Plan merging within the asprilo framework

Tarek Ramadan[1]

University of Potsdam, 14469 Potsdam, Germany
tramadan@uni-potsdam.de

Abstract. The Multi-Agent Path Finding problem is a multi-faceted problem that shows up in many areas of real life. In this project, the problem was addressed using the asprilo framework in a warehouse-like environment. Plans from different agents, robots in this case, were to be merged collision-free. In this report, the focus was on deviating as little as possible from the original plans. The project was carried out by various groups under the supervision of Etienne Tignon.

1 Introduction

This report is about a project on the Multi-Agent Path Finding problem, using the asprilo [1] framework provided by potassco [2]. Both Multi-Agent Path Finding and asprilo are briefly introduced in section 2. In section 3, my own approach is described and analyzed. This is followed by a comparison to the mergers of the other groups using benchmarks that we agreed on in advance. The report ends with a conclusion and a short outlook.

2 Multi-Agent Path Finding and the asprilo framework

Multi-Agent Path Finding (MAPF) describes many different problems in which collision-free paths of multiple agents have to be found from their starting positions to given goal positions. In this project, we worked on the MAPF problem with robots in a warehouse-like environment. The asprilo framework provided a platform to create instances, plan paths and visualize both. Instances consist of nodes that can be populated with robots, shelves and picking stations (see Fig. 1). Furthermore, shelves can be filled with products and orders can be placed. Asprilo offers different domains that simulate different levels of a warehouse. We worked in the M-domain ("Movement-Only Domain"), although there are groups that have designed their merger to work on higher domains and are able to move shelves to picking stations. In the given problem plans of several robots had to be merged collision-free. The paths of the individual robots are predefined and were generally planned without the knowledge of other robots within the instance.

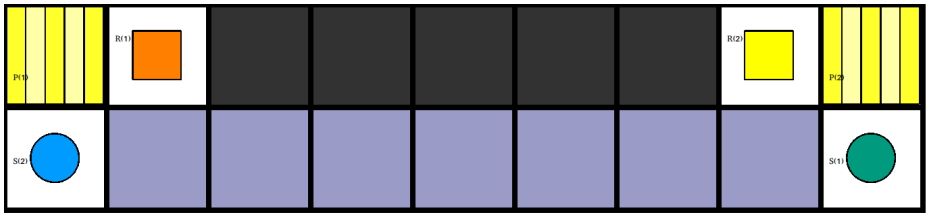

Fig. 1. R(x) identifies the respective robot, S(x) the shelf and P(x) the picking station.

2.1 Representation of Instances

In asprilo instances are displayed as nodes on which robots, shelves and picking stations can be placed. An active node is represented by the predicate `init(object(node,N), value(at,(X,Y)))` where `N` is the node number and `(X,Y)` the node coordinates. Robots, shelves and picking stations are presented in a similar way.

```
init(object(robot,R), value(at,(X,Y))).
init(object(shelf,S), value(at,(X,Y))).
init(object(pickingStation,P), value(at,(X,Y))).
```

Products and orders follow a similar principle, although they do not carry coordinates but information about the location of the product or the order process. The `A` stands for the product ID, the `S` for the shelf ID, the `O` for the order ID and the `a` for the number of (ordered) products.

```
init(object(product,A),value(on,(S,a))).
init(object(order,O),value(line,(A,a))).
```

2.2 Representation of Plans

Plans associated with an instance are represented with the occurs/3 predicate. It signals which robot `R` has executed the movement `D` at time `T`.

```
occurs(object(robot,R),action(move,D),T).
```

3 Plan merging approach

3.1 Strategy

Over the period of the project, my strategy has evolved, with the main goal always remaining the same. And that is to re-plan as little as possible while keeping as much of the original paths as possible.

With my first approach, I tried to categorize collisions and force the robots into certain patterns so that collisions could always be solved in the same way.

The goal here was to take the paths before and after the collisions without any changes and only reschedule the part in between. This approach unfortunately did not prove effective, as the leap to instances with more than two robots seemed nearly impossible with the overhead of classifying collisions.

The second approach avoids categorization and aims to fill the collision gap with randomly generated movement. These generated moves are then minimized to obtain an optimal result. In this report, I will focus on the random movement approach, since the merger of the first approach is only able to solve benchmarks with two robots.

3.2 Random movement approach

The merger works in a simple way. First, collisions are detected and from this the time step `T` of the first collision is extracted. The plans of all robots up to this time are taken over, since they are collision-free.

If the goal of a robot is less than four time steps away, all steps of the robot are assigned with random movements. If the goal is temporally far from the collision, movements are generated depending on a constant `rm`, which must ensure that the paths of the robots do not overlap after the collision, because the original plans of the respective robots are taken over from the time step `T+1` after the first collision.

Collision detection In general, there are two different types of collisions that can occur in the given environment. Robots can try to occupy the same node at the same time (Line 1) and they can try to swap places (Line 2).

Collisions are detected with the following encoding. This is done by comparing the imported positions of the robots. Note that duplicate detection of collisions is avoided with `not collision((X,Y),T,robot(R'),robot(R))` and `not collision(_,T,robot(R'),robot(R))` respectively.

```
1 collision((X,Y),T,robot(R),robot(R')) :-
        position_im(robot(R),(X,Y),T),
        position_im(robot(R'),(X,Y),T),
        R!=R', not collision((X,Y),T,robot(R'),robot(R)).
2 collision((X,Y),T,robot(R),robot(R')) :-
        position_im(robot(R),(X,Y),T),
        position_im(robot(R'),(X',Y'),T),
        position_im(robot(R),(X',Y'),T+1),
        position_im(robot(R'),(X,Y),T+1),
        R!=R', not collision(_,T,robot(R'),robot(R)).
```

Time management The merger adjusts the time dynamically. It does not need any input, because the time is calculated based on the first collision and the maximum time steps of the robots (Line 3 and 4). The maximum time steps

of the individual robots are acquired in Line 6 and 7. This plays a significant role in the Checkpointing section of the merger.

Line 5 describes the already mentioned time steps in which movements can be generated. These are dependent on the time of the first collision and the constant rm, which must be given to the merger.

```
3 time(1..T+1) :- first_collision(T).
4 time(T+1..(R+S)) :-
        first_collision(T), max_range(R,_), steps(S,_).
5 random_time((T')..(T'+P)) :- first_collision(T'), P=rm.

6 1{max_range(T,robot(R)):
    position_im(robot(R),_,T), position_im(robot(R),_,T'), T>=T'}1
        :- position_im(robot(R),_,_).
7 :- max_range(T,robot(R)), position_im(robot(R),_,T),
    position_im(robot(R),_,T'), T<T'.
```

Checkpointing In this section checkpoints are set for each robot to reach. These depend on how far the first collision is from the maximum time step of a robot.

If the goal is close, the checkpoint is set at the end position (Line 8).

If the goal is far, the checkpoint is set to the position the robot wanted to reach one time step after the first collision (Line 9).

```
8 checkpoint(C,robot(R),2) :-
        position_im(robot(R),C,T'), first_collision(T),
        max_range(T',robot(R)), T'-T<4.
9 checkpoint(C,robot(R),1) :-
        position_im(robot(R),C,T+1), first_collision(T),
        max_range(T',robot(R)), T'-T>=4.
```

Movement generation and minimization Movement is generated within the given time frame (Line 10). From the movements, positions are determined using a modification of the original asprilo encoding [3]. These positions are in turn used to determine the number of steps per robot needed to reach the specified checkpoint (Line 11). The steps are then minimized with the same priority for each robot (Line 12).

```
10 { random_move(robot(R),D,T) : direction(D) } :-
        random_time(T), isRobot(robot(R)),
        first_collision(T'), T>=T'.
11 steps(S,robot(R)) :-
        random_position(robot(R),C,T), checkpoint(C,robot(R),_),
        first_collision(T'), S=T-(T'-1).
12 #minimize {S: steps(S,_) }.
```

Plan adopting and replaning If the plans of the individual robots can be merged without collision, they are adopted without changes (Line 13).

If there are collisions, the plans of the robots are adopted up to the time of the first collision (Line 14). Line 15 describes the application of the generated movement as a dependence of the minimized steps `S`. Depending on the checkpoint, the rest of the original plan is then adopted if necessary (Line 16).

```
13 move(robot(R),D,T) :-
        move_im(robot(R),D,T), not collision(_,_,_,_).
14 move(robot(R),D,T) :-
        move_im(robot(R),D,T), first_collision(T'), T<T'.

15 move(robot(R),D,T) :-
        random_move(robot(R),D,T), steps(S,robot(R)),
        first_collision(T'), T<=(T'-1)+S.
16 move(robot(R),D,T+S-2) :-
        move_im(robot(R),D,T), steps(S,robot(R)),
        first_collision(T'), checkpoint(_,robot(R),1), T>T'+1.
```

Original encoding This part of the original encoding was used, although in an edited form, to translate the generated movements of the robots into positions and to check them for correctness. It proved efficient to do so.

```
% - move/3 ------------------------------------------------------
 position(R,C,T) :-
        move(R,D,T), position(R,C',T-1),     nextto(C',D,C).
 :- move(R,D,T), position(R,C ,T-1), not nextto(C ,D,_).

% - inertia -----------------------------------------------------
 position(R,C,T) :-
        position(R,C,T-1), not move(R,_,T), isRobot(R), time(T).

% - edge collision ----------------------------------------------
 moveto(C',C,T) :-
        nextto(C',D,C), position(R,C',T-1), move(R,D,T).
 :- moveto(C',C,T), moveto(C,C',T), C < C'.

% - vertex collision --------------------------------------------
 :- { position(R,C,T) : isRobot(R) }  > 1, position(C), time(T).

% - auxiliaries -------------------------------------------------
 :- { position(R,C,T) } != 1, isRobot(R), time(T).
```

4 Comparison to other Groups

In total, five different groups [4–8] participated in this project. We agreed on 20 benchmarks (four from each group) to be tested by each group with each

merger. The benchmarks were named uniformly and provide information about the group from which they originate. `benchmark_3_2` for example is benchmark 2 from group 3. Note that `benchmark_3_4` is only listed in the table for those groups that were also able to solve the benchmark, since the benchmark is not solvable in the M-Domain.

The group order is as follows.

```
Group 1: T. Ramadan
Group 2: J. Bruns, T. Schmidt, H. Weichelt
Group 3: M. Funke, M. Wiedenhöft
Group 4: N. Kämmer, M. Wawerek
Group 5: A. Salewsky
```

Benchmark results Although the benchmarks were conducted very thoroughly, it should be noted that each group can handle their merger the best. It is therefore not out of question that a merger performs worse than possible in my benchmark results. I encourage you to get a better picture by visiting the git repositories of the other groups [5–8].

In the following, the terminology used is elaborated. "Unsatifiable" describes the case where the merger fails to find an answer and subsequently stops searching for a solution. A search is "killed" when the solver grounds too much, whereupon the search is aborted.

The benchmark results are presented below.

Table 1. Benchmark results of group 1.

Benchmark	Robots	Horizon	Constant	Time
benchmark_1_1	3	5	4	0.023s
benchmark_1_2	2	19	16	0.233s
benchmark_1_3	3	7	6	0.085s
benchmark_1_4	2	15	8	0.034s
benchmark_2_1	4	5	4	0.025s
benchmark_2_2	3	4	3	0.019s
benchmark_2_3	50	-	20	killed
benchmark_2_4	30	-	20	killed
benchmark_3_1	2	5	2	0.013s
benchmark_3_2	2	4	3	0.012s
benchmark_3_3	4	6	4	0.020s
benchmark_4_1	4	17	7	0.079s
benchmark_4_2	8	11	5	0.115s
benchmark_4_3	5	4	2	0.200s
benchmark_4_4	6	11	3	1.686s
benchmark_5_1	2	5	4	0.012s
benchmark_5_2	4	-	10	Unsatisfiable
benchmark_5_3	2	14	9	0.026s
benchmark_5_4	8	-	8	Unsatisfiable

Table 2. Benchmark results of group 2.

Benchmark	Robots	Horizon	Time
benchmark_1_1	3	5	Unsatisfiable
benchmark_1_2	2	19	Unsatisfiable
benchmark_1_3	3	9	0.076s
benchmark_1_4	2	15	0.268s
benchmark_2_1	4	7	0.0750s
benchmark_2_2	3	4	0.0161s
benchmark_2_3	50	29	116.088s
benchmark_2_4	30	62	180.634s
benchmark_3_1	2	7	Unsatisfiable
benchmark_3_2	2	4	0.0162s
benchmark_3_3	4	8	0.0933s
benchmark_4_1	4	11	0.075s
benchmark_4_2	8	13	1.530s
benchmark_4_3	5	10	0.110s
benchmark_4_4	6	21	2.018s
benchmark_5_1	2	5	0.022s
benchmark_5_2	4	3	0.015s
benchmark_5_3	2	6	0.024s
benchmark_5_4	8	10	0.217s

Table 3. Benchmark results of group 3.

Benchmark	Robots	Horizon	Time
benchmark_1_1	3	5	Unsatisfiable
benchmark_1_2	2	19	Unsatisfiable
benchmark_1_3	3	10	11.847s
benchmark_1_4	2	16	3.420s
benchmark_2_1	4	10	Unsatisfiable
benchmark_2_2	3	6	1.318s
benchmark_2_3	50	40	killed
benchmark_2_4	30	100	killed
benchmark_3_1	2	7	0.344s
benchmark_3_2	2	4	0.058s
benchmark_3_3	4	9	2.892s
benchmark_3_4	2	15	1.976s
benchmark_4_1	4	17	178.564s
benchmark_4_2	8	-	killed
benchmark_4_3	5	10	460.838s
benchmark_4_4	6	-	killed
benchmark_5_1	2	5	0.195s
benchmark_5_2	4	3	Unsatisfiable
benchmark_5_3	2	14	1.273s
benchmark_5_4	8	9	killed

Table 4. Benchmark results of group 4.

Benchmark	Robots	Horizon	Time
benchmark_1_1	3	7	0.019s
benchmark_1_2	2	19	0.256s
benchmark_1_3	3	9	0.027s
benchmark_1_4	2	16	0.048s
benchmark_2_1	4	10	Unsatisfiable
benchmark_2_2	3	10	0.013s
benchmark_2_3	50	40	killed
benchmark_2_4	30	100	killed
benchmark_3_1	2	7	0.017s
benchmark_3_2	2	4	0.006s
benchmark_3_3	4	9	0.024s
benchmark_3_4	2	15	0.024s
benchmark_4_1	4	15	0.305s
benchmark_4_2	8	15	0.424s
benchmark_4_3	5	15	0.525s
benchmark_4_4	6	25	0.327s
benchmark_5_1	2	5	0.006s
benchmark_5_2	4	3	0.0075s
benchmark_5_3	2	14	0.034s
benchmark_5_4	8	9	0.068s

Table 5. Benchmark results of group 5.

Benchmark	Robots	Horizon	Constant	Time
benchmark_1_1	3	5	8	Unsatisfiable
benchmark_1_2	2	19	5	Unsatisfiable
benchmark_1_3	3	9	2	0.137s
benchmark_1_4	2	15	4	0.470s
benchmark_2_1	4	7	5	4.417s
benchmark_2_2	3	4	1	0.033s
benchmark_2_3	50	40	10	killed
benchmark_2_4	30	100	10	killed
benchmark_3_1	2	7	1	0.011s
benchmark_3_2	2	4	1	0.007s
benchmark_3_3	4	8	5	2.615s
benchmark_4_1	4	-	14	Unsatisfiable
benchmark_4_2	8	15	6	303.478s
benchmark_4_3	5	10	1	2.550s
benchmark_4_4	6	23	1	367.966s
benchmark_5_1	2	5	1	0.012s
benchmark_5_2	4	3	3	0.059s
benchmark_5_3	2	14	9	107.974s
benchmark_5_4	8	9	3	22.383s

Evaluation Each merger has problem areas. Especially `benchmark_2_3` and `benchmark_2_4` have caused problems for all mergers except the one of group 2. The merger of group 4 can solve almost all benchmarks in a very short time and performs best in the overall performance. In general, the mergers of groups 1, 2 and 4 perform well on benchmarks that can be solved by the respective merger. The mergers of groups 3 and 4 show weaknesses in some benchmarks, but are able to solve most benchmarks.

One thing is very clear from the benchmark tests. It is very easy to get too focused on certain instances in a project of this type and disregard other perspectives. This is evident in the discrepancy between some of the performances of almost all of the mergers.

5 Conclusion

The strategy developed as part of the project work shows clear strengths, although very little time was invested in optimization. It performed decently in the benchmark test with very competitive times in some benchmarks. The merger has theoretical and practical weaknesses, as not all benchmarks could be solved and there are many instances where the strategy is suboptimal. However, these can be solved by further optimization, such as enhanced movement generation in critical instances.

References

1. asprilo Homepage, https://www.asprilo.github.io/. Last accessed 23 Mar 2021
2. Potassco Homepage, https://www.potassco.org/. Last accessed 23 Mar 2021
3. asprilo Encoding, https://www.github.com/potassco/asprilo-encodings. Last accessed 23 Mar 2021
4. Git repository of T. Ramadan, https://www.github.com/tramadan-up/asprilo-project. Last accessed 23 Mar 2021
5. Git repository of J. Bruns, T. Schmidt, H. Weichelt, https://www.github.com/tzschmidt/PlanMerger. Last accessed 23 Mar 2021
6. Git repository of M. Funke, M. Wiedenhöft, https://www.github.com/Zard0c/ProjektMAPF. Last accessed 23 Mar 2021
7. Git repository of N. Kämmer, M. Wawerek, https://www.github.com/NikKaem/mapf-project. Last accessed 23 Mar 2021
8. Git repository of A. Salewsky, https://www.github.com/salewsky/Plan-Merging. Last accessed 23 Mar 2021