Declaring Local-Search Neighbourhoods in MiniZinc

Gustav Björdal[1]
Pierre Flener,[1] Justin Pearson,[1] Peter J. Stuckey,[2] and Guido Tack[3]
[1] Uppsala University, [2] University of Melbourne, [3] Monash University

Goal

- Extend the MiniZinc language with declarative neighbourhoods.
- Allow rapid prototyping with local search strategies.
- Recreate known local search strategies in MiniZinc.

Goal

- Extend the MiniZinc language with declarative neighbourhoods.
- Allow rapid prototyping with local search strategies.
- Recreate known local search strategies in MiniZinc.

This presentation

Some background and a short walkthrough of our new MiniZinc syntax.

Gustav Björdal
Uppsala University

MiniZinc

Gustav Björdal
Uppsala University

MiniZinc

Constraint-based declarative modelling language.

Solver and technology independent: more than 15 backends.

High-level syntax.

Free and open-source under MPL 2.0.

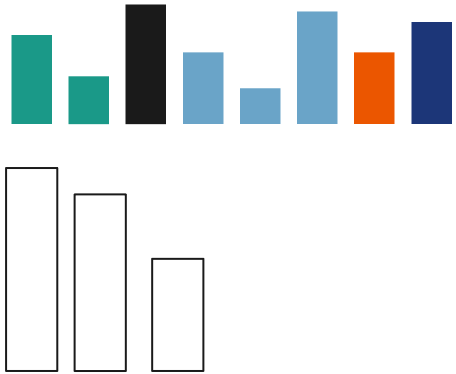
Annual MiniZinc Challenge.

www.minizinc.org

Running Example – Steel Mill Slab Design

Given

- orders of steel, each having a *size* and *colour*, to be cut from slabs
- available slab sizes



Gustav Björdal
Uppsala University

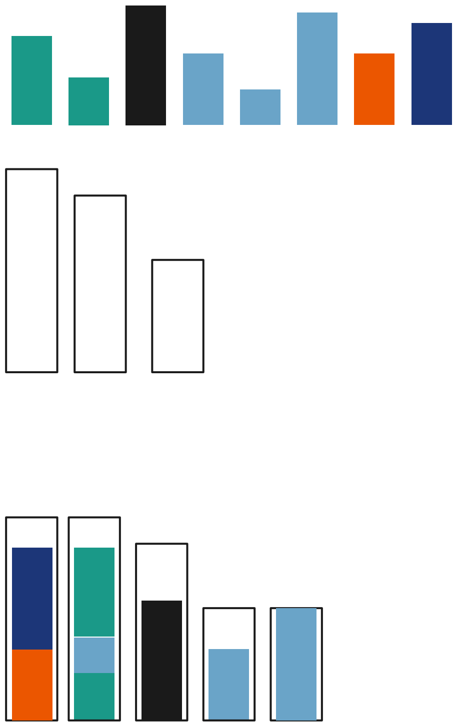
Running Example – Steel Mill Slab Design

Given

- orders of steel, each having a *size* and *colour*, to be cut from slabs
- available slab sizes

Decide

- how many slabs to use
- the size of each slab
- the slab each order is placed in

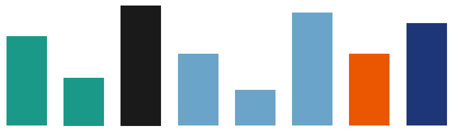


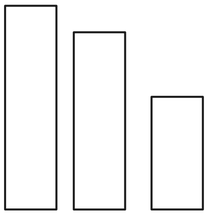
Gustav Björdal
Uppsala University

Running Example – Steel Mill Slab Design

Given

- orders of steel, each having a *size* and *colour*, to be cut from slabs
- available slab sizes



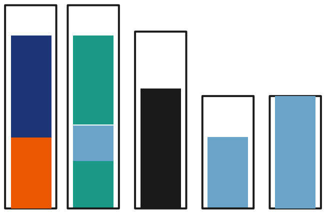


Decide

- how many slabs to use
- the size of each slab
- the slab each order is placed in

Such that

- at most two colours on each slab
- the total slack in the slabs is minimal

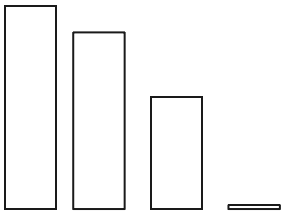


Gustav Björdal
Uppsala University

Running Example – Steel Mill Slab Design

Given

- orders of steel, each having a *size* and *colour*, to be cut from slabs
- available slab sizes

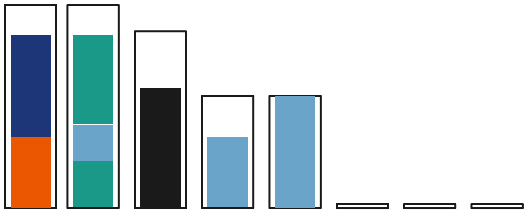


Decide

- ~~how many slabs to use~~
- ~~the size of each slab~~
- the slab each order is placed in

Such that

- at most two colours on each slab
- the total slack in the slabs is minimal



Gustav Björdal
Uppsala University

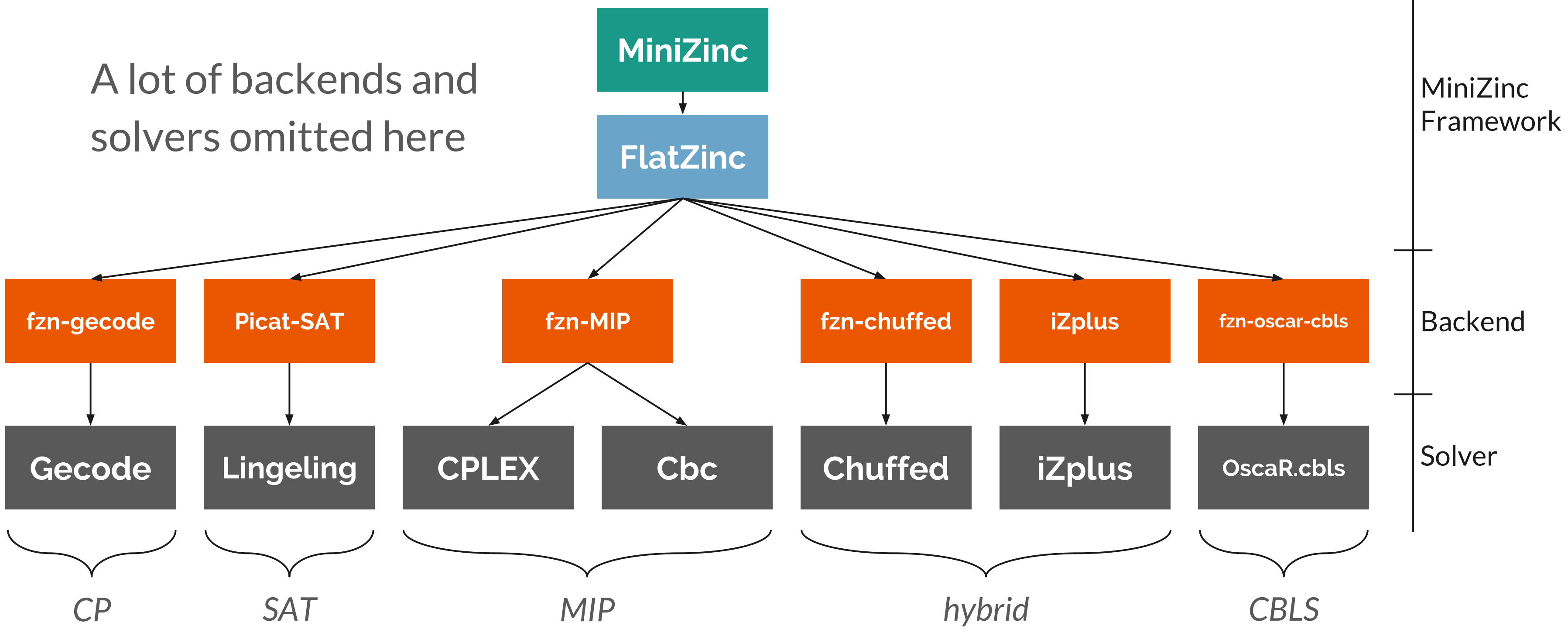
MiniZinc Model

```
% Some parameters omitted
...
array [Orders] of int: size;
array [Orders] of Colors: color;
array [0..maxCapa] of 0..maxCapa: slack = ... ;
% Variables:
array [Orders] of var Slabs: placedIn;
% Constraints:
array [Slabs]  of var 0..maxCapa: load;
constraint bin_packing_load(load, placedIn, size);
array [Slabs]  of var 0..2: nColors;
constraint forall(s in Slabs)(nColors[s] = ... );
% Objective:
var int: objective = sum(s in Slabs)(slack[load[s]]);
solve minimize objective;
```

Gustav Björdal
Uppsala University

Current Landscape

A lot of backends and solvers omitted here



Gustav Björdal
Uppsala University

(Constraint-Based) Local Search

Local Search

1. Start from an initial assignment of all variables
2. While some condition holds:
   a. Generate a neighbourhood of similar assignments
   b. Move to a best neighbour
3. Return the best solution found

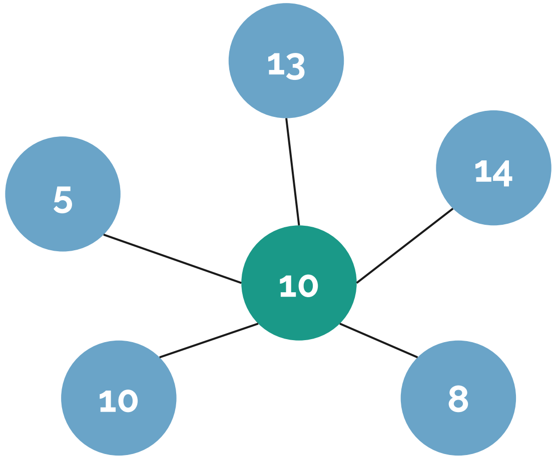
Gustav Björdal
Uppsala University

Local Search

1. **Start from an initial assignment of all variables**
2. While some condition holds:
   a. Generate a neighbourhood of similar assignments
   b. Move to a best neighbour
3. Return the best solution found

Local Search

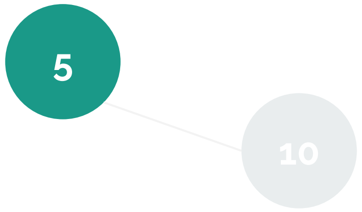
1. Start from an initial assignment of all variables
2. While some condition holds:
   a. **Generate a neighbourhood of similar assignments**
   b. Move to a best neighbour
3. Return the best solution found



Gustav Björdal
Uppsala University

Local Search

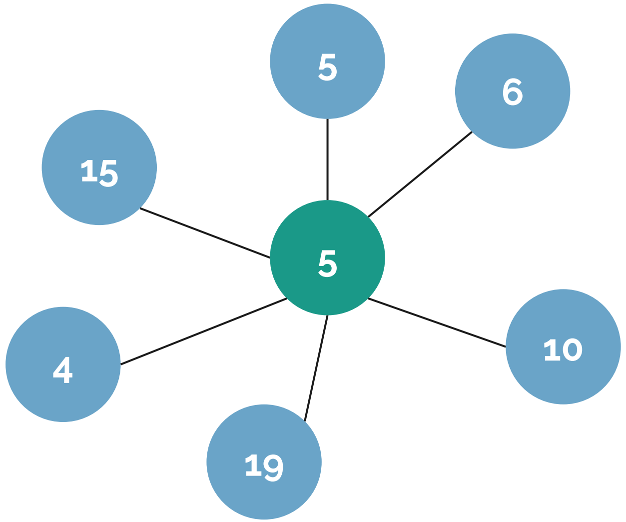
1. Start from an initial assignment of all variables
2. While some condition holds:
   a. Generate a neighbourhood of similar assignments
   b. Move to a best neighbour
3. Return the best solution found



Gustav Björdal
Uppsala University

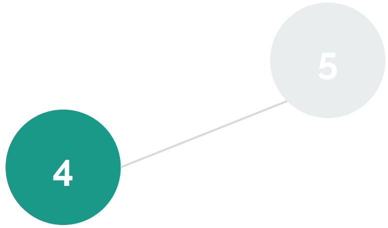
Local Search

1. Start from an initial assignment of all variables
2. While some condition holds:
   a. **Generate a neighbourhood of similar assignments**
   b. Move to a best neighbour
3. Return the best solution found



Local Search

1. Start from an initial assignment of all variables
2. While some condition holds:
   a. Generate a neighbourhood of similar assignments
   b. **Move to a best neighbour**
3. Return the best solution found

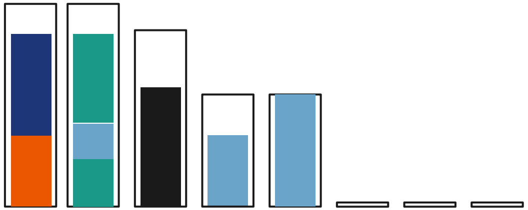


Gustav Björdal
Uppsala University

Local Search

1. Start from an initial assignment of all variables
2. While some condition holds:
   a. Generate a neighbourhood of similar assignments
   b. Move to a best neighbour
3. **Return the best solution found**

Steel Mill Slab Design

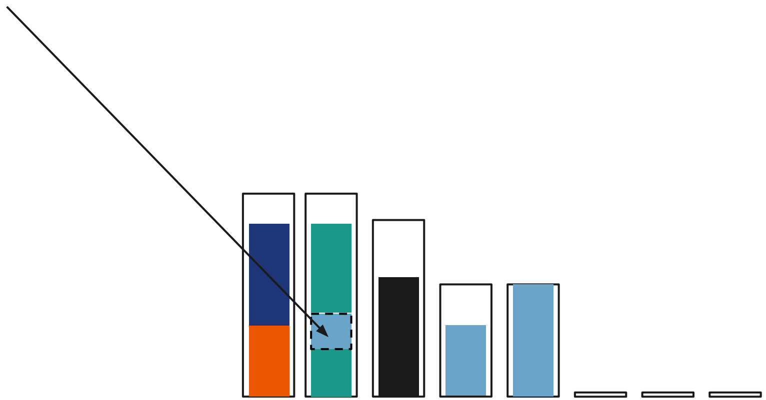


An initial assignment

Gustav Björdal
Uppsala University

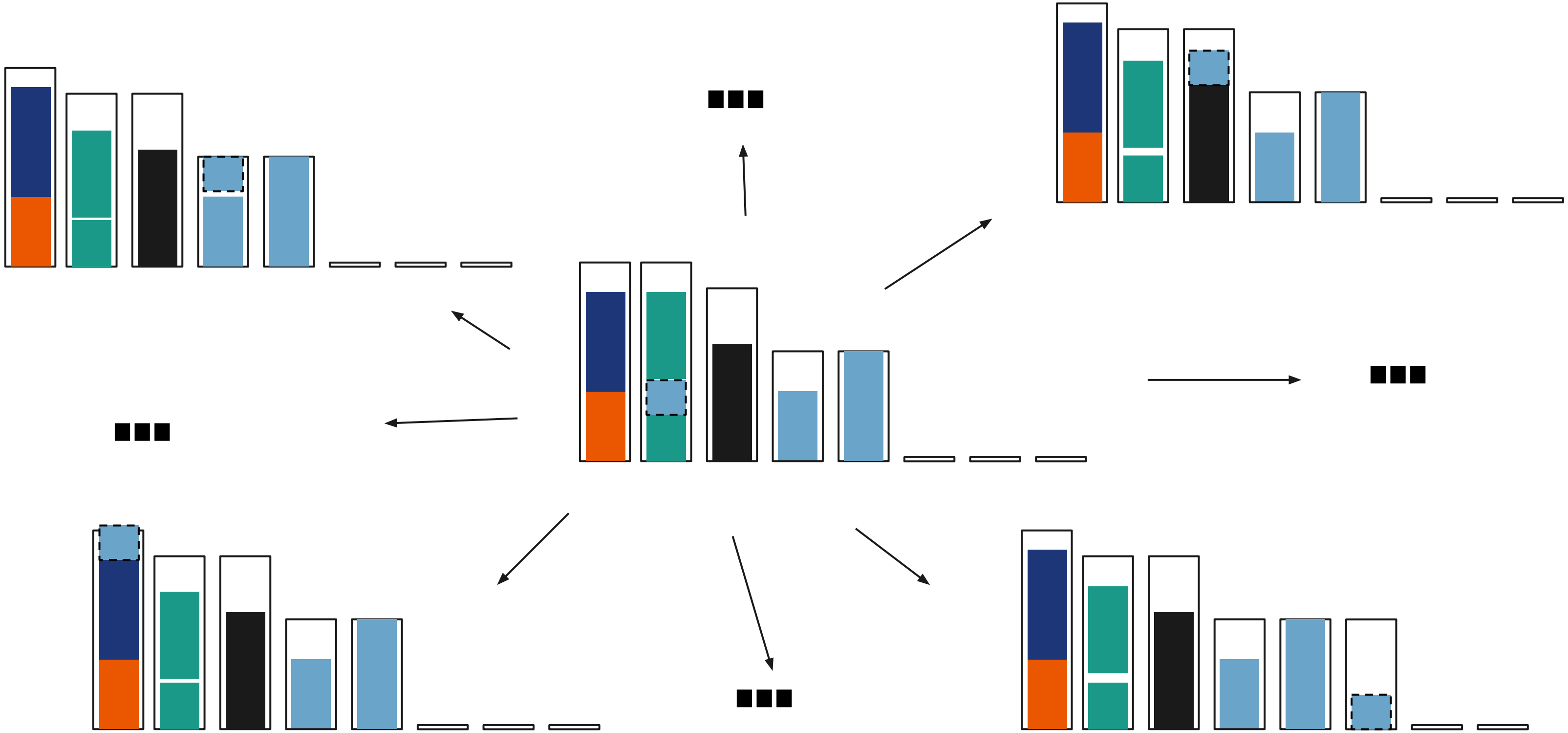
Steel Mill Slab Design

Move an order to another slab.



Gustav Björdal
Uppsala University

Steel Mill Slab Design



Design Aspects

Initialisation and neighbourhood

- How do we initialise?
- What are the moves?

Heuristic

- How do we explore the neighbourhood?

Meta-heuristic

- How do we prevent the search from getting stuck?

Gustav Björdal
Uppsala University

Constraint-Based Local Search (CBLS)

CBLS = CP-style declarative modelling + local search

A CBLS framework has:

- a library of reusable components
- a language for writing a local search procedure

Constraint-Based Local Search (CBLS)

CBLS = CP-style declarative modelling + local search

A CBLS framework has:

- a library of reusable components
- a language for writing a local search procedure

CBLS for MiniZinc

MiniZinc is a modelling language: must use black-box local search.
Works well but can be hit and miss.

Constraint-Based Local Search (CBLS)

CBLS = CP-style declarative modelling + local search

A CBLS framework has:

- a library of reusable components
- a language for writing a local search procedure

CBLS for MiniZinc

MiniZinc is a modelling language: must use black-box local search.
Works well but can be hit and miss.

Idea

Allow modellers to define (part of) a local search strategy in MiniZinc.

Gustav Björdal
Uppsala University

Declarative Neighbourhoods in MiniZinc

What is a Neighbourhood?

A neighbourhood is a set of moves.

For example:

$\{X \leftarrow v \mid v \in 1 \,..\, 10\}$ i.e., $\{X \leftarrow 1,\ X \leftarrow 2,\ \ldots\ ,\ X \leftarrow 10\}$

What is a Neighbourhood?

A neighbourhood is a set of moves.

For example:

$\{X \leftarrow v \mid v \in 1..10\}$ i.e., $\{X \leftarrow 1,\ X \leftarrow 2,\ \ldots\ ,\ X \leftarrow 10\}$

A neighbourhood is not about:

- how to explore the neighbourhood
- how to evaluate the quality of a move
- which move to select

These are heuristics.

Gustav Björdal
Uppsala University

What is a Neighbourhood?

A neighbourhood is a set of moves.

For example:

$\{X \leftarrow v \mid v \in 1..10\}$ i.e., $\{X \leftarrow 1,\ X \leftarrow 2,\ \dots\ ,\ X \leftarrow 10\}$

A neighbourhood is not about:

- how to explore the neighbourhood
- how to evaluate the quality of a move
- which move to select

These are heuristics.

We provide syntax only for defining the neighbourhood, not heuristics.

Basic Syntax

Move operators

```
X := v
X[i] := v
X :=: Y
X[i] :=: Y[j]
```

Basic Syntax

Move operators

```
X := v
X[i] := v
X :=: Y
X[i] :=: Y[j]
```

Declaring a set of moves

```
moves(v in 1..10)(X := v)
```

Gustav Björdal
Uppsala University

Basic Syntax

Move operators

```
X    := v
X[i] := v
X    :=: Y
X[i] :=: Y[j]
```

Declaring a set of moves

```
moves(v in 1..10)(X := v)
```

Compound moves

```
moves(i,j in Idx)(Xs[i] := Xs[j] /\ Xs[j] := Xs[i])
```

Gustav Björdal
Uppsala University

Pre- and Post-Conditions

Moves can have pre- and post-conditions in the form of CSPs.

Pre-condition

```
moves(i in Idx, v in Dom where Xs[i] > v)(Xs[i] := v)
```

Gustav Björdal
Uppsala University

Pre- and Post-Conditions

Moves can have pre- and post-conditions in the form of CSPs.

Pre-condition

```
moves(i in Idx, v in Dom where Xs[i] > v)(Xs[i] := v)
```

Post-condition

```
moves(i in Idx, v in Dom)(
    Xs[i] := v /\ ensuring(alldifferent(Xs))
```

Initialisation

A neighbourhood can have an initialisation post-condition that must hold upon the initial assignment (and restarts).

Permutation neighbourhood

```
initially(alldifferent(Xs)) /\
moves(i, j in Idx where i < j)(Xs[i] :=: Xs[j])
```

Union of Neighbourhoods

The union of two neighbourhoods can be expressed.

Swaps and assigns

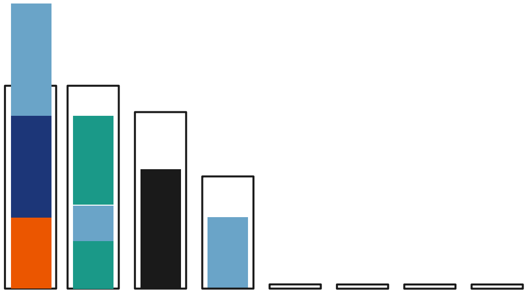
```
moves(i in Idx, v in Dom)(Xs[i] := v)
union
moves (i, j in Idx where i < j)(Xs[i] :=: Xs[j])
```

Example 1 – Basic Neighbourhood

```
moves(o in Orders, s in Slabs)(placedIn[o] := s)
```

Example 1 – Basic Neighbourhood

```
moves(o in Orders, s in Slabs)(placedIn[o] := s)
```

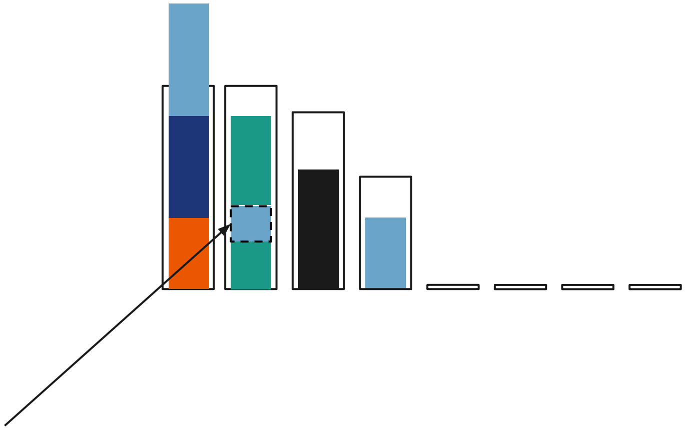


An initial assignment

Gustav Björdal
Uppsala University

Example 1 – Basic Neighbourhood

```
moves(o in Orders, s in Slabs)(placedIn[o] := s)
```

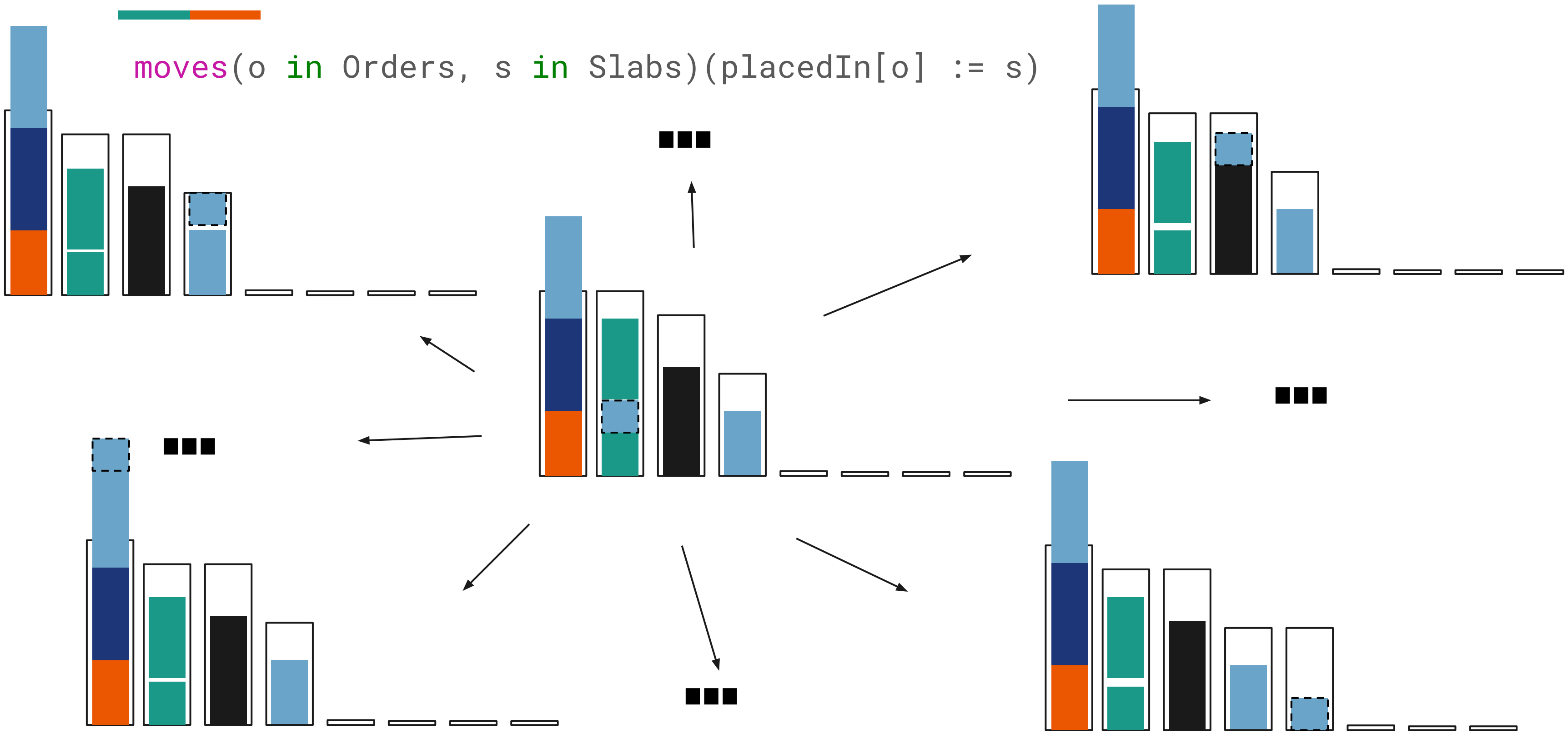


Move an order to another slab.

Gustav Björdal
Uppsala University

Example 1 – Basic Neighbourhood

```
moves(o in Orders, s in Slabs)(placedIn[o] := s)
```



Gustav Björdal
Uppsala University

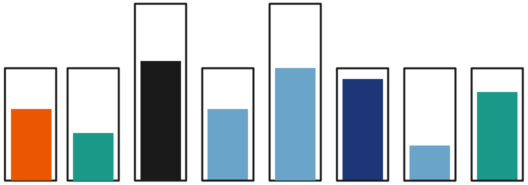
Example 2 – Only Feasible Solutions

```
initially(forall(o in Orders)(placedIn[o] = o))
/\
moves(o in Orders, s in Slabs where size[o] + load[s] <= maxCapa)(
         placedIn[o] := s /\ ensuring(nColors[s] <= maxColors))
```

Example 2 – Only Feasible Solutions

```
initially(forall(o in Orders)(placedIn[o] = o))
/\
moves(o in Orders, s in Slabs where size[o] + load[s] <= maxCapa)(
          placedIn[o] := s /\ ensuring(nColors[s] <= maxColors))
```

An initial assignment

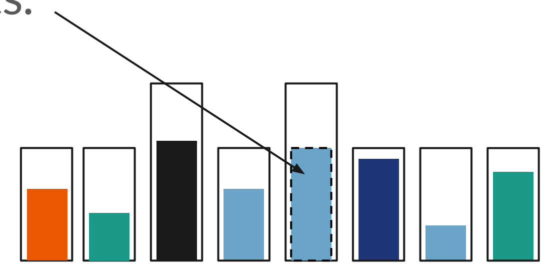


Gustav Björdal
Uppsala University

Example 2 – Only Feasible Solutions

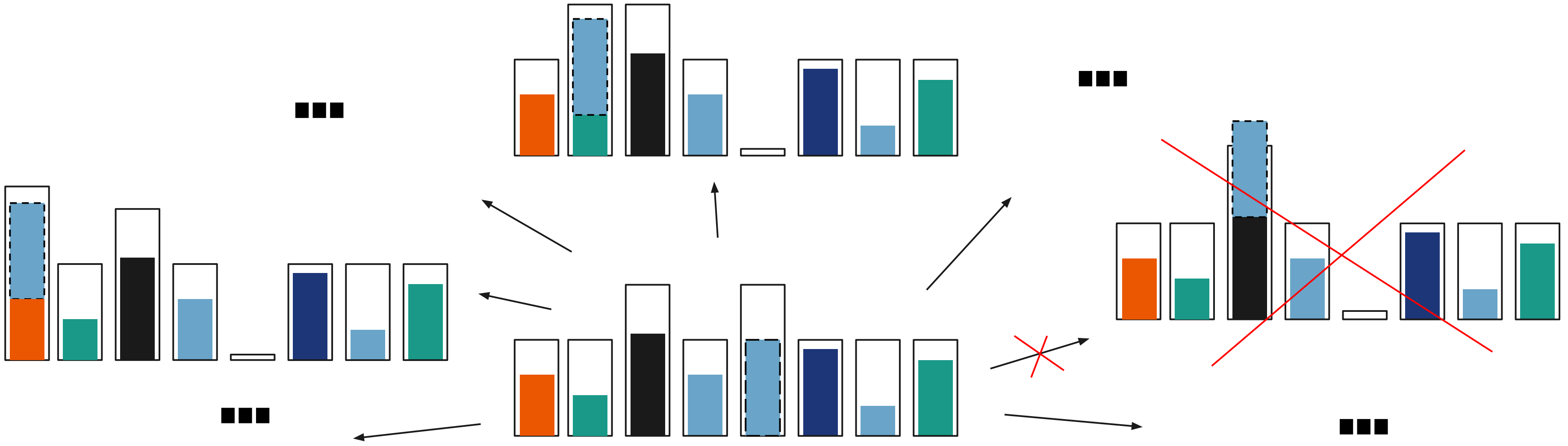
```
initially(forall(o in Orders)(placedIn[o] = o))
/\
moves(o in Orders, s in Slabs where size[o] + load[s] <= maxCapa)(
         placedIn[o] := s /\ ensuring(nColors[s] <= maxColors))
```

Move an order to another slab, preserving the colour and capacity constraints.



Example 2 – Only Feasible Solutions

```
initially(forall(o in Orders)(placedIn[o] = o))
/\
moves(o in Orders, s in Slabs where size[o] + load[s] <= maxCapa)(
         placedIn[o] := s /\ ensuring(nColors[s] <= maxColors))
```



Gustav Björdal
Uppsala University

Under the Hood

Some Highlights

We extend FlatZinc with flat functions.

Initialisation done using the CP solver OscaR.cp.

Move pre- and post-conditions checked using OscaR.cbls constraint system.

Gustav Björdal
Uppsala University

Experimental Evaluation

Setup

1. Steel mill slab design is solved using CBLS in [1].
   We recreated their neighbourhoods in MiniZinc.
2. We added neighbourhoods to existing MiniZinc models.

We only compare the results versus fzn-oscar-cbls (black-box).

1. Schaus, P., Van Hentenryck, P., Monette, J.N., Coffrin, C., Michel, L., Deville, Y.: Solving steel mill slab problems with constraint-based techniques: CP, LNS, and CBLS. Constraints 16(2), 125–147 (April 2011).

Results Compared to Black-Box

About 20% improvement in objective value compared to black-box neighbourhood.

Results Compared to Black-Box

About 20% improvement in objective value compared to black-box neighbourhood.

For more complex neighbourhoods, we see overall:

- a decrease in terms of iterations per second
- a speedup in terms of time until best solution found
- an improvement in terms of quality

Results Compared to Black-Box

About 20% improvement in objective value compared to black-box neighbourhood.

For more complex neighbourhoods, we see overall:

- a decrease in terms of iterations per second
- a speedup in terms of time until best solution found
- an improvement in terms of quality

For neighbourhoods similar to the black-box ones:

- an overall slowdown
- this is expected

Gustav Björdal
Uppsala University

Conclusion & Future Work

Conclusion & Future Work

A good starting point

- Grey-box
- Declarative neighbourhoods are expressive and powerful
- One can now experiment with neighbourhoods in MiniZinc

More to do

- White-box
- Declarative language for expressing heuristics and meta-heuristics
- Overheads to trim

Questions?

Thank You!