BACHELOR'S THESIS

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF "BACHELOR OF SCIENCE (B.SC.) INFORMATIK(COMPUTER SCIENCE)/COMPUTATIONAL SCIENCE" TO THE FACULTY OF KNOWLEDGE PROCESSING AND INFORMATION SYSTEMS AT THE UNIVERSITY OF POTSDAM

SOLVING ABSTRACTIONS OF MULTI-AGENT PATHFINDING PROBLEMS WITH ANSWER SET PROGRAMMING

TAREK RAMADAN

Supervised by Professor Torsten Schaub, Klaus Strauch

Contents

Abstract. Multi-Agent Pathfinding Probleme sind eine besondere Art der Multi-Agent Planning Probleme. Multi-Agent Pathfinding stellt ein wichtiges Problem im bereich der Künstlichen Intelligenz dar. In dieser Bachelorarbeit werden Abstraktionen von Multi-Agent Pathfinding Problemen mit verschiedenen Methoden gelöst. Dazu wird das Answer Set Programming System clingo zusammen mit dem Benchmarking Framework asprilo benutzt. Hinzu kommen diverse Python-Skripte, die das Arbeiten mit clingo, asprilo und den Abstraktionen vereinfachen. Diese Arbeit baut auf der Arbeit von Adrian Salewsky auf, der sich in seiner Bachelorarbeit mit den Abstraktionen selbst beschäftigt hat. Zum Vergleich der entwickelten Methoden in dieser Arbeit wird das von asprilo bereitgestellte Encoding benutzt.

1 Introduction

In this thesis, the Answer Set Programming system clingo was used to solve the Multi-Agent Pathfinding Problem on abstractions of instances that were provided by the work of Adrian Salewsky. Multi-Agent Pathfinding is a very important class of problems in Artificial Intelligence. Applications of it can be found in robotics (automated warehouses), gaming (movement of non-player characters) and logistics/transportation. Multi-Agent Pathfinding is prevalent where the movement and the coordination of multiple agents in a shared environment is involved. Clingo and asprilo, which are both developed at the University of Potsdam, were used to represent, solve and visualize abstractions of warehouse-like instances of the Multi-Agent Pathfinding problem. This thesis expands on Adrian Salewsky's thesis "Map Abstraction for Multi-Agent Pathfinding problems with Answer Set Programming" where he used clingo and asprilo to find abstractions of instances. The purpose of an abstraction in this context is to narrow the scope of instances and serves to limit possible movement of agents. The goal of this thesis is to find techniques for solving the Multi-Agent Pathfinding problem while using the information given by the abstraction. This section introduces Answer Set Programming, Multi-Agent Pathfinding, clingo and asprilo. It is followed by a short summary of Adrian's work in Section 2. The main part of the thesis, solutions to those abstractions, is expanded upon in Section 3. Asprilo provides its own solver that is used to compare results of this thesis' solutions in Section 4. Lastly, this thesis ends with a short reflection in Section 5. All programs used in this thesis can be found at the GitHub repository to this thesis [1]. Adrian's work can be found at his GitHub repository [2].

1.1 Answer Set Programming and clingo

Answer Set Programming (ASP) is a form of logical programming that follows the declarative programming paradigm. In ASP problems don't need to be specified to be solved, the declaration of the problem using logical rules and constraints is sufficient. The goal is to find a solution that is consistent with the given set of rules and constraints. The solution is represented as an answer set, which is a set of logical atoms.
Clingo is an ASP System developed at the University of Potsdam. It combines the grounder gringo and the solver clasp to ground and solve logic programs [3].
The following code shows a simple clingo program. This program describes a problem with 3 chairs and 3 persons, where each person has to be seated without two sitting on the same chair. In lines 1-4 the atoms $chair(1).$ $chair(2).$ $chair(3).$ and $person(john).$ $person(liz).$ $person(lucy).$ are declared. Line 5 specifies that the program chooses a chair for each person. The constraints in lines 6 and 7 demand that every person has to be seated and that two persons can't be seated on the same chair respectively. Line 7 can be read as: "It can't be the case that $P1$ is seated on C, $P2$ is seated on C and that $P1$ and $P2$ are not the same person". Lines 8 and 9 specify the output of the program.

```
1 chair(1..3).
2 person(john).
3 person(liz).
4 person(lucy).
5 {seated(P, C) : chair(C)} :- person(P).
6 :- person(P), not seated(P, _).
7 :- seated(P1,C), seated(P2,C), P1!=P2.
8 #show.
9 #show seated/2.
```

1.2 Multi-Agent Pathfinding and asprilo

Multi-Agent Pathfinding (MAPF) is the problem of finding collision-free solutions for a set of agents given their respective start and end positions. There are different versions of a MAPF problem, which depend on specifications and limitations. For example, in this thesis, the focus is on non-anonymous MAPF problems. Unlike anonymous MAPF problems, non-anonymous MAPF problems define a fix goal position for each agent. There are also different types of conflicts, like the "following conflict" [4]. Only edge conflicts (agents traverse the same edge at

the same time), vertex conflicts (agents occupy the same vertex at the same time) and swapping conflicts (agents swap positions) are considered conflicts in this thesis.
To visualize MAPF problems, the benchmark framework asprilo [5] is used. Asprilo is very sophisticated and includes a various amount of domains. In this thesis, the focus lies on the M-domain. Information about picking stations, products and orders, which are used by the other domains to simulate a more realistic warehouse scenario, are disregarded and the focus on the movement of the agents is emphasized. Using asprilo, a simplified MAPF Problem can be described as a grid of nxm nodes, k robots and k shelves. Since this thesis heavily relies asprilo relies on its encoding, in the following agents are referred to as robots. The example in Figure 1 shows a $3x3$ grid, two robots (squares) and two shelves (circles). A solution for this instance would include a collision-free path for each robot to its respective shelf. A snippet of the representation of the example in Figure 1 is given in the following code. Each object (node, robot, shelf) has an ID and a position at (X, Y). Since the focus is on non-anonymous MAPF, it is assumed that the ID of a robot and its shelf align. A more thorough explanation of the asprilo encoding can be found in Section 3.

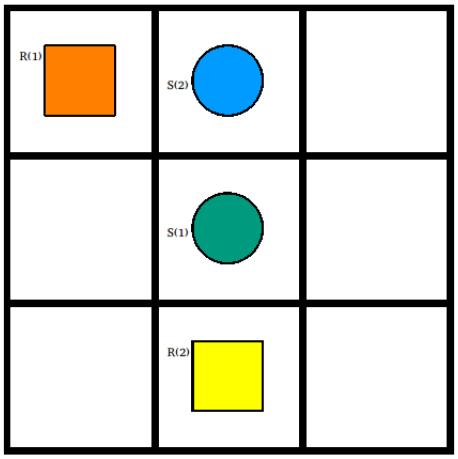

Figure 1: Simple MAPF problem on a 3x3 grid involving 2 robots.

```
1 init(object(node, 1), value(at, (1, 1))).
2 ...
3 init(object(robot,1), value(at,(1,1))).
4 init(object(shelf,1), value(at,(2,2))).
5 ...
```

2 Abstractions of Multi-Agent Pathfinding Problems

As stated in the introduction, this thesis builds upon Adrian Salewsky's Bachelor's Thesis, in which he used modified asprilo encodings to compute abstractions of instances. His thesis entails three different methods. Shortest Path, Node Combining and Reachable Nodes.
The Shortest Path method determines the shortest path for each robot and prunes the instance accordingly. The result of such an abstraction can be seen in Figure 2. Node Combining is split up into two approaches, calls "Open Node Combining" and "Complete Node Combining". The aim of this method is to combine several nodes into one node [2]. Open Node Combining only works for instances without obstacles, while the size of the combined subgrid is variable. Complete Node Combining solves the issue with the obstacles, but is only able to combine $2x2$ subgrids. Like Shortest Path, Reachable Nodes determines the shortest path of each robot. It then computes deviations from that path, assigning a value to the distance of certain nodes to the shortest path of a robot.

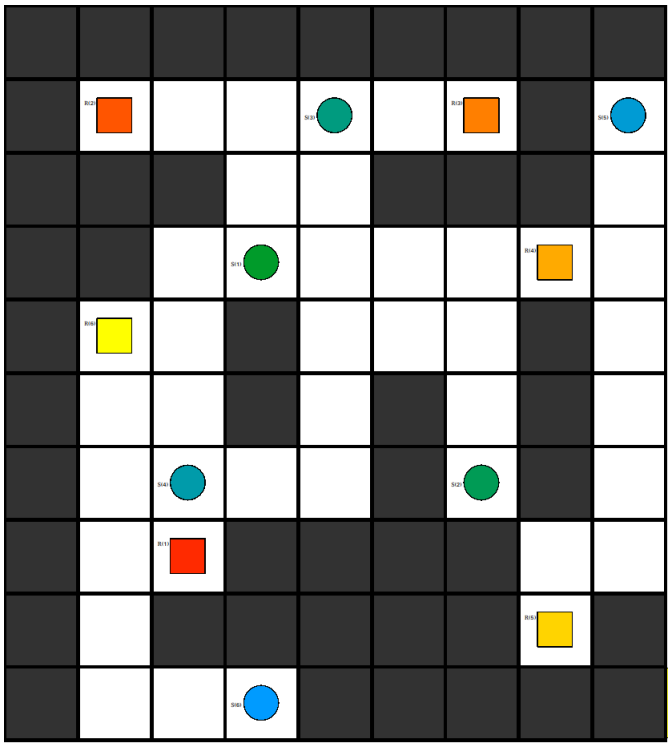

Figure 2: Abstraction of an MAPF instance.

3 Solving Techniques

The techniques developed in this thesis can be divided into two parts. Besides modified versions of the original asprilo encoding, the first part is an approach of implementing an A* style algorithm. The second part entails methods that try to find certain overlaps between robots to group them up and divide the abstraction. It continues with an algorithm that is very similar to the Conflict-based Search Algorithm. The clingo API for Python was used to combine different clingo encodings and relay information between them. For example, each abstraction is processed first to ensure that the different solvers are not heavily dependent on the representation of the abstraction and are capable of working with each of the abstractions.

3.1 Preparation of the Abstraction

Depending on the solver, each abstraction is processed in a before it is solved. All abstractions are translated first, since Adrian used different names for certain predicates. Additional information provided by the abstraction, like the predicate, $imp\_position$ are then used in various ways which is explained in the respective section of the solver. For all solvers, the Manhattan distance is calculated and provided for the solver, since the encoding needs a time step input in the form of a horizon. The horizon of an instance is the time step at which every robot occupies its respective shelf. In this context, the Manhattan distance refers to the distance from each starting point to the destination.

3.2 Original Encoding

Usually the original encoding is split up into different parts, out of simplicity they were merged into one file for the sake of this thesis. In the following they are described according to their division. An instance is represented by the predicate $init$. It assigns each object (node, robot, shelf) an ID and a coordinate (X, Y) on the grid.

3.2.1 Input

Importing an instance into the encoding translates the $init$ predicate. In this case, it is not a simple renaming, since certain information is added or omitted. For example, in line 5 almost every information of a node besides its coordinate is omitted. In lines 6-7 information is added in the form of the time step 0. The original encoding is more sophisticated, but since this thesis focuses on the M-domain, the other parts of the instance can be dropped completely.

```
1 robot(R):- init(object(robot,R),_).
2 shelf(S):- init(object(shelf,S),_).
3 isRobot(robot(R)) :- robot(R).
4 isShelf(shelf(S)) :- shelf(S).
5 position((X,Y)):- init(object(node,_),value(at,(X,Y))).
6 position(robot(R),(X,Y),0) :-
7     init(object(robot,R),value(at,(X,Y))).
8 position(shelf(S),(X,Y)) :-
9     init(object(shelf,S),value(at,(X,Y))).
```

3.2.2 Action

This part of the encoding is responsible for generating movement of robots. After defining the time scope (horizon) with the predicate time in line 1, the predicates direction and $nextto$ are defined in line 2-5. The predicate $nextto$ establishes a relation between connected nodes. The predicate move in line 6 chooses a direction for each robot at each time step. In lines 8-11, the predicate $position$ takes on the movement described in the predicate move and calculates the positions for each robot at each time step. Lines 13-14 describes the case in which a robot doesn't have a move predicate for a given time step. This is translated into the robot standing still.

```
 1 time(1..horizon).
 2 direction((X,Y)) :- X=-1..1, Y=-1..1, |X+Y|=1.
 3 nextto((X,Y),(DX,DY),(X',Y')) :- direction((DX,DY)),
 4     position((X,Y)),position((X',Y')),
 5     (X,Y)=(X'-DX,Y'-DY), (X',Y')=(X+DX,Y+DY).
 6 { move(R,D,T) : direction(D) } 1 :- isRobot(R), time(T).
 7
 8 position(R,C,T) :- move(R,D,T), position(R,C',T-1),
 9     nextto(C',D,C).
10 :- move(R,D,T), position(R,C,T-1),
11     not nextto(C,D,_).
12
13 position(R,C,T) :- position(R,C,T-1), not move(R,_,T),
14     isRobot(R), time(T).
```

The next section of the action encoding deals with edge and vertex collisions. Lines 1-3 ensure that

edges are not traversed at the same time by two different robots by lexicographically comparing the coordinates of the $moveto$ predicate in line 3. In Line 4 a constraint is set that prohibits a robot from occupying more than one position at the same time.

```
1 moveto(C',C,T) :- nextto(C',D,C), position(R,C',T-1),
2       move(R,D,T).
3 :- moveto(C',C,T), moveto(C,C',T), C < C'.
4 :- { position(R,C,T) : isRobot(R) } > 1, position(C), time(T).
```

3.2.3 Goal and Output

The original encoding works with orders and products. Since they are not necessary for this thesis, this part had to be modified. Lines 1-2 declare a robot to be processed if it occupies the same node as its shelf at the last time step (horizon). In Line 5 the encoding warrants that a robot needs to be processed.

```
1 processed(R,robot(R)) :- position(shelf(S),(X,Y)),
2       isRobot(robot(R)), position(robot(R),(X,Y),horizon),
3           R==S.
4 processed(R)    :- processed(R,robot(R)).
5 :- robot(R), not processed(R).
```

Last but not least, the desired output is described. The predicate occurs is important for the visualizer viz, which is also provided by the asprilo framework. Screenshots of the instances used in this thesis are produced by it.

```
1 #show.differently
2 #show init/2.
3 #show occurs(object(robot,R), action(move,D),      T) :
4       move(robot(R),D,T).
```

3.3 A*

A* is a pathfinding algorithm that uses a combination of different cost functions to find the shortest path between two nodes in a graph. Typically, A* uses the actual cost to reach a node $g(x)$ and a heuristic $h(x)$, like an estimated cost to reach the goal node, to compute the f-value for each node. In this implementation, $g(x)$ is determined in lines 1-6 by using the position predicates of the robots. With the $nextto$ predicate, nodes that are reachable by one step are

established. Since robots are allowed to stand still, a difference between actual movement and standing still has to be made. $h(x)$ is computed by using the Manhattan distance to the goal node in lines 8-10. With $g(x)$ and $h(x)$, $f(x) = g(x) + h(x)$ is calculated in lines 12-13. $f(x)$ is used to minimize moves with the minimize statement in line 18, after the movement of the robots is made dependent on the f-value in lines 15-17. The rest of the encoding is very similar to the original encoding.

```
 1 g(C,C', R, G) :- position(R,_,0),
 2       position(R,C',G'), nextto(C',_,C),
 3       C==C', G=G'.
 4 g(C,C', R, G) :- position(R,_,0),
 5       position(R,C',G'), nextto(C',_,C),
 6       C!=C', G=G'+1.
 7
 8 h((X,Y),R,T,H) :- position(R,(Xv,Yv),T),
 9       nextto((Xv,Yv),(DX,DY),(X,Y)),
10       goal(R,(X',Y')), H=|X-X'|+|Y-Y'|.
11
12 f(C,R,T,F) :- position(R,C',T), g(C,C',R,G),
13       h(C,R,T,H), F=G+H.
14
15 {move(R,D,F,T): direction(D)} 1 :- robot(R),
16       time(T), position(R,C',T-1), f(C,R,T-1,F),
17       nextto(C',D,C).
18 #minimize {F,R,T : move(R,_,T,F)}.
```

3.4 Grouping Methods

Two different but similar grouping techniques were developed in this thesis. Using the predicate $imp\_position$, which provides information on the shortest path of a robot, robots are grouped in two different ways. First, the important positions are used to group robots up if they share important positions. The second method looks for collisions and groups robots up accordingly.

3.4.1 Grouping by Important Positions

This method establishes a reflexive, symmetric and transitive relation between robots that share important positions. This is done in lines 1-2 with the share predicate. The code in line 3 picks

one robot of the group as a representative. The predicate $group$ in the following lines 5-7 is used to make sure that robots that don't share any positions with other robots are declared a group and that there aren't two representatives of the same group. Without going into much detail, the rest of the code add information about the group of the robot to the predicates $robot$, $shelf$ and $positon\_$ while confirming that every robot is part of a group.

```
1 1{share(R,R')} :- imp_position(R,(X,Y)), imp_position(R',(X,Y)).
2 share(R,R') :- share(R,RR), share(RR,R').
3 in_exgroup(R) :- share(R,R'), R!=R'.
4
5 {group(R) : in_exgroup(R)}.
6 group(R) :- not in_exgroup(R), robot(R).
7 :- group(R), group(R'), share(R,R'), R!=R'.
8
9 grpd(G,R) :- group(G), share(G,R).
10 :- robot(R), not grpd(_,R).
11
12 robot(R,G) :- grpd(G,R).
13 shelf(S,G) :- grpd(G,S).
14 position_((X,Y),G) :- imp_position(R,(X,Y)),
15       init(object(node,N),value(at,(X,Y))), grpd(G,R).
```

3.4.2 Grouping by Collisions

Grouping up robots by detected collisions works very similar to grouping up robots by important nodes. The only difference is that with this method, the important positions are used to find collisions. The way those collisions are found is the same as described in the next section. With information about collisions, the robots are then divided into two groups, collision-free and collision robots. The positions of collision-free robots can be adopted directly from the abstraction. The group of robots involved in a collision can't be divided further without a great amount of work, since resolving a collision between two robots can lead to further unpredictable collisions in the solving process. Since the plan for the collision-free robots is adopted, it is ensured that those robots stay collision-free.

3.5 Conflict-based Search (CBS) Algorithm

The Conflict-based Search (CBS) Algorithm [6] is a two-level search algorithm. The two levels are called the high-level search and the low-level search. At the high-level, CBS detects conflicts and adds constraints to the instance. At the low-level, CBS finds a solution for a single agent. The combination of those two levels ensures a valid solution. CBS is an iterative approach to the MAPF problem. To be able to mimic the CBS Algorithm the solving process is split up into two parts, finding collisions and using a heuristic to choose a constraint and solving the instance for a single robot. The preparation for this method only consists of renaming predicates, besides calculating the Manhattan distance for each robot. To solve the instance for a single robot, a modified version of the A* style encoding is used. The constraint in line 1 prohibits a robot from entering a restricted node at the coordinate $C == C'$ at time step T.

```
1       :- position(R,C,T), constraint(C',R,T), C==C'.
```

To find vertex collisions, the rule in lines 1-2 looks for two different robots visiting the same node at the same time. In lines 3-5 the case of two robots swapping nodes is handled. A heuristic calculating the Manhattan distance from the collision node to the goal node of the respective robot is defined in lines 6-7. It is necessary to exclude robots swapping nodes from this heuristic, since there is a special case where two robots occupy the shelf of the other robot. In this case, the heuristic would be the same for both robots, resulting in an endless cycle of passing on the constraint to the other robot.

```
1 collision(R1,R2,(X,Y),T) :- position(R1,(X,Y),T),
2       position(R2,(X,Y),T), R1!=R2.
3 collision(R1,(X',Y'),T+1) :- position(R1,(X,Y),T),
4       position(R2,(X',Y'),T), position(R1,(X',Y'),T+1),
5       position(R2,(X,Y),T+1), R1!=R2, not collision(R2,(X,Y),T+1).
6 h((X,Y),R1,T,H) :- collision(R1,R2,(X,Y),T), goal(R1,(X',Y')),
7       H=|X-X'|+|Y-Y'|.
```

The predicate s selects a heuristic h for each collision in line 1. Lines 2-3 make sure that the heuristic with the lower value is selected and that each heuristic is selected. Then the predicate constraint is defined in lines 4-5 using the information from above. Lastly, robots that received a constraint in the next step of the solution process are marked. This information is needed to be able to replan the right robots.

```
1 1 {s(C,R,T,H) : h(C,R,T,H)} 1 :- collision(C,T).
2 :- s(C,R,T,H), h(C,_,T,H'), H'<H, H!=H'.
```

```
3 :- h(C,_,T,_), not s(C,_,T,_).
4 constraint(C,R,T) :- s(C,R,T,_).
5 constraint(C,R,T+1) :- collision(R,C,T).
6 robot_c(R) :- constraint(_,R,_).
```

3.6 Python Scripts

Different Python scripts named $original.py$, $cbs.py$ and $group.py$ were developed, used to run the encoding and to assure that the communication between the different encodings works. Several auxiliary functions were used to manage the input and the output of the clingo programs. For example, one of those functions combines two files and saves the content in a third file. All Python scripts use the clingo API to ground and solve the instance. The scripts $original.py$ and $group.py$ are very straight forward. Besides the auxiliary functions, they only contain two main functions. One of those functions prepares the abstraction. It extracts information about the Manhattan distance of the robots, and in the case of $group.py$ it even extracts information on the group representative. This information is then combined with the prepared abstraction and the encoding to solve the problem. Since $cbs.py$ aims at iteratively solving the problem, the script needs additional functions. One to call the encoding that finds collision and sets constraints, and two additional functions to handle a list of robots that need to be replanned. To generate the abstractions, the Python script $hor\_inc.py$ was used. This script is provided by Adrian. For a detailed explanation, a look at his thesis is recommended.

4 Benchmarking

4.1 Benchmarks

Although asprilo provides an instance generator and Adrian was kind enough to allow the usage of his Python script that generates instances, all the used instances were created by hand using a template. The aim of this procedure was to be able to increase the difficulty of a layout without relying on the randomness of a map generator. To test the solvers, four different layouts with the size of $8x8$ nodes, $16x16$ nodes, $32x16$ nodes and $64x16$ nodes were used. Both Adrian's encoding and some of the encodings that were developed in this thesis struggle with the $64x16$ node layout. As of this, no bigger layout was tested. The dimensions of the bigger layouts were purposefully chosen to resemble warehouse-like instances. To increase the difficulty, obstacles were added to force a possible congestion and shelves of robots were purposefully placed to

increase possible collisions. The benchmarks were run on an Ubuntu 22.04.2 LTS 64-bit machine with an Intel® Core™ i5-1035G1 CPU and 8 GB of RAM using clingo 5.5.0. Unfortunately, the Turing cluster of the University of Potsdam [7] could not be used to benchmark the solvers, since problems regarding the access arose that could not be resolved in time before the submission date.

4.2 Results

Only two methods were able to complete a benchmark that is comparable. The modified original encoding and the grouping by important instances method. Grouping by collision, find invalid solutions. The CBS style algorithm takes a lot of time to ground and solve problems. The way the code is structured, all constraints that are chosen are never forgotten, the search can reach a dead end. The A* approach in clingo also disappoints. Using it without minimizing the f-value is bearable, which means it's only capable of finding subpar solutions.
The results of the benchmarks are contained in Table 1-4. To make a better comparison, not only the runtime but also just the solution time (time without grounding) is displayed. All values are in ms. The grouping method of this thesis is only able to beat the original encoding in certain scenarios. Such an instance can be seen in Figure 3. Instances that can be separated and planned individually are the perfect scenario for the grouping method. But there are still possible improvements that can be made. For instance, the method can be parallelized, or the grounding time could be lowered by efficiently using the clingo API. This also applies for the CBS approach, where parallelization was attempted. The overhead of communication between workers slowed the program down instead of speeding it up. In the end, the original asprilo encoding performed better on the abstraction than the grouping method, both in grounding and in solving the abstraction.

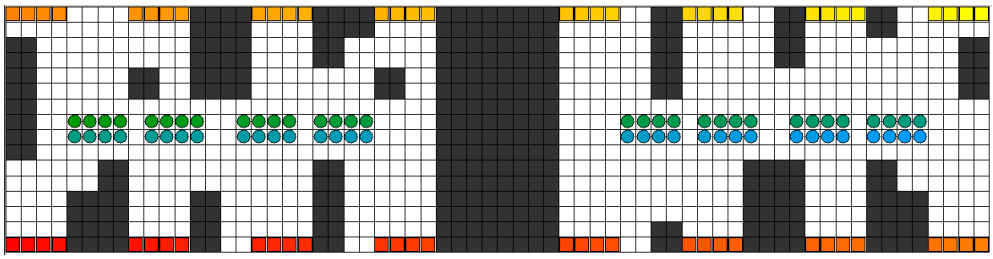

Figure 3: Layout 64x16(4). An Instance where Grouping Method outperforms Original Encoding.

5 Conclusion

This paper attempted at solving abstractions of Multi-Agent Pathfinding problems. Not all developed methods were good enough to compete with the original encoding of the asprilo framework. The attempts to mimic the Conflict-based Search Algorithm and the A* algorithm showed potential, even though they weren't able to complete the benchmark. Of the grouping methods of this thesis, only one was able to compete. Grouping robots by shared important nodes can compete with the original encoding in certain scenarios. And with some adjustments, like parallelization and optimization, it should be able to compete with it regardless of the scenario. Solving abstractions of Multi-Agent Pathfinding problems has a lot of potential. Using the abstraction to divide the instance further shows good results in big divisible abstractions.

layout	sp	cnc	rn
8x8(3)	29	25	35
8x8(4)	33	27	34
16x16(1)	52	44	61
32x16(1)	324	258	NA
32x16(2)	903	1081	NA
32x16(3)	3130	3668	NA
64x16(1)	4644	535	NA
64x16(4)	2586	1057	NA

Table 1: Method: Grouping By Important Positions, Runtime in ms

layout	sp	cnc	rn
8x8(3)	2.5	2.2	2.6
8x8(4)	3.4	2.5	4
16x16(1)	4.9	3.5	5.1
32x16(1)	18.4	14.9	None
32x16(2)	31.1	38.6	None
32x16(3)	164.7	219.2	None
64x16(1)	124	26.9	None
64x16(4)	90.2	50	None

Table 2: Method: Grouping By Important Positions, Solution Time in ms

layout	sp	cnc	rn
8x8(3)	13	16	29
8x8(4)	10	14	18
16x16(1)	12	15	32
32x16(1)	116	161	None
32x16(2)	366	694	None
32x16(3)	1247	2178	None
64x16(1)	318	556	None
64x16(4)	773	1230	None

Table 3: Method: Modified Original Encoding, Runtime in ms

layout	sp	cnc	rn
8x8(3)	1.5	1.8	2.2
8x8(4)	1.5	2.2	2.2
16x16(1)	1.6	2.3	4.4
32x16(1)	6.7	9.5	None
32x16(2)	13.2	27.9	None
32x16(3)	52.2	128.8	None
64x16(1)	18.9	34.5	None
64x16(4)	38.2	58.6	None

Table 4: Method: Modified Original Encoding, Solution Time in ms

References

[1] https://github.com/tramadan-up/mapf-ba

[2] https://github.com/salewsky/MAPF-Project

[3] https://potassco.org/

[4] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, & Roman Bartak. (2019). Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks.

[5] https://potassco.org/asprilo/

[6] Guni Sharon, Roni Stern, Ariel Felner, & Nathan R. Sturtevant (2015). Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence, 219, 40-66.

[7] https://www.cs.uni-potsdam.de/bs/research/labs.html#turing

Eidesstattliche Erklärung / Statutory Declaration

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publi- cations concerned. The thesis in the same or similar form has not been submitted to any examina- tion body and has not been published. This thesis was not yet, even in part, used in another exam- ination or as a course performance.

Weinheim, March 28, 2023 Tarek Ramadan