



# Soft Constraint Programming in MiniBrass

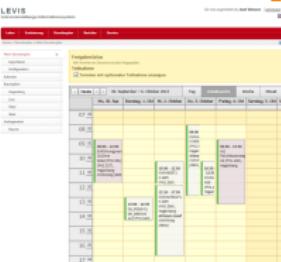
Alexander Schiendorfer @ FH Hagenberg, 12.05.2016



# Entscheidungsprobleme



“Wie viele Muffins von welcher Sorte?”



“Wann findet welche Vorlesung statt?”



“Wer macht wann welche Aufgaben?”



“Was machen wir am Wochenende?”

## Entscheidungen

- Die Anzahl Schokomuffins oder Bananenmuffins
- Die Vorlesungen im Studienplan
- Die Freitags- und Samstagsaktivität

## Entscheidungen (*Variablen*)

- Die Anzahl Schokomuffins oder Bananenmuffins
- Die Vorlesungen im Studienplan
- Die Freitags- und Samstagsaktivität

## Möglichkeiten

- Schokomuffins:  $\{0, \dots, 20\}$
- Vorlesung "Algorithmen 1":  $\{\text{HS1}, \text{HS2}, \text{HS3}, \dots\}$

## Entscheidungen (*Variablen*)

- Die Anzahl Schokomuffins oder Bananenmuffins
- Die Vorlesungen im Studienplan
- Die Freitags- und Samstagsaktivität

## Möglichkeiten (*Domänen*)

- Schokomuffins:  $\{0, \dots, 20\}$
- Vorlesung "Algorithmen 1":  $\{\text{HS1}, \text{HS2}, \text{HS3}, \dots\}$

## Abhängigkeiten

- Das benötigte Mehl für  $x$  Schoko- und  $y$  Bananenmuffins darf 250g nicht übersteigen.
- In einem Raum kann gleichzeitig nur eine Veranstaltung stattfinden.
- Es darf nur 1 SchnitzNight pro Wochenende geben.

## Entscheidungen (*Variablen*)

- Die Anzahl Schokomuffins oder Bananenmuffins
- Die Vorlesungen im Studienplan
- Die Freitags- und Samstagsaktivität

## Möglichkeiten (*Domänen*)

- Schokomuffins:  $\{0, \dots, 20\}$
- Vorlesung "Algorithmen 1":  $\{\text{HS1}, \text{HS2}, \text{HS3}, \dots\}$

## Abhängigkeiten (*Constraints*)

- Das benötigte Mehl für  $x$  Schoko- und  $y$  Bananenmuffins darf 250g nicht übersteigen.
- In einem Raum kann gleichzeitig nur eine Veranstaltung stattfinden.
- Es darf nur 1 SchnitzNight pro Wochenende geben.

## Präferenzen

- Am Freitag, 08:00 *sollte* keine Algorithmenübung stattfinden
- Bernd möchte ins Steakhouse, Ada zur Burgerei → Adas Präferenz ist *wichtiger*
- Ich möchte weder putzen noch staubsaugen; Putzen ist aber *schlimmer*

## Präferenzen (*Soft Constraints*)

- Am Freitag, 08:00 *sollte* keine Algorithmenübung stattfinden
- Bernd möchte ins Steakhouse, Ada zur Burgerei → Adas Präferenz ist *wichtiger*
- Ich möchte weder putzen noch staubsaugen; Putzen ist aber *schlimmer*

und/oder

## Ziele

- Maximiere den Ertrag durch Schoko/Bananenmix
- Maximiere die Anzahl der vorlesungsfreien Tage

ein **Constraint-Satisfaction-(Optimization)-Problem** (CSP/COP)

## Präferenzen (*Soft Constraints*)

- Am Freitag, 08:00 *sollte* keine Algorithmenübung stattfinden
- Bernd möchte ins Steakhouse, Ada zur Burgerei → Adas Präferenz ist *wichtiger*
- Ich möchte weder putzen noch staubsaugen; Putzen ist aber *schlimmer*

und/oder

## Ziele (*Zielfunktionen*)

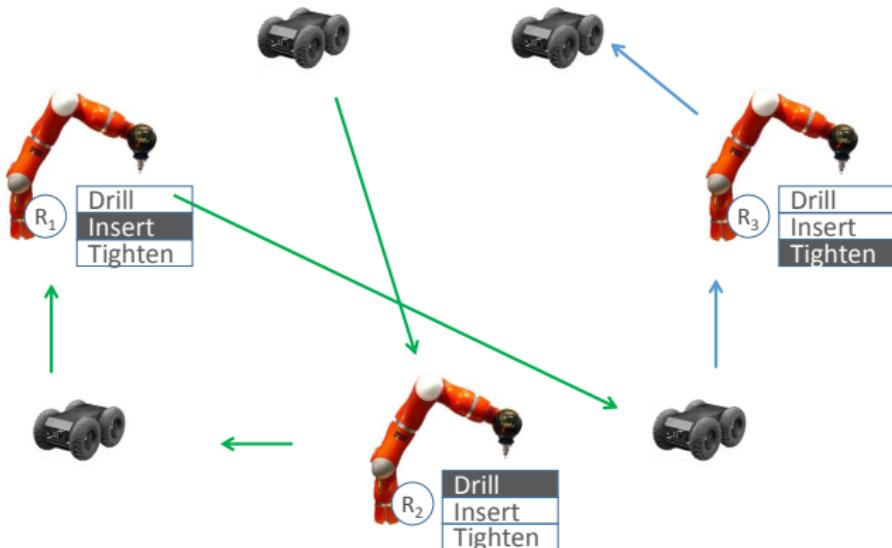
- Maximiere den Ertrag durch Schoko/Bananenmix
- Maximiere die Anzahl der vorlesungsfreien Tage

ein **Constraint-Satisfaction-(Optimization)-Problem** (CSP/COP)

# Autonome Systeme I: Adaptive Produktion



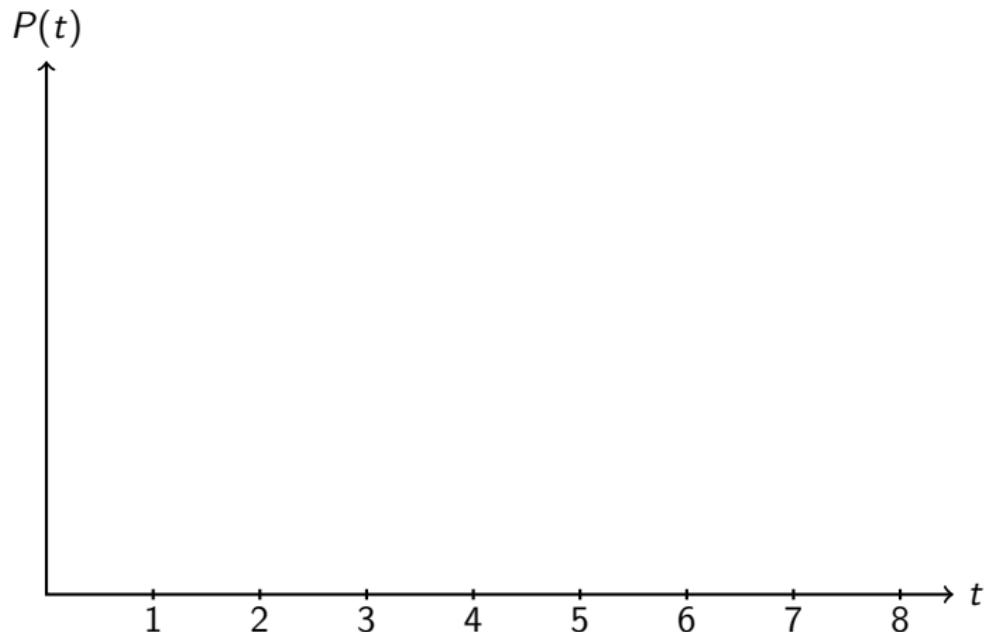
**Ziel:** Weise Robotern Aufgaben so zu, dass sich ein korrekter Ressourcenfluss ergibt



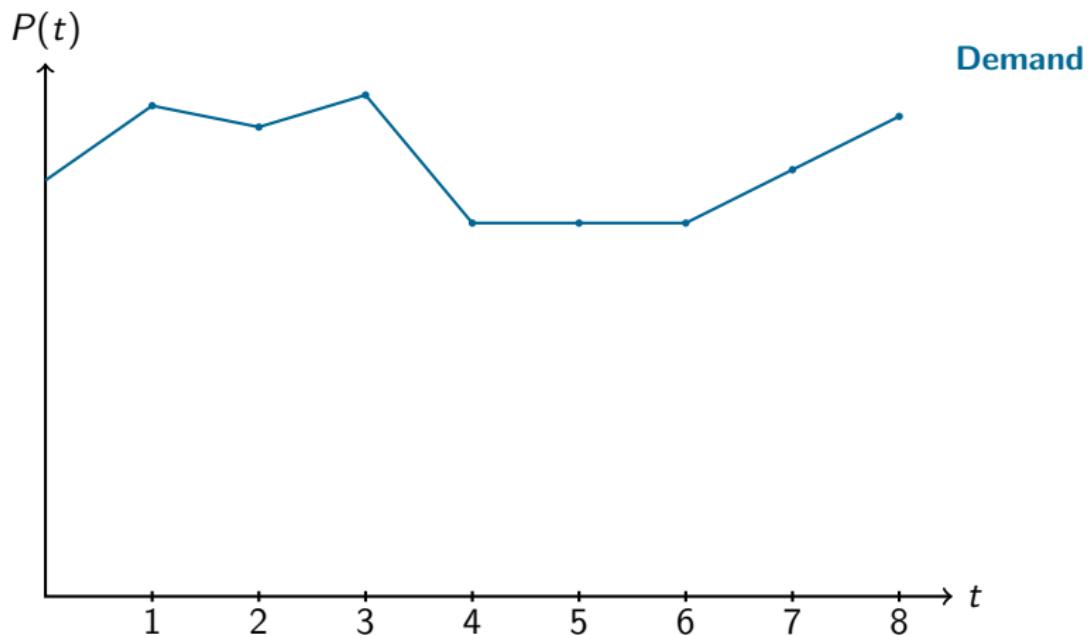
# Autonome Systeme II: Energiemanagement



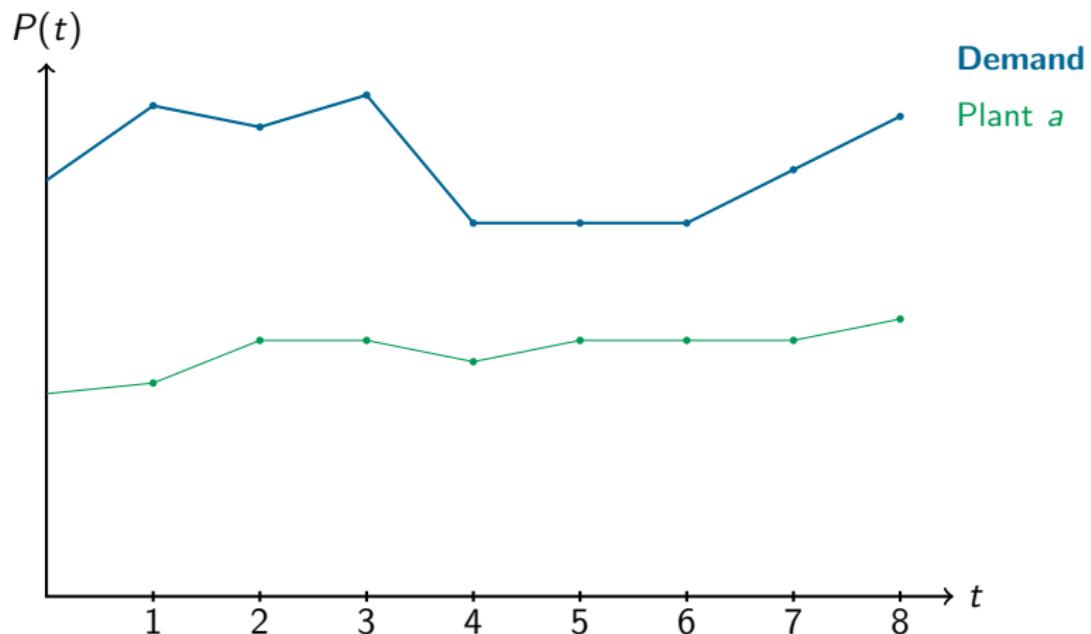
**Ziel:** Plane Kraftwerke so ein, dass sie die **Last** gemeinsam erfüllen



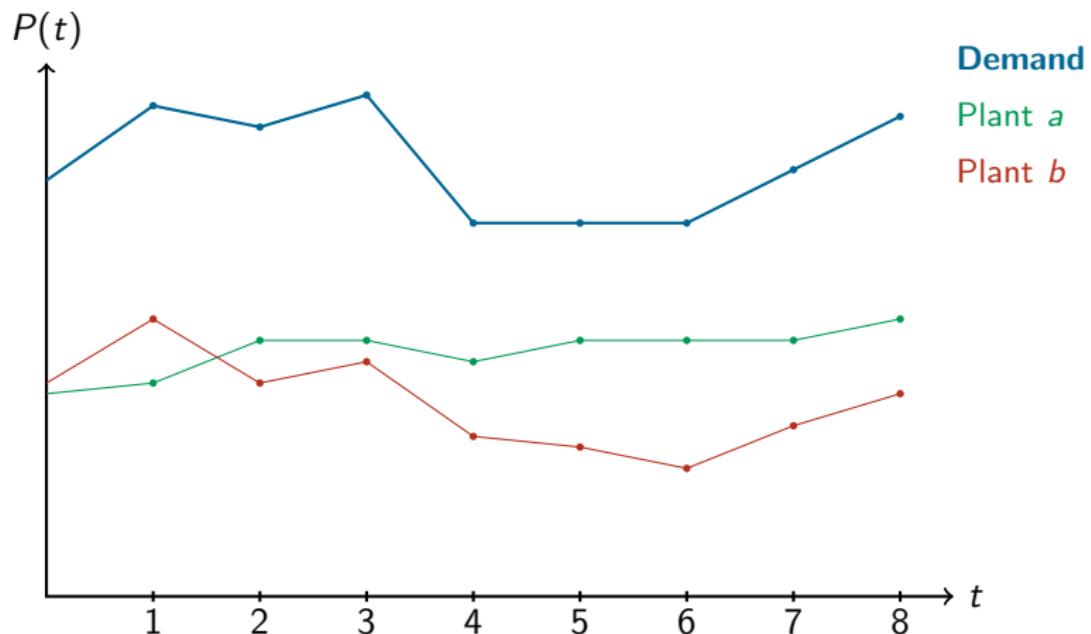
**Ziel:** Plane Kraftwerke so ein, dass sie die **Last** gemeinsam erfüllen



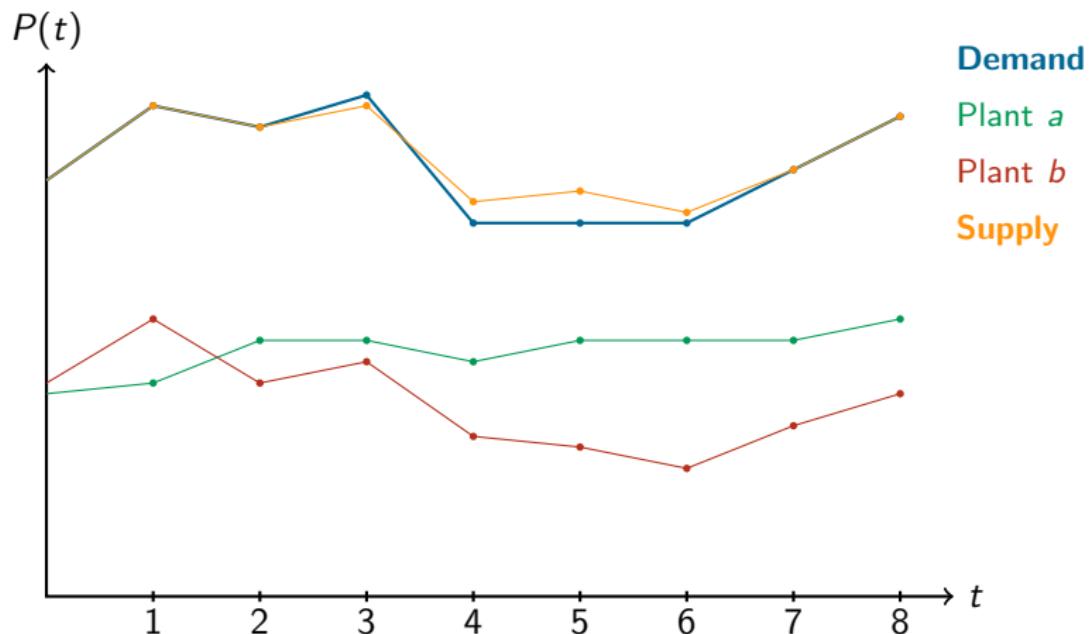
**Ziel:** Plane Kraftwerke so ein, dass sie die **Last** gemeinsam erfüllen



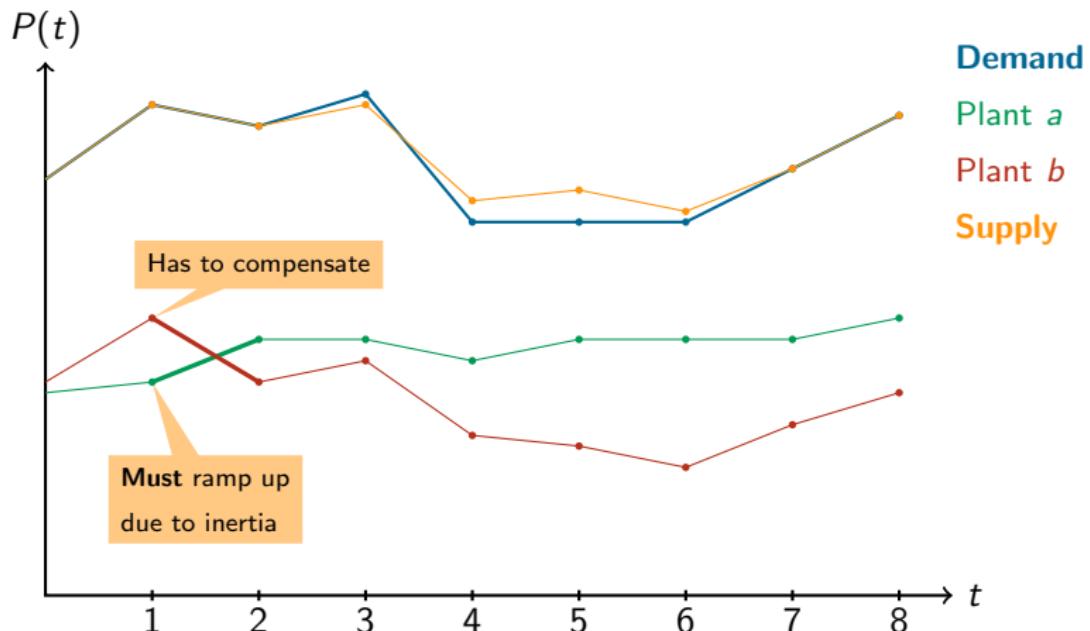
**Ziel:** Plane Kraftwerke so ein, dass sie die **Last** gemeinsam erfüllen



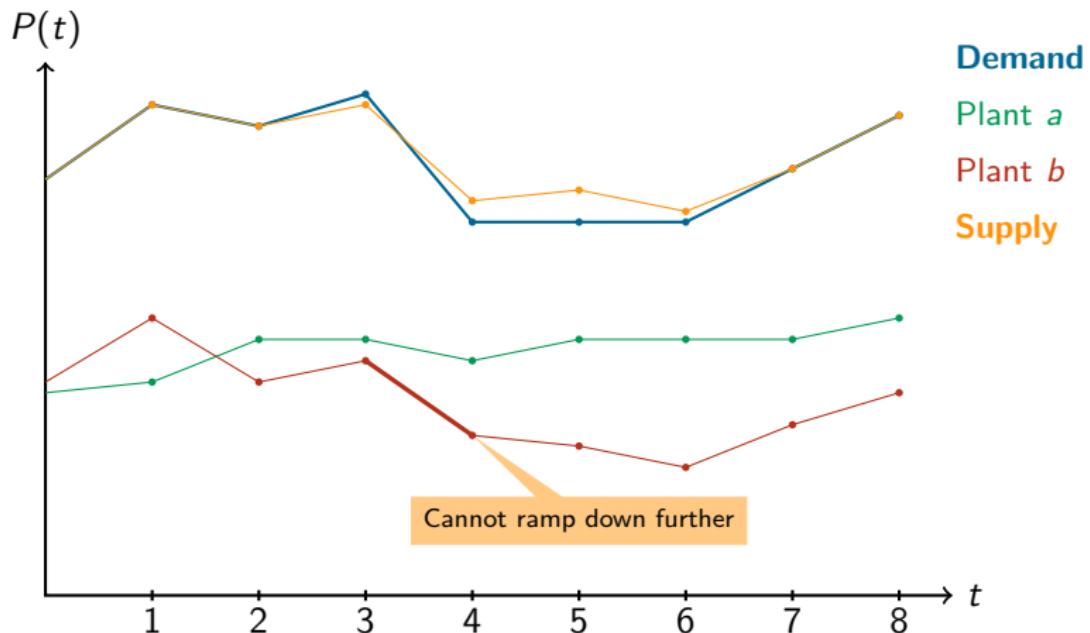
**Ziel:** Plane Kraftwerke so ein, dass sie die **Last** gemeinsam erfüllen



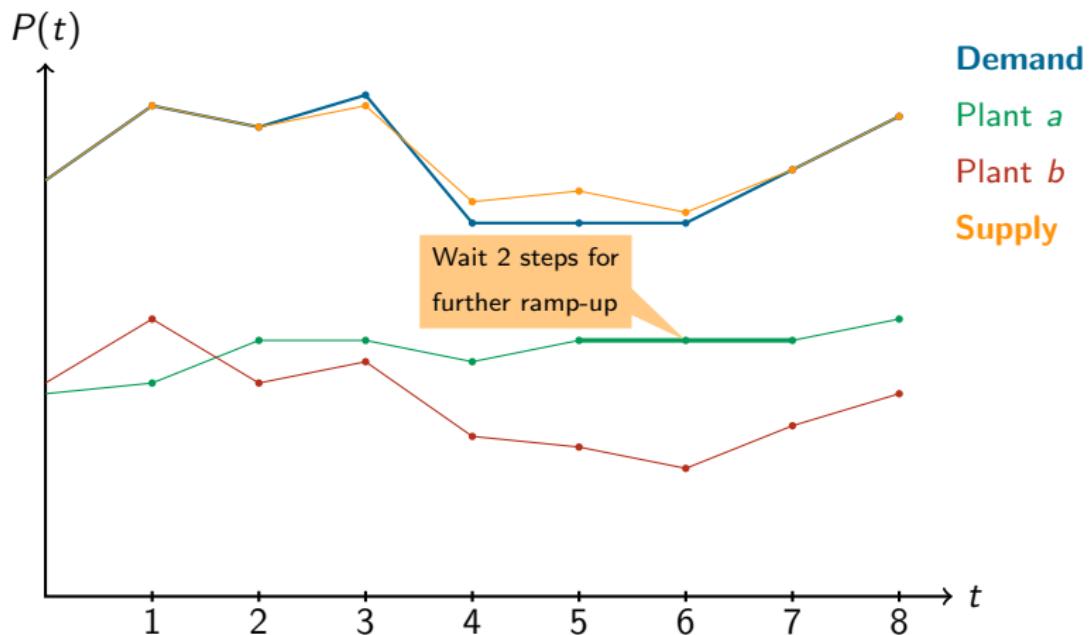
**Ziel:** Plane Kraftwerke so ein, dass sie die **Last** gemeinsam erfüllen



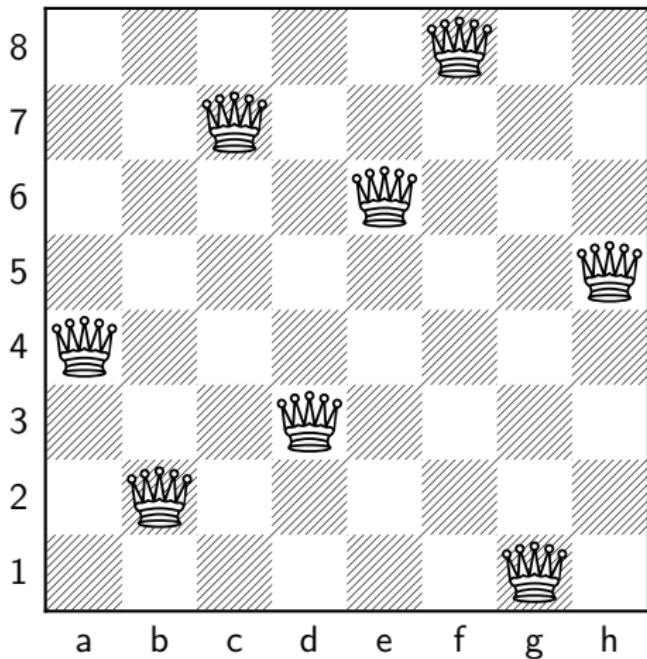
**Ziel:** Plane Kraftwerke so ein, dass sie die **Last** gemeinsam erfüllen



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



# Ein klassisches CSP



Entwickle eigenen Algorithmus *from scratch*:

```
PlaceQueens($col: IntArray  ↴n: int  ↴i: int)
    var j: int
begin
    for j := 1 to n do
        if QueenFits(↓col ↓i ↓j) then
            col[i] := j -- place queen i in column j
            if i = n then
                WriteSolution(↓col ↓n)
                halt
            end -- if
            PlaceQueens($col ↓n ↓i + 1)
            col[i] = 0 -- remove queen i
        end -- if
    end -- for
end PlaceQueens
```

(Pomberger, Dobler, 2008)

Alternative: Schreibe *ein* Modell – teste *viele* Algorithmen!

```
include "globals.mzn";
int: n = 8;
array[1..n] of var 1..n: queens;

solve satisfy;

constraint all_different(queens);
constraint all_different([queens[i]+i | i in 1..n]);
constraint all_different([queens[i]-i | i in 1..n]);
```

2 3 4 5        0 -1 -2 -3

3 4 5 6        1 0 -1 -2

4 5 6 7        2 1 0 -1

5 6 7 8        3 2 1 0

queens = array1d(1..8 ,[4, 6, 1, 5, 2, 8, 3, 7]);

-----

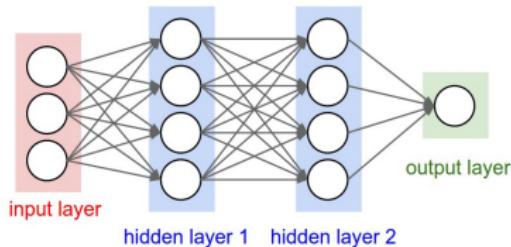
## Constraint Programming

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

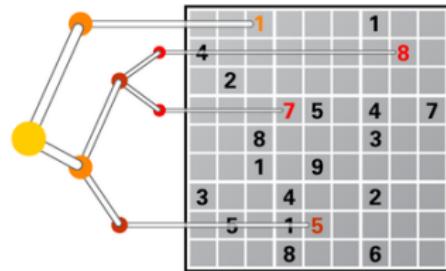
## Soft Constraint Programming

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

## Data-driven AI



## Decision-driven AI



- Machine Learning
- Signal Processing
- Computer Vision

- Constraint Programming
- Combinatorial Optimization
- Heuristic Optimization
- Planning / Scheduling

- ① Einmal (formal) modelliert – vielseitig gelöst
  - Constraint-Solving
  - SAT
  - MIP
  - Heuristiken
- ② Effiziente, (normalerweise) gut getestete Algorithmen in Lösern verbaut
  - Gleiche Algorithmen für viele Probleme
  - Spezialalgorithmen für wiederkehrende Teilprobleme (alldifferent)
- ③ Prototyping
  - Problemspezifikation wird klarer
  - Spezialalgorithmus für konkretes Problem kann nachentwickelt werden

```
SELECT firstname, lastname  
FROM employees  
WHERE age < 30
```

statt

```
Collection<Person> youngs = new ArrayList<>();  
for(Person p : allEmployees) {  
    if(p.getAge() < 30)  
        youngs.add(p);  
}
```

## Motivation

Constraint-Modellierung für Optimierungsprobleme  $\approx$  SQL für Datenzugriff

## Theorem

Das zu einem CSP gehörende Entscheidungsproblem ist NP-vollständig.

### MY HOBBY: EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT	
~~ APPETIZERS ~~	
MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80
~~ SANDWICHES ~~	
BARBECUE	6.55



## Rationale

Eine Sprache – viele Solver

### Unterstützte Solver

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



# Ein erstes Modell

```
var 0..2: x;  
var 0..2: y;  
  
constraint x < y;  
  
solve maximize x + y;
```

<http://www.minizinc.org>

```
x = 1;  
y = 2;  
-----  
=====
```

# Etwas interessanter ...

```
var 0..100: b; % no. of banana muffins
var 0..100: c; % no. of chocolate muffins

% flour
constraint 250*b + 200*c <= 4000;
% bananas
constraint 2*b <= 6;
% sugar
constraint 75*b + 150*c <= 2000;
% butter
constraint 100*b + 150*c <= 500;
% cocoa
constraint 75*c <= 500;

% maximize our profit
solve maximize 400*b + 450*c;

output ["no. of banana muffins = ", show(b), "\n", "% b = 2"
        "no. of chocolate muffins = ", show(c), "\n"]; % c = 2
```

- Taskzuweisungsproblem (*task allocation problem*)
  - $n$  Roboter
  - $m$  Tasks
  - Gebe jedem Roboter einen *unterschiedlichen* Task, und maximiere den Gewinn
- Beispielproblem:
  - $n = 4, m = 5$

	t1	t2	t3	t4	t5
r1	7	1	3	4	6
r2	8	2	5	1	4
r3	4	3	7	2	5
r4	3	1	6	3	6

```
% problem data
int: n; set of int: ROBOTS = 1..n;
int: m; set of int: TASKS = 1..m;
array[ROBOTS,TASKS] of int: profit;

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

% goal
solve maximize sum(r in ROBOTS) (profit[r, allocation[r]] );

% have robots work on different tasks
constraint alldifferent(allocation);
```

# Wie funktioniert's?

Im Wesentlichen ...

4						9
		5	8			
7	9	3	2		8	1
5		6	9		4	
	4		3	1		
2		1	4		5	
6	8	4	3		7	5
		9	6			
9						3

- ① Streiche alle Kandidaten raus
- ② Fülle alle Felder aus, bei denen nur noch 1 Zahl in Frage kommt
- ③ Wenn du nicht mehr weiterkommst → probiere aus

# Wie funktioniert's?

Im Wesentlichen ...

4						9
		5	8			
7	9	3	2		8	1
5		6	9		4	
	4		3		1	
2		1	4		5	
6	8	4	3		7	5
		9	6			
9						3

- ① Streiche alle Kandidaten raus (*Constraint Propagation*)
- ② Fülle alle Felder aus, bei denen nur noch 1 Zahl in Frage kommt
- ③ Wenn du nicht mehr weiterkommst → probiere aus

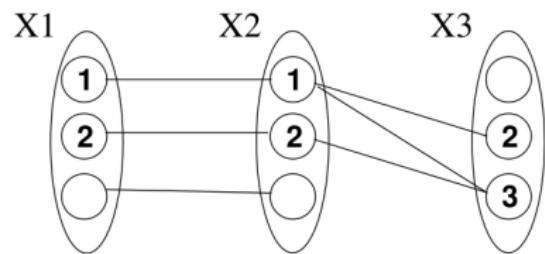
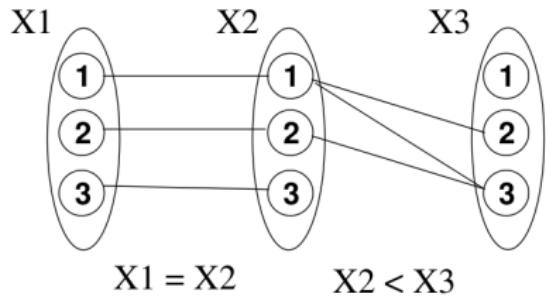
# Wie funktioniert's?

Im Wesentlichen ...

4						9
		5	8			
7	9	3	2		8	1
5		6	9		4	
	4		3	1		
2		1	4		5	
6	8	4	3		7	5
		9	6			
9						3

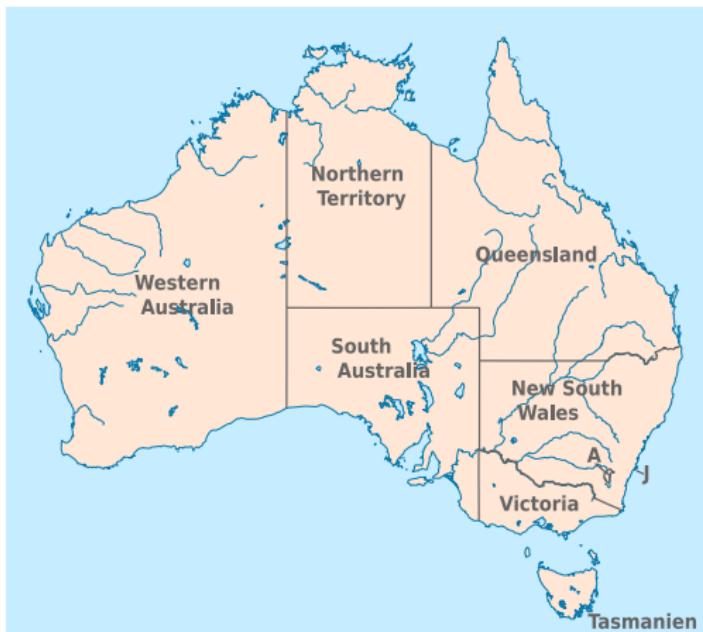
- ① Streiche alle Kandidaten raus (*Constraint Propagation*)
- ② Fülle alle Felder aus, bei denen nur noch 1 Zahl in Frage kommt
- ③ Wenn du nicht mehr weiterkommst → probiere aus (*Backtracking Search*)

# Constraint-Propagation: Beispiel



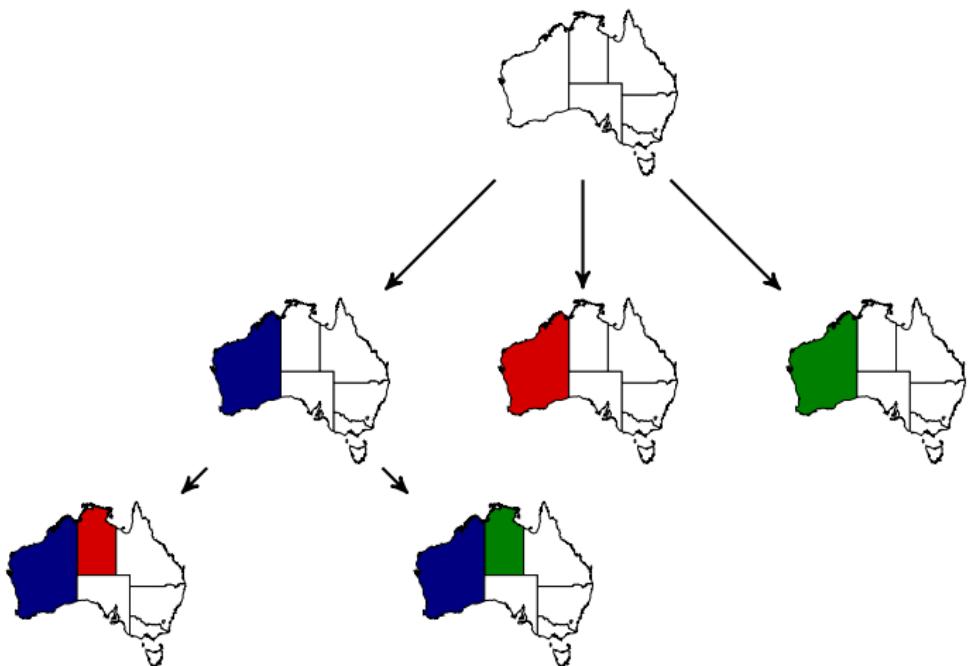
Entferne Werte, die zu keiner Lösung führen können.

# Suche am Beispiel Map Coloring

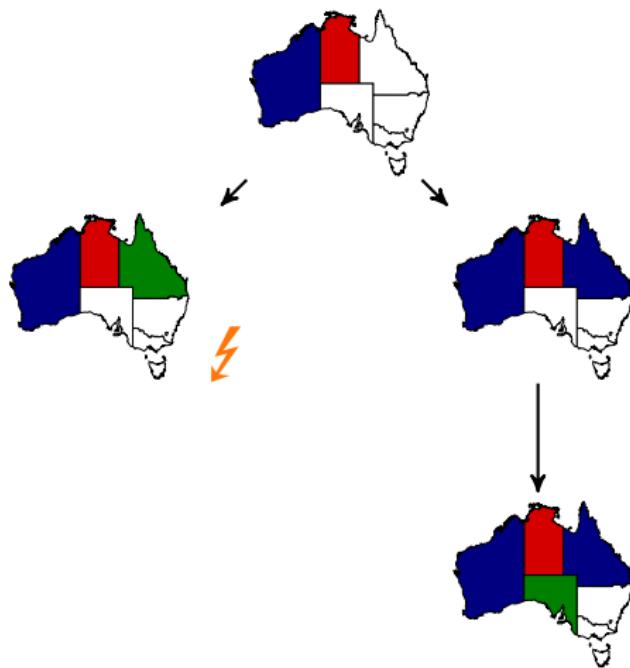


$$X = \{WA, NT, \dots, V\}, D_x = \{r, g, b\}, C = \{WA \neq NT, NT \neq SA, \dots\}$$

# Systematische Suche I



# Systematische Suche II



Wähle Länder (Variablen) und Farben (Werte) in *geschickter* Reihenfolge aus

- *Minimum-Remaining-Values* (MRV): Wähle das Land mit der kleinsten verbleibenden Domäne
- *Most-constrained* (MC): Wähle das Land mit den meisten angrenzenden Ländern
- *Minimum reduction* (MR): Wähle eine Farbe, die andere Länder minimal einschränkt

Dadurch frühe Sackgassenerkennung und sinnvolle Wahl, um Suchbaum zu begrenzen.

Beispiel für MRV+MC (MC bricht Unentschieden nach MRV):



Alle unbelegt



SA in 5 Constraints

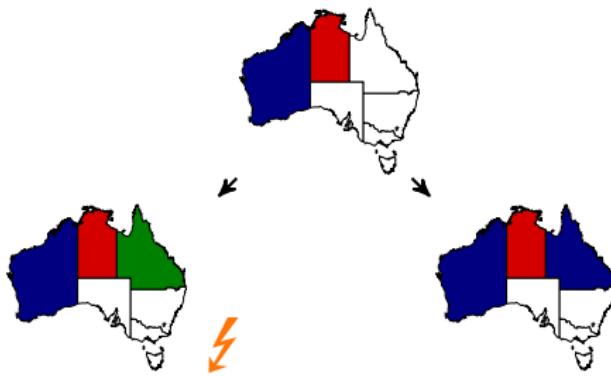


$|D_{QL}|$  nur 2



$|D_{NT}|$  nur 1

Wähle die Belegung, die die wenigsten Domäneneinschränkungen zur Folge haben.



Es bleibt **kein** Wert für SA.

Es bleibt ein Wert für SA.

# Erstes Modell: Revisited

```
var 0..2: x;
var 0..2: y;

constraint x < y;

solve
:: int_search([x,y], input_order, indomain_max, complete)
maximize x + y;
```

```
x = 1;
y = 2;
-----
=====
```

In der Praxis: **überbestimmte** Probleme

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

$$c_1 : x + 1 = y$$

$$c_2 : z = y + 2$$

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

- Nicht alle Constraints können gleichzeitig erfüllt werden

In der Praxis: **überbestimmte** Probleme

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

$$c_1 : x + 1 = y$$

$$c_2 : z = y + 2$$

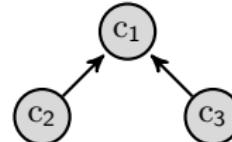
$$c_3 : x + y \leq 3$$

- Nicht alle Constraints können gleichzeitig erfüllt werden
  - e. g.,  $c_2$  erzwingt  $z = 3$  und  $y = 1$ , im Konflikt mit  $c_1$
- Wir **wählen** zwischen Zuweisungen, die  $\{c_1, c_3\}$  oder  $\{c_2, c_3\}$  erfüllen.

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

Ansatz (Schiendorfer et al., 2013)

- Definiere Relation  $R$  über Constraints  $C$  um anzugeben, welche Constraints wichtiger sind als andere, e. g.
  - $c_1$  wichtiger als  $c_2$
  - $c_1$  wichtiger als  $c_3$
- Wie **viel** wichtiger soll  $c_1$  sein?
  - Wichtiger als nur  $c_2$  oder  $c_3$  allein?
  - Wichtiger als  $c_2$  und  $c_3$  zusammen?



[View on GitHub](#)

## MiniBrass\*

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

[tar.gz](#) [.zip](#)

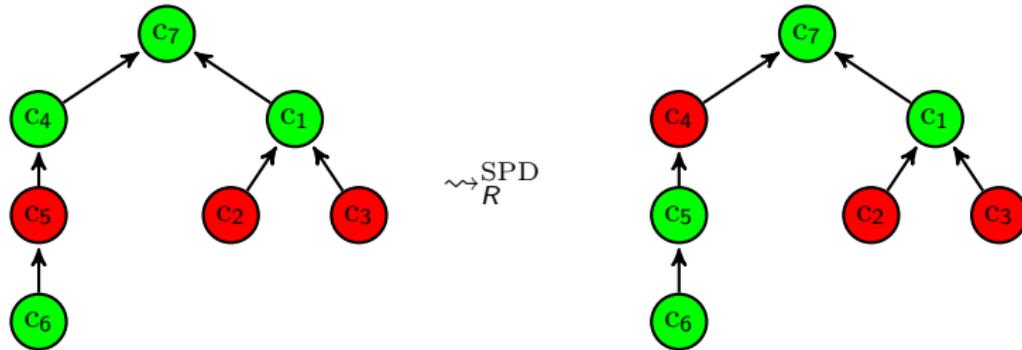
### Optimization with Preferences

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.

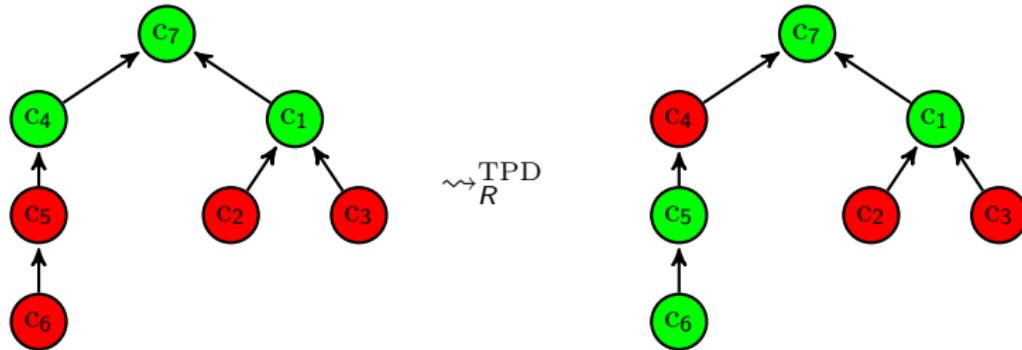
We provide support for over-constrained problems or problems where desirable properties can be modeled as optional (soft) constraints. Importance is expressed only by

<http://isse-augsburg.github.io/constraint-relationships/>

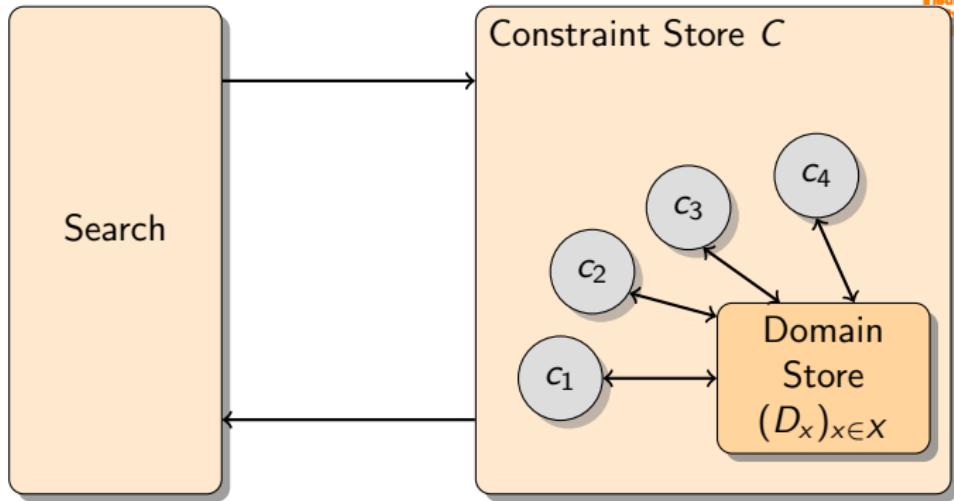
isWorseThan-Relation für Mengen verletzter Constraints



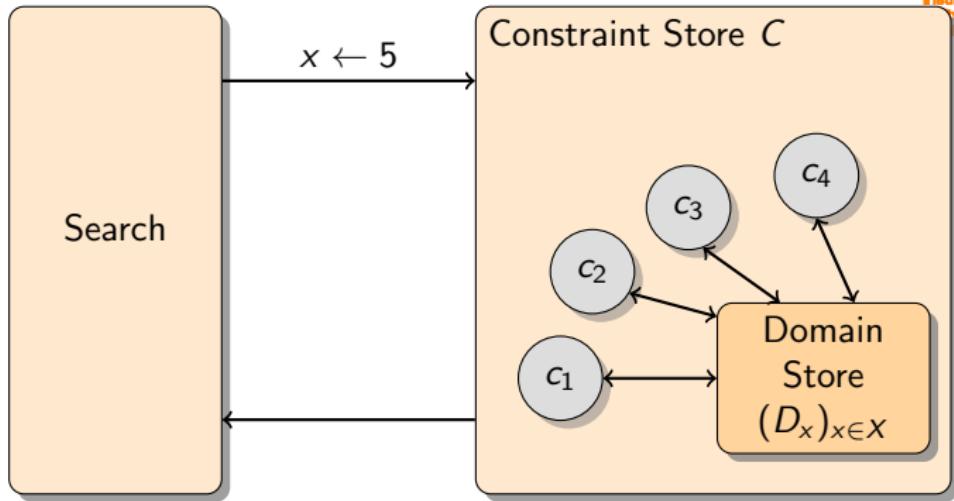
isWorseThan-Relation für Mengen verletzter Constraints



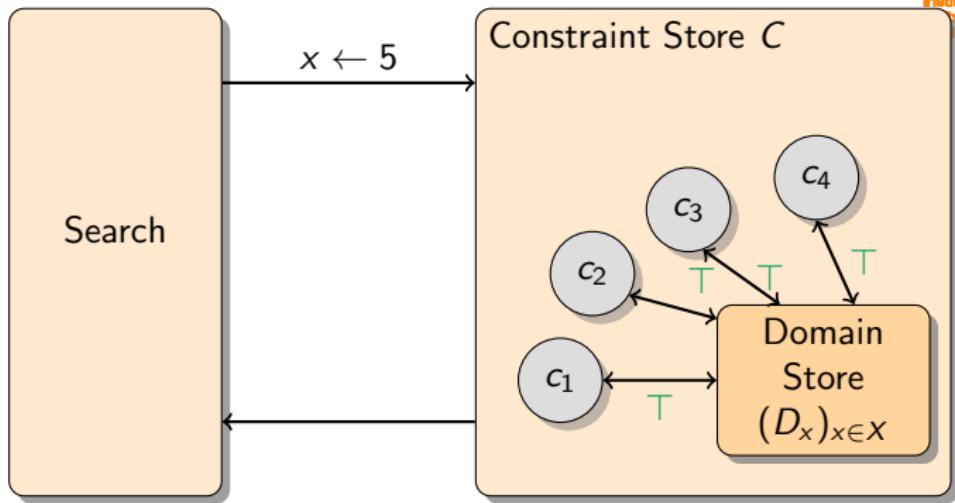
# Traditional Constraint Solving



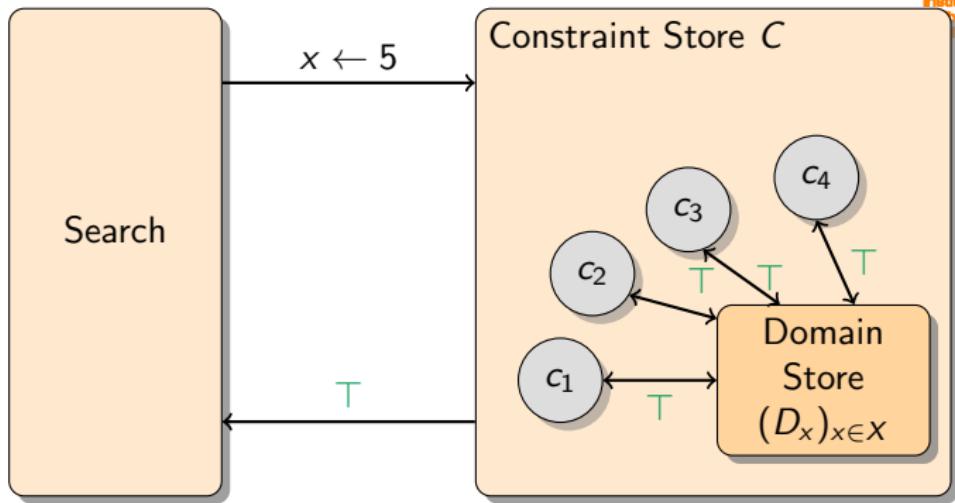
# Traditional Constraint Solving



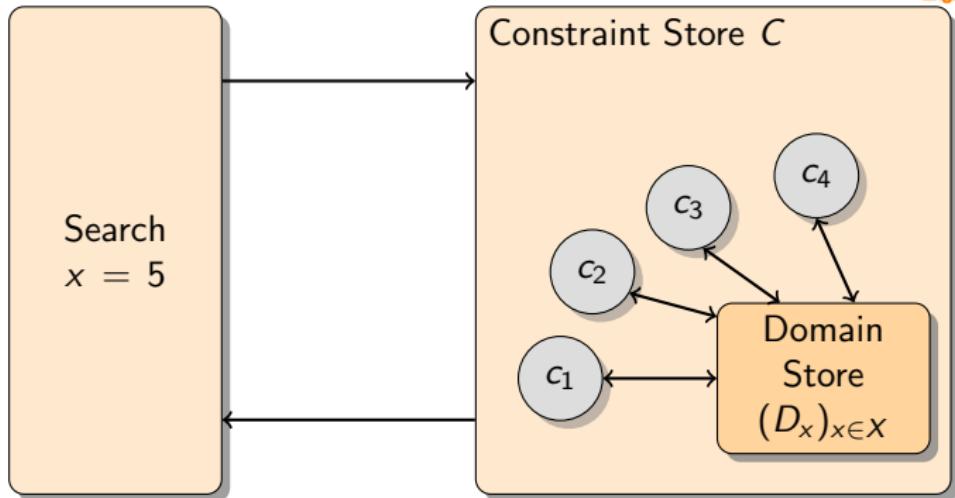
# Traditional Constraint Solving



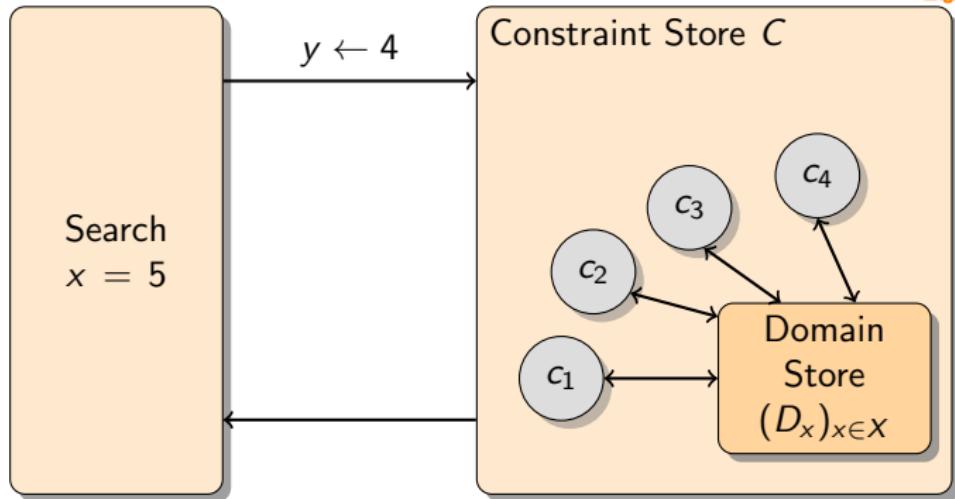
# Traditional Constraint Solving



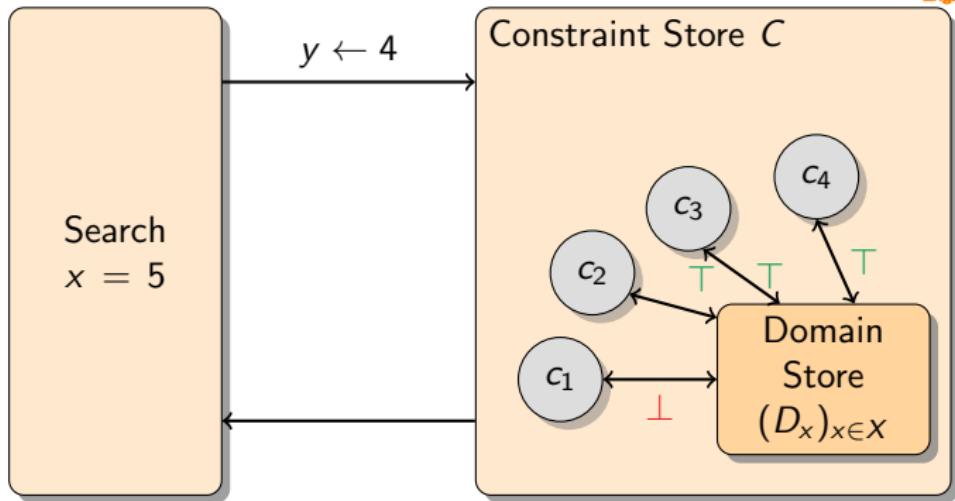
# Klassisches Constraint-Solving



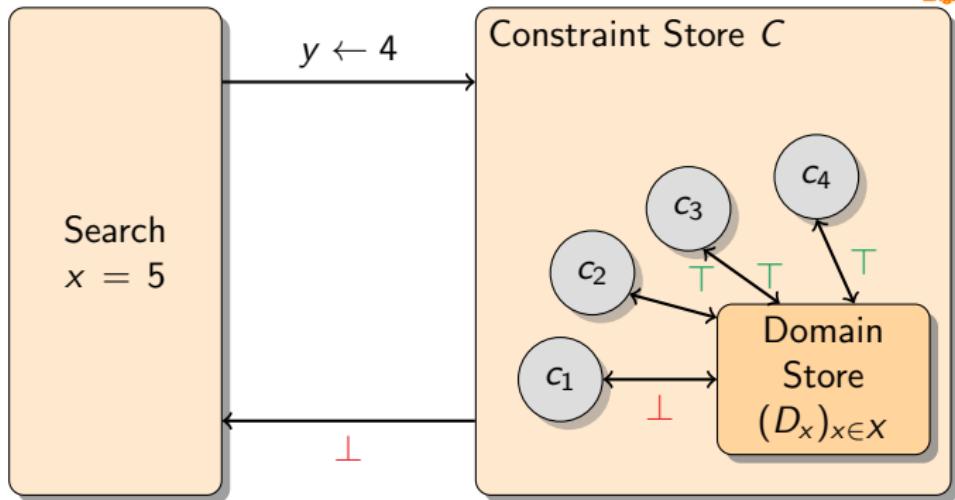
# Klassisches Constraint-Solving

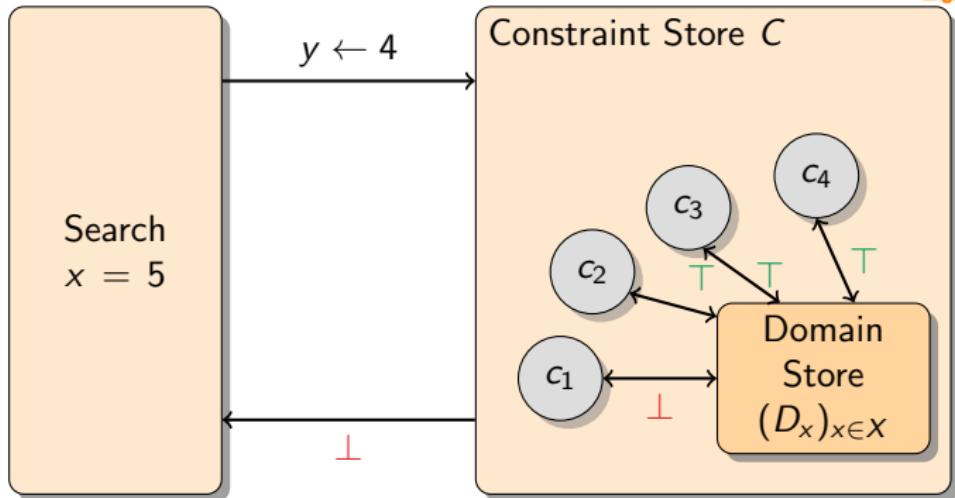


# Klassisches Constraint-Solving



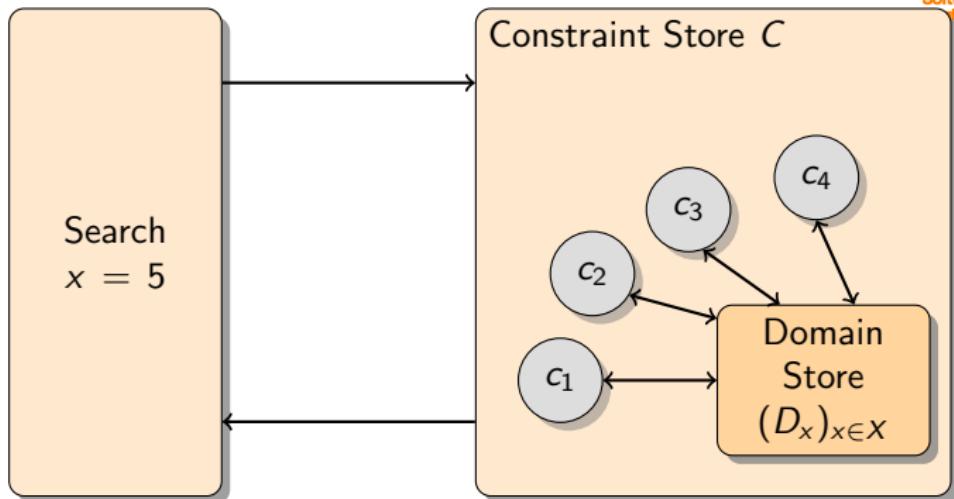
# Klassisches Constraint-Solving



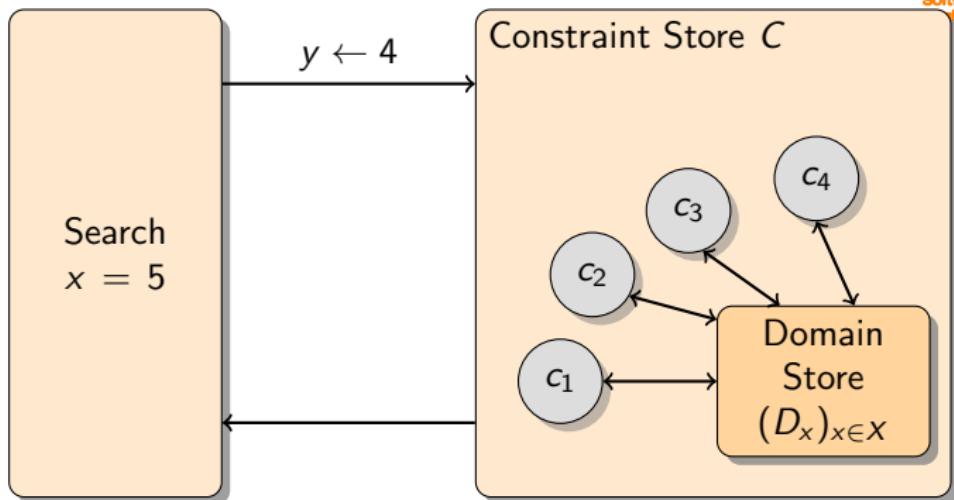


- Eine Menge von Erfüllungsgraden,  $\mathbb{B} = \{\perp, \top\}$
- Eine Kombinationsoperation  $\wedge$
- Ein neutrales Element  $\top$
- Eine partielle Ordnung  $(\mathbb{B}, \leq_{\mathbb{B}})$  mit  $\top <_{\mathbb{B}} \perp$

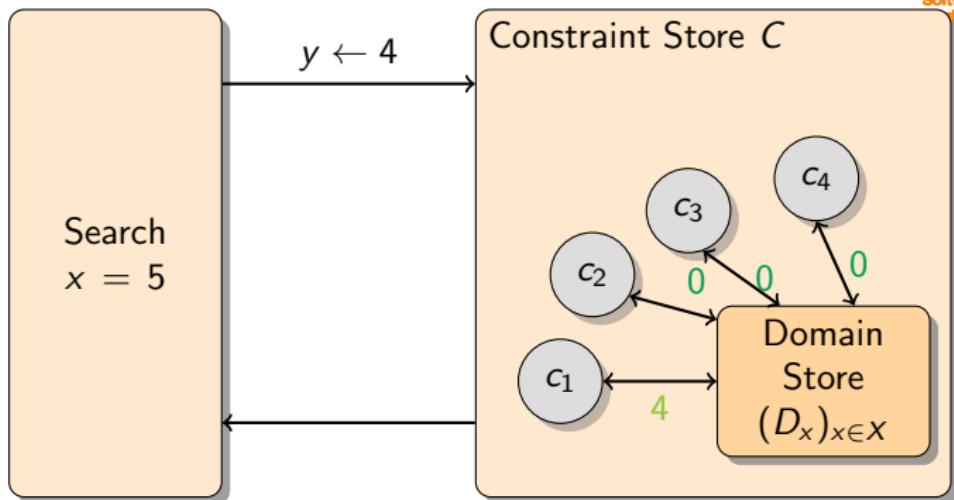
# Soft-Constraint-Solving



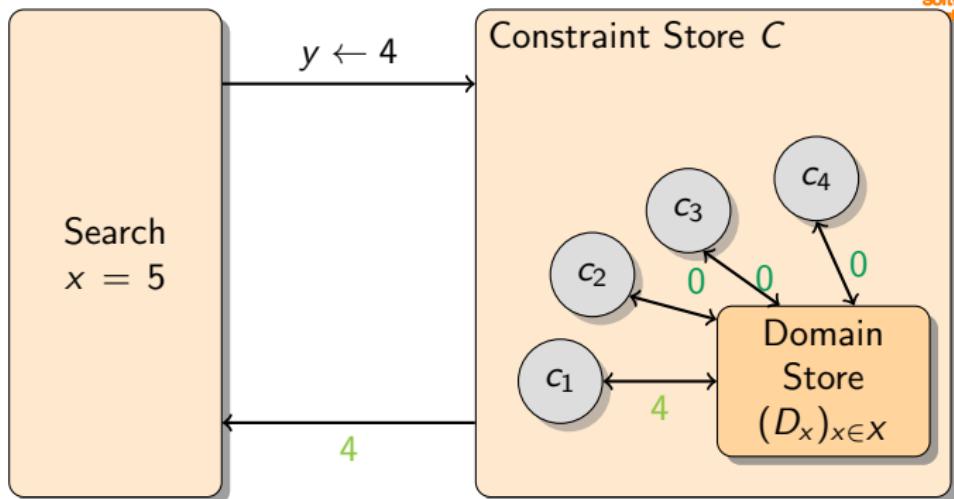
# Soft-Constraint-Solving

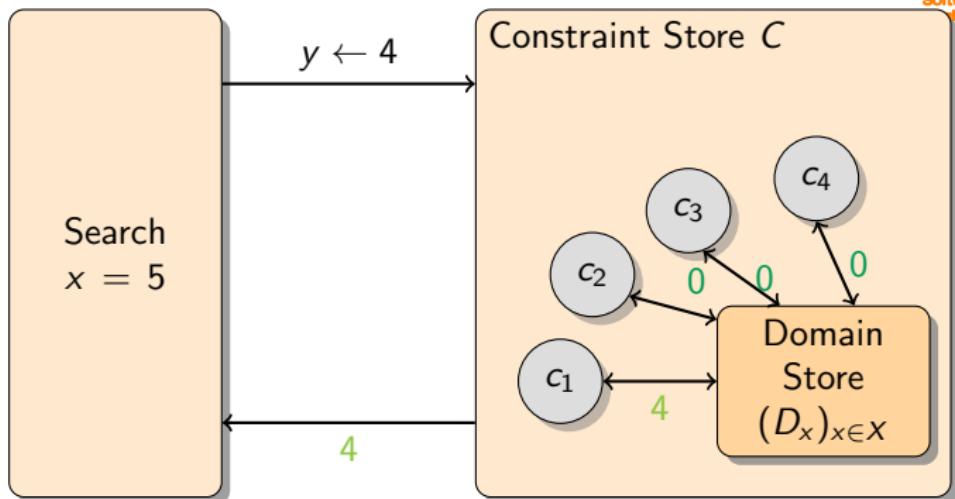


# Soft-Constraint-Solving



# Soft-Constraint-Solving





- Eine Menge von Erfüllungsgraden, z.B.,  $\{0, \dots, k\}$
- Eine Kombinationsoperation  $+$
- Ein neutrales Element 0
- Eine partielle Ordnung  $(\mathbb{N}_0, \geq)$  mit 0 als Top

Zugrundeliegende **algebraische Struktur**: 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

- $M \dots$  Elemente
- $\cdot_M \dots$  Kombination von Bewertungen
- $\varepsilon_M \dots$  neutrales, “bestes” Element
- $\leq_M \dots$  Ordnung, links “schlechter”

## Konkret

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

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

# SoftConstraints in MiniZinc

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

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

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

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

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

# SoftConstraints in MiniBrass

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

PVS: cr1 = new ConstraintRelationships("cr1") {
    soft-constraint c1: x < 500;
    soft-constraint c2: x > 500;

    crEdges : |- [| c1, c2 |] -|;
    useSPD: false ;
};

solve cr1;
```

**Ziel:** Weise Prüfungsslots an Studenten zu sodass:

- Jeder Student ist zufrieden mit seinem Datum (*stimmt zumindest zu*)
- Die Anzahl von Prüfungstagen ist minimiert (um das Zeitbudget der Prüfer 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			

- Präferenzen von Studenten sollten nicht unterschiedlich hoch gewichtet werden
- Lösung (Prüfplan) ist eine geteilte Entscheidung

# Exam Scheduling: Core Model

See exam-scheduling-approval.mzn:

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

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

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

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

# Exam Scheduling: Preferences

See exam-scheduling-approval.mzn:

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

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

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

- Gesammelte Präferenzen von 33 Studenten
- verteilt über 12 mögliche Termine (6 Tage, Vormittag und Nachmittag)
  - *Approval* set
  - *Impossible* set
- Kombiniert mittels **Approval Voting** (schöne wahltheoretische Eigenschaften!)
- Höchstens 4 pro Termin
- Findet optimale Lösung sofort (61 msec)
  - *Jeder* Student stimmt Termin zu
  - Wird mit der minimalen Anzahl von 9 Terminen erreicht
- Eingesetzte Strategie:

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

## Take-away

- ① Innovative Software steht vor *Constraint-Optimierungsproblemen*
- ② Modellierung für leistungsfähige Algorithmik
- ③ Algebraische Strukturen sind “in” ;-)

schiendorfer@isse.de

Try it in your own projects!

- Gadducci, F., Hözl, M., Monreale, G., and Wirsing, M. (2013).  
Soft constraints for lexicographic orders.  
In Castro, F., Gelbukh, A., and González, M., editors, *Proc. 12<sup>th</sup> 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. 33<sup>rd</sup> SGAI Int. Conf. Innovative Techniques and Applications of Artificial Intelligence (AI'13)*, pages 241–255. Springer.