Optimizing Highly Constrained Truck Loadings

Using a Self-Adaptive Genetic Algorithm

Sander van Rijn Edgar Reehuis
Michael Emmerich Thomas Bäck

LIACS, Leiden University

IEEE CEC 2015
May 26

Introduction

We consider a variation of the Container Loading Problem regarding the delivery of a strongly heterogeneous set of boxes by side-loaded truck to DIY stores:

- 30∼130 boxes per truck
- 1∼25 different customers per trip

We refer to a truck as a container, and a configuration of boxes in a container as a loading

Introduction (cont.)

A human planner takes 10∼15 minutes to make a loading

Existing automated solvers were incompatible with the constraints and objectives

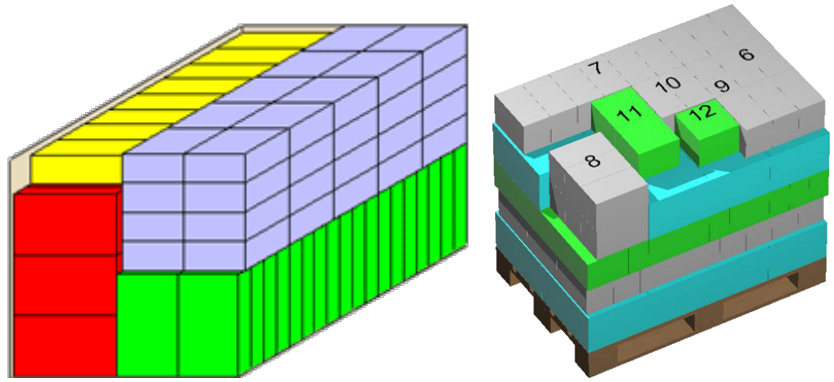


Figure: Examples from other automated solvers

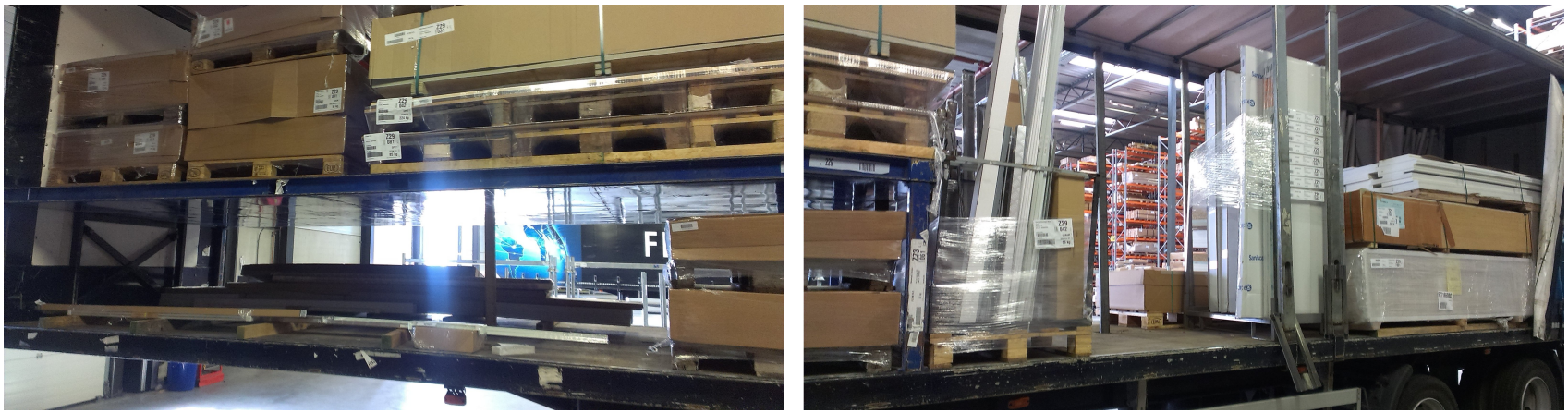


Figure: An actual loading in a truck trailer with a so-called 'bridge'

Constraints and Objectives

There are many criteria that determine the quality of a loading. We distinguish two kinds of criteria:

Constraints and Objectives

There are many criteria that determine the quality of a loading. We distinguish two kinds of criteria:

- Violating a hard constraint makes a loading invalid

Constraints and Objectives

There are many criteria that determine the quality of a loading. We distinguish two kinds of criteria:

- Violating a hard constraint makes a loading invalid
- A loading will receive a penalty in the fitness function for not following an objective

Constraints and Objectives (cont.)

The hard constraints we consider are the following:

- [Two boxes may not occupy the same space]
- Boxes may have limited overhang
- Boxes may only be stacked in stacks
- Boxes may only be stacked according to type-dependent rules

Constraints and Objectives (cont.)

The penalties that are used in determining the fitness:

- Number of boxes left out
- Weight distribution (left/right)
- "Stack pattern" (potential damage)
- Inconvenient client order
- Keeping boxes for one client on the same (preferred) side
- Stack height

Approach

A Monte Carlo search can produce many valid, but likely far from optimal loadings

Due to the large search space and complex evaluation, we choose a Genetic Algorithm with a discrete representation

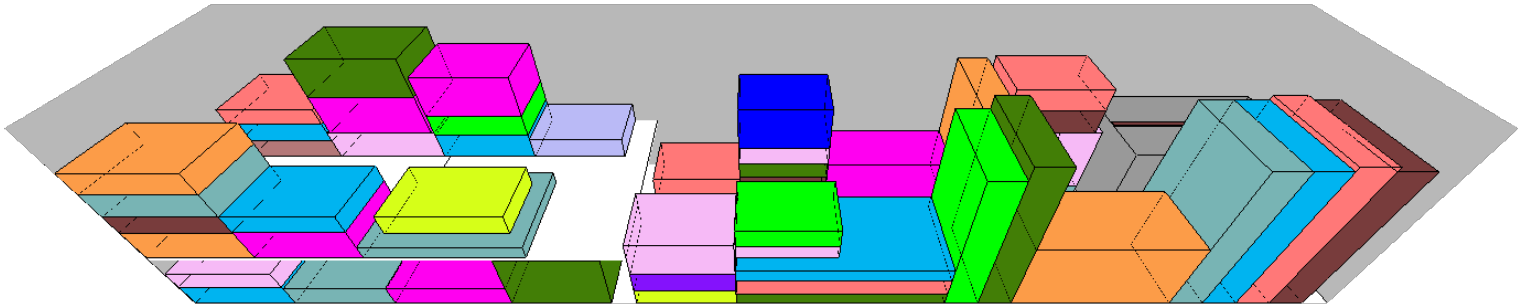


Figure: A randomly generated loading fitting only around 2/3 of all intended boxes for this container

Genetic Algorithm

We apply a GA using only a self-adaptive mutation, according to [Kruisselbrink et al. 2011]

Furthermore, we reset the step size if no improvement occurs after a certain number of generations since the last improvement

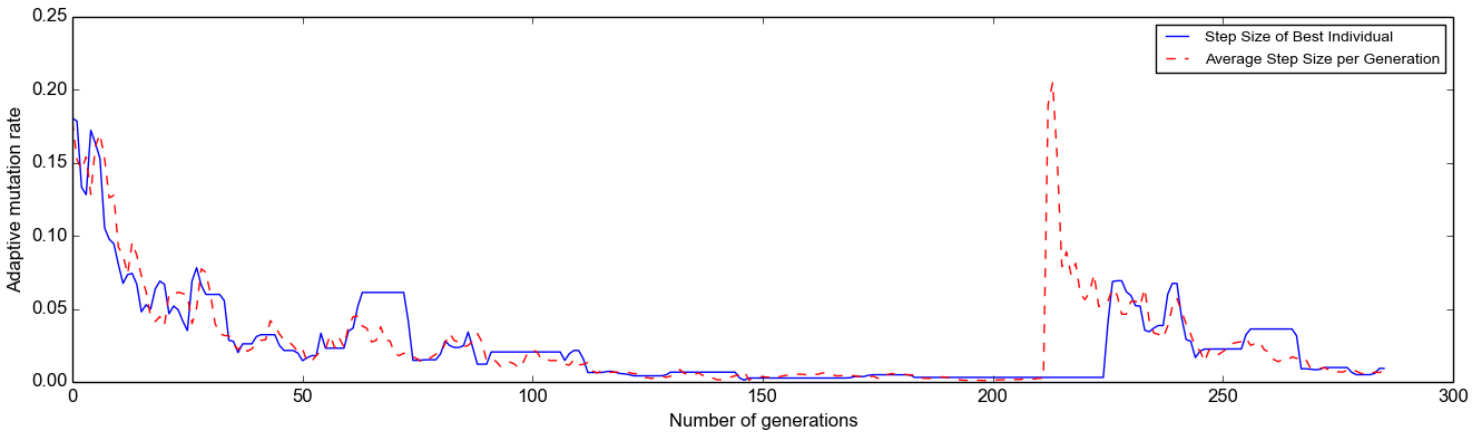


Figure: An example of a step size reset

Representation

A naive representation, only controlling placement order, would result in insufficient freedom to recreate most loadings

Representation

A naive representation, only controlling placement order, would result in insufficient freedom to recreate most loadings

Using 3D-coordinates in millimeter precision would give 6.8×10^{10} possible coordinates for a box: too much freedom, especially since many of these are effectively equivalent anyway!

Representation

A naive representation, only controlling placement order, would result in insufficient freedom to recreate most loadings

Using 3D-coordinates in millimeter precision would give 6.8×10^{10} possible coordinates for a box: too much freedom, especially since many of these are effectively equivalent anyway!

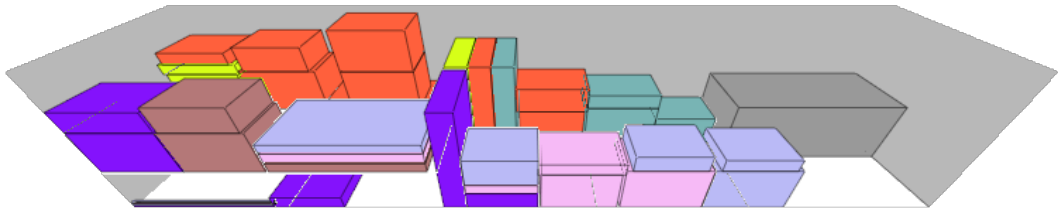
Instead we observe a human planner and derive a representation from their actions:

Add box *b* to the stack *s* in area *a*

Representation (cont.)

Emulate manual placement using triples:
(BoxID, Area, Stack)

```
(40, 1, 1)
(39, 1, 1)
(38, 3, 1)
   ....
( 3, 6, 5)
( 2, 6, 5)
( 1, 6, 6)
```



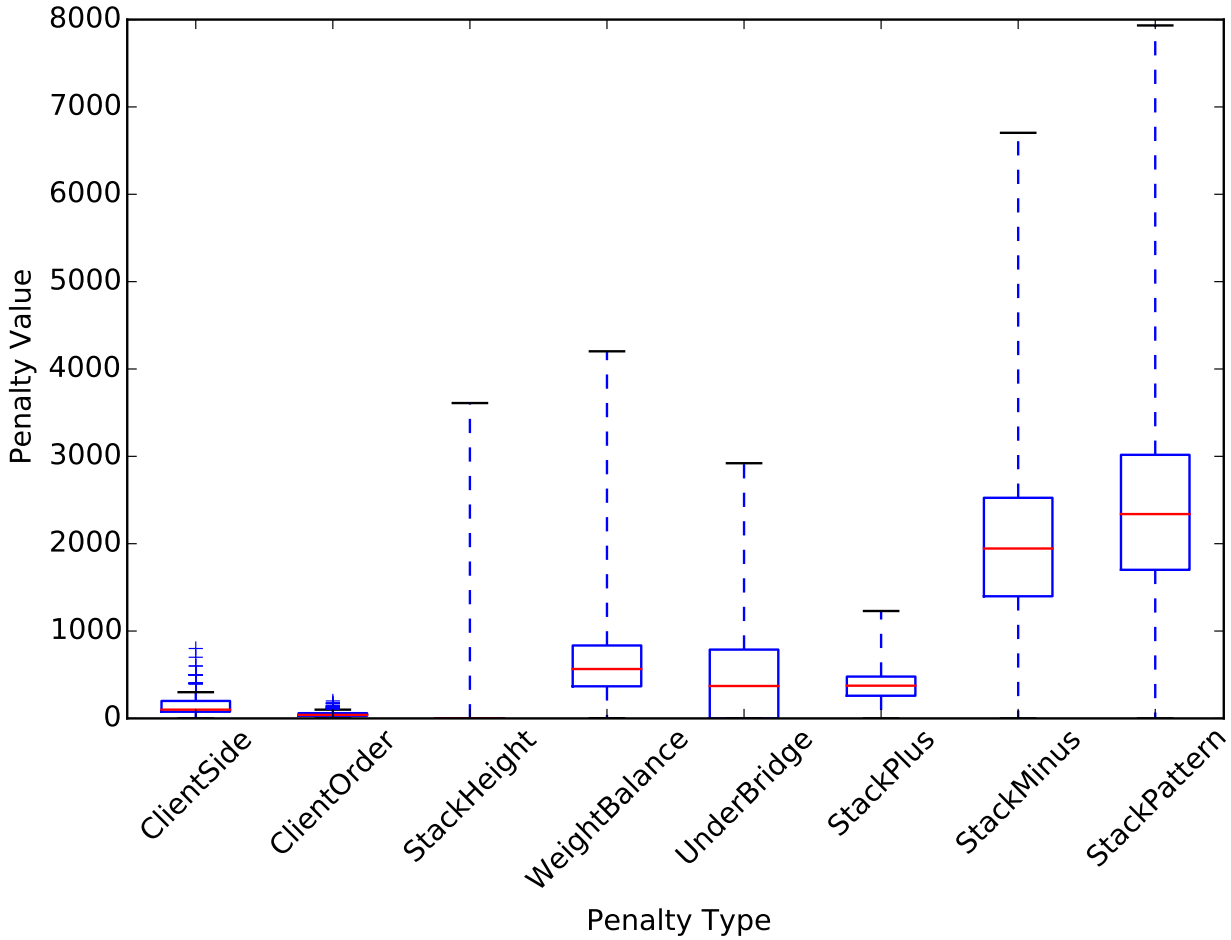
Experiments

Test set:	528 real world cases
Benchmark:	comparison with human score
Evaluation budget:	10,000

Tests:	
Strategy:	static, adaptive
Populations:	(1,10), (5, 35)
Mutations:	swap, insert, 50/50

Results

How are the relative weights of the penalties distributed in loadings?



Results (cont.)

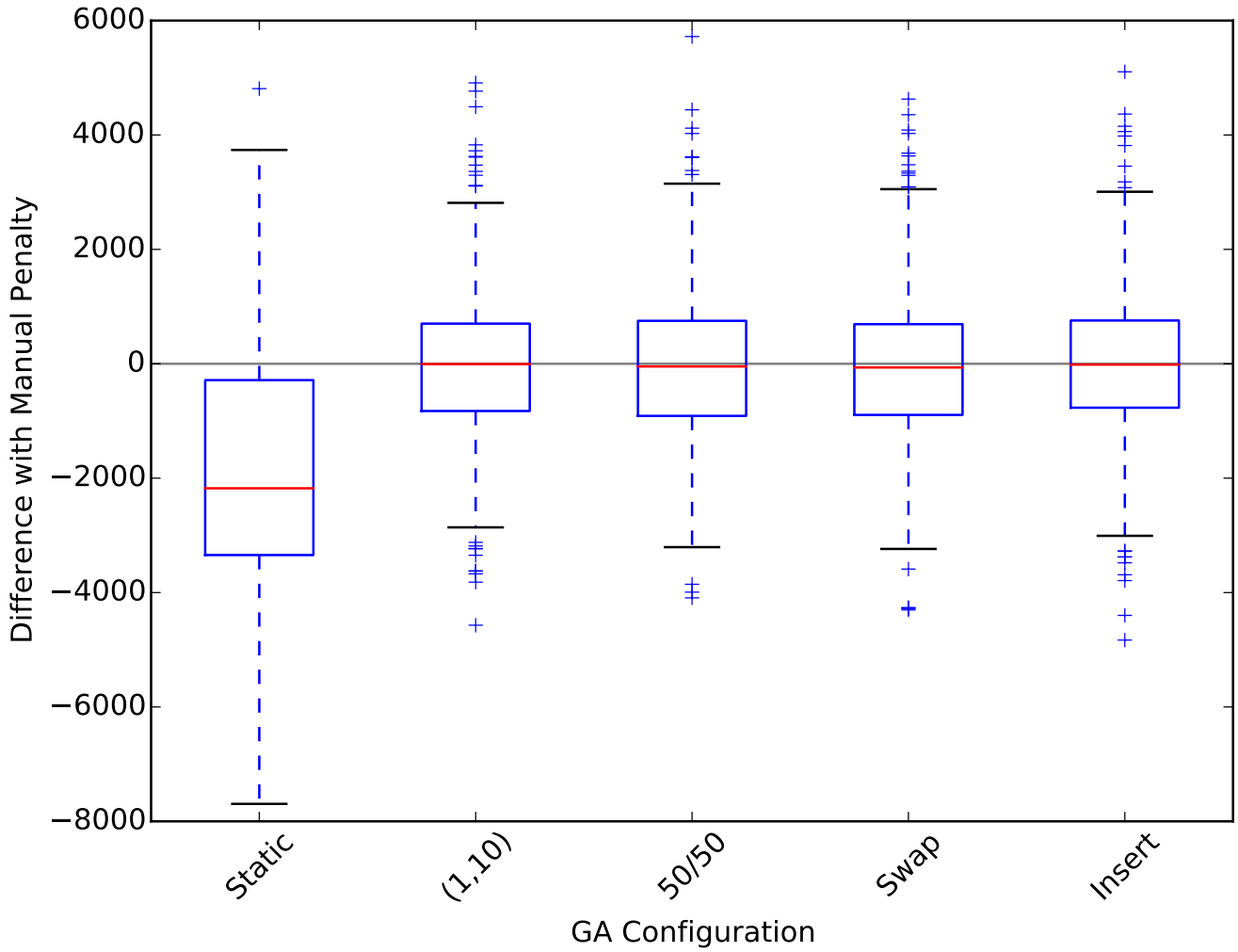
In how many out of the 528 cases is our GA (strictly) better than the human planner?

Penalty	Static	(1,10)	50/50	Insert	Swap
ClientSide	119	203	208	201	219
ClientOrder	122	115	109	95	108
StackHeight	39	42	41	37	41
WeightBalance	231	224	234	226	209
UnderBridge	81	163	165	164	173
StackPlus	146	181	210	173	188
StackMinus	205	248	258	250	263
StackPattern[1]	185	241	259	234	242
Average	141	177	185	172	180

[1]) StackPattern penalty = StackPlus penalty + StackMinus penalty

Results (cont.)

Spread of $penalty_{manual} - penalty_{GA}$ (positive: GA is better)



Results (cont.)

How often did a run stagnate?
How many stagnations occurred per stagnating runs?

Strategy	Stagnating Runs	Stagnations/Run
Static	110	3.0
(1,10)	418	3.5
50/50	308	3.1
Insert	337	3.0
Swap	277	3.0

How many loadings did not fit all boxes?
What percentage of the 32,486 boxes were not placed?

	Static	(1,10)	50/50	Insert	Swap
Incomplete Loadings	247	76	76	80	76
% Boxes Not Placed	1.48	0.27	0.28	0.30	0.27

Results (cont.)

A comparison of typical actual loadings:

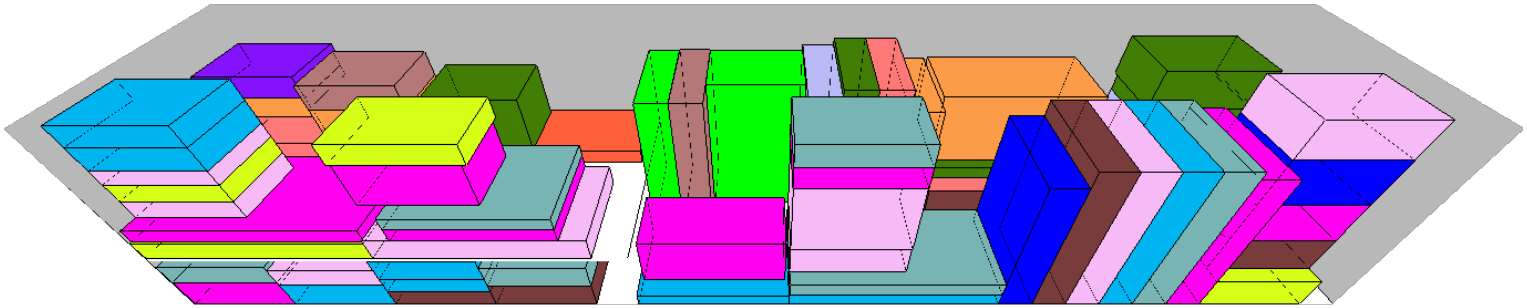


Figure: A loading as generated by the self-adaptive GA

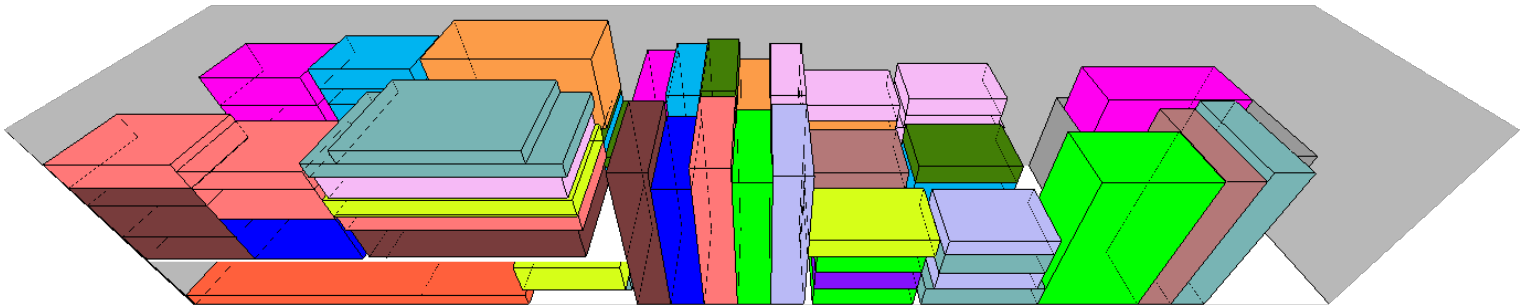


Figure: A loading as created by a human planner

Conclusion

- The devised representation is effective at describing loadings
- A self-adaptive GA clearly outperforms a GA with a static mutation rate
- There is little difference between the performance of different GA configurations that use self-adaptation
- The lack of formally defined objective measures makes it difficult to construct a fitness function that will help the GA produce desired loadings
- Some of the currently GA-generated loadings can be directly used

Future Work

- Improve the fitness function
- Find optimal parameters for penalties
- Improve mutation operator by automated learning from human experience

Thank you for your attention