MiniBrass: Soft Constraint Programming

Alexander Schiendorfer et al.

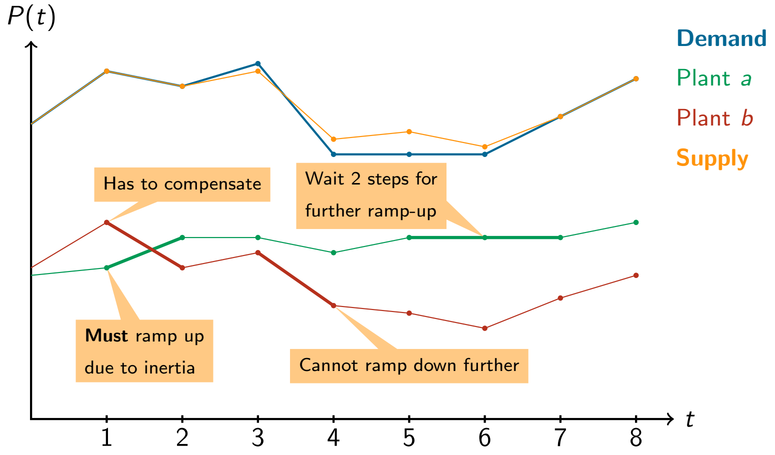
Universität Augsburg University

Einordnung

Wo wir letztes Jahr waren ...

Fahrplanerstellungsproblem

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



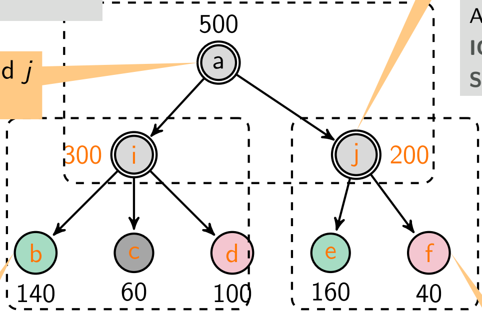
Hierarchisches Energiemanagement

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

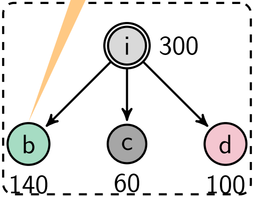
Wie vermeide ich meinen Speicher über 90% zu füllen?

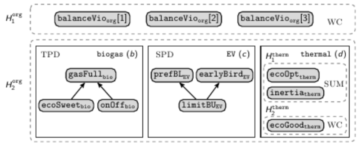
Constraint Relationships / PVS
SGAI'13, ICTAI'14
Wirsing'15, Constraints'16

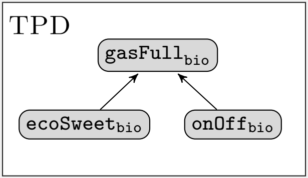
Wie beschreibe ich meine Abläufe?

Constraint Relationships

Wie vermeide ich meinen Speicher über 90% zu füllen?







Ziel: Integration von Individualpräferenzen.

Präferenzen im Constraint-Solving

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

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

In der Praxis: unerfüllbare Probleme

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

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

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

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

(Soft) Constraints in der Energie

Institute for
Software & Systems
Engineering

Harte Constraints aus Supply Automata:

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

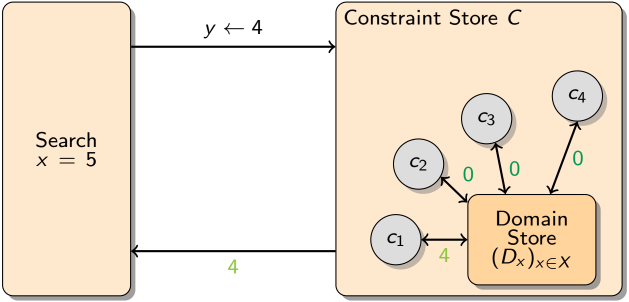
Weiche Constraints anlagenspezifisch (z.B. Präferenz für 350 bis 390 KW):

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

oder Änderungsgeschwindigkeit

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

Soft-Constraint-Solving



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

Genannt valuation structure (**?**), bei totaler Ordnung, ansonsten partial valuation structure (**?**). Ähnlich: (**?**): c-Semiringe

Partial Valuation Structures

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

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

Abstrakt

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

Konkret

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

(**?**; **?**)

PVS–Idee

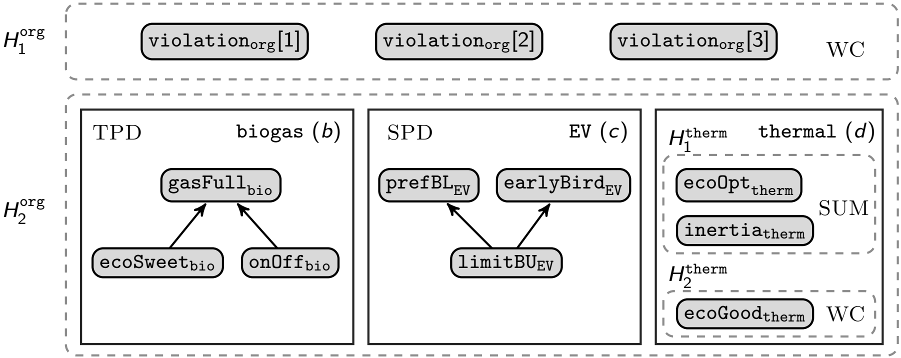
Konkrete PVS	M	$\oplus_M$	$\leq_M$	ε_M
Weighted CSP (WCSP)	$\mathbb{N}$	$+$	$\geq$	0
Fuzzy CSP	$[0,1]$	min	$\leq$	1
Inclusion Max CSP	2^{C_s}	$\cup$	$\supseteq$	$\emptyset$
Constraint Relationships[1]	$\mathcal{M}^{\text{fin}}(C_s)$	$\cup$	$\supseteq_{\text{SPD}}$	$\emptyset$

Hauptidee

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

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

Kombinationen (?)



Die Präferenzstruktur dieses Problems:

$$V_{\mathrm{org}_1} \ltimes (P_{\mathtt{biogas}} \times P_{\mathtt{EV}} \times (P^1_{\mathtt{thermal}} \ltimes P^2_{\mathtt{thermal}}))$$

SPD . . . Single-Pred.-Dom.
TPD . . . Single-Pred.-Dom.
SUM . . . Sum of Errors
WC . . . Worst-Case Error

Praxis I

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 submittted (Toulbar)

- the competion has been postponed

15/75

Praxis II

Im Constraint Programming:

- Fokus auf *klassischen* Constraint-Lösern
- Erweiterung auf einfache Optimierung (Branch & Bound)
- Zielfunktion kann skalare Variable (`int` oder `float`) sein
- `toulbar2` ist der einzige dedizierte Weighted-CSP-Solver

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 wollen aber heterogene PVS → MiniZinc

Warum MiniZinc?

Institute for Software & Systems Engineering

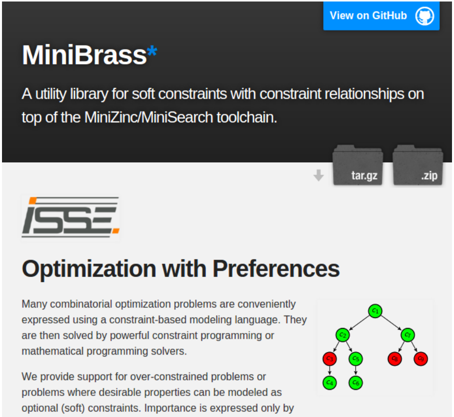
Rationale

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)
- ...

MiniBrass



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

MiniBrass: HelloWorld

Institute for Software & Systems Engineering

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;
```

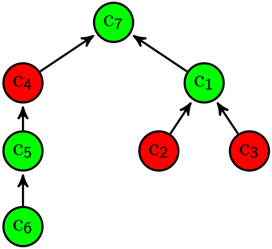
```
Solution:  x = 1; y = 2; z = 1
Valuations:  mbr_overall_cr1 = 2..2
----------
==========
```

Single-Predecessor-Dominance (SPD) Lifting

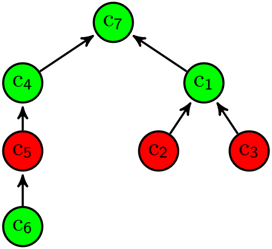
isWorseThan-Relation für Mengen verletzter Constraints (**?**)

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

$V \uplus \{c_{\mathrm{imp}}\} \supset_{\text{SPD}} V \uplus \{c_{\neg\mathrm{imp}}\} \quad \text{if } c_{\neg\mathrm{imp}} \rightarrow c_{\mathrm{imp}}$



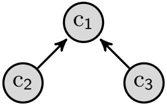
$\supset_{\text{SPD}}$



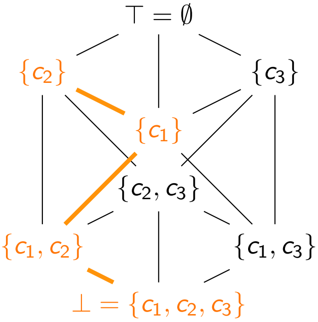
- Bekannt als *Smyth-Ordnung* (Powerdomains) (**?**, Ch. 9)
- Entsteht aus *freier Konstruktion* über Constraint-Relationship. (**?**)

Constraint-Optimierung mit PVS

Der partiell geordnete Bewertungsraum




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

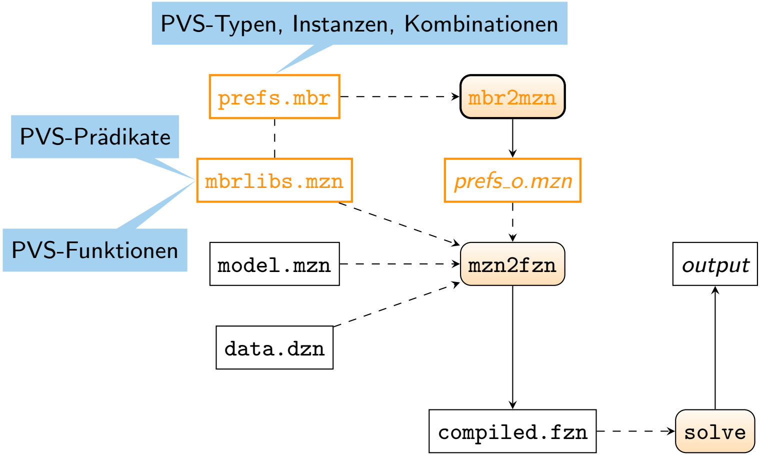



```
x = 1; y = 1; z = 1
Valuation = 1..2
----------
x = 1; y = 1; z = 3
Valuation = 1..1
----------
x = 1; y = 2; z = 1
Valuations = 2..2
----------
==========
```

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

Institute for Software & Systems Engineering

Architektur



PVS-Typdefinitionen

Institute for Software & Systems Engineering

```
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: $\texttt{times} : S^n \to E$
- Ordnungsrelation: $\texttt{isWorse} \subseteq E \times E$

PVS-Definitionen

Innerhalb von ../mbr_types/cr_type.mzn:

```
function var set of int:
  link_invert_booleans(array[int] of var bool: b...

% gives us access to constraint relationship predicates
include "soft_constraints/spd_worse.mzn";
include "soft_constraints/tpd_worse.mzn";

predicate is_worse_cr(var set of int: violated1,
                      var set of int: violated2,
                      par int: nScs, array[int, 1..2] of par int: crEdges,
                      par bool: useSPD) =
let { par set of int: softConstraints = 1..nScs; } in (
    if useSPD then
      spd_worse(violated1, violated2, softConstraints, crEdges)
    else
      tpd_worse(violated1, violated2, softConstraints, crEdges)
    endif);
```

PVS-Instanziierung

```
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 geparst, bis auf `mbr.`-Präfixe)
- Parameter aus `PVSType` müssen Wert erhalten

Weitere PVS-Typen

```
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-Instanziierung Weighted

Institute for
Software & Systems
Engineering

```
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]`)
- Aber können wir sie nicht auch berechnen? Aus Constraint Relationships?
- Homomorphismus $\varphi : \mathrm{PVS}_{\mathrm{cr}} \to \mathrm{PVS}_{\mathrm{weighted}}$
  - $\varphi(\top_{\mathrm{cr}}) = \top_{\mathrm{weighted}}$
  - $\varphi(m \cdot_{\mathrm{cr}} n) = \varphi(m) \cdot_{\mathrm{weighted}} \varphi(n)$
  - $m \leq_{\mathrm{cr}} n \to \varphi(m) \leq_{\mathrm{weighted}} \varphi(n)$

PVS und Morphismen

Kategoriell gesehen,

- (Konkrete) Kategorie PVS:
  - Objekte $\rightarrow$ partielle Bewertungsstrukturen (PVS)
  - Morphismen $\rightarrow$ PVS-Homomorphismen
- Für eine gegebene partielle Ordnung ("über Constraints") P
  - $\mathit{PVS}\langle P\rangle$ ... SPD-geordnete PVS (Multimengen von Elementen aus P als Träger)
  - $\mathit{Weighted}(P)$ ... Weighted PVS ($\mathbb{N}$ als Träger)

$$\mathit{PVS}\langle P\rangle \xrightarrow{w} \mathit{Weighted}(P)$$

Freie PVS über einer partiellen Ordnung

Institute for Software & Systems Engineering

- (Konkrete) Kategorie POSet:
  - Objekte → partiell geordnete Mengen
  - Morphismen → monotone Funktionen

$$
\begin{tikzcd}
P \arrow[r, "\eta_P^{\mathrm{PVS}}"] \arrow[dr, "\varphi"'] & PO(PVS\langle P\rangle) \arrow[d, "PO(\varphi^{\#\mathrm{PVS}})"] \\
& PO(M)
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
PVS\langle P\rangle \arrow[d, dashed, "\varphi^{\#\mathrm{PVS}}"] \\
M
\end{tikzcd}
$$

Freie Konstruktionen

- no junk
- no confusion

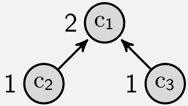
Morphismen in MiniBrass

```
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 <= 3';
```

PVS-Kombinationen: Pareto

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 bool: x; var bool: y;
PVS: wcsp1 = new WeightedCsp("wcsp1") {
   soft-constraint c1: 'y = false' ;
   k : '20';
};

PVS: wcsp2 = new WeightedCsp("wcsp2") {
   soft-constraint c1: 'x = false' ;
   k : '20';
};

solve wcsp1 pareto wcsp2; % returns x = false, y = false
```

PVS-Kombinationen: Lex

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: cr1 = new CostFunctionNetwork("cr1") {
   soft-constraint c1: 'x' ;
   soft-constraint c2: '3 - y' ;
   k : '20';
};
PVS: cr2 = new CostFunctionNetwork("cr2") {
   soft-constraint c1: 'y' ;
   soft-constraint c2: '3 - x' ;
   k : '20';
};
solve cr1 lex cr2; % returns x = 1, y = 3
% dually cr2 lex cr1 yields x = 3, y = 1
```

Evaluationsergebnisse

Kooperationen

Institute for Software & Systems Engineering

Konzepte, Sprachdesign MiniBrass

- AS, Alexander Knapp, Gerrit Anders, Oliver Kosak

Anwendungen, Multiagenten-Einsatz

- Alexander Schubert (MSc-Thesis: Einsatz von Voting-Verfahren), Markus Tolls (MSc-Thesis: Formalisierung von Task-Allocation-Problemen)

Outreach

Institute for Software & Systems Engineering

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

References I