Merging Plans of Robots in the asprilo Framework

Adrian Salewsky

University of Potsdam

Abstract. The asprilo framework was created to make plans for robots in a warehouse. It uses the language Answer Set Programming which is a good choice for such a problem because this language was made for solving NP-hard problems. The solution works well for a small number of robots but at some point the computation time gets too high. This is why the idea came up to create plans for a small number of robots and then combine the plans to get a working solution. This paper shows my approach on how to solve the plan merging problem.

1 Introduction

The asprilo framework is used to make plans for robots in a warehouse, i.e. when does a robot pick up or shelf or when does a robot move which way. It uses the language Answer Set Programming (ASP). Since there is no order in which ASP computes things, the predicates which describe the actions of robots and the positions of objects contain an argument T which declares at which time step the predicate holds. This project was done in the M-domain of the asprilo framework. In the M-domain, the goal is that every shelf has a robot on the same node at the last time step (the last time step is called horizon). This domain simplifies the framework to Multi-Agent Path Finding (MAPF). The framework works well for a small number of robots but for a high number of robots, the computation time gets too high. This is why the plan came up to combine plans with a small number of robots to a plan with all the robots. During the combining of these plans, all the conflicts that arise between robots must be solved. This paper shows my approach on how to solve this plan merging problem in ASP. In the following section, I will show my general idea on how to solve conflicts. In the section after that, I will show my ASP encoding and explain its functions. Section 4 contains the comparison of my approach with that of other groups that have been working on the same project. I will end my paper with a conclusion.

2 Idea

The main idea behind my solution is that every time a conflict arises, one of the robots participating in the conflict randomly changes his movement during that time step and pushes his original plan back. To do that, the program needs to

detect the conflicts and choose a robot who has to solve the conflict by generating a new move. The problem that arises is that there is no option to overwrite predicates in ASP. Therefore, a new argument was needed for every predicate describing a plan. This new argument is called $conflict\_nr$ and the higher this argument is in a predicate the newer is the plan containing that predicate. This idea is a very simple approach to solving the problem since it completely adapts the original plan and only inserts additional moves. This way the resulting solution might not have an optimal move set for every robot but it should be able to theoretically solve any problem instance.

3 Answer Set Programming Encoding

3.1 Initiating Predicates

- conflict depth
- horizon
- initiating position ...

3.2 Conflict Detection and Selection

- show how its done
- talk about minimization

3.3 Conflict Solving

- show possibilities
- show how plan gets changed

3.4 Integrity Constraints

- show which ones
- talk about efficiency

3.5 Output

- show new_move
- show new_occurs

3.6 Extra Features

There are two extra features I have dealt with. The first extra feature is the possibility of declaring that certain robots do not have to change their plan at all. This is done by adding the predicate $strict\_plan(R\_ID)$ with R_ID being the ID of a robot. The only thing that gets changed in the encoding is in the conflict detection part:

```
conflict(R1,T,A) :- position(R1,C1,T-1,A), position(R1,C2,T,A), position(R2,C2,T,A),
                    R1!=R2, conflict_nr(A), not strict_plan(R1).
```

The only things that gets added is that a robot can only have a conflict that he might have to solve if he does not have the predicate *strict_plan*. It is the same for the second type of conflict. If two robots have a conflict and they both have strict plans there is no solution for that particular problem. This is encoded by using additional integrity constraints:

```
:- position(R1,C1,T-1,A), position(R1,C2,T,A), position(R2,C2,T,A), R1!=R2,
   conflict_nr(A), strict_plan(R1), strict_plan(R2).
```

The other conflict type looks the same. As this is a very hard constraint and can very easily lead to unsatisfiable problems, I have decided to expend on the idea to let the user control which robot gets chosen as the one who has to solve the conflict. This brings me to my second feature: Introducing a priority system. The user can add predicates $priority(R\_ID, P)$ where R_ID is the ID of a robot again and P is the assigned priority of that robot. The solver then always chooses the robot with the lower priority to solve a conflict. If two robots have the same priority, either can be chosen.

```
conflict(R1,T,A) :- position(R1,C1,T-1,A), position(R1,C2,T,A), position(R2,C2,T,A),
                    R1!=R2, conflict_nr(A), p_priority(R1,P1),
                    p_priority(R2,P2), P1<=P2.
p_priority(R,P) :- priority(R,P).
p_priority(R,0) :- not priority(R,_), isRobot(robot(R)).
```

The second conflict looks like the first one. The reason I change *priority* to *p_priority* is that this way I can introduce a default value. If the user does not choose any priority value for a robot, the priority gets set to 0.
Both features change how the conflict solving robot is chosen. Using them can be useful in certain scenarios, especially the second one, but it should be noted that those features can easily make an instance unsatisfiable.

4 Comparison with other approaches

- advantages and disadvantages of my approach compared to other approaches based on the benchmarking

5 Conclusion

This paper explained my approach on how to solve the plan merging problem for the asprilo framework. The solution is able to solve almost all of the benchmarks with the exception being a very niche benchmark that is unlikely to appear in the real world. There are two main problems in my solution. The first is that it is very time inefficient. The second is that the user has to in put the time

horizon and the maximum conflict number. If he does not know it, he will likely take an upper bound which increases the computation time. While those two problems are definitely a major deficit of my solution, the trade-off is that it is generally usable and it finds a solution with the minimum amount of conflicts needed to solve a problem instance. To avoid the time inefficiency, the user can also take a parallel approach, i.e. combining the plans of 2 robots at a time, then combining these plans again and so on. This way, the conflict number of the problem instances is likely lower than when combining all plans at once. This also means that the computation time is reduced. Another possible usage of this solution is adding new robots to a warehouse where a plan for the other robots already exists. Since only a small number of robots is likely added, the conflict number is probably low enough to get a solution in an acceptable time.
My solution to solving the plan merging problem has a lot of possibilities of getting improved, like time efficiency and taking it to the A-domain of the framework. Nevertheless, it is an approach that solves the problem and could be used in certain scenarios.

References