar2
- Eingesetzte Probleme (insgesamt 15 Instanzen)
 - Soft-Queens
 - Photo-Platzierung
 - Talent-Scheduling
 - On-Call-Rostering
 - Multi-Skilled Project Scheduling Problem
- Timeout: 10 Minuten

Eingabeformat: wcsp

SAMPLE-PROB 3 3 3 4

3 3 3

2 0 1 2 2

0 1 0

1 2 0

2 1 2 1 1

0 2 0

2 0 1 1 6

0 0 0

0 1 0

0 2 0

1 0 0

1 1 0

2 0 0

Eingabeformat: wcsp

SAMPLE-PROB 3 3 3 4

3 3 3

2 0 1 2 2

0 1 0

1 2 0

2 1 2 1 1

0 2 0

2 0 1 1 6

0 0 0

0 1 0

0 2 0

1 0 0

1 1 0

2 0 0

$\Pi \cup M \beta \Sigma \mathbb{R}$
 $J \Lambda \subset K$

Fixing issues with large-domain variables from FlatZinc in
toulbar2 #35


 [9ahbit](#) merged 6 commits into [eomahony:master](#) from [Alexander-Schiendorfer:master](#) on Feb 14

Photo-Platzierung und Soft-Queens

Werte in Klammern geben die beste gefundene Lösung nach Timeout an; Da Weighted CSP → Minimierung. Zeiten in Sekunden.

	OR-Tools	Gecode	Choco	JaCoP	G12	toulbar2
Photo						
photo1	0.18	0.19	0.41	0.54	3.38	0.4
photo2	1.06	2.98	0.52	2.92	35.3	0.55
Soft-Queens						
$n = 8$	0.03	0.03	0.46	0.18	0.03	0.27
$n = 16$	0.03	0.04	0.5	0.22	0.05	0.28
$n = 30$	0.04 (0)	600 (4)	0.55 (0)	187.86 (0)	600 (4)	0.58 (0)

Talent-Scheduling und On-Call Rostering

Werte in Klammern geben die beste gefundene Lösung nach Timeout an; Da Weighted CSP → Minimierung. Zeiten in Sekunden.

	OR-Tools	Gecode	Choco	JaCoP	G12	toulbar2
Talents						
small	0.03	0.03	0.35	0.16	0.04	2.28
concert	0.05	0.05	0.47	0.24	0.07	16.98
film103	2.23	67.69	7.48	3.01	9.3	—
Rostering						
4s-10d ³	0.14	0.17	1.53	0.64	0.22	0.81
4s-23d	2.59	2.92	5.68	4.06	4.49	3.98
10s-50d	600 (6)	600 (6)	600 (14)	600 (10)	600 (10)	87.18 (1)

³s bezeichnet "staff", d "days"

Multi-Skilled Project Scheduling Problem

Werte in Klammern geben die beste gefundene Lösung nach Timeout an; Da Weighted CSP → Minimierung. Zeiten in Sekunden.

	OR-Tools	Gecode	Choco	JaCoP	G12	toulbar2
MSPSP						
easy_01	0.2	0.32	1.26	0.94	0.27	–
medium_01	0.19	0.22	1.35	0.67	0.21	–
hard_02	0.37	0.33	1.59	1.01	0.37	600 (–)
hard_04	0.27	0.25	1.62	0.86	0.28	–

- ① Weitgehende algebraische Konzepte zur Modellierung von Präferenzstrukturen in der Literatur
- ② Constraint Relationships und Weiterentwicklung von PVS-Produkten für Hierarchien ✓

- ❶ Weitgehende algebraische Konzepte zur Modellierung von Präferenzstrukturen in der Literatur
- ❷ Constraint Relationships und Weiterentwicklung von PVS-Produkten für Hierarchien ✓
- ❸ Es gibt genau einen dedizierten Soft-Constraint-Solver, `toulbar2` für weighted CSP; keine Sprache, nur Binärformat

- ❶ Weitgehende algebraische Konzepte zur Modellierung von Präferenzstrukturen in der Literatur
- ❷ Constraint Relationships und Weiterentwicklung von PVS-Produkten für Hierarchien ✓
- ❸ Es gibt genau einen dedizierten Soft-Constraint-Solver, `toulbar2` für weighted CSP; keine Sprache, nur Binärformat
- ❹ Modellierungssprache für Präferenzen entwickelt, die Soft-Constraints auf klassische Constraints reduziert ✓
 - Kann auch Weighted Probleme direkt für `toulbar2` übersetzen

- ① Weitgehende algebraische Konzepte zur Modellierung von Präferenzstrukturen in der Literatur
- ② Constraint Relationships und Weiterentwicklung von PVS-Produkten für Hierarchien ✓
- ③ Es gibt genau einen dedizierten Soft-Constraint-Solver, `toulbar2` für weighted CSP; keine Sprache, nur Binärformat
- ④ Modellierungssprache für Präferenzen entwickelt, die Soft-Constraints auf klassische Constraints reduziert ✓
 - Kann auch Weighted Probleme direkt für `toulbar2` übersetzen
- ⑤ Evaluierung demonstriert Vorteile durch Solver-Unabhängigkeit: Für z.B. Scheduling-Probleme sind Reduktionen schneller als dedizierte Soft-Constraint-Solver

Konzepte, Sprachdesign MiniBrass

- AS, Alexander Knapp, Gerrit, Oliver

Anwendungen, Multiagenten-Einsatz

- Alexander Schubert (MSc-Thesis: Einsatz von Voting-Verfahren), Markus Tolls (MSc-Thesis: Formalisierung von Task-Allocation-Problemen)
- TeamBots (Ludwig, Miroslav)
- Testen von SO-Systemen (Benedikt, Hella, Axel)

Outreach

- Vortrag Helmholtz-Zentrum München
- Vortrag FH Hagenberg
- Tutorial @ SASO 2016

Amadio, R. M. and Curien, P.-L. (1998).

Domains and Lambda-Calculi.

Cambridge Tracts in Theoretical Computer Science 46. Cambridge University Press.

Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., and Fargier, H. (1999).

Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison.

Constraints, 4(3):199–240.

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.

Knapp, A. and Schiendorfer, A. (2014).

Embedding Constraint Relationships into C-Semirings.

Technical Report 2014-03, Institute for Software and Systems Engineering,
University of Augsburg.

[http://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/
index/index/docId/2684](http://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/index/index/docId/2684).

Knapp, A., Schiendorfer, A., and Reif, W. (2014).

Quality over Quantity in Soft Constraints.

In *Proc. 26th Int. Conf. Tools with Artificial Intelligence (ICTAI'2014)*,
pages 453–460.

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.

Schiex, T., Fargier, H., and Verfaillie, G. (1995).

Valued Constraint Satisfaction Problems: Hard and Easy Problems.

In *Proc. 14th Int. Joint Conf. Artificial Intelligence (IJCAI'95)*, Vol. 1, pages 631–639. Morgan Kaufmann.