Solving Soft Constraint Problems in MiniBrass

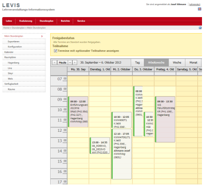
Alexander Schiendorfer et al.

Universität Augsburg
University

Decision Problems

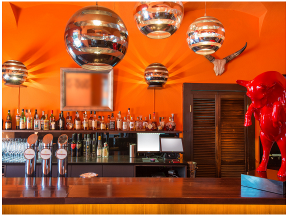
Institute for
Software & Systems
Engineering

“How many muffins of each kind?”



“When does which lecture take place?”

“Who performs which tasks?”



“What’s up on the weekend?”

What do these problems have in common?

Decisions *(variables)*

- The number of chocolate or banana muffins
- The lectures in the curriculum
- Friday and saturday activity

Possibilities *(domains)*

- Chocolate muffins: $\{0, \ldots, 20\}$
- Lecture "Algorithms 101": $\{\mathrm{LH1}, \mathrm{LH2}, \mathrm{LH3}, \ldots\}$

Dependencies *(constraints)*

- The required flour for x chocolate and y banana muffins may not exceed 250g.
- There can only be one class per room (at a time).
- There can only be one all-you-can-eat buffet per weekend.

What do these problems have in common?

Institute for Software & Systems Engineering

Preferences *(soft constraints)*

- There *should* be no algorithm lab on Friday, 8am
- Bernd would like to have steaks, Ada prefers sushi → Ada's preference is *more important*
- I'd rather not clean nor vacuum. But if I have to do either, cleaning is *worse*

and/or

Goals *(utility functions)*

- Maximize the revenue of our chocolate/banana mix
- Maximize the number of lecture-free days

a constraint satisfaction (optimization) problem (CSP/COP),
discrete if some decisions are over integers

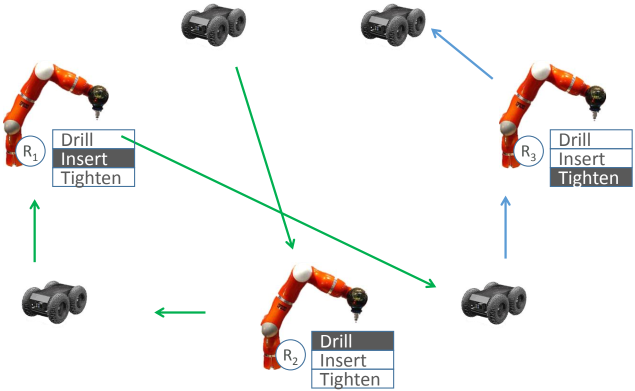
Discrete Optimization Problems in FAS* (1)

Institute for Software & Systems Engineering

Adaptive production cells

Discrete Optimization Problems in FAS* (1)

Goal: Assign tasks to robots such that a correct resource flow emerges



(Seebach et al., 2010), SASO

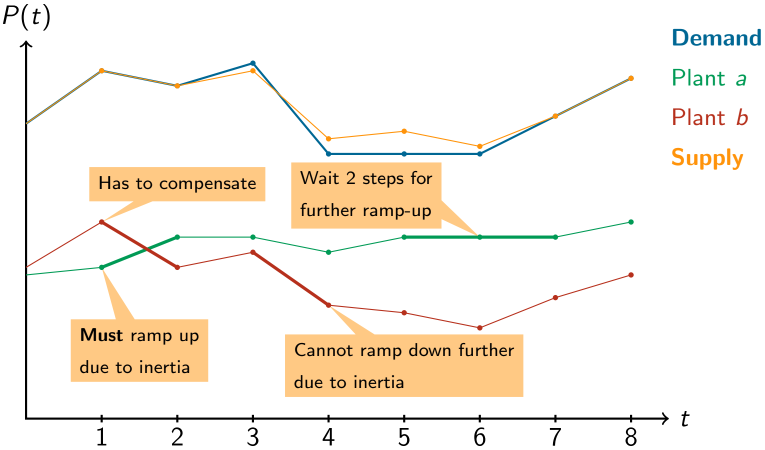
Discrete Optimization Problems in FAS* (2)

Institute for
Software & Systems
Engineering

Decentralized energy management

Discrete Optimization Problems in FAS* (2)

Goal: Schedule power plants such that they meet the demand

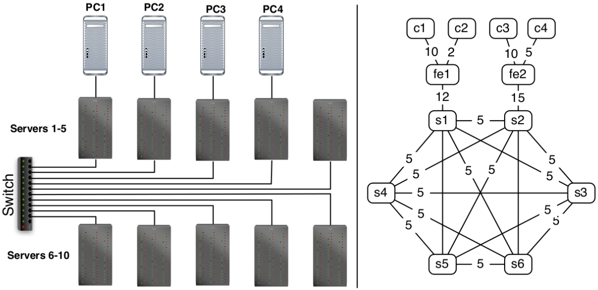


(Anders et al., 2013), SASO

Discrete Optimization Problems in FAS*

Institute for
Software & Systems
Engineering

ESDS Deployment Problem (*eventually serializable data services*)



(Michel et al., 2008), CPAIOR

Why is it useful to you?

Institute for Software & Systems Engineering

As a tool ...

- If you identify discrete optimization problems in your (self-organizing, autonomic, cloud) application, you can solve them with reliable tools.
- Modeling languages provide *easy* access to powerful solvers
- $\rightarrow$ *independent* of the concrete technology (SAT, CSP, Mathematical Programming)

As a research opportunity ...

- Studying interactions of complex networks of optimizing agents offers interesting problems (*global systems science*)
- Integration of optimization problems into architectures for decision-making
- Solving large-scale problems by decomposition through self-organization

Soft Constraint Programming in MiniBrass

Institute for Software & Systems Engineering

Constraint programming (first half of the tutorial)

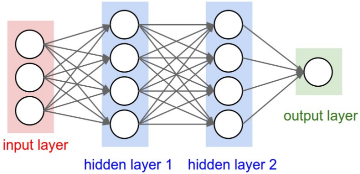
- Declarative programming (similar to SQL, Prolog)
- Separation of **model** and *algorithm*
- Suitable for combinatorial problems with hard constraints (e.g. physics!)
- Modeling language MiniZinc

Soft constraint programming (second half of the tutorial)

- Modeling of **user preferences**
- Find solutions that are *as good as possible*
- What does “good“ mean?
- Modeling language MiniBrass

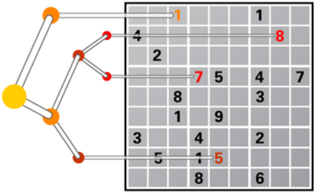
Artificial Intelligence: Categories

Data-driven AI



- Machine Learning
- Signal Processing
- Computer Vision

Decision-driven AI



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

Our first model

Institute for Software & Systems Engineering

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

constraint x < y;

solve maximize x + y;
```

http://www.minizinc.org

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

A second example

```
var 0..2: x; % this notation is shorthand for the set {0,1,2}
var 0..2: y;
var 0..1: z; % {0,1}

% c1
constraint x != y /\ y != z /\ x != z;
% c2
constraint x + 1 = y;

solve satisfy;
```

Can you give a solution to this problem?

- $\Theta = \{(x \to 1, y \to 2, z \to \mathbf{?}), (x \to 0, y \to 1, z \to \mathbf{?})\}$ satisfy c_2;
- $(x \to 0, y \to 1, z \to ?)$ cannot be extended to a solution since z has to be 0 or 1 $\to$ has to violate c_1
- Hence, the only solution is $(x \to 1, y \to 2, z \to 0)$

Why modeling?

Institute for Software & Systems Engineering

1. Once stated (formally) – solved many times
   - Constraint-Solving
   - SAT – Boolean Satisfiability
   - MIP – Mixed Integer Programming
   - Heuristic Optimization – Genetic etc.
2. Efficient and reliable algorithms provided by solvers
   - Same algorithms for a variety of different problems
   - Dedicated algorithms for recurring combinatorial substructures (`alldifferent`)
3. Prototyping
   - Specification becomes much clearer
   - Bespoke algorithm for concrete problem can be developed later

Modeling in other CS domains

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

instead of

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

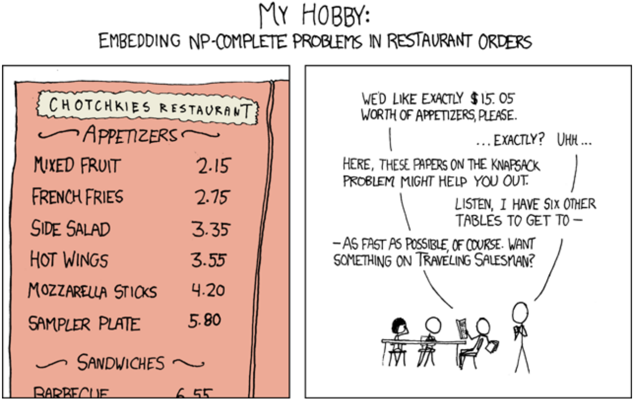
Motivation

Constraint modeling for optimization problems ≈ SQL for database access

NP-Completeness

Theorem

The decision problem associated to a constraint satisfaction/optimization problem is NP-complete.



https://xkcd.com/287/

Why MiniZinc?

Institute for
Software & Systems
Engineering

Rationale

One modeling language – many solvers

Supported solvers (selection)

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

A relevant example . . .

Institute for

```
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
```

Basic MiniZinc: Parameters and Variables

Institute for Software & Systems Engineering

Parameters: input data for the problem, constants

```
int: n;
n = 3;
% equivalent to:
int: n = 3;
par int: n = 3;
% or with bounded domains
0..10: m; % a fixed value (input by a user) between 0 and 10
% set types
set of int: AGENTS = 1..n; % {1, ..., n}
```

Decision Variables: variables to be assigned by a solver

```
var int: n;
var 0..10: x;
% immediate declaration (useful for dependent expressions)
var int: y = 2*x;
```

Basic MiniZinc: Arrays and Sets

Institute for Software & Systems Engineering

Arrays: for parameters/variables of different sizes

```
int: n;
array[1..n] of int: height; % height[i] denotes the height of truck i
% also for decisions
array[1..n] of var bool: x; % x[i] holds if we select item i
% typical usage
set of int: ITEMS = 1..n;
array[ITEMS] of var bool: x;
```

Sets: set type for parameters and decisions (only set of int/subtype int)

```
set of int: AGENTS = 1..n;
set of int: CAP = 1..m;
array[AGENTS] of set of CAP: offers; % [{1}, {2}, {1,2}]

var set of AGENT: selectedTeam;
```

Basic MiniZinc: Types

Allowed primitive types for variables/parameters are

- Integer `int` or subtypes:
  - range `1..n`
  - explicit set `{1,5,7}`
- Floating point number `float` or subtypes:
  - range `0.0 .. 1.0`
  - explicit set `{1.0, 2.5, 3.7}`
- Boolean `bool`
- Arrays, Sets

Constraints, Objective and Output

Constraints: to restrict assignments

```
var 0..10: x; var 0..10: y; var 0..10: z;
constraint x + y = z;
constraint x mod 2 = 0;
% 'forall' concept for arrays
par int: n; array[1..n] of var 0..10: t;
% each t[i] should be even:
constraint forall(i in 1..n) (t[i] mod 2 = 0);
```

Solve Item: provides the objective (one per model)

```
solve satisfy; % for a satisfaction problem
solve maximize sum(i in 1..n) (t[i]);
solve minimize x + y;
```

String Output: one output item per model

```
var 0..10: x;
output ["The value of x is: \(x)"]
```

MiniZinc: HelloWorld

Exercise 1
Build a MiniZinc model `xopt.mzn` with a decision variable x taking values from 0 to 10, with constraints to ensures that x is divisible by 4, which outputs the value of x that gives the minimum value of $(x-7)^2$.
Test it using the precompiled IDE-bundle. Suppose you cannot use the `mod` function, how would you alternatively model that x is divisible by 4?

MiniZinc: Arrays

Exercise 2 Define a MiniZinc model `array.mzn` which takes an integer parameter n defining the length of an array of numbers x taking values from 0 to 9. Constrain the array so the sum of the numbers in the array is equal to the product of the numbers in the array. Output the resulting array. Test your model using the "all solutions" setting active in the IDE.

Add a constraint to ensure that the numbers in the array are non-decreasing, i.e. $x[1] \leq x[2] \leq \ldots \leq x[n]$. This should reduce the number of similar solutions. How big a number can you solve with your model? Why do you think this happens?

Deciding functions

Institute for Software & Systems Engineering

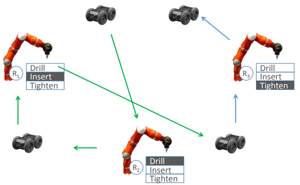
- A very important, recurring problem is *deciding* a finite function.
- Given
  - Domain `DOM`
  - Codomain `COD`
- Find a function $f : \texttt{DOM} \rightarrow \texttt{COD}$
- Satisfying some criteria (e.g. *injectivity*)

Examples:

- Task allocation (maps tasks to workers, exactly one worker per task)
- Exam schedules (map students to appointments, certainly different appointments)

Task allocation in practice

Institute for
Software & Systems
Engineering



- Task allocation problem
  - n robots
  - m tasks
  - Assign each robot a *different* task and maximize the profit
- Example:
  - $n = 4,\ m = 5$

	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

Task allocation: Model

Institute for Software & Systems Engineering

```
% 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 forall(r1 in ROBOTS, r2 in ROBOTS where r1 < r2)
  ( allocation[r1] != allocation[r2] );
```

Judgement

```
constraint forall(r1 in ROBOTS, r2 in ROBOTS where r1 < r2)
  ( allocation[r1] != allocation[r2] );
```

- In principle, *fine*, but ...
  - Not very concise – can be messed up
  - $O(n^2)$ individual constraints between two variables
- But in fact, this substructure is so common that it deserves a *name*
  - `alldifferent`

```
constraint alldifferent(allocation);
```

- $\mathrm{alldifferent}([x_1, \ldots, x_n]$ is true if and only if all variables x_1 to x_n take a different value
- e.g. $\mathrm{alldifferent}([2, 3, 5])$ holds but $\mathrm{alldifferent}([2, 2, 3])$ does not.

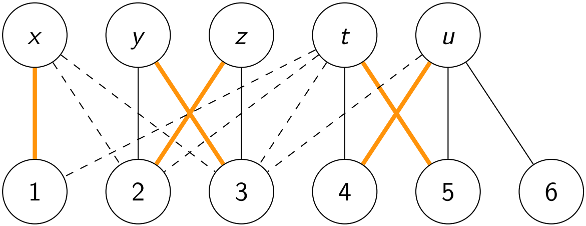
Task allocation: Improved Model

Institute for Software & Systems Engineering

```
% 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]] );

include "alldifferent.mzn"; % we have to import from the library
% have robots work on different tasks
constraint alldifferent(allocation);
```

AllDifferent – Propagation

```
var {1,2,3}: x;     var {2,3}: y;    var {2,3}: z;
var {1,2,3,4,5}: t; var {3,4,5,6}: u;
constraint alldifferent([x,y,z,t,u]);
```




```
var {1}: x;     var {2,3}: y;    var {2,3}: z;
var {4,5}: t;    var {4,5,6}: u;
constraint alldifferent([x,y,z,t,u]);
```

Other Globals

Assume an employee scheduling problem (rostering)
(Shifts: 1 = morning, 2 = afternoon, 3 = night):

	Monday	Tuesday	Wednesday	Thursday	Friday
Nurse 1	off	1	2	3	2
Nurse 2	1	off	1	1	1
Nurse 3	1	2	off	1	2
Nurse 4	2	2	2	off	3
Nurse 5	3	3	2	3	off

- Consider work regulations
    - At least one nurse has to be assigned to every shift per day
    - Each nurse may at most work two night shifts
    - Each nurse needs to have one day off

Cardinality Constraint

Institute for Software & Systems Engineering

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

- $\text{cardinality}(X \mid v, l, u)$
  - $X = \{x_1, \dots, x_n\}$
  - $v = (v_1, \dots, v_m)$
  - $l = (l_1, \dots, l_m)$ contain *lower* and $u = (u_1, \dots, u_m)$ *upper* bounds for values v_i
- For instance, for each nurse i:
  - $\text{cardinality}([\text{worksShift}[i, \cdot]] \mid (0, 1, 2, 3), (1, 0, 0, 0), (5, 5, 5, 2))$
- In MiniZinc:

```
constraint forall(i in NURSES) (
      global_cardinality_low_up([worksShift[i,d] | d in DAYS],
                              [0,1,2,3], [1,0,0,0], [1,5,5,2] ) );
```

MiniZinc: Group Photo

Exercise 3

Given a group of *n* people, we must arrange them for a photo. The best photo is when people are next to their friends, so the aim is to arrange them so that each person is next to (to the left or right) with as many friends as possible. The data for the problem is given as

```
n = <size of problem> ;
array[1..n,1..n] of var bool:  friend;
```

where `friend[f1, f2]` means `f1` and `f2` are friends. You can assume that the friend array is symmetric. You should output a list of the people in their position to maximize the number of adjacent friends. For example given the data `groupphoto1.dzn`, you should output the placement of the guests as well as the objective value, i.e.,

```
Obj = 7; [4, 3, 5, 6, 8, 7, 1, 2]
```

References I

Anders, G., Steghöfer, J.-P., Siefert, F., and Reif, W. (2013).
A trust-and cooperation-based solution of a dynamic resource allocation problem.
In *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*, pages 1–10. IEEE.

Michel, L., Shvartsman, A., Sonderegger, E., and Van Hentenryck, P. (2008).
Optimal deployment of eventually-serializable data services.
In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 188–202. 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.