Generating Compound Moves for Local Search by Systematic Search

Gustav Björdal,
joint work with Pierre Flener and Justin Pearson
Research Overview

We have previously developed a local-search backend to the modelling language MiniZinc.

Such a backend must automatically infer a search strategy from a model.

I will here discuss a case where this inference can go wrong and what can be done about it.
TSP with Time Windows
TSP with Time Windows (TSPTW)

Given a graph with weighted edges and a time window for each node:
TSP with Time Windows (TSPTW)

Given a graph with weighted edges and a time window for each node:
Given a graph with weighted edges and a time window for each node:

find a shortest Hamiltonian path, such that each node is visited within its time window.
TSP with Time Windows (TSPTW)

Given a graph with weighted edges and a time window for each node:

find a shortest Hamiltonian path, such that each node is visited within its time window.
Given a graph with weighted edges and a time window for each node:
find a shortest Hamiltonian path, such that each node is visited within its time window.

TSP with Time Windows (TSPTW)
TSP with Time Windows (TSPTW)

Given a graph with weighted edges and a time window for each node:

find a shortest Hamiltonian path, such that each node is visited within its time window.
Representing a Route

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Representing a Route

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>D</th>
<th>F</th>
<th>E</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Modelling TSPTW in MiniZinc
Model Overview

Data
- Nodes
- Distances
- Time windows and duration

Variables
- The route
- Arrival times

Constraints
- Visit each node once
- Within time windows

Objective
- Total distance
Data

set of int: Nodes = 1..7;
set of int: Positions = 1..7;  % positions in the route
array[Nodes, Nodes] of int: distance = ...;
array[Nodes] of int: open = ...;
array[Nodes] of int: close = ...;
Data

set of int: Nodes = 1..7;
set of int: Positions = 1..7;
array[Nodes, Nodes] of int: distance = ...;
array[Nodes] of int: open = ...;
array[Nodes] of int: close = ...;

distance[2, 5] is the time we spend at node 2 plus the travel time from node 2 to node 5.
Data

set of int: Nodes = 1..7;
set of int: Positions = 1..7;
array[Nodes, Nodes] of int: distance = ...;
array[Nodes] of int: open = [0, 0, ...];
array[Nodes] of int: close = [200, 40, ...];
Variables

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;
Variables

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;

route =

<table>
<thead>
<tr>
<th>?</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>?</th>
</tr>
</thead>
</table>
Variables

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;

route = [1 2 3 4 5 6 7]

route[1] = first node we visit
route[2] = second node we visit
...
route[13] = last node we visit
Constraints

We do not visit the same node twice:

constraint alldifferent(route);
Constraints

We do not visit the same node twice:

$constraint \text{alldifferent}(\text{route});$

We arrive at the node at the first position after it opens:

$constraint \text{arrivalTime}[1] \geq \text{open}[\text{route}[1]];$
We arrive at the node at position $i$ after it opens and after we are done at the node at position $i-1$ plus the travel time:

```plaintext
constraint forall(i in Positions where i != 1)(
    arrivalTime[i] >= open[route[i]]
    \/
    arrivalTime[i] >= arrivalTime[i-1] + distance[route[i-1], route[i]]
);
```
Constraints (cont.)

We arrive at the node at position $i$ after it opens and after we are done at the node at position $i-1$ plus the travel time:

```
constraint forall(i in Positions where i != 1)(
    arrivalTime[i] >=
    max(open[route[i]],
        arrivalTime[i-1] + distance[route[i-1], route[i]])
);
```
Constraints (cont.)

We arrive at each node before it closes:

constraint forall (i in Positions) (arrivalTime[i] <= close[route[i]]) ;
Minimise the Distance

```plaintext
var int: totalDistance;
constraint totalDistance = sum(i in Positions where i != 7)(
    distance[route[i], route[i+1]]
);
solve minimize totalDistance;
```
A TSPTW Model in MiniZinc

set of int: Nodes = 1..7;
set of int: Positions = 1..7;
array[Nodes, Nodes] of int: distance = ...;
array[Nodes] of int: open = [0, 0, ...];
array[Nodes] of int: close = [200, 40, ...];
A TSPTW Model in MiniZinc

...  
array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;

constraint alldifferent(route);
constraint arrivalTime[1] >= open[route[1]];
constraint forall(i in Positions where i != 1)(arrivalTime[i] >= ...);
constraint forall(i in Positions)(arrivalTime[i] <= ...);
var int: totalDistance;
constraint totalDistance = ...;
solve minimize totalDistance;
The MiniZinc Framework
The MiniZinc Pipeline

A lot of backends and solvers omitted here

MiniZinc

FlatZinc

Picat-SAT
fzn-gecode
fzn-MIP
fzn-chuffed
fzn-oscar-cbls

Lingeling
Gecode
CPLEX
Cbc
Chuffed
OscaR.cbls

SAT
CP
MIP
hybrid
local search

Solver

Backend

Technology

MiniZinc Framework
Model Once, Solve Everywhere

MiniZinc offers a unified modelling language for all technologies.

However, backends are not always robust to model variations.
Local-Search Backends on this Model

If we solve the TSPTW model with local search, then we see:
Local-Search Backends on this Model

If we solve the TSPTW model with local search, then we see:

<table>
<thead>
<tr>
<th>instance</th>
<th>fnz-oscar-cbls</th>
<th>Yuck</th>
<th>LocalSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>n40w120</td>
<td>–</td>
<td>468</td>
<td>–</td>
</tr>
<tr>
<td>n40w140</td>
<td>–</td>
<td>391</td>
<td>–</td>
</tr>
<tr>
<td>n40w160</td>
<td>–</td>
<td>411</td>
<td>–</td>
</tr>
</tbody>
</table>
## Local-Search Backends on This Model

If we solve the TSPTW model with local search, then we see:

<table>
<thead>
<tr>
<th>instance</th>
<th>fzn-oscar-cbls</th>
<th>Yuck</th>
<th>LocalSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>n40w120</td>
<td>–</td>
<td>468</td>
<td>–</td>
</tr>
<tr>
<td>n40w140</td>
<td>–</td>
<td>391</td>
<td>–</td>
</tr>
<tr>
<td>n40w160</td>
<td>–</td>
<td>411</td>
<td>–</td>
</tr>
</tbody>
</table>

not a MiniZinc backend
Local-Search Backends on This Model

If we solve the TSPTW model with local search, then we see:

<table>
<thead>
<tr>
<th>instance</th>
<th>fnz-oscar-cbls</th>
<th>Yuck</th>
<th>LocalSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>n40w120</td>
<td>–</td>
<td>468</td>
<td>–</td>
</tr>
<tr>
<td>n40w140</td>
<td>–</td>
<td>391</td>
<td>–</td>
</tr>
<tr>
<td>n40w160</td>
<td>–</td>
<td>411</td>
<td>–</td>
</tr>
</tbody>
</table>

Local search is usually good for this kind of problem: why not here?!
Constraints (cont.)

We arrive at the node at position $i$ after it opens and after we are done at the node at position $i-1$ plus the travel time:

constraint forall (i in Positions where i != 1)(
    arrivalTime[i] >=
    max(open[route[i]],
        arrivalTime[i-1] + distance[route[i-1], route[i]]
    )
);
constraint forall(i in Positions where i != 1)(
    arrivalTime[i] >=
    max(open[route[i]],
        arrivalTime[i−1] + distance[route[i−1], route[i]]);

or

constraint forall(i in Positions where i != 1)(
    arrivalTime[i] =
    max(open[route[i]],
        arrivalTime[i−1] + distance[route[i−1], route[i]]);
## Comparing the Equality and Inequality Models

<table>
<thead>
<tr>
<th>instance \ model</th>
<th>fzn-oscar-cbls</th>
<th>Yuck</th>
<th>LocalSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>n40w120</td>
<td>ineq</td>
<td>468</td>
<td>ineq</td>
</tr>
<tr>
<td>n40w140</td>
<td>ineq</td>
<td>391</td>
<td>ineq</td>
</tr>
<tr>
<td>n40w160</td>
<td>ineq</td>
<td>411</td>
<td>ineq</td>
</tr>
</tbody>
</table>
# Comparing the Equality and Inequality Models

<table>
<thead>
<tr>
<th>instance \ model</th>
<th>fzn-oscar-cbls</th>
<th>Yuck</th>
<th>LocalSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eq</td>
<td>ineq</td>
<td>eq</td>
</tr>
<tr>
<td>n40w120</td>
<td>+434</td>
<td>-</td>
<td>436</td>
</tr>
<tr>
<td>n40w140</td>
<td>+328</td>
<td>-</td>
<td>+328</td>
</tr>
<tr>
<td>n40w160</td>
<td>352</td>
<td>-</td>
<td>+348</td>
</tr>
</tbody>
</table>
What’s Going on Here?

We significantly improved this model by changing one constraint.

But can this also be done for other models/problems?

What if such a fix cannot be made?

What’s actually going on here?
Local Search
Solving TSP with Local Search

We want to search for a best assignment of variables.

Variables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Random Initial Assignment

We want to search for a best assignment of variables.

Pick a random assignment and evaluate it.
Random Initial Assignment

We want to search for a best assignment of variables.

Pick a random assignment and evaluate it.
Random Initial Assignment

We want to search for a best assignment of variables.

Pick a random assignment and evaluate it

Variables

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Cost: $37 + 2$
Random Initial Assignment

We want to search for a best assignment of variables.

Pick a random assignment and evaluate it

Variables

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G \\
1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
\]

Cost: \(37 + 2 \cdot 1000 = 2037\)
Generate Neighbours

Explore all similar assignments we get upon swapping two values.

Variables

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Cost: $37 + 2 \cdot 1000 = 2037$
Generate Neighbours

Explore all similar assignments we get upon swapping the values of two variables.

Cost: $37 + 2 \cdot 1000 = 2037$
Generate Neighbours

Explore all similar assignments we get upon **swapping** the values of two variables.

Variables

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Cost: \(47 + 1 \cdot 1000 = 1047\)
Generate Neighbours

Explore all similar assignments we get upon swapping the values of two variables.

Cost: $37 + 2 \cdot 1000 = 2037$
Generate Neighbours

Explore all similar assignments we get from swapping the values of two variables.

Cost: $37 + 2 \cdot 1000 = 2037$
Generate Neighbours

Explore all similar assignments we get from **swapping** the values of two variables.

Cost: $64 + 0.1000 = 64$
Generate Neighbours

Explore all similar assignments we get from **swapping** the values of two variables. Try all swaps!

Cost: ?? + ?·1000 = ????
Select a Best Neighbour
Select a Best Neighbour and Move To It
Select a Best Neighbour and Move To It

64
Repeat

64

A  B  C  G  E  F  D
1  2  3  4  5  6  7
Design Aspects

Initialisation and neighbourhood
- How do we make the initial assignment?
- How are the neighbours obtained?

Cost function
- How do we evaluate the quality of an assignment?

(Meta-)Heuristic
- Which neighbour is selected?
- How do we prevent the search from getting stuck?
Design Aspects

Initialisation and neighbourhood
- How do we make the initial assignment? Random
- How are the neighbours obtained? Swap

Cost function
- How do we evaluate the quality of an assignment? $x+y$

(Meta-)Heuristic
- Which neighbour is selected? Best
- How do we prevent the search from getting stuck?
From MiniZinc to Local Search
From This

... array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;

constraint alldifferent(route);
constraint arrivalTime[1] >= open[route[1]];
constraint forall(i in Positions where i != 1)(arrivalTime[i] >= ...);
constraint forall(i in Positions)(arrivalTime[i] <= ...);
var int: totalDistance;
constraint totalDistance = ...;
solve minimize totalDistance;
To This

Initialisation and neighbourhood
- How do we make the initial assignment?
- How are the neighbours obtained?

Cost function
- How do we evaluate the quality of an assignment?

(Meta-)Heuristic
- Which neighbour is selected?
- How do we prevent the search from getting stuck?
How Local-Search Backends to MiniZinc Do It

Determine:

1. which variables to make moves on,
2. which moves to make,
3. how to evaluate the quality of an assignment, and
4. a (meta-)heuristic.
Variables

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;
var int: totalDistance;
Variables

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;
var int: totalDistance;

Some variables are functionally defined by other variables.
Variables

Some variables are functionally defined by other variables.

constraint totalDistance = sum(i in Positions where i != 7)(
    distance[route[i], route[i+1]]
);

Search variables

Functionally defined variables
Variables

The rest are search variables.

- **route[i]**
- **arrivalTime[i]**

- **totalDistance**

  Search variables

  Functionally defined variables
Initialisation and Neighbourhood

We must initialise and make moves on the search variables. Constraints can hint at moves.
Initialisation and Neighbourhood

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;
constraint alldifferent(route);
Initialisation and Neighbourhood

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;
constraint alldifferent(route);

The alldifferent constraint tells us to initialise ‘route’ to different values and do swap moves.

For arrivalTime[i] we can initialise randomly and do assign moves.
Initialisation and Neighbourhood

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;
constraint alldifferent(route);

The alldifferent constraint tells us to initialise ‘route’ to different values and do swap moves.

For arrivalTime we can initialise randomly and do assign moves.

A move is either: swap two values in route
or: change a value in arrivalTime
Cost Function

array[Positions] of var Nodes: route;
array[Positions] of var int: arrivalTime;
constraint alldifferent(route);
constraint arrivalTime[1] >= open[route[1]];
constraint forall(i in Positions where i != 1)(arrivalTime[i] >= ...);
constraint forall(i in Positions)(arrivalTime[i] <= ...);
var int: totalDistance;
constraint totalDistance = ...;
solve minimize totalDistance;
Cost Function

constraint arrivalTime[1] >= open[route[1]];  
constraint forall(i in Positions where i != 1)(arrivalTime[i] >= ...);  
constraint forall(i in Positions)(arrivalTime[i] <= ...);  
var int: totalDistance;  
constraint totalDistance = ...;  
solve minimize totalDistance;

Cost: **objective** + **penalty**
(Meta-)Heuristic

Tabu search
The Problem with Automatically Inferred Neighbourhoods
Neighbourhood and Moves

The *alldifferent* constraint tells us to initialise ‘route’ to different values and do *swap moves*.

For *arrivalTime* we can initialise randomly and do *assign moves*.

But the latter is really bad!
Bad Moves

Let’s do a swap

Move counter: 0
Bad Moves

Let’s do a swap

Move counter: 0
Bad Moves

Let’s do a swap

Move counter: 1

Assume invalid edges have distance 0.
Bad Moves

The \text{arrivalTime}[i]\ are now wrong!

Move counter: 1

Assume invalid edges have distance 0.
Bad Moves

We should now do assign moves...

Move counter: 1

Assume invalid edges have distance 0.
We should now do assign moves...

Move counter: 2

Assume invalid edges have distance 0.
Bad Moves

We should now do assign moves...

Move counter: 3

Assume invalid edges have distance 0.
We should now do assign moves...

Assume invalid edges have distance 0.

Move counter: 6
It takes 6 very coordinated moves to repair all the `arrivalTime[i]` values.

But there is randomness involved in selecting moves, so no hope here!
Bad Moves

We should not search locally for the \( \text{arrivalTime}[i] \) values!

The \( \text{arrivalTime}[i] \) are auxiliary and mainly represent side information.
What About the Equality Model?

```
constraint forall (i in Positions where i != 1)(
    arrivalTime[i] =
    max(open[route[i]],
        arrivalTime[i-1] + distance[route[i-1], route[i]]);

instead of

constraint forall (i in Positions where i != 1)(
    arrivalTime[i] >=
    max(open[route[i]],
        arrivalTime[i-1] + distance[route[i-1], route[i]]);
```
More Functionally Defined Variables

- route[i]
- arrivalTime[i]

Search variables

- totalDistance

Functionally defined variables
More Functionally Defined Variables

In the equality model, the arrivalTime[i] are functionally defined!

- route[i]
- totalDistance
- arrivalTime[i]

Search variables

Functionally defined variables
Another Way to Look at It

In the inequality model, we actually have a third category: auxiliary variables.
Auxiliary Variables

For some models, we cannot do the ineq->eq reformulation in order to eliminate auxiliary variables.

How do we:

1. detect auxiliary variables in a model?
2. avoid searching locally over them?
Generating Compound Moves
Generating Compound Moves

Detect auxiliary variables in a model:
- Add annotation to MiniZinc for specifying search variables.
- Assume variables without a good move are auxiliary.

Avoid searching locally over auxiliary variables:
- Use a constraint programming solver to compute their value after every move on search variables.

We explored many different configurations of this hybridisation.
# Numbers!

<table>
<thead>
<tr>
<th>TSPTW</th>
<th>fzn-oscar-cbls</th>
<th>fzn-oscar-cbls</th>
<th>fzn-oscar-cbls</th>
<th>Yuck</th>
<th>Chuffed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>original</td>
<td>CMG config1</td>
<td>CMG config2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n20w180</td>
<td>377</td>
<td>377&lt;sup&gt;1&lt;/sup&gt;</td>
<td>+253</td>
<td>261</td>
<td>263&lt;sup&gt;10&lt;/sup&gt;</td>
</tr>
<tr>
<td>n20w200</td>
<td>347</td>
<td>373&lt;sup&gt;2&lt;/sup&gt;</td>
<td>+233</td>
<td>233</td>
<td>234&lt;sup&gt;10&lt;/sup&gt;</td>
</tr>
<tr>
<td>n40w120</td>
<td>–</td>
<td>–</td>
<td>+434</td>
<td>437</td>
<td>464&lt;sup&gt;9&lt;/sup&gt;</td>
</tr>
<tr>
<td>n40w140</td>
<td>–</td>
<td>–</td>
<td>+328</td>
<td>367</td>
<td>388&lt;sup&gt;10&lt;/sup&gt;</td>
</tr>
<tr>
<td>n40w160</td>
<td>–</td>
<td>–</td>
<td>+348</td>
<td>362</td>
<td>393&lt;sup&gt;10&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CVRP</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A-n37-k5</td>
<td>2614</td>
<td>2870&lt;sup&gt;9&lt;/sup&gt;</td>
<td>2925</td>
<td>875</td>
<td>983&lt;sup&gt;9&lt;/sup&gt;</td>
</tr>
<tr>
<td>A-n64-k9</td>
<td>5431</td>
<td>5659&lt;sup&gt;4&lt;/sup&gt;</td>
<td>5518</td>
<td>2868</td>
<td>3472&lt;sup&gt;8&lt;/sup&gt;</td>
</tr>
<tr>
<td>B-n45-k5</td>
<td>3638</td>
<td>4121&lt;sup&gt;6&lt;/sup&gt;</td>
<td>4201</td>
<td>972</td>
<td>1182&lt;sup&gt;10&lt;/sup&gt;</td>
</tr>
<tr>
<td>P-n16-k8</td>
<td>489</td>
<td>503&lt;sup&gt;2&lt;/sup&gt;</td>
<td>450</td>
<td>481</td>
<td>481&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
</tbody>
</table>
Any questions?
Thank you for listening!