Plan Merging in Asprilo's m-Domain

Leo Pinetzki Jarek Liesen

University of Potsdam

May 19, 2020

Table of Contents

Table of Contents

What is *asprilo*?

- Framework for intra-logistics & warehouse automation based on Answer Set Programming (ASP)
- Consists of
  - domain specifications (a, b, c and m)
  - plan checker
  - instance generator
  - visualizer
- Goal: Allow for generation of plans, that account for robot movement and fulfillment of orders

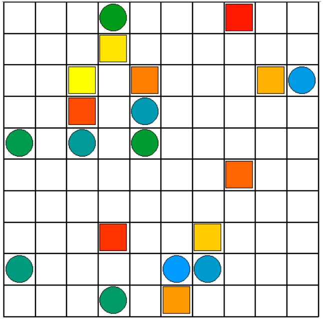


Figure: an asprilo warehouse in its visualizer

asprilo's m-Domain

- The simplest domain, “movement only”

square
: A robot

circle
: A destination for any robot

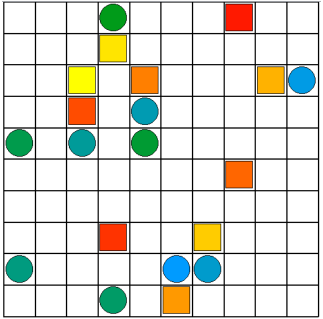
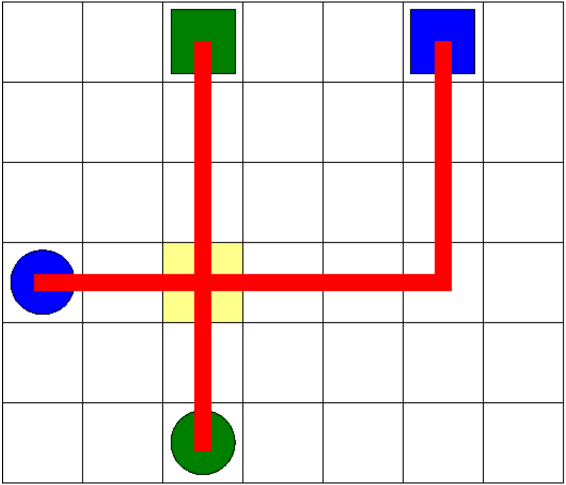


Figure: an asprilo instance of the m-domain

Table of Contents

What is Plan Merging?

Unlike global planning, distributed planning needs plan merging



What is Plan Merging?

Unlike global planning, distributed planning needs plan merging

Advantages of plan merging are:

- Reduced complexity and computation time
- Avoiding re-planning when adding plans
- Plans can be computed in parallel

Important considerations are:

- Conflicts between single plans, such as robot collisions
- Plan has to stay valid
- Optimization of merged plan

Our Approach for Plan Merging

Standard, global approach:

1. **Centralized Plan generation**
   . . . for all robots by allowing arbitrary movements and forbidding collisions and unfulfilled orders

Our distributed approach:

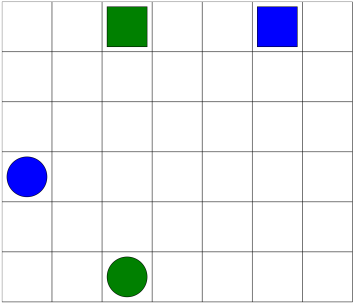
1. **Target assignment**
   . . . optimizing for the smallest single distance between robot and shelf
2. **Plan generation**
   . . . without collision constraints
3. **Plan merging**

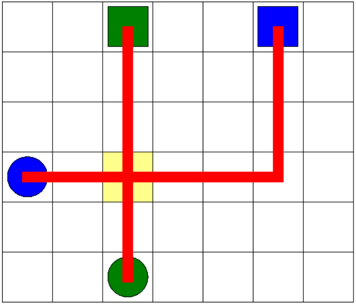
Target Assignment

```
{ assigned(R,S) : isShelf(S) } = 1 :- isRobot(R).

distance(R,S,D) :- position(R,(RX,RY),0),
                   position(S,(SX,SY),0),
                   assigned(R,S),
                   D = |RX - SX| + |RY - SY|.

#minimize { D : distance(R,S,D) }.
```

`assigned(R,S)` : robot `R` is assigned to shelf `S`

- Standard m-encoding but without collision constrains
  - assignes each robot a destination
  - plans moves without avoiding collisions
- Parallel independent planning for all robots at once





`planmov(R,D,T)` : robot `R` plans to take step in direction `D` at time step `t`

`planpos(R,C,T)` : robot `R` plans to be at cell `C` at time step `T`

Plan Merging

- We generally allow for arbitrary movement, but forbid certain moves using different strategies
- Strategies add constraints based on the plans
- Strategies can be freely combined to further reduce the search space

atoms after mergin of all plans:

`move(R,D,T)` : robot `R` takes step in direction `D` at time step `t`

`position(R,C,T)` : robot `R` is at cell `C` at time step `T`

Plan Merging: *basefile.lp*

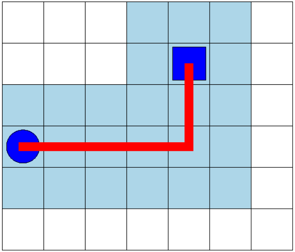
- Merge baseline, and included in every merge call
- Contains move generation and position calculation
- Avoids moves that would lead to crashing of robots
- Generates itself a valid plan

```
{ move(R,D,T) : direction(D) } 1 :- isRobot(R),
        time(T).
:- error(_).

...
```

Table of Contents

Corridor

```
1  surrounds((-s..s,-s..s)).
2  possible(R,(X+DX,Y+DY)):-
       planpos(R,(X,Y),_),
       surrounds((DX,DY)).
3  :- isRobot(R), position(R,C,_),
       not possible(R,C).
```



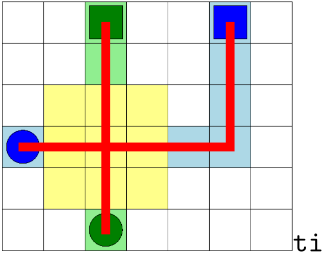
s = 1

time(0)

Idea: Allow each robot only to move to squares close to its planned trajectory

Bottleneck

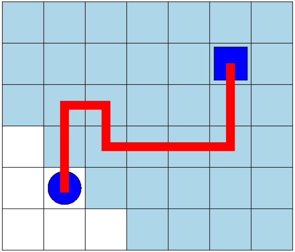
```
1  possible(R,C) :- planpos(R,C,_).
2  possible(R1,(X+DX,Y+DY)) :-
       planpos(R1,(X,Y),_),
       planpos(R2,(X,Y),_),
       R2 != R1, surrounds((DX,DY)).
3  :- isRobot(R), position(R,C,_),
       not possiblePosB(R,C).
```



Idea: Constraint moves even further than corridor, by only allowing to deviate from the plan at conflict cells

Deathlaser

```
1  :- position(R,(X1,Y1),T),
       planpos(R,(X2,Y2),T),
       assigned(R,S),
       position(S,(SX,SY),0),
      D1 = |SX-X1| + |SY-Y1|,
      D2 = |SX-X2| + |SY-Y2|,
      D1 > D2.
```

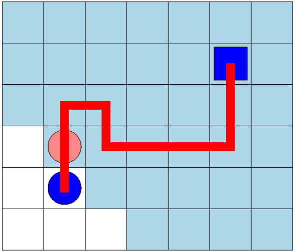


time(0)

Idea: Disallow robot to fall behind its plan in terms of distance to its goal

Deathlaser

```
1  :- position(R,(X1,Y1),T),
       planpos(R,(X2,Y2),T),
       assigned(R,S),
       position(S,(SX,SY),0),
       D1 = |SX-X1| + |SY-Y1|,
       D2 = |SX-X2| + |SY-Y2|,
       D1 > D2.
```

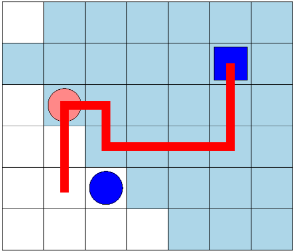


time(1)

Idea: Disallow robot to fall behind its plan in terms of distance to its goal

Deathlaser

```
1 :- position(R,(X1,Y1),T),
      planpos(R,(X2,Y2),T),
      assigned(R,S),
      position(S,(SX,SY),0),
     D1 = |SX-X1| + |SY-Y1|,
     D2 = |SX-X2| + |SY-Y2|,
     D1 > D2.
```

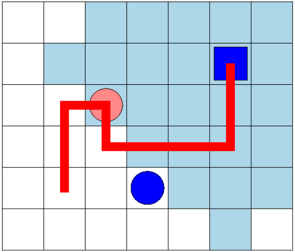


time(2)

Idea: Disallow robot to fall behind its plan in terms of distance to its goal

Deathlaser

```
1  :- position(R,(X1,Y1),T),
       planpos(R,(X2,Y2),T),
       assigned(R,S),
       position(S,(SX,SY),0),
       D1 = |SX-X1| + |SY-Y1|,
       D2 = |SX-X2| + |SY-Y2|,
       D1 > D2.
```

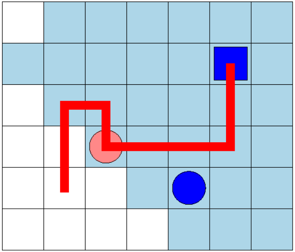


time(3)

Idea: Disallow robot to fall behind its plan in terms of distance to its goal

Deathlaser

```
:- position(R,(X1,Y1),T),
    planpos(R,(X2,Y2),T),
    assigned(R,S),
    position(S,(SX,SY),0),
    D1 = |SX-X1| + |SY-Y1|,
    D2 = |SX-X2| + |SY-Y2|,
    D1 > D2.
```

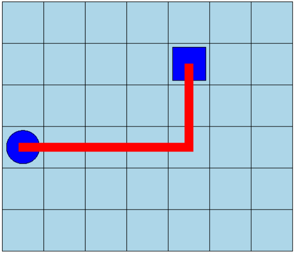


time(4)

Idea: Disallow robot to fall behind its plan in terms of distance to its goal

Homesick

```
1  homesick(T) :- time(T),
       T \ interval = 0.
2  :- homesick(T), isRobot(R),
        position(R,C,T),
        not planpos(R,C,_).
```

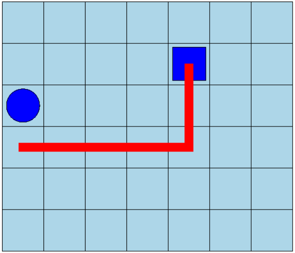


`interval = 3`
`time(0)`

Idea: Make robots converge to their original trajectory every few moves

Homesick

```
1 homesick(T) :- time(T),
      T \ interval = 0.
2 :- homesick(T), isRobot(R),
      position(R,C,T),
      not planpos(R,C,_).
```

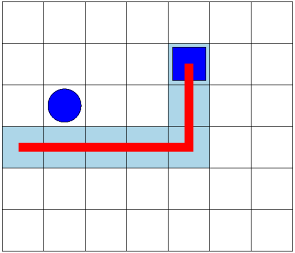


```
interval = 3
time(1)
```

Idea: Make robots converge to their original trajectory every few moves

Homesick

```
homesick(T) :- time(T),
    T \ interval = 0.
:- homesick(T), isRobot(R),
     position(R,C,T),
     not planpos(R,C,_).
```



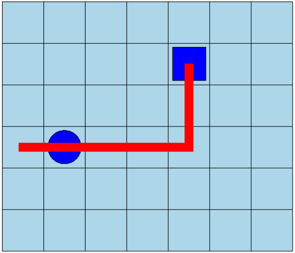
`interval = 3`

`time(2)`

Idea: Make robots converge to their original trajectory every few moves

Homesick

```
homesick(T) :- time(T),
    T \ interval = 0.
:- homesick(T), isRobot(R),
     position(R,C,T),
     not planpos(R,C,_).
```



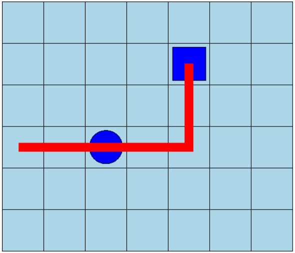
`interval = 3`

`time(3)`

Idea: Make robots converge to their original trajectory every few moves

Homesick

```
homesick(T) :- time(T),
    T \ interval = 0.
:- homesick(T), isRobot(R),
    position(R,C,T),
    not planpos(R,C,_).
```

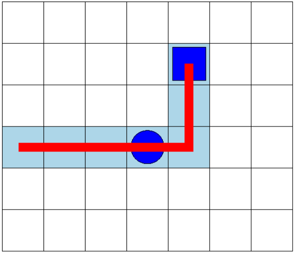


interval = 3
time(4)

Idea: Make robots converge to their original trajectory every few moves

Homesick

```
1  homesick(T) :- time(T),
       T \ interval = 0.
2  :- homesick(T), isRobot(R),
       position(R,C,T),
       not planpos(R,C,_).
```

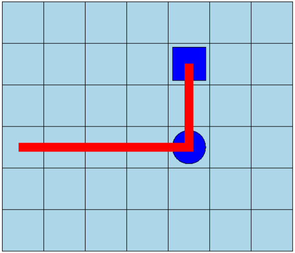


`interval = 3`
`time(5)`

Idea: Make robots converge to their original trajectory every few moves

Homesick

```
homesick(T) :- time(T),
     T \ interval = 0.
:- homesick(T), isRobot(R),
     position(R,C,T),
     not planpos(R,C,_).
```



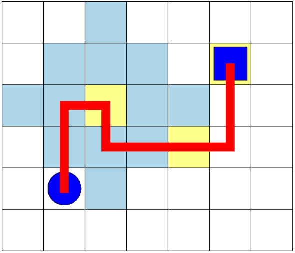
interval = 3

time(6)

Idea: Make robots converge to their original trajectory every few moves

Checkpoints

```
1 :- checkpoint(R,CP,C),
       position(R,CP,T),
       not reached(R,CP,T),
       T = CP*step.
2 :- goalpoint(R,CP,C),
       reached(R,CP,T1),
       move(R,D,T2), T1 > T2.
```

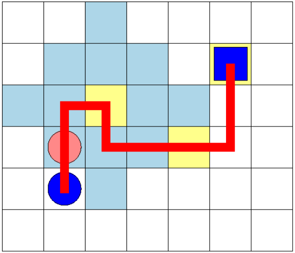


```
interval = 3
time(0)
```

Idea: Force robots to visit their planned cell every few moves

Checkpoints

```
1 :- checkpoint(R,CP,C),
     position(R,CP,T),
     not reached(R,CP,T),
     T = CP*step.
2 :- goalpoint(R,CP,C),
     reached(R,CP,T1),
     move(R,D,T2), T1 > T2.
```

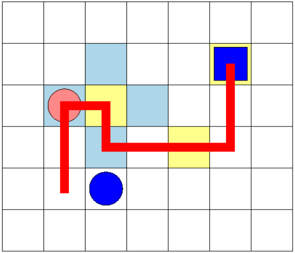


interval = 3
time(1)

Idea: Force robots to visit their planned cell every few moves

Checkpoints

```
1 :- checkpoint(R,CP,C),
      position(R,CP,T),
      not reached(R,CP,T),
      T = CP*step.
2 :- goalpoint(R,CP,C),
      reached(R,CP,T1),
      move(R,D,T2), T1 > T2.
```

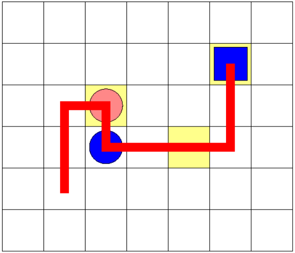


`interval = 3`
`time(2)`

Idea: Force robots to visit their planned cell every few moves

Checkpoints

```
1 :- checkpoint(R,CP,C),
      position(R,CP,T),
      not reached(R,CP,T),
      T = CP*step.
2 :- goalpoint(R,CP,C),
      reached(R,CP,T1),
      move(R,D,T2), T1 > T2.
```



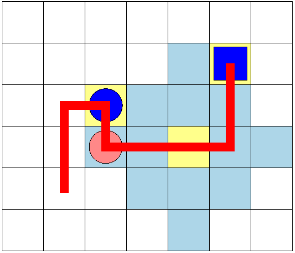
interval = 3

time(3)

Idea: Force robots to visit their planned cell every few moves

Checkpoints

```
1  :- checkpoint(R,CP,C),
      position(R,CP,T),
      not reached(R,CP,T),
      T = CP*step.
2  :- goalpoint(R,CP,C),
      reached(R,CP,T1),
      move(R,D,T2), T1 > T2.
```

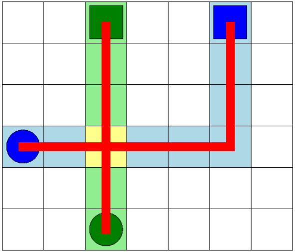


```
interval = 3
time(4)
```

Idea: Force robots to visit their planned cell every few moves

Crossroad

```
1  :- not orderedPlan(R,_,O,D),
      orderedMoves(R,_,O,D).

2  :- not orderedMoves(R,_,O,D),
      orderedPlan(R,_,O,D).

3  :- orderedPlan(R,T,O,_),
      orderedMoves(R,T',O,_),
      |T - T'| > maximumOffset.
```

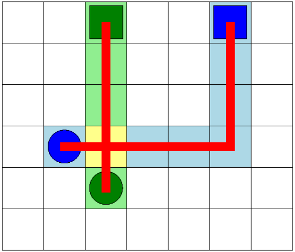


`time(0)`

Idea: Allow to change the timeing of moves but keep their order

Crossroad

```
1  :- not orderedPlan(R,_,O,D),
      orderedMoves(R,_,O,D).

2  :- not orderedMoves(R,_,O,D),
      orderedPlan(R,_,O,D).

3  :- orderedPlan(R,T,O,_),
      orderedMoves(R,T',O,_),
      |T - T'| > maximumOffset.
```

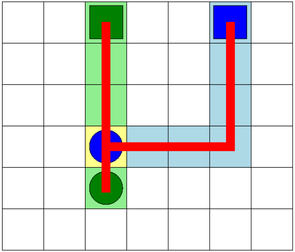


time(1)

Idea: Allow to change the timeing of moves but keep their order

Crossroad

```
1  :- not orderedPlan(R,_,O,D),
      orderedMoves(R,_,O,D).

2  :- not orderedMoves(R,_,O,D),
      orderedPlan(R,_,O,D).

3  :- orderedPlan(R,T,O,_),
      orderedMoves(R,T',O,_),
      |T - T'| > maximumOffset.
```

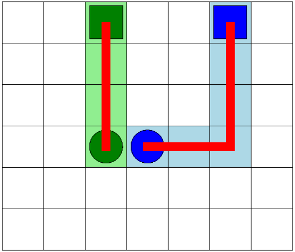


time(2)

Idea: Allow to change the timeing of moves but keep their order

Crossroad

```
1  :- not orderedPlan(R,_,O,D),
      orderedMoves(R,_,O,D).

2  :- not orderedMoves(R,_,O,D),
      orderedPlan(R,_,O,D).

3  :- orderedPlan(R,T,O,_),
      orderedMoves(R,T',O,_),
      |T - T'| > maximumOffset.
```



time(3)

Idea: Allow to change the timeing of moves but keep their order

Table of Contents

Benchmark Setup

- Executed on V-Server with the following specs:
  - 16 VCores Intel Xenon E5-2680-v3
  - 32GB RAM
  - Debian 18.04.4 LTS
  - Python 3.7.7
  - clingo 5.4.0
- Additional strategy "replan" was benchmarked:
  complete replanning after target assignment and plan generation
  $\rightarrow$ Baseline for other strategies
  Only basefile.lp without any strategy

Benchmark Instances

- Three types of quadratic instances according to asprilo's m-domain specifications:

Sparse	10% of edge length as robots
Normal	100% of edge length as robots
Cluttered	10% of number of cells as robots

- 10 instances each for edge lengths of 10, 20, 30, . . . , 100
- For cluttered edge lengths of over 50 were infeasible to ground, so we have generated additional instances between 30 and 50

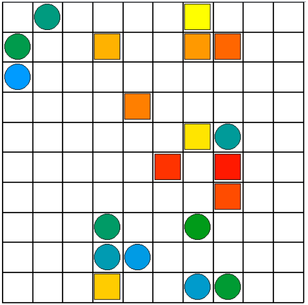


Figure: cluttered 10x10

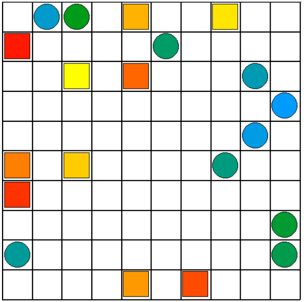


Figure: normal 10x10

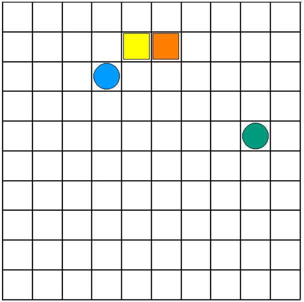


Figure: sparse 10x10

Hypothesis on Single Strategies

Corridor	Bottleneck	Checkpoints
$\sim$ ok performance	$+$ better than Corridor	$\sim$ ok performance

Homesick	Deathlaser	Crossroad
$+$ very good performance	$+$ very good performance	$-$ bad performance

Configuration Clingo 5.4.0 + baseline.lp + Strategy

Hypothesis on Strategy Combinations

	Corridor	Bottleneck	Checkp.	Homesick	Deathlaser
Bottleneck	$-$ Corridor redundant				
Checkp.	$+$	$-$ Checkpoints redundant			
Homesick	$\sim$	$-$ Homesick redundant	$\sim$ Highly depends on hyper-parameters		
Deathlaser	$+$	$+$	$-$ Deathlaser redundant	$+$ $=$ Checkpoints	
Crossroad	$-$ no benefit	$-$ no benefit	$-$ no benefit	$-$ no benefit	$+$

Configuration Clingo 5.4.0 + baseline.lp + Strategies

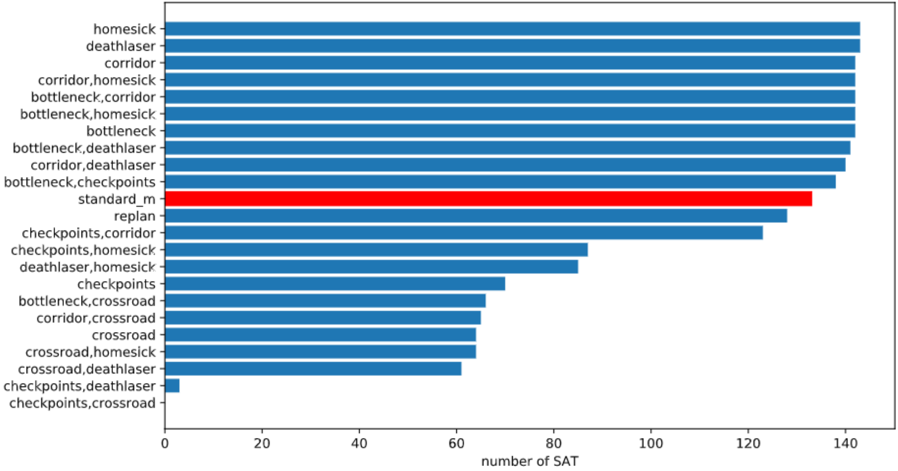
General Benchmark Results

- Total time determined by grounding time of merge step
- Combining strategies did not improve the performance
- Strategies performed very differently on different instance types
- Solving time of the merge is strongly correlated with number of rules and choice rules

General Benchmark Results

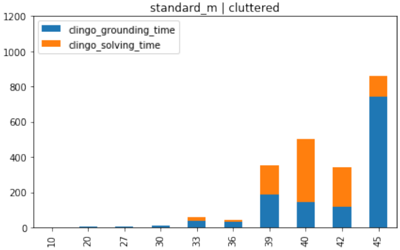
- Checkpoints strategy almost always timed out, combinations with checkpoints performed very poorly
- Combination of checkpoints and crossroad solved no instances
- Crossroad is the worst non-combined strategy
- Standard m-encoding performed the best on cluttered instances

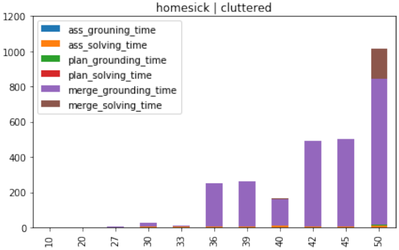
Benchmark Results



timeout 20min

Comparison with Standard m-Encoding





- Best performance on cluttered instances
- By far the lowest grounding times of all strategies on cluttered, comparable on other types
- Generally a much higher solving time than well-performing strategies

Comparison with Hypothesis

Corridor	Bottleneck	Checkpoints
+ good performance	+ worse than Corridor	— bad performance

Homesick	Deathlaser	Crossroad
+ best performance	+ second best performance	— worst performance

Comparison with Hypothesis

	Corridor	Bottleneck	Checkp.	Homesick	Deathlaser
Bottleneck	+ good for normal and sparse				
Checkp.	— limited by implementation of checkpoints	— worse than Bottleneck on ist own			
Homesick	∼ works well on normal and sparse	∼ works well on normal and sparse	∼ performed well on sparse		
Deathlaser	+ good on cluttered	+ very good for normal and cluttered	—	—	
Crossroad	—	—	—	—	—

Brief comparrison of stats

Grouding related stats (means):

strategies	grounding time	merge constraints	merge variables
homesick	124	42301.9	225590.7
standard-m	125	1353323	6227959

Solving related stats (means):

strategies	solving time	merge conflicts
homesick	3.226768	148.2
standard-m	29	559.2105

We have much more detailed information. If one is interested we can show them afterwards!

Conclusion

- Combining strategies is not worth
- No strategy performed well on all instance types
- Using normalized scores, homesick and bottleneck alone performed the best, checkpoints and its combinations performed the worst
- The m-encoding performed surprisingly well due to small grounding times
- Our hypothesis focused on the performance of solving rather then grounding

Future Work

- Optimization of grounding in merge step!
- Implementation of more strategies that ground faster
- Optimization of hyperparameters