



Soft and Hard Constraints in Large Self-Organizing Systems

Alexander Schiendorfer et al.

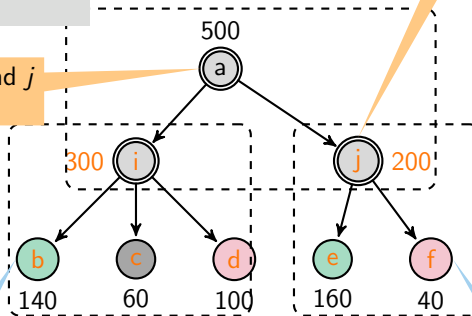


Regio-zentrale Fahrpläne
ICAART'14, SAOS'14
Marktbasiert
TAAS'15

Wie kann ich *e* und *f* repräsentieren?

Abstraktion
ICAART'14, TCCI'15
SASO'15

Was sollen *i* und *j*
beisteuern?



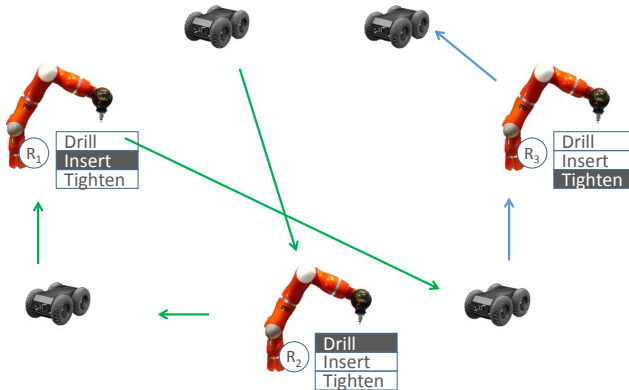
Supply Automata
SEN-MAS'14
TCCI'15

Wie vermeide ich
meinen Speicher
über 90% zu füllen?

Constraint Relationships / PVS
SGAI'13, ICTAI'14
Wirsing'15, Constraints'17

Wie beschreibe ich
bevorzugte Abläufe?

Ziel: Weise Tasks an Roboter zu, sodass ein korrekter **Ressourcenfluss** entsteht



(Seebach et al., 2010)

Ziel: Belege (endlich viele) Variablen aus X mit einem aus (endlich vielen) Werten aus D sodass alle Constraints C erfüllt werden.

Beispiel

- n Roboter, m Tasks
- Gebe jedem Roboter einen *unterschiedlichen* Task, stelle sicher, dass jeder Task belegt ist

```
% problem data
int: n; set of int: ROBOTS = 1..n;
int: m; set of int: TASKS = 1..m;

% decisions
array[ROBOTS] of var TASKS: allocation;

% goal
solve satisfy;

% have robots work on different tasks
constraint alldifferent(allocation);
constraint forall(t in TASKS) (exists(r in ROBOTS) (allocation[r] = t));
```

Ziel: Suche die beste Belegung, sodass eine **skalare** Zielfunktion $f : [X \rightarrow D] \rightarrow \mathbb{Z}$ minimiert (oder maximiert) wird.

Beispiel

- n (steuerbare) Supplier, m (steuerbare) Consumer decken Residuallast

```
% problem data
int: n; set of int: SUPPLIERS = 1..n;
array[SUPPLIERS] of int: costs;
int: m; set of int: CONSUMERS = 1..m;
int: residualLoad;

% decisions
array[SUPPLIERS] of var 0..100: supply;
array[CONSUMERS] of var 0..100: demand;

% goal
solve minimize sum(s in SUPPLIERS)(costs[s]*supply[s]);

% have robots work on different tasks
constraint sum(supply) - sum(demand) - residualLoad = 0;
```

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

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

Constraint Programming

- Deklarative Programmierung (ähnlich SQL, Prolog)
- Trennung von **Modell** und *Algorithmus*
- Geeignet für kombinatorische Probleme unter harten Bedingungen (Physik!)
- Modellierungssprache **MiniZinc**

Soft Constraint Programming

- Modellierung von **Präferenzen**
- Finde Lösungen, die *so gut wie möglich* sind
- Was bedeutet "gut"?
- Modellierungssprache **MiniBrass**

Rationale

Eine Sprache – viele Solver

Unterstützte Solver

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



Was machen wir nun mit **inertia_{therm}** und **ecoSweet_{bio}**?

Max-CSP Erfülle so viele Constraints wie möglich (Freuder and Wallace, 1992)

Weighted CSP Minimiere die Summe der verletzten Constraints nach Gewicht
(Shapiro and Haralick, 1981)

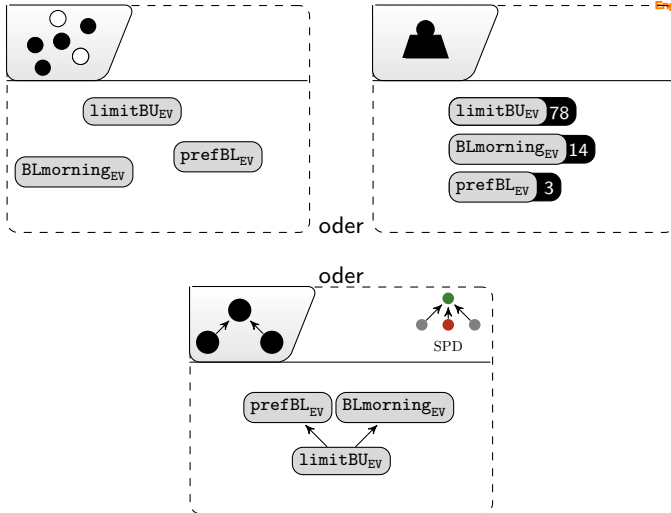
Fuzzy CSP Erfülle den (minimalen) Erfüllungsgrad (zwischen 0 und 1) über alle
Soft Constraints (Ruttkay, 1994)

... und natürlich

Constraint Preferences (früher *Constraint Relationships*): Definiere partielle
Wichtigkeitsordnung über Constraints; erhebe diese zu Mengen von
verletzten oder erfüllten Constraints (Schiendorfer et al., 2013)

Zentrale Frage

→ Was sind die Gemeinsamkeiten? Was müssen wir “minimal” tun?



Wir benötigen . . .

- Eine Menge M von Erfüllungsgraden, z.B. $[0.0, 1.0]$ oder $\{0, 1, \dots k\}$ oder 2^{C_s} .
- Eine partielle Ordnung \leq_M über M : $m \leq_M n$ drückt aus, dass m *schlechter* als n ist
- Eine Kombinationsoperation \cdot_M , um zwei Elemente aus M miteinander zu verrechnen
- Ein bestes Element ε_M , um *volle Zufriedenheit* auszudrücken

Gemeinsam nennen wir $(M, \cdot_M, \varepsilon_M, \leq_M)$ eine **partielle Bewertungsstruktur**.

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

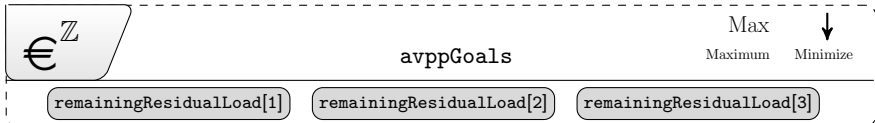
Konkrete PVS-Typen	M	\cdot_M	\leq_M	ε_M
Weighted CSP (WCSP)	\mathbb{N}	$+$	\geq	0
Cost Function Network (CFN)	$\{0, \dots, k\}$	$+/max$	\geq	0
Fuzzy CSP	$[0, 1]$	min	\leq	1
Inclusion Max CSP	2^{C_s}	\cup	\supseteq	\emptyset
Constraint Preferences (CP) ¹	$\mathcal{M}^{\text{fin}}(C_s)$	\sqcup	\supseteq_{SPD}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

Hauptidee

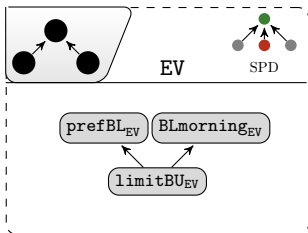
Implementiere Lösungsverfahren für Constraint-Probleme, die durch Bewertungsstrukturen geordnet sind. Instantiiere für konkrete Probleme.

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

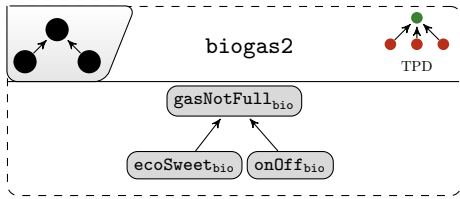
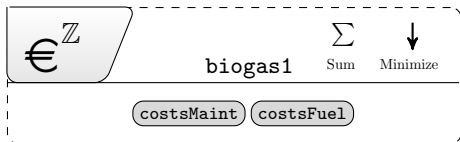
× lex



× pareto





× lex





[View on GitHub](#)

MiniBrass*

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



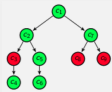


Universität
Augsburg
University

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();
```

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

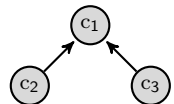
=====

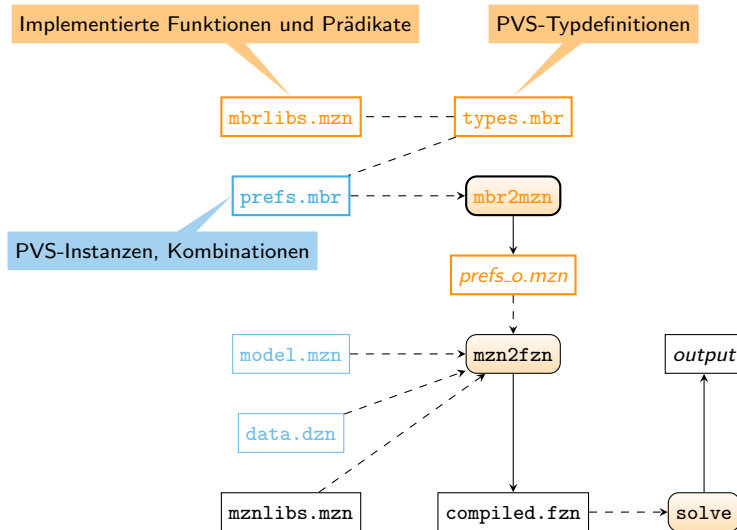
Präferenzmodell (MiniBrass)

```
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;
```





MiniBrass wird für verschiedene Anwendungen eingesetzt:

- Energiefallstudie
- Studenten-Mentoren-Matching
- Prüfungsterminfindung
- Multi-User-Multi-Display
- Rekonfigurierbare Roboterteams
- Potentiell: Rollenallokation im ODP

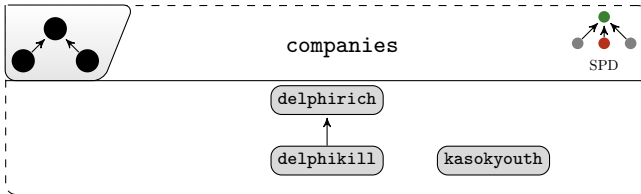
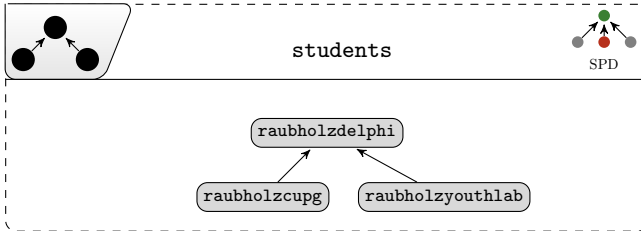
Ziel: Teile Mentees (z.B. Studenten) Mentoren zu (z.B. Firmen), sodass

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

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 (*harter Ausschluss* von Lösungen)

× lex



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

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

PVS: companies = new ConstraintPreferences("companies") {
    soft-constraint delphi_kill: 'worksAt[kill] = delphi';
    soft-constraint delphi_rich: 'worksAt[rich] = delphi';
    soft-constraint kasokyouth: 'worksAt[kasok] = airtrain';

    crEdges : '[| mbr.delphi_kill, mbr.delphi_rich |]';
    useSPD: 'true' ;
};

solve students lex companies;
% solve companies lex students;
```

Pro:

- Wünschenswerte Bedingungen als boolesche Eigenschaften ("Constraints")
- Nicht alle erfüllbar
- Nicht alle **vergleichbar**
- **Komparative** Bewertung möglich ("A ist mir wichtiger als B")

Contra:

- Problem hat numerisches Ziel (Kosten, Verletzungen als Metrik)
- Feingranulares Tuning für Gewichte von Constraints nötig
- Ich will nur *eine* Lösung am Ende.

Ziel: Weise Prüfungstermine an Studenten zu, sodass

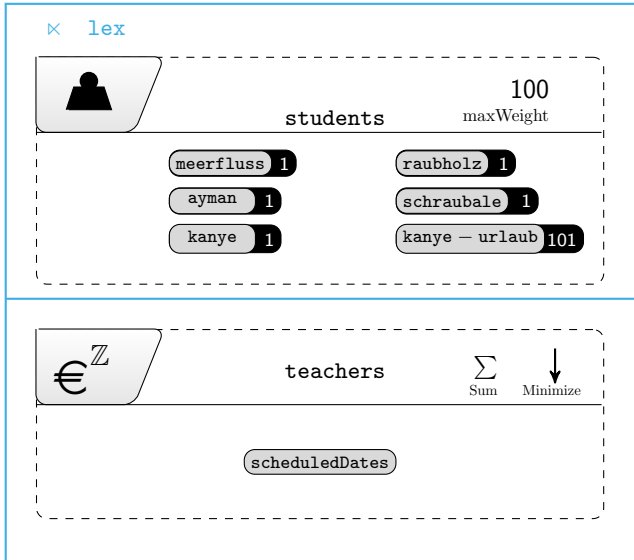
- Jeder Student stimmt seinem Termin zu
- Die Anzahl verschiedener Termine wird minimiert (um das Zeitinvestment der Dozenten zu schonen)



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



```
PVS: students = new WeightedCsp("students") {
    maxWeight: '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 ayman: 'sd[ayman] in {monday, tuesday}';
    soft-constraint kanye: 'sd[kanye] in {monday, wednesday}';
    % hard by weight (less than bottom)
    soft-constraint kanye-urlaub: 'sd[kanye] != 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

Pro:

- Feintuning von Gewichten für Soft Constraints (evt. erlernt vom Benutzer)
- Skalare Kostenfunktion ist offensichtlich (Produktionskosten, Spanne der Task-Zuweisung)
- Fehler können per Distanz angegeben und minimiert werden (50 KW Abweichung ist schlimmer als 10 KW Abweichung)
- Lösungszeit ist kritisch – keine Auswahlmöglichkeiten nötig

Contra:

- Es gibt tatsächlich unvergleichbare Kriterien.
- Ich will mir für eine Ordnung keine Gewichtung “ausdenken” müssen – besonders wenn sich Soft Constraints häufig ändern.
- Viele Entscheidungsvarianten gewünscht (nicht nur “skalares Optimum”) – z.B. in Offline-Entscheidungssituationen

Freuder, E. C. and Wallace, R. J. (1992).

Partial Constraint Satisfaction.

Artif. Intell., 58(1–3):21–70.

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>.

Ruttkay, Z. (1994).

Fuzzy constraint satisfaction.

In *Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on*, pages 1263–1268. IEEE.

Schiendorfer, A., Knapp, A., Anders, G., and Reif, W. (2017).

Minibrass: Soft constraints for minizinc.

Constraints, 22(3):377–402.

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.
- Seebach, H., Nafz, F., Steghofer, J.-P., and Reif, W. (2010).
A software engineering guideline for self-organizing resource-flow systems.
In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 194–203. IEEE.
- Shapiro, L. G. and Haralick, R. M. (1981).
Structural descriptions and inexact matching.
IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 504–519.