University of Potsdam

Final Thesis for a Bachelor's Degree in Computational Science

Map Abstraction for Multi-Agent Pathfinding problems with Answer Set Programming

Adrian Salewsky

supervised by

Prof Torsten Schaub

Arvid Becker

01.04.2022

Abstract.

1 Introduction

2 Asprilo

3 Abstraction Methods

I have developed several abstraction methods. All of them use some of the asprilo encoding as a base. Since the input instances look the same, I use a reduced version of the *input.lp* file to transform it into predicates, that are easier to understand. It is a reduced version because I do not care about orders, products or packing stations. The only relevant things are robots, shelves and nodes.
The goal condition is the same as in the modified asprilo encoding.
The output predicates are *new_init* and *imp_position*. The *new_init* predicates describe the new instance with a reduced node count. The form of *new_init* resembles that of the *init* predicate in the input instance. There are *new_init* predicates for robots, shelves and nodes. The ones for robots and shelves are the same as in the input instance (with of course a "new" being added) since the start and goal conditions are not changed during the abstraction. The predicates for the nodes contain only the nodes that still exist after the abstraction. The *imp_position*(R, C) predicate states that robot R is on cell C at some point while building the abstraction. This can be used as extra information when solving the instance with the use of this abstraction. The output is the same for every method except for the "Reachable Nodes" method. In that method the output is slightly different which I will explain in the respective subsection.
The first part of the abstraction building is always solving the instance without caring for conflicts. Therefore the encoding takes parts of the original asprilo encoding again. The difference is that the constraint rules which prohibit conflicts are ignored. Furthermore there are two lines extra:

```
:- move(R,_,T1), not move(R,_,T2), time(T2), isRobot(robot(R)), T2<T1.
#minimize {1,(R,T): move(R,_,T)}.
```

The first line states that a robot must always move until he is finished moving completely. This makes it so that a robot does not randomly wait instead of directly going to his goal.
The second line is used to minimize the amount of moves each robot does. This is used to ensure that the solver looks for the shortest paths of the robots.
In the following subsections, I will explain the idea behind each abstraction method and show the encoding that is only used in that method respectively.

3.1 Shortest Path

The first method is to look for the shortest path for each robot to its respective goal without considering possible conflicts as was explained in the section above. Each position the robot visits is then used to create the *imp_position* predicates.

```
imp_position(R,C) :- position(R,C,_), isRobot(robot(R)).
```

In this method, it is also possible to add the time step at which the robot visits the position as an argument to the *imp_position* predicate. However, this would mean that a solver that uses the abstraction to solve the instance would need to specifically be designed for the “Shortest Path” method. Always using the same structure for each method means that one solver can be created to work for all methods. It is also not possible to look at how many arguments the predicate has because the “Reachable Nodes” method needs a third argument for something different, which I will explain in the respective subsection. If it is not of interest to have a solver that solves by using the different abstractions the same way, the line shown above can easily be modified to contain the information about the time step. The output would then have to be modified as well.
Of the methods explained in this paper, this is the one that results in the smallest abstraction. It could also be viewed as a special case of the other methods. If there is a shortest path in an instance, this method will find it so it can be used for all solvable MAPF problems.

3.2 Node Combining

3.3 Reachable Nodes

4 Benchmarking

5 Conclusion

References

A Affidavit