Model Checking of Multi-agent Path Planning algorithm using Formal Methods

Eashwar Sathyamurthy
Systems Engineering
University of Maryland
College Park, United States
eashwar@umd.edu

***Abstract*—The paper presents how model checking tools can validate a coupled multi-agent path planning algorithm. First, the robots' actions, robot-robot collision, and obstacles in the environment are represented in Linear Temporal Logic (LTL). Then the multi-agent path planning algorithm is converted into a model capable of determining robots' subsequent actions. A safety property of all the robots reaching their goal without robot-robot and environmental collision is formulated using linear temporal logic. Finally, the algorithm's model is validated to check whether it satisfies the negation of the safety property. The results obtained in this paper show that the model solves the multi-agent path planning problem. The paper also visualizes the obtained solution to emphasize the correctness of the algorithm and the approach.**

***Index Terms*—Linear Temporal Logic, coupled-multi-agent path planning, formal methods.**

I. Introduction

Multi-agent systems have a wide range of applications in surveillance, maintaining security, and transportation mainly due to their ability to divide and perform complex tasks concurrently in a robust manner. However, the flexibility and robustness make the planning of such systems difficult. The path planning of multi-agent systems consists of two parts: The ability of individual agents to plan the path to its goal and Agents avoiding collision with other agents while traversing its path. Collision avoidance makes it difficult for all the agents to follow an optimal path. Hence, the path planning for multi-agent usually results in sub-optimal solutions. The ability to produce solutions that are optimal or sub-optimal can help distinguish multi-agent path planning algorithms. Optimality or sub-optimality criteria of an algorithm can be determined by verification. One way to perform verification is by representing the solution as finite states and using model-checking to validate it [1]. This paper presents how model checking can verify coupled multi-agent path planning algorithms using formal logic.

Multi-agent path planning can be divided into two categories: a) Decoupled Planning b) Coupled Planning. In decoupled planning, the path planning is performed in a low dimensional search space [6] [7]. Hence in most cases, the solutions are either sub-optimal or cease to exist, but scalability is achieved. Coupled planning considers the entire configuration space to plan the path for all agents [4] [5]. As a result, all the possible routes are explored to find the optimal path. The main advantage of coupled planning is its ability to produce optimal trajectory (if it exists). At the same time, the complexity increases exponentially as the number of agents increases. Thus, scalability is an issue. Therefore, there is an inherent trade-off between optimality and scalability in multi-agent path planning.

Validation of the path planning algorithm helps to check its optimality, scalability, and completeness in different search spaces. Model-checking tools help validate the path planning algorithm by specifying characteristics of the environment and agents' movement in linear temporal logic. Spin is a model-checking tool (that uses a high-level language called PROMELA to define system descriptions) for validation [9]. Spin checks the LTL specification by evaluating it be true or false. If the LTL specification is incorrect, spin provides a counterexample to justify why it is validated wrong, which is also the reason for choosing spin.

The paper's main contribution is to show how model checking tools can validate a multi-agent path planner and produce optimal solutions. The optimal solutions are derived from the counterexample produced by the spin when the specification is validated false.

The paper is structured the following way: Section 2 is the literature review, Section 3 is Preliminaries, Section 4 is Problem Formulation, Section 5 is Implementation Details, Section 6 is Results, and Section 7 is the Conclusion and Future Works.

II. Literature Review

Many multi-agent path planning algorithms were proposed by extending the work done on single-agent path planning to multiple agents. Carpin *et al.* developed the sampling-based Rapidly Exploring Random Trees(RRT) algorithm for multiple-agent by finding the agents' paths simultaneously, [4] which resulted in asymptotically optimal solutions. Grenouilleau *et al.* modified the classic A* algorithm to find multiple agents' paths having multiple goals [10]. An improved version of coupled A* called local repair A* proposed by Hart *et al.* plans the individual paths of each robot using A* without considering other robots. [14] If the robots come into collision, re-planning paths of the robots involved in a collision happen. David Silver introduced the notion of cooperative pathfinding by proposing Cooperative A*(CA*), Hierarchical Coop-

erative A*(HCA*), and Windowed Hierarchical Cooperative A*(WHCA*) algorithms. These algorithms solve multi-agent path planning problems by including wait action, making the robot wait for other robots to pass by to avoid the collision [15]. The coupled path planning algorithm used in this paper for validation is coupled-A*, which is very similar to [10] considering each agent has only one goal.

Model-checking is a method to verify finite-state systems formally. The applicability of model-checking was emphasized by Larsen *et al.* by developing an efficient automatic model-checking algorithm for real-time systems by combining compositional and symbolic model-checking techniques [12]. Clarke *et al.* briefly explained how model checking could be used for algorithmic verification and debugging [13]. Holzmann *et al.* introduced a method of extracting verification models from source code to verify an extensive software application formally [16] [17] [18]. This paper utilizes the applicability of model-checking to verify coupled multi-agent path planning algorithms.

Previous works have been done to incorporate model-checking into multi-agent path planning. Hoek *et al.* proposed a novel model checking algorithm using Alternating Temporal Epistemic Logic, a temporal logic incorporated with knowledge operators, to perform multi-agent path planning for epistemic goals [2]. Bordini *et al.* developed a rational-agent programming language called AgentSpeak based on AgentSpeak(L)(an abstract agent-oriented programming language), capturing agents' mentalistic notions such as beliefs, desires, and intentions [3]. Giunchiglia *et al.* introduced and briefly described a model-checking paradigm for planning [11]. This paper uses the same idea as [11] to convert coupled multi-agent path planning algorithm into the model-checking paradigm.

III. PRELIMINARIES

This section provides the syntax and semantics of linear temporal logic [19] used to formulate the multi-agent path planning problem and provides a visual representation of the search space used. The semantics of this logic is defined in the discrete domain.

Let $AP = \{ap_1, ap_2, .., ap_n\}$ be a finite set of atomic propositions.

A. Syntax of LTL formulas

A linear temporal logic LTL formula over AP is inductively defined as follows:

- All atomic propositions $ap_i \in AP$ is a LTL formula
- If β and γ are formulas, then $\beta \vee \gamma, \neg\beta, \beta$ U γ, β X γ are also LTL formulas.

B. Semantics of LTL formulas

The semantics of LTL formulas are given over w, words in the set 2^{AP}. Thus, a LTL formula can be satisfied by infinite sequences of set of propositions. The words are of the form $w = w_1, w_2, w_3, ..,$ where $w_i \in 2^{AP}, i \geq 1$. The satisfaction($\models$) relation between a word(w) and a LTL formula (β) is given by:

- $w_i \models AP$ if $AP \in w_i$;
- $w_i \models \neg\beta$ if $w_i \not\models \beta$;
- $w_i \models \beta \vee \gamma$ if $w_i \models \beta$ or $w_i \models \gamma$;
- $w_i \models$ Xβ if $w_{i+1} \models \beta$(β is true in the next time step)
- $w_i \models \beta$ U γ if there exists $j \geq i$ such that $w_j \models \gamma$ and $\forall i \leq k \leq j$, we have $w_k \models \beta$

The symbols $\neg$ and $\vee$ represent negation and disjunction, respectively. Additional logical operators in LTL formula include $\wedge$ (conjunction), $\implies$ (implication) and $\iff$ (equivalence). X and U represent next and until temporal operators, respectively. The formula Xβ means β will be true in the next time step. The formula β U γ means that γ will eventually become true, and β will remain true until that happens. Additional temporal operators present in LTL formulas are $\Diamond$ (eventually) and $\Box$ (always). The formula $\Diamond\beta$ means that β will eventually become true infinite time. The formula $\Box\beta$ means that β will always be true over w. Combining $\Diamond$ and $\Box$ can improve the fluidity of LTL logic formulas. The formula $\Diamond\Box\beta$ means β is true infinitely often and $\Box\Diamond\beta$ means β means β becomes eventually true and stays true forever. The upcoming section formulates the multi-agent path planning problem using the above-defined LTL logic.

IV. PROBLEM FORMULATION

Consider n robots with a common search space *S*. Each robot is assigned a start and goal position in *S*. Each robot can move in eight possible directions given East, West, North, South, North-East, North-West, South-East, and South-West. The goal is to find an optimal path from start to goal positions for all the robots while avoiding collision with the environment. The below figure shows the environment:

The robots are represented by r^i where $i = \{1, 2, ..n\}$. The constraint space of the environment is represented by *E*. The environmental constraints are assumed to be static. *E* contains all the coordinate points in search space *S*, which are not traversed by the robots. The start and goal points of the robots are represented as follows: s^i and g^i where $i = \{1, 2, ..n\}$. $\pi^* = \{\pi_1^*, \pi_2^*, .., \pi_n^*\}$ represent the optimal path traversed by the robots to reach the goal from their start position. Each path π_i^* in π^* consists of finite number of coordinate points $p^i[j]$ where $j = \{1, 2, ..m\}$, m is a finite number, through which the robot *i* passes. The multi-agent path planning problem can be represented as a constraint optimization problem given by

$$\begin{aligned} \min_{\pi} \quad & \sum_{i=1}^{n} \pi_i \\ \text{s.t.} \quad & \pi_i \cap E = \emptyset \\ & \pi_i \cap \pi_j = \emptyset \quad \forall i \neq j \end{aligned}$$

Similarly, the multi-agent path planning problem can be formulated as a model checking problem by specifying the robots' action and environment and collision constraints in LTL logic as follows:

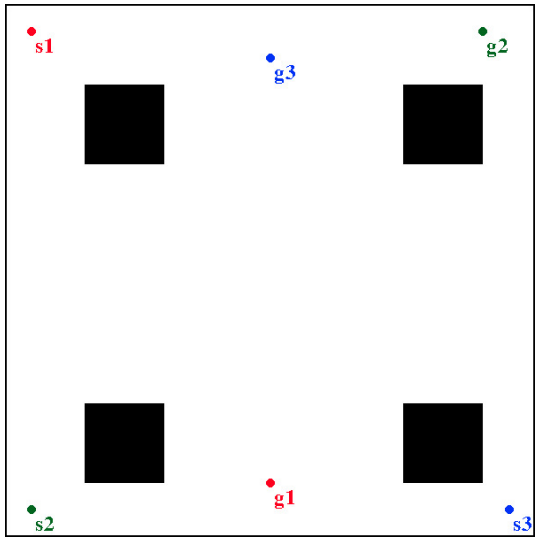


Fig. 1: The above figure shows the search space *S*, which considers three robots with start and goal positions given by $s1, s2, s3$, and $g1, g2, g3$ respectively. The black boxes are environment constraints *E*, which is discussed later in this section.

.

$$A \wedge B \wedge C \tag{1}$$

This following points below elaborates the LTL specification:

- $A = \Diamond\Box(\bigwedge_i(p^i[j] = g^i))$ represents that the eventually all the robots will reach their goals and stay in that position forever.
- $B = \neg(\bigvee_i(p^i[j] \wedge E))$ represents robots avoiding environmental constraints. This specification will make sure that all robots at any point during their path traversal will not collide with the obstacles in the environment.
- $C = (\neg(\pi_i \wedge \pi_j)\{i \neq j\})$ This specification will avoid any robot-robot collisions in the search space.

So, the LTL specification 1 would imply that eventually, all the robots will reach their goals without environment and robot-robot collision and stay in their respective goal positions indefinitely. The next section will explain how the coupled multi-agent path planning algorithm model is implemented and validated and the results obtained from validation.

V. Implementation Details

The model of a couple of multi-agent path planning algorithms is created using the Spin model checking tool coded in PROMELA. The algorithm is then verified by checking LTL specifications1. The PROMELA model validates the specification to be false and produces a counter-example, which is the solution to the coupled multi-agent path planning problem. The counter-example is stored in a text file for visualization using the Pygame package in Python. The below block diagram describes the implementation:

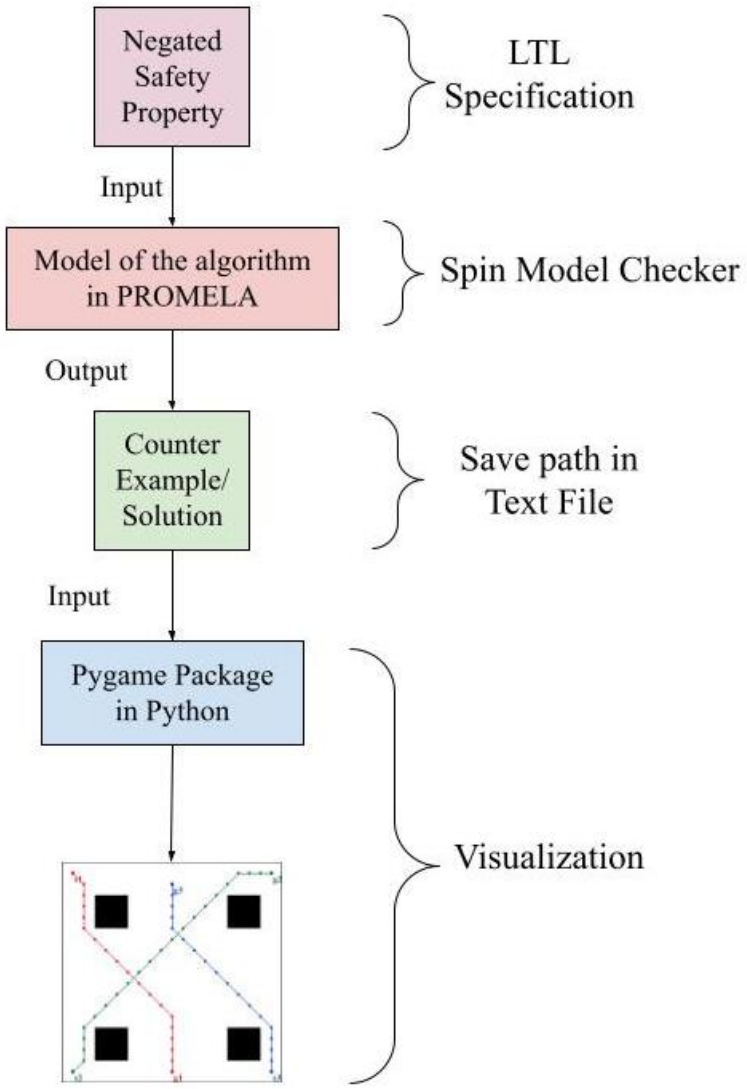
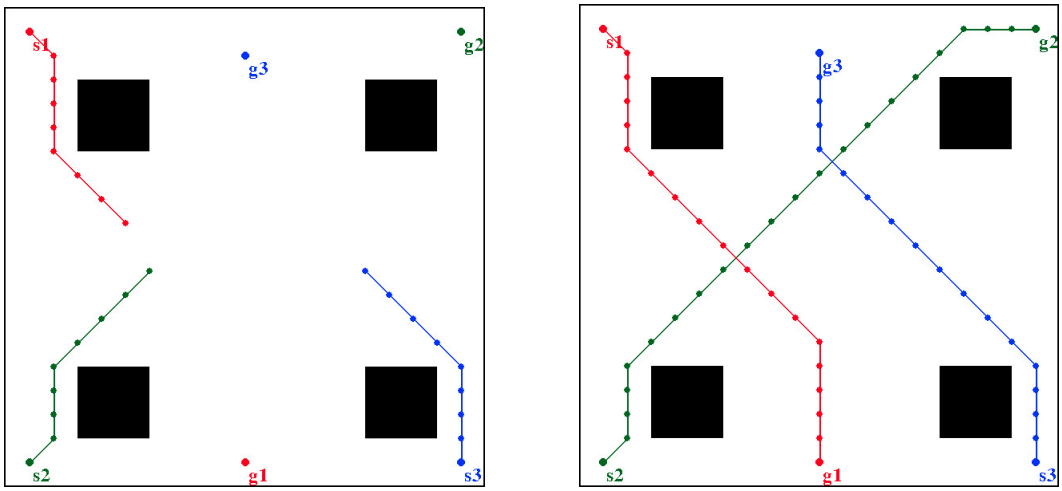


Fig. 2: The figure provides a high level block diagram representation of the implementation process.

VI. Results

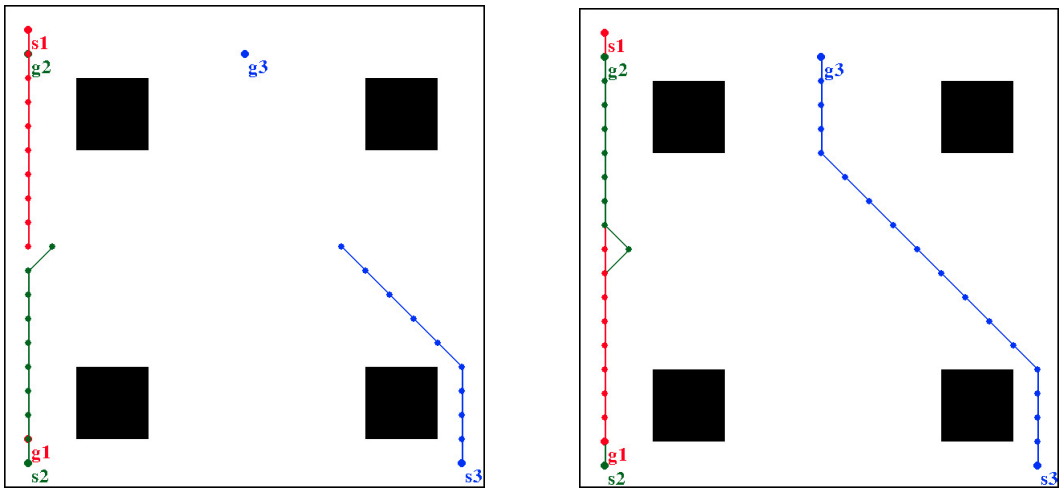
This section provides the results obtained by verifying coupled multi-agent path planning algorithm on a 3.50 GHz Intel Quad-Core i7-6700HQ, 8 GB RAM, 1TB HDD, NVIDIA GeForce GTX 960M laptop. The version of the model checking tool used is Spin Version 6.5.1.

The results shown considers three robots r^1, (red) r^2 (green),r^3 (blue) highlighting the robot-robot (Figure 4) and environmental collision avoidance (Figure 3) of coupled A* algorithm. The path shows the visualization of the counter-example produced by Spin, which is the solution to the multi-agent path planning problem. Please refer to the Appendix section for more information regarding results.



(a) (b)

Fig. 3: Figure 3a shows that robots traversing towards their goal. The dots in the optimal paths π_i^* are the coordinates $p^i[j]$ which robots traverse through. The intersection between the paths in Figure 3b does not indicate robot-robot collision as the robots traverse through the nodes at different time intervals.



(a) (b)

Fig. 4: Figure 5a highlights the collision avoidance between robot r^1 (red) and r^2 (green). Robot r^2 diverges its path when there is a possible occurrence of collision with r^1. The figure 5b shows all robots reach their goals without collision.

VII. Conclusion and Future Works

Coupled A* multi-agent path planning algorithm is validated using a spin model checker. The counter-example produced by spin is visually shown using the Pygame package in the python programming language. The coupled path planning algorithm produced optimal paths for all robots in the collision and collision-free scenarios. This can be attributed to the admissible heuristics of the algorithm. Future works include validating the coupled multi-agent path planning algorithm for many robots and creating a Spin model of decoupled multi-agent path planning algorithms, comparing coupled and decoupled algorithms performance using the solutions produced by Spin.

References

[1] Model-checking. (2021, March 13). Retrieved April 14, 2021, from https://en.wikipedia.org/wiki/Model_checking

[2] Van der Hoek, W., Wooldridge, M. (2002). Tractable multiagent planning for epistemic goals. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems Part 3 - AAMAS '02. doi:10.1145/545056.545095*

[3] R. H. Bordini, M. Fisher, M. Wooldridge, and W. Visser, "Model checking rational agents," in *IEEE Intelligent Systems*, vol. 19, no. 5, pp. 46-52, Sept.-Oct. 2004, doi: 10.1109/MIS.2004.47.

[4] S. Carpin and E. Pagello. On parallel RRTs for multi-robot systems. In *Proc. 8th Conf. Italian Association for Artificial Intelligence*, pages 834–841. Citeseer, 2002.

[5] Robert W Ghrist and Daniel E Koditschek. Safe, cooperative robot dynamics on graphs. *SIAM Journal on Control and Optimization*, 40(5), 2002.

[6] K. Kant and S.W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72, 1986.
[7] Ryan Malcom. Multi-robot path-planning with subgraphs. In *Australasian Conference on Robotics and Automation*, 2006.
[8] M. Kloetzer and C. Belta, "Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions," in *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 320-330, April 2007, doi: 10.1109/TRO.2006.889492.
[9] Holzmann, G. (n.d.). Formal verification. Retrieved April 13, 2021, from http://spinroot.com/spin/whatispin.html
[10] Grenouilleau, F., Hoeve, W., &; Hooker, J. (n.d.). A multi-label a* algorithm for multi-agent pathfinding. Retrieved April 14, 2021, from https://ojs.aaai.org//index.php/ICAPS/article/view/3474
[11] Giunchiglia F., Traverso P. (2000) Planning as Model Checking. In: Biundo S., Fox M. (eds) Recent Advances in AI Planning. ECP 1999. Lecture Notes in Computer Science, vol 1809. Springer, Berlin, Heidelberg. https://doi.org/10.1007/10720246_1
[12] Larsen K.G., Pettersson P., Yi W. (1995) Model-checking for real-time systems. In: Reichel H. (eds) Fundamentals of Computation Theory. FCT 1995. Lecture Notes in Computer Science, vol 965. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-60249-6_41
[13] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. 2009. Model-checking: algorithmic verification and debugging. Commun. ACM 52, 11 (November 2009), 74–84. DOI:https://doi.org/10.1145/1592761.1592781
[14] Hart, P., Nilsson, N. and Raphael, B. (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions Systems Science and Cybernetics, 4, 100-107. http://dx.doi.org/10.1109/TSSC.1968.300136
[15] David Silver. 2005. Cooperative pathfinding. In Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'05). AAAI Press, 117–122.
[16] Holzmann, G. J., & H. Smith, M. (2001). Software model checking: Extracting verification models from source code. Software Testing, Verification, and Reliability, 11(2), 65-79.
[17] Holzmann GJ. Designing executable abstractions.Proceedings of Formal Methods in Software Practice. ACM Press:Ft. Lauderdale, FL, U.S.A., 1998; 103–108
[18] Holzmann GJ. Logic verification of ANSI-C code withSpin.Proceedings of SPIN 2000 Workshop(Lecture Notes inComputer Science, vol. 1885), September 2000, Stanford University. Springer: Berlin, 2000; 131–147
[19] Wikimedia Foundation. (2021, April 20). Linear temporal logic. Wikipedia. https://en.wikipedia.org/wiki/Linear_temporal_logic.

APPENDIX

The appendix contains details regarding the coupled A* algorithm, additional results obtained from Spin, and difficulties faced during the project.

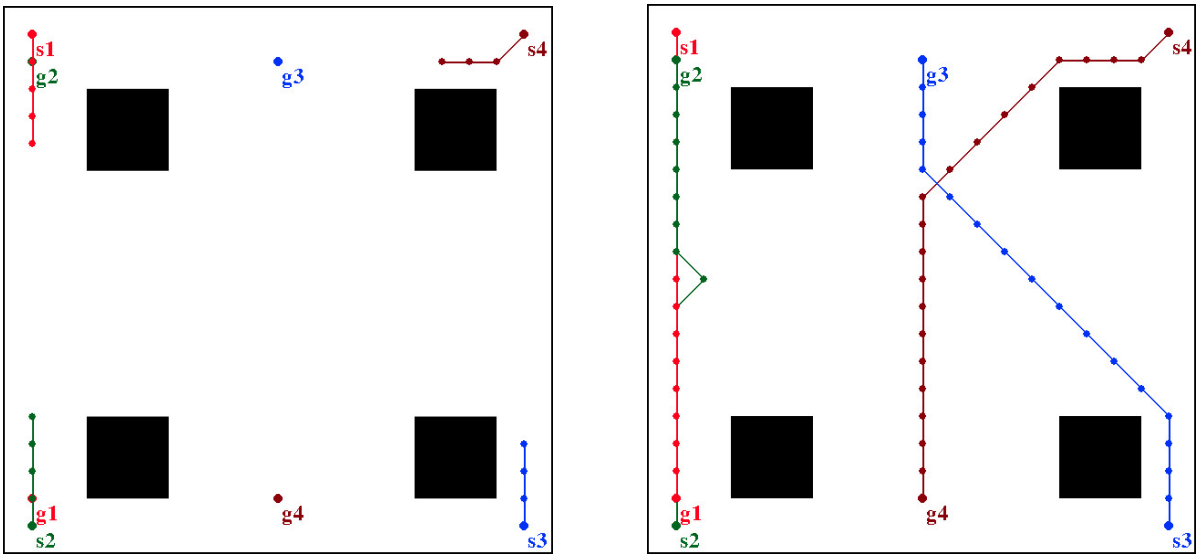
A. *Coupled A* algorithm*

Coupled A* algorithm is an extension of the A* algorithm implemented on multiple agents. The algorithm comes in the coupled category because each robot knows the position of other robots in the search space. Actions taken by all the robots are joint, meaning all the robots move towards their goal, simultaneously reducing the global path length. In each iteration, robot checks the possibility of robot-robot and environmental collision while traversing towards its goal. The solution produced by the algorithm is always optimal because of its admissible heuristics. The pseudo-code of the algorithm is given below:

Algorithm 1 Pseudocode for Coupled A*

Require: v_I - Initial Configurations of the robots
v_F - Final Configurations of the robots
for $v_K \in V$ **do**
$v_K.cost\_to\_come \leftarrow$ INF
end for
$v_I \leftarrow 0$
$v_I.back\_ptr \leftarrow \varnothing$
pQueue = $\{v_I\}$
{Adding intial configuration to the priority queue}
while True **do**
$v_K =$ pQueue.pop(0)
{Priority queue Sort nodes in ascending order of v.cost_to_come + v.cost_to_go}
if $v_K = v_F$ **then**
return back_track(v_K)
{Back tracking the solution path using $v_K.back\_ptr$}
end if
if collision(v_K) $= \varnothing$ **then**
for $v_l \in neighbors(v_K)$ **do**
if $v_K.cost\_to\_come + distance(v_K, v_l) < v_l.cost\_to\_come$ **and** $collision(v_l) = \varnothing$ **then**
$v_l.cost\_to\_come \leftarrow v_K.cost\_to\_come + distance(v_K, v_l)$
$v_l.back\_ptr \leftarrow v_K$
end if
end for
end if
end while
return No path exists

Coupled A* algorithm used priority Queue data structure to sort the neighbors based on distance in ascending order automatically. The prerequisite of the algorithm given in pseudo-code is only the robot's start and goal configuration. The collision() function checks for both environmental and robot-robot collision for a particular node in consideration. The algorithm stops when all the robots reach their goal.



(a) (b)

Fig. 5: The figure shows the visual output for multi-agent path planning problem for 4 robots r^1, r^2, r^3, r^4 with start and goal positions as $s1, s2, s3, s4$ and $g1, g2, g3, g4$ respectively.

```
eashwar@eashwar:~/Desktop/prob$ spin -a finalproj.pml
eashwar@eashwar:~/Desktop/prob$ cc -o pan pan.c
eashwar@eashwar:~/Desktop/prob$ ./pan

(Spin Version 6.5.1 -- 31 July 2020)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 272 byte, depth reached 8690, errors: 0
     8691 states, stored
        0 states, matched
     8691 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    2.487       equivalent memory usage for states (stored*(State-vector + overhead))
    2.722       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  131.171       total actual memory usage


unreached in proctype planner
        (0 of 160 states)

pan: elapsed time 0.01 seconds
eashwar@eashwar:~/Desktop/prob$
```

Fig. 6: This figure shows the commands used to run the spin program. The figure shows that the program runs without deadlocks or syntax errors.

B. *Additional Results*

This section mainly includes the results which are not included in the main paper. The figure 5 shows the optimal produced by Spin for four robots. This result is the extension of the work shown in the main paper for three robots. This result is mainly included to show that the algorithm can be extended to many robots by correctly describing the constraints and defining the start and goal positions of the robot.

C. *Spin Output*

This section provides information regarding the commands used to run the Spin program and obtain the output. The figure 6 shows that the spin program runs without syntax errors and infinite loops, which means that the model has found the solution for the multi-agent path planning problem.

D. *Difficulties Faced*

This section contains difficulties faced during the creation of the model for coupled multi-agent path planning algorithm.

1) ***Limited Data structures****:* As Spin is mainly a model checking tool rather than a programming language, the choice of data structures was limited. Arrays are the only data structure present. This complicated the way the model was created for the algorithm. In any path planning algorithm, creating a node, usually in the form of a linked list and backtracking through the linked list to obtain the solution path, is considered standard practice. This was not possible in Spin. This problem was solved by printing the points in the solution path onto the terminal while the model is finding the optimal solution path shown in the figure 7.

```
eashwar@eashwar:~/Desktop/prob$ spin finalproj.pml
      (1, 1), (1, 19), (19, 19), (19, 1)
      (1, 2), (1, 18), (19, 18), (18, 2)
      (1, 3), (1, 17), (19, 17), (17, 2)
      (1, 4), (1, 16), (19, 16), (16, 2)
      (1, 5), (1, 15), (19, 15), (15, 2)
      (1, 6), (1, 14), (18, 14), (14, 3)
      (1, 7), (1, 13), (17, 13), (13, 4)
      (1, 8), (1, 12), (16, 12), (12, 5)
      (1, 9), (1, 11), (15, 11), (11, 6)
      (1, 10), (2, 10), (14, 10), (10, 7)
      (1, 11), (1, 9), (13, 9), (10, 8)
      (1, 12), (1, 8), (12, 8), (10, 9)
      (1, 13), (1, 7), (11, 7), (10, 10)
      (1, 14), (1, 6), (10, 6), (10, 11)
      (1, 15), (1, 5), (10, 5), (10, 12)
      (1, 16), (1, 4), (10, 4), (10, 13)
      (1, 17), (1, 3), (10, 3), (10, 14)
      (1, 18), (1, 2), (10, 2), (10, 15)
      (1, 18), (1, 2), (10, 2), (10, 16)
      (1, 18), (1, 2), (10, 2), (10, 17)
      (1, 18), (1, 2), (10, 2), (10, 18)
     All robots reached their goals
1 process created
eashwar@eashwar:~/Desktop/prob$
```

Fig. 7: This figure shows the coordinate points printed on the terminal while the model is finding the optimal path.

2) ***Difficulty defining constraints****:* As shown in the figure 5, all the environmental constraints are polygonal in shape. But in the real world, the constraints are rigid. The main reason for choosing polygonal constraints is the difficulty of defining a non-rigid environmental constraint in Spin. Non-rigid constraints are usually characterized by defining its boundary and intersecting all the half-planes formed by the boundary. This is not possible in Spin due to its limited library resources.

3) ***Interfacing with C****:* Spin provides an option to use the C programming language in between a spin program by using the command **c_code{}**. A normal C program with no syntax errors can be executed within the curly braces of the **c_code** command. But, all the variables declared within the **c_code** are not accessible outside the command and vice versa. This makes the interfacing between C and Spin harder.

4) ***Extracting data from trial file****:* Initially, the optimal path produced was supposed to be accessed from the trial file produced during the execution of the Spin program. But, the trial file did not produce the solution path in a format suitable for visualization using Python. Because of this reason, the coordinates displayed on the figure 7 were copied onto a text file. Then visualization was done by processing that text file.

These are some of the additional details provided, which are not mentioned in the main paper.