

Elevator Project Report

Target-Based Incremental Approach

Atreya Shankar, Luis Glaser

`{shankar,glaser}@uni-potsdam.de`

BM3: Advanced Problem Solving Techniques (ASP)

University of Potsdam, WiSe 2018/2019

Prof. Dr. Torsten Schaub

April 8. 2019

Table of Contents

- 1 Introduction
- 2 Target-Assignment Methodologies (Static)
- 3 Translating Targets to Actions (Iterative)
- 4 Optimizations
- 5 Results

Introduction

```

1 % Initialize and create targets
2 move[] := []
3 move[] := []
4 holds(0,0) :- init(A).
5 targets(elevator(X),Y) :- holds(at(elevator(X),Y),0), holds(request(deliver(X),Y),0).
6 [targets(elevator(X),Y) : holds(at(elevator(X),Y),0) :- holds(request(call(X),Y),0), not holds(request(
7 deliver(X),Y),0).
8 % constraints to ensure all tasks are targetted, and no double tasking
9 :- targets(elevator(X),Y), targets(elevator(X'),Y'), X' != X, holds(request(call(X'),Y'),0), not holds(request(
10 deliver(X'),Y'),0).
11 :- holds(request(call(X),Y),0), not targets(X,Y).
12 % calculate number of tasks per elevator
13 number(elevator(X),N) :- N = #sum[Y:targets(elevator(X),Y)], holds(at(elevator(X),Y),0).
14 % distribution parameter for elevator vs. requests, given multiple targets
15 param(elevator(X),M,N) :- holds(at(elevator(X),Y),0), N = #sum[Y-Y:holds(at(elevator(X),Y),0), targets(ele
16 vator(X),Y)], N = #min[Y-Y:holds(at(elevator(X),Y),0), targets(elevator(X),Y)], L > 1, number(elevator
17 (X),L).
18 % action if only one target exists
19 target1(elevator(X),Y) :- targets(elevator(X),Y), N = 1, number(elevator(X),N).
20 target2(elevator(X),Y) :- target1(elevator(X),Y), N = 1, number(elevator(X),N).
21 % action if multiple targets exist, elevator inside target span
22 target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q), N = #min([M]:[K]), param(ele
23 vator(X),M,K), N%>=0, Q = Y+M, not target2(elevator(X),Q), L > 1, number(elevator(X),L), #false:target
24 1(elevator(X),Q), Q<0.
25 target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q), N = #min([M]:[K]), param(ele
26 vator(X),M,K), N%>=0, Q = Y-M, not target2(elevator(X),Q), L > 1, number(elevator(X),L), #false:target
27 1(elevator(X),Q), Q<0.
28 target2(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q), N = #max([M]:[K]), param(ele
29 vator(X),M,K), N%>=0, Q = Y+M, target1(elevator(X),Q'), Q' != Q, L > 1, number(elevator(X),L), #false
30 targets(elevator(X),Q'), Q'>0.
31 target2(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q), N = #max([M]:[K]), param(ele
32 vator(X),M,K), N%>=0, Q = Y-M, target1(elevator(X),Q'), Q' != Q, L > 1, number(elevator(X),L), #false
33 :targets(elevator(X),Q'), Q'<0.
34 % action if multiple targets exist, elevator outside target span
35 target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q), N = #min([M]:[K]), param(ele
36 vator(X),M,K), N%>=0, Q = Y+M, L > 1, number(elevator(X),L).
37 target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q), N = #min([M]:[K]), param(ele
38 vator(X),M,K), N%>=0, Q = Y-M, L > 1, number(elevator(X),L).

```

Fig. 1. Snippet of encoding in elevator.lp

- Encoding utilizes target-based incremental approach
- Approach deemed faster over a pure combinatorial approach
- Encoding succeeds on all Yeti cases

Target Assignment

```
% initialize and create targets
```

```
targets(elevator(X),Y')  
:- holds(at(elevator(X),Y),0), holds(request(deliver(X),Y'),0).
```

```
{targets(elevator(X),Y'): holds(at(elevator(X),_),0)}  
:- holds(request(call(_),Y'),0),  
not holds(request(deliver(_),Y'),0).
```

```
% ensure no double targets and all requests targetted
```

```
:- targets(elevator(X),Y), targets(elevator(X'),Y), X' != X,  
holds(request(call(_),Y),0), not holds(request(deliver(_),Y),0).
```

```
:- holds(request(_),Y),0), not targets(_),Y).
```

Distribution Parameter Generation

```
% calculate number of tasks per elevator
```

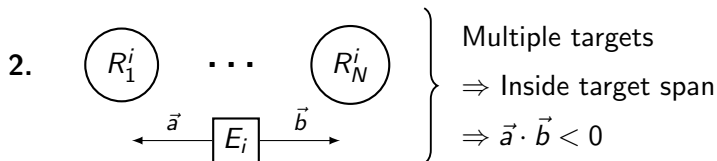
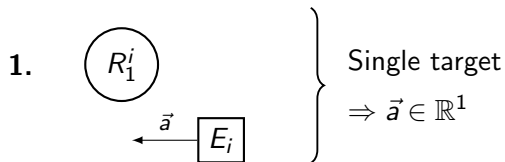
```
number(elevator(X),N) :- N = #sum{1,Y:targets(elevator(X),Y)},  
holds(at(elevator(X),_),0).
```

```
% distribution parameter given multiple targets
```

```
param(elevator(X),M,N) :- holds(at(elevator(X),_),0),  
M = #max{Y-Y':holds(at(elevator(X),Y),0),  
targets(elevator(X),Y')},  
N = #min{Y-Y':holds(at(elevator(X),Y),0),  
targets(elevator(X),Y')}, L > 1, number(elevator(X),L).
```

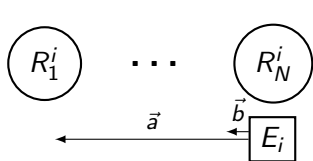
- Parameters are useful in deciphering position of elevator wrt. corresponding targets

Target Distribution Cases I



Target Distribution Cases II

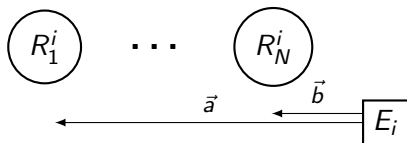
3.



Multiple targets

 \Rightarrow On target span $\Rightarrow \vec{a} \cdot \vec{b} = 0$

4.



Multiple targets

 \Rightarrow Outside target span $\Rightarrow \vec{a} \cdot \vec{b} > 0$

End-Targets Assignment I

Case 1 (Single Target):

```
target1(elevator(X),Y) :- targets(elevator(X),Y), N = 1,  
number(elevator(X),N).
```

```
target2(elevator(X),Y) :- target1(elevator(X),Y), N = 1,  
number(elevator(X),N).
```

- End-Targets are defined to be the same for consistency
- Necessary measure to keep end-targets framework

End-Targets Assignment II

Case 2/3 (Inside/On Span):

```
target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #min{|M|;|K|}, param(elevator(X),M,K), M*K <= 0, Q = Y+N,
not target2(elevator(X), Q), L > 1, number(elevator(X),L),
#false:targets(elevator(X),Q'), Q'>Q.
```

```
target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #min{|M|;|K|}, param(elevator(X),M,K), M*K <= 0, Q = Y-N,
not target2(elevator(X), Q), L > 1, number(elevator(X),L),
#false:targets(elevator(X),Q'), Q'<Q.
```

```
target2(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #max{|M|;|K|}, param(elevator(X),M,K), M*K <= 0, Q = Y+N,
target1(elevator(X), Q''), Q''!= Q, L > 1, number(elevator(X),L),
#false:targets(elevator(X),Q'), Q'>Q.
```

```
target2(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #max{|M|;|K|}, param(elevator(X),M,K), M*K <= 0, Q = Y-N,
target1(elevator(X), Q''), Q''!= Q, L > 1, number(elevator(X),L),
#false:targets(elevator(X),Q'), Q'<Q.
```

End-Targets Assignment III

Case 4 (Outside Span):

```
target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #min{|M|;|K|}, param(elevator(X),M,K), M*K > 0, Q = Y+N, L > 1,
number(elevator(X),L).
```

```
target1(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #min{|M|;|K|}, param(elevator(X),M,K), M*K > 0, Q = Y-N, L > 1,
number(elevator(X),L).
```

```
target2(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #max{|M|;|K|}, param(elevator(X),M,K), M*K > 0, Q = Y+N,
target1(elevator(X), Q''), Q'' != Q, L > 1, number(elevator(X),L),
#false:targets(elevator(X),Q'), Q'>Q.
```

```
target2(elevator(X),Q) :- holds(at(elevator(X),Y),0), targets(elevator(X),Q),
N = #max{|M|;|K|}, param(elevator(X),M,K), M*K > 0, Q = Y-N,
target1(elevator(X), Q''), Q'' != Q, L > 1, number(elevator(X),L),
#false:targets(elevator(X),Q'), Q'<Q.
```

Move Calculation

```
% calculate per-elevator moves
```

```
moves(elevator(X),N) :- N = #sum{|Y-Y'|;|Y'-Y''|},  
holds(at(elevator(X),Y),0), target1(elevator(X),Y'),  
target2(elevator(X),Y'')).
```

```
% calculate cumulative moves
```

```
allmoves(N) :- N = #sum{M:moves(elevator(_),M)}.
```

- Deterministic (pre-iterative) approach to quantify moves
- Useful for optimization/minimization purposes

Target Actions: Moving

% moving to target1

```
do(elevator(X),move(V),t) :- holds(at(elevator(X),Y),t-1),
target1(elevator(X),Y'), holds(request(deliver(X),Y'),t), move(V),
|Y+V-Y'|<|Y-Y'|, not do(elevator(X),serve,t).
```

```
do(elevator(X),move(V),t) :- holds(at(elevator(X),Y),t-1),
target1(elevator(X),Y'), holds(request(call(_),Y'),t), move(V),
|Y+V-Y'|<|Y-Y'|, not do(elevator(X),serve,t).
```

% moving to target2

```
do(elevator(X),move(V),t) :- holds(at(elevator(X),Y),t-1),
target2(elevator(X),Y'), holds(request(deliver(X),Y'),t),
target1(elevator(X),Y''), move(V), |Y+V-Y'|<|Y-Y'|,
not holds(request(deliver(X),Y''),t), not holds(request(call(_),Y''),t),
not do(elevator(X),serve,t).
```

```
do(elevator(X),move(V),t) :- holds(at(elevator(X),Y),t-1),
target2(elevator(X),Y'), holds(request(call(_),Y'),t),
target1(elevator(X),Y''), move(V), |Y+V-Y'|<|Y-Y'|,
not holds(request(deliver(X),Y''),t), not holds(request(call(_),Y''),t),
not do(elevator(X),serve,t).
```

Target Actions: Serving

```
% serving delivery request
```

```
do(elevator(X),serve,t) :- holds(request(deliver(X),Y),t-1),  
holds(at(elevator(X),Y),t-1), targets(elevator(X),Y).
```

```
% serving call request
```

```
do(elevator(X),serve,t) :- holds(request(call(_),Y),t-1),  
holds(at(elevator(X),Y),t-1), targets(elevator(X),Y).
```

```
% serving floor
```

```
serving(Y,t) :- do(elevator(X),serve,t),  
holds(at(elevator(X),Y),t-1).
```

Target Actions: Transfer Positions/Requests

```
% transfer elevator positions
```

```
holds(at(elevator(X),Y+V),t) :- holds(at(elevator(X),Y),t-1),  
do(elevator(X),move(V),t).
```

```
holds(at(elevator(X),Y),t) :- holds(at(elevator(X),Y),t-1),  
do(elevator(X),serve,t).
```

```
holds(at(elevator(X),Y),t) :- holds(at(elevator(X),Y),t-1),  
not do(elevator(X),_,t).
```

```
% transfer requests
```

```
holds(request(call(V),Y),t) :- holds(request(call(V),Y),t-1),  
not serving(Y,t).
```

```
holds(request(deliver(X),Y),t) :- holds(request(deliver(X),Y),t-1),  
not holds(at(elevator(X),Y),t-1).
```

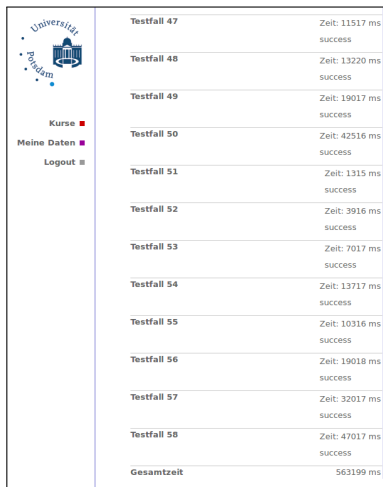
Optimizations

```
% minimize deterministic cumulative moves
```

```
#minimize{N:allmoves(N)}.
```

- Pre-iterative calculation of `allmoves(N)` makes minimization more efficient
- Reduction of target search-space in pre-solving phase
- Overall target assignment methodology more efficient than pure combinatorical approach

Results



Testfall 47	Zeit: 11517 ms	success
Testfall 48	Zeit: 13220 ms	success
Testfall 49	Zeit: 19017 ms	success
Testfall 50	Zeit: 42516 ms	success
Testfall 51	Zeit: 1315 ms	success
Testfall 52	Zeit: 3916 ms	success
Testfall 53	Zeit: 7017 ms	success
Testfall 54	Zeit: 13717 ms	success
Testfall 55	Zeit: 10316 ms	success
Testfall 56	Zeit: 19018 ms	success
Testfall 57	Zeit: 32017 ms	success
Testfall 58	Zeit: 47017 ms	success
Gesamtzeit	563199 ms	

- Encoding succeeds on all 58 Yeti test-cases
- Total runtime $\approx 560s$
- Longest runtime on instance 31 $\approx 85s$
- This was due to a large search space, resulting in 2048 optimal solutions

Fig. 2. Snippet of Yeti performance for encoding