Journal of Experimental & Theoretical Artificial Intelligence

ISSN: 0952-813X (Print) 1362-3079 (Online) Journal homepage: http://www.tandfonline.com/loi/teta20

DiMPP: a complete distributed algorithm for multi-agent path planning

Satyendra Singh Chouhan & Rajdeep Niyogi

To cite this article: Satyendra Singh Chouhan & Rajdeep Niyogi (2017) DiMPP: a complete distributed algorithm for multi-agent path planning, Journal of Experimental & Theoretical Artificial Intelligence, 29:6, 1129-1148, DOI: 10.1080/0952813X.2017.1310142

To link to this article: https://doi.org/10.1080/0952813X.2017.1310142

JOURNAL OF EXPERIMENTAL & THEORETICAL ARTIFICIAL INTELLIGENCE, 2017
VOL. 29, NO. 6, 1129–1148
https://doi.org/10.1080/0952813X.2017.1310142

Taylor & Francis
Taylor & Francis Group

DiMPP: a complete distributed algorithm for multi-agent path planning

Satyendra Singh Chouhan and Rajdeep Niyogi

Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India

ABSTRACT

Multi-agent path planning (MAPP) is a challenging task that aims to find conflict free paths for all the agents in a given domain. Priority-based decoupled approach is one of the several approaches to solve a MAPP problem. It works as follows: first, find paths of individual agents and then restructure these paths based on some priority of the agents. Most of the existing decoupled approaches use centralised algorithms. However, multi-agent systems are inherently distributed, where agents have limited information and each agent may not know the total number of agents in the system. Some of these aspects are incorporated in DMAPP. DMAPP is an existing fully distributed algorithm that works in three phases: (i) individual path planning, (ii) priority decision-making and (iii) plan restructuring. However, DMAPP is incomplete, i.e. DMAPP may fail to find a solution, even if it exists. In this paper, we present a new distributed algorithm (DiMPP) which is complete. The computer simulations performed on some well-known benchmark domains reveal that DiMPP outperforms DMAPP in the number of problem instances solved. For larger problem instances, the time taken by DiMPP is orders of magnitude less than that of some existing centralised algorithms.

ARTICLE HISTORY
Received 30 May 2016
Accepted 12 February 2017

KEYWORDS
Multi-agent path planning; decoupled approach; plan restructuring; prioritised path planning

1. Introduction

Multi-agent path planning (MAPP) is an interesting research problem in artificial intelligence. In MAPP, the objective is to find conflict free paths for all the agents. It has practical applications in several areas such as video games, inventory management, warehouse management and robotics (Bernardini, Fox, & Long, 2014; Cirillo, Pecora, Andreasson, Uras, & Koenig, 2014; Geramifard, Chubak, & Bulitko, 2006; Parker, 2008; Wurman, D'Andrea, & Mountz, 2008). Finding an optimal solution for MAPP is PSPACE-hard (Hopcroft, Schwartz, & Sharir, 1984).

MAPP approaches can be divided into coupled and decoupled approaches. In the former, a centralised planner computes a path in the joint state space of all the agents. A centralised planner can compute optimal plans (Goldenberg, Felner, Stern, Sharon, Sturtevant, Holte, & Schaeffer, 2014; Sharon, Stern, Felner, & Sturtevant, 2015a; Standley, 2010; Wagner & Choset, 2015). However, this approach becomes computationally infeasible as the number of agents increases. Decoupled approaches are more practical in use (Erdmann & Lozano-Perez, 1987; Silver, 2005).

Decoupled approaches may not be complete, albeit, scales well with the number of agents. It works in phases: (i) computing the plans of individual agents and (ii) plan coordination (restructuring). Existing decoupled approaches (Liu, Sun, & Zhu, 2014; Sharon, Stern, Felner, & Sturtevant, 2015b; Van Den Berg & Overmars, 2005; Wagner & Choset, 2015; Wilt & Botea, 2014; Yu, Peng, & Zhang, 2014) use

CONTACT Satyendra Singh Chouhan satycdec@iitr.ac.in

centralised algorithms. In Yu et al. (2014) path coordination is done based on the priority of agents. Only the priority of the agents is evaluated in a distributed manner.

Multi-agent systems are inherently distributed where agents have limited information and each agent may not know the total number of agents in the system (De Weerdt & Clement, 2009). These aspects are incorporated in DMAPP (a fully distributed multi-agent path planning algorithm) (Chouhan & Niyogi, 2015). It works in three phases: (i) individual path planning, (ii) priority decision-making and (iii) plan restructuring.

DMAPP is an incomplete algorithm. The disadvantage of an incomplete algorithm is that it may fail to find a solution even if it exists. DMAPP is incomplete because of the plan restructuring phase.

In DMAPP, only one priority order is checked in the restructuring phase to find a solution. If it cannot find a solution, it terminates and reports that solution is not found. If the agent cannot restructure its plan, we cannot conclude that a solution does not exist, since there may be some other ordering of the agents for which a solution exists. A detailed discussion of DMAPP algorithm (via example) is given in Section 4.4, where we show why it is incomplete.

Thus, in this paper we present a new distributed algorithm (DiMPP) which is complete, i.e. if there exists a solution to MAPP problem then it will definitely find it. For this, we suggest a distributed plan restructuring phase in Section 5. We provide a detailed analysis of DiMPP algorithm and also give the comparative analysis of DiMPP with DMAPP. In addition, we also present the performance evaluation of DiMPP with some existing state-of-the-art centralised path planning algorithms.

The rest of this paper is organised as follows. In Section 2 we discuss the basic definitions. In Section 3, we present the related work in MAPP. Section 4 presents the background of DMAPP algorithm and limitation of existing approach. Section 5 presents the DiMPP algorithm. The experimental results are given in Section 6. Conclusions are given in Section 7.

2. Definitions and terminologies

Definition 1: [Plan of an agent] Let Σ_i be a finite set of actions of agent i. A plan P of the agent i is a sequence of actions $P = a_1; a_2; \ldots; a_n$, where each $a_i \in \Sigma_i$.

Definition 2: [MAPP problem] In MAPP, the objective is to find a set of plans ($MA - plan$) such that, $\forall\ (i, j)$, plan of agent i is not conflicting with the plan of agent j.

Definition 3: [Input to MAPP problem] The inputs are (i) A directed graph $G(V, E)$. Vertices of the graph are possible locations of the agents and the edges are the possible transition between locations. (ii) k agents. Each agent i, has a start vertex $start_i \in V$ and a goal vertex $goal_i \in V$.

Definition 4: [MAPP solution] A set of paths for all agents i, j, $i \neq j$ such that plan of agent i (P_i) does not conflict with that of agent j (P_j).

In this paper, we use the word 'solution' that denotes MAPP solution as given in Definition 4.

Let us take an example of a four-connected grid domain as shown in Figure 1. Here, vertices are the cells in the grid (36 numbers). The possible actions are 'move' and 'wait'. An agent can move to at most four adjacent locations. In the figure, Agent 1 can move to the four adjacent locations from the start location marked s_1. Similarly, Agent 2 can move to the three possible adjacent locations from the start location marked s_2. The plans of the individual agents are shown in the figure by arrows. The cell marked by a circle is the point of conflict between the two plans. A conflict occurs since a location is occupied by more than one agent at a particular time. Another type of conflict happens when one agent tries to move from location x to location y and another agent tries to move from y to x at the same time.

2.1. Cost function

In the literature, there are various cost functions assumed in MAPP such as SIC (sum of individual cost), Makespan, Energy etc. SIC is the sum of the path lengths of the agents. Path length is the number of time steps required to reach a goal. Makespan is the maximum of the individual path length of the

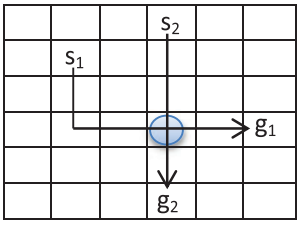


Figure 1. A conflict between plans at time step 3.

agents. Energy is the sum of the total distance travelled by all the agents. An optimal solution for MAPP would be the minimal sum of the costs.

In this paper, we use SIC as the cost function.

3. Related work

MAPP approaches can be divided into coupled and decoupled approaches. The coupled approaches use centralised algorithms. However, decoupled approaches use centralised as well as distributed algorithms. Most of the existing works in decoupled approach use centralised algorithms; very few works use distributed algorithms.

In centralised MAPP, a single agent (planner) generates plans for all the agents in the system. The planner knows the initial and the goal states of all the agents. Thus, the planner can resolve the conflicts between plans of the individual agents. In DMAPP, each agent computes its individual plan based on its initial and goal states. Then each agent communicates, by exchanging messages, with other agents to resolve conflicts between the individual plans.

3.1. Coupled approach

In this approach, a centralised planner searches in the joint state space of all the agents. For example, MA-A* (multi-agent A*) is a classical state-of-the-art MAPP algorithm that is based on the coupled approach. In MA-A*, each state in the search space is a composite state of the states of all the agents. The size of the state space is exponential in the number of agents (n). In addition, the branching factor of a given state may be exponential in n. For example, consider a state on a 4-connected grid with 10 agents. Each agent can take a maximum 4 possible 'move' actions and 1 'wait' action. The maximum number of nodes generated at the state will be 5^{10}. Consequently, to find a solution in such large search space is not practical.

In Standley (2010), an independence detection (ID) framework is introduced to reduce the number of agents with independent detection. ID tries to identify the independent groups of agents. Two groups of agents are independent if there is no conflict between the solutions of the groups. Some works related to ID are presented in Wagner and Choset (2011) and Ferner, Wagner, and Choset (2013). Another line of work is to reduce the number of nodes generated in the state space (Felner et al., 2012). In Goldenberg et al. (2014), EPEA* algorithm (an extension of A* algorithm) is proposed to circumvent the problem of generating surplus nodes (nodes which are generated but never expanded). EPEA* uses *a priori* domain knowledge to determine the surplus nodes.

3.2. Priority-based decoupled approach

Priority-based decoupled approach (Erdmann & Lozano-Perez, 1987) works in a sequential manner. Initially, the highest priority agent computes a path. Subsequently, lower priority agents compute their paths according to the paths of the higher priority agents. In Ter Mors, Witteveen, Zutt, and Kuipers (2010), a priority-based decoupled approach is proposed for route planning. It is based on free time windows, which is a time interval during which the node is not occupied by other agents

and sequentially finds the path of each agent. In Silver (2005), a Windowed Hierarchical Cooperative A* (WHCA*) algorithm is proposed. WHCA* is based on dynamic priority assignment approach where each agent gets the highest priority for a period of time. However, deciding the time period for which the agents get the highest priority is still a challenging problem.

In Sturtevant and Buro (2006), WHCA* is extended by abstracting the search space to reduce the runtime cost. In Geramifard et al. (2006), a path planning approach is proposed for real-time strategy games for eight-connected grid world. The algorithm given in Geramifard et al. (2006) first calculates the individual paths by A* algorithm; then collisions are resolved based on a biased cost function and the priority of the agents are known *a priori*.

In Wang and Botea (2009, 2011), a decoupled approach (MAPP) is proposed for multi-agent planning. Individual paths of agents are calculated along with alternate plans. A distance-based heuristic is used to determine the priority of the agents. MAPP is complete for a class of problems called SLIDEABLE.

3.3. Path coordination-based decoupled approach

In Röger and Helmert (2012), a path finding approach is suggested that can find a solution in polynomial time. In Peasgood, Clark, and McPhee (2008), a decoupled approach is suggested based on a spanning tree. It moves all the agents to the leaf nodes of the spanning tree. Then it adjusts the positions of the agents such that the paths of the agents from the root to the leaf node are collision free. However, Peasgood et al. (2008) assumes that the number of agents is less than the number of leaf nodes. Some other works based on spanning tree are given in Surynek (2009), Khorshid, Holte, and Sturtevant (2011), and Masehian and Nejad (2009). Path coordination based on the graph decomposition is suggested in (Ryan, 2006, 2007). These approaches partition the graph into special subgraphs such as cliques, singleton, and extra Hall graph. The individual paths are first computed and then these paths are coordinated on the subgraphs.

3.4. Distributed multi-agent path planning

There are very few works in distributed MAPP (Chouhan & Niyogi, 2015; Regele & Levi, 2006). In Regele and Levi (2006), a priority-based distributed algorithm is proposed for cooperative multi-agent path planning (CMAPP) that works in planning cycles. The priority of the agents dynamically changes during the planning cycles. In a planning cycle, each agent searches its path based on the current priority. If an agent cannot find a path, it raises its priority and tries to find the path again. In each cycle only one priority adjustment is possible. Whenever an agent changes its priority, it must be communicated to all the agents. This approach can lead to a deadlock. The algorithm is incomplete.

DMAPP is a recent fully distributed algorithm (Chouhan & Niyogi, 2015). It works in three phases: (i) individual path planning, (ii) priority decision-making and (iii) plan restructuring. Plan restructuring phase of DMAPP is incomplete. Hence DMAPP is incomplete in nature. A detailed discussion of DMAPP is given in Section 4.

In this paper, we propose a distributed multi-agent path planning algorithm (DiMPP) where the first two phases are as in DMAPP and suggest a new distributed plan restructuring algorithm that is complete. Thus the proposed algorithm is complete. Table 1 summarises the comparison between DiMPP and some state-of-the-art priority-based MAPP algorithms. In Table 1, CMAPP refers to the work presented in Regele and Levi (2006).

Table 1 shows that WHCA*, CMAPP and DMAPP are incomplete algorithms, i.e. these algorithms may fail to find a solution to MAPP problem even if it exists. MAPP is complete only for a class of problems called SLIDEABLE. Thus, MAPP is partially complete. However, DiMPP (proposed algorithm) is complete, i.e. if a solution of a MAPP problem exists, then it will definitely find it.

Table 1. Comparison between DiMPP and some state-of-the-art priority-based MAPP algorithms.

	WHCA* Sturtevant and Buro (2006)	MAPP Wang and Botea (2009, 2011)	CMAPP Regele and Levi (2006)	DMAPP Chouhan and Niyogi (2015)	DiMPP (Proposed in this paper)
Phase (i) Individual path planning	Centralised	Centralised	Distributed	Distributed	Distributed same as DMAPP
Phase (ii) priority decision-making	Centralised*	Centralised*	Distributed	Distributed	Distributed same as DMAPP
Phase (iii) plan coordination (restructuring)			Distributed	Distributed	**Distributed (proposed in this paper)**
Algorithm properties	Incomplete	Partially complete	Incomplete	Incomplete	**Complete**

legend *: No phase (ii) in the algorithm. Priorities of the agents are predetermined.

4. Background

In this section, we start with the introduction of DMAPP (Chouhan & Niyogi, 2015) algorithm. Next, we present the limitation of DMAPP by taking an illustrative example. DMAPP works in three phases: individual path planning, distributed decision-making and plan restructuring.

4.1. Phase (i): Path planning

In this phase, given an initial and the goal state, each agent computes its path using single agent path planning algorithm. Each agent performs an informed search from initial to goal state. DMAPP uses A* algorithm for individual path planning with Manhattan distance as a heuristic. In this phase, paths are computed by considering the static obstacles in the environment and the presence of other agents are not considered. Therefore, their individual plans may be conflicting with each other. For the details of the algorithm we refer to Chouhan and Niyogi (2015).

4.2. Phase (ii): Distributed Priority decision-making

After the first phase, each agent has its individual plan. These plans may be conflicting with each other. Therefore, the agents need to coordinate (restructure) their plans by considering the plans of the other agents. Thus the order (priority of the agents) in which the restructuring will be done has to be decided. DMAPP algorithm uses a distributed priority decision-making procedure. DMAPP makes the following assumptions:

- The network topology for massage passing is a unidirectional ring of nodes, numbered 1 to n in the clockwise direction.
- The messages can only be sent in a clockwise direction.
- Each agent have a unique identifier. Agents are associated with nodes, however they do not know their indices as well as neighbours in the node.
- All the computations/communications are done in a synchronous mode.
- The communication system is lossless.

In DMAPP, the priority of each agent is decided based on the individual plan length of the agents. Highest priority is given to the agent that has the longest plan length (Van Den Berg & Overmars, 2005).

Each agent i has a set S_i that stores the tuple $\langle Agent\_id_i, plan\_length_i \rangle$. Initially, each agent has its own tuple in the set. The set is sorted (in decreasing order) according to plan length. If two agents have the same plan length then the tie is broken by the agent ID. Here, the goal of every agent is to know the plan length of all the other agents. By this, each agent would come to know the priority of the agents for plan restructuring. The algorithm stops when each agent knows the plan length of

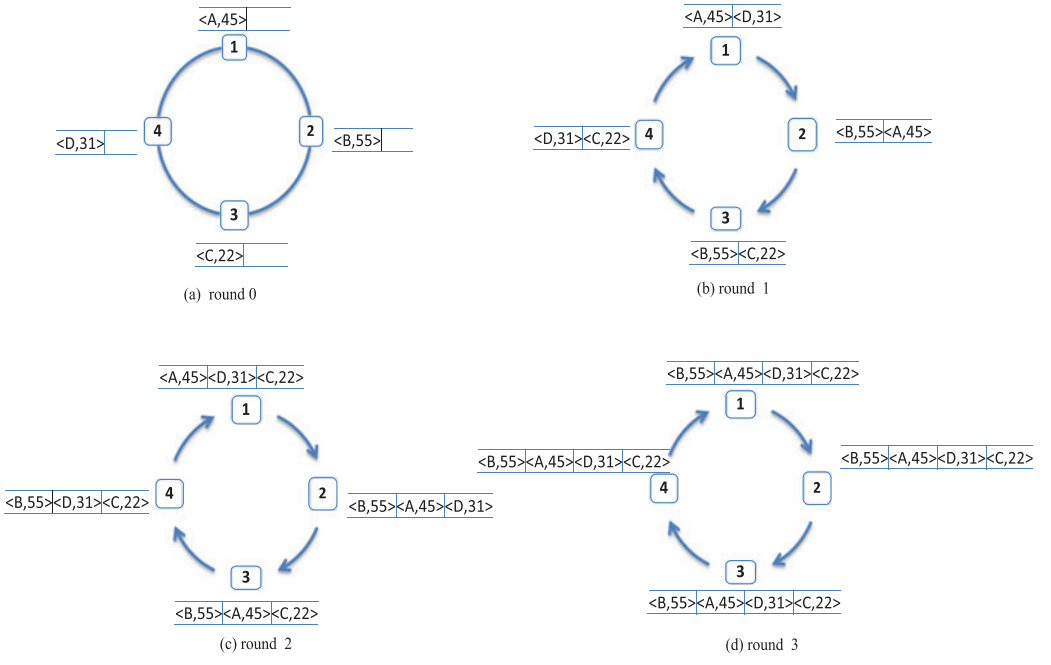


Figure 2. Distributed priority decision-making process.

every other agent. For the details of the algorithm and its properties, we refer to Chouhan and Niyogi (2015). We now consider the following example to illustrate phase (ii).

Example 1: Consider a network of four agents in a unidirectional ring, numbered 1–4 in the clockwise direction. The plan length of the agents (1,2,3,4) with agent ids (A,B,C,D) are 45, 55, 22 and 31 respectively. The distributed priority decision-making procedure is shown in Figure 2.

Here rounds are synchronous, i.e. all the agents send messages to its clockwise neighbours at the same time. For example, at time $t1$, when agent 1 sends a message to agent 2, at the same time $t1$ agent 2 sends a message to 3, agent 3 sends a message to 4, and agent 4 sends a message to 1.

Similarly, all the agents will receive messages from its anti-clockwise neighbours at the same time. For example, at time $t2$, when agent 1 receives a message from 4 then at the same time ($t2$) agent 2 receives a message from 1 and so on.

This activity (send and receive for each agent) takes place in one round.

Initially, all the agents have their own tuple in S_i (Figure 2(a)). Now, at round 1, each agent sends its tuple to next agent in the clockwise direction. Receiving agent check that the received tuple is already presented in S_i or not. If not, then it adds the received tuple (new tuple) into its own set (Figure 2(b)). After round 3, all the agents have same tuples in the set. The algorithm runs for 4 rounds (Figure 2(a)–(d)) and in the last round, each agent receives its own tuple (termination condition).

In the above example, the set $S_i = \{\langle B, 55\rangle, \langle A, 45\rangle, \langle D, 31\rangle, \langle C, 22\rangle\}$ is same for all the agents. Hence, each agent knows the priority order in which the plan restructuring (phase (iii)) will be done and the number of agents in the system. In the above example priority of the agents will be $\langle B\ A\ D\ C\rangle$.

In this way, each agent knows for the restructuring phase, who would be the communicating agents. For example, agent B (at node 1) knows that it will receive a message from agent A (node 2), then it would restructure its plan, and then send a message to agent D (node 4).

4.3. Phase (iii): Plan restructuring

After phase (ii),

- Each agent knows the total number of agents.

- Every agent knows the priority order (of agents) in which restructuring will be done.

The plan restructuring phase starts from the highest priority agent and goes up to the lowest priority agent. Plan restructuring is done in the following sequence.

- Initially the highest priority agent triggers the plan restructuring phase by updating the *MA*−*plan* with its own plan and sends the *MA* − *plan* to the next lower priority agent.
- Intermediate agents receive the *MA* − *plan* from the higher priority agents and checks if there is any conflict between the plans. If there is any conflict, it recursively tries to solve all the conflicts between plans. If it succeeds, it sends the *MA* − *plan* to the next agent in the priority order. Otherwise, it sends a failure signal to all the agents.
- Lowest priority agent restructures its plan according to *MA* − *plan*. If it succeeds, it sends an 'ok' message to all the agents. Otherwise, it sends a 'failure' message.

4.3.1. Handling conflicts

Whenever a conflict is detected between the paths of different agents, an agent does not recompute the whole path again. Rather, it recomputes the affected part of the plan and merges it into the original plan. Figure 3 shows the possible conflicts in a grid map. There are two types of conflicts that can occur between the paths of agents: (i) vertex conflict and (ii) edge conflict. Most of the vertex conflicts can be handled using wait action (Figure 3(a)). However, there may be a case where a plan cannot be restructured by introducing wait action (Figure 3(b)). In this case, the part of the plan, where conflict occurs is recomputed and merged into the original plan (Figure 3(c)). Edge conflict is handled by recomputing the affected part of the plan and merging into the original plan.

Restructuring is done by only one agent at a time. Therefore, unlike existing algorithms such as CBS (Sharon et al., 2015a), DMAPP algorithm is not exponential with respect to the number of conflicts between paths. For the details of the restructuring phase of DMAPP, we refer to Chouhan and Niyogi (2015).

4.4. Limitation of DMAPP

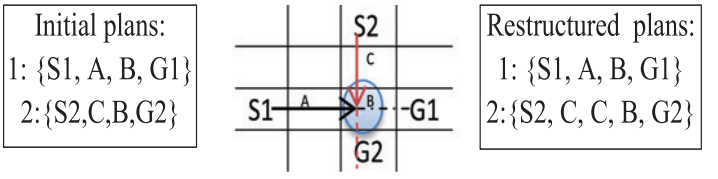
The plan restructuring phase of DMAPP is incomplete, i.e. DMAPP may fail to find a solution (Definition 4) even if it exists. This is because DMAPP considers one priority order and tries to restructure the plans. If it cannot find a solution, we cannot conclude that a solution does not exist since there may be other priority orders where a solution may be found.

We take different ways of a assigning priority (e.g. priority based on the index of an agent and priority based on plan length of an agent) and show that DMAPP is incomplete.

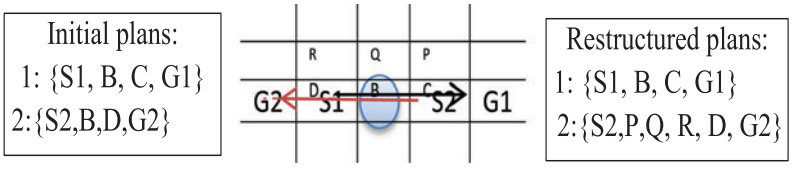
Example 2: Priority based on index of an agent consider a four-connected grid domain shown in Figure 4(a). There are three agents *A*, *B* and *C*. Assume that the priorities of the agents are based on their indices, i.e. *A* is the highest priority agent and *C* is the lowest priority agent. Initial and final position of the agents are shown in Figure 4(a). Initially, all the agents calculate their individual paths (Figure 5). In the paths, a particular position of an agent is indicated by the (x, y) coordinate. Here, x and y are the row and column numbers respectively.

Now from Figure 5, it is clear that paths of *B* and *C* are conflicting at time step 2, 3 and 4. The priority order of the agents is $\langle A\ B\ C \rangle$, i.e. *A* has the highest priority and *C* has the lowest priority. Therefore, *C* will try to restructure its path according to the paths of *A* and *B*. We assume that once an agent reaches the goal it remains there. A possible restructuring would be as follows.

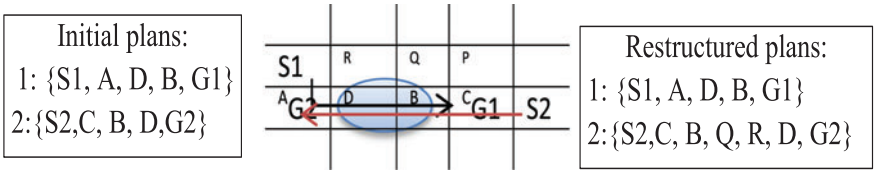
- At time step 2, Agent *B* will be at (2, 3) therefore agent *C* cannot move to (2, 3). From Figure 6(a) it is clear that, agent *C* has only one option to go back to the position (1, 4) at time step 2.
- At time step 3, *C* moves to position (1, 3) (see Figure 6(b)).
- At time step 4, *C* moves to (2, 3) and B reaches its goal (4, 3). Now *B* will be at the goal state (4, 3). From Figure 6(c) and (d) it is clear that Agent *C* cannot reach its goal, because *B* blocks the only possible path for *C*. Hence, plan restructuring fails.



a) Vertex conflict (case 1)



b) Vertex conflict (case 2)



c) Edge conflicts

Figure 3. Different cases of conflict between paths (illustrated via grid map). Here S_i and G_i are start and the goal positions respectively.

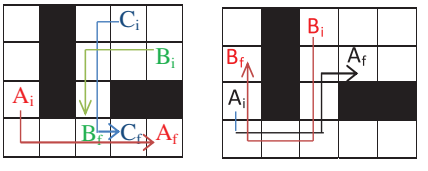


Figure 4. A simple 4-connected grid domain. Agent's initial and final position are shown by X_i and X_f, respectively (where X can be any agent, i.e. A, B, or C). Arrows indicate the initial individual plans of each agent irrespective of other agents.

Time Steps

Agent	0	1	2	3	4	5
A	(3,1)	(4,1)	(4,2)	(4,3)	(4,4)	**(4,5)**
B	(2,5)	(2,4)	(2,3)	(3,3)	**(4,3)**	
C	(1,4)	(1,3)	(2,3)	(3,3)	(4,3)	**(4,4)**

Figure 5. Initial plans of agents A, B, and C.

However, there exist conflict free paths for this situation.

Example 3: Priority based on plan length (DMAPP) considers the grid world domain given in Figure 4(b). Now, assume that there are two agents A and B with initial and final positions as shown in Figure 4 (b). The priority order of the agents is based on individual plan lengths (as suggested in DMAPP). In phase (i) of DMAPP, individual plans are computed. The plan length of A and B are 6 and 7, respectively, (Figure 4(b)). Now, according to DMAPP algorithm, B will send its plan to A. Since

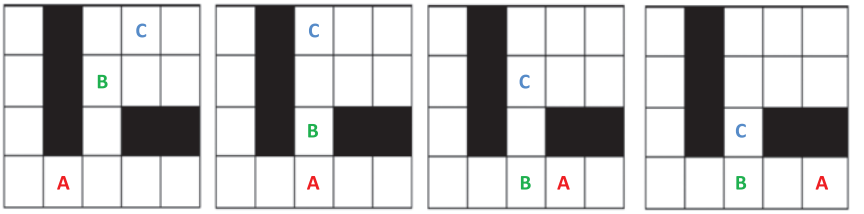


Figure 6. Movement of *A*, *B*, and *C* at each time step.

both plans are conflicting, agent *A* tries to restructure its plan with respect to the plan of *B*. The plan restructuring phase of DMAPP fails because *B*'s plan blocks the movement of *A*. However, there exist conflict free plans for this situation.

5. Proposed work

In this section we present the DiMPP algorithm. First, we present the motivation behind the proposed algorithm. Next, we present the DiMPP with theoretical analysis.

5.1. Motivation for DiMPP

In Example 2 (Section 4.4), all agents could have achieved their goals with the priority order $\langle A\ C\ B\rangle$. Similarly, in Example 3 (Section 4.4), both agents could achieve their goals if *A* has higher priority than *B*.

5.1.1. Completeness issue

A naive way to ensure completeness would be to check each priority order. In the above example, there would be six possible priority orders $\langle A\ B\ C\rangle$, $\langle A\ C\ B\rangle$, $\langle B\ C\ A\rangle$, $\langle B\ A\ C\rangle$, $\langle C\ A\ B\rangle$ and $\langle C\ B\ A\rangle$. If there are *n* number of agents then there would be *n*! possible priority orders. Now, checking *n*! priority orders would be computationally infeasible as the number of agents increases.

Let us consider Example 2 given in Section 4.4. The initial priority order of the agents is $\langle A\ B\ C\rangle$. However, plan restructuring fails because *C* is unable to find a plan with respect to the plan of *B*. Now let us take an another priority order $\langle B\ A\ C\rangle$ where *B* has higher priority than *C* (same as in Example 2). In this case, *B* will send its plan to *A*. *A* will check and restructure its plan (if necessary). Similar to earlier case, here also the plan of *B* remains the same. Next, *C* will try to change its plan according to the plans of *B* and *A*. But, *C* will again fail to find a plan because of agent *B*.

Thereby, it is obvious that there is no need to check $\langle B\ A\ C\rangle$ if we have already checked $\langle A\ B\ C\rangle$, since in both the cases the precedence order of *B* and *C* remains same. Based on the above observation we can reduce the number of checks to cover all the *n*! combinations. Now consider an example to see how many priority orders we need to check to cover all the *n*! priority order combinations.

Example 4: Consider four agents *A*, *B*, *C*, and *D* in the MAPP problem. The underlying network topology for message passing is a unidirectional ring consisting of *n* nodes, numbered 1 to *n* in the clockwise direction. Counting is modulo *n*. Agents are arranged in the alphabetical order, i.e. agent *A* is at node 1 and agent *D* is at the last node. Messages can only be sent to clockwise neighbour (Figure 7). Now, there are only four possibilities of the priority orders: $\langle A\ B\ C\ D\rangle$, $\langle B\ C\ D\ A\rangle$, $\langle C\ D\ A\ B\rangle$, and $\langle D\ A\ B\ C\rangle$ in this topology (only clockwise communication is possible).

However, the possible combination of the priority orders with *A*, *B*, *C* and *D* are 24 (shown in Figure 8). Now, we show that the given four possible orders are sufficient to cover all the 24 possible priority orders.

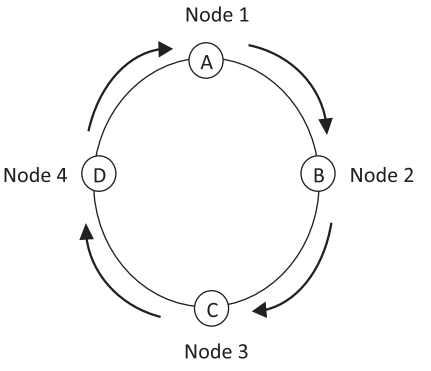


Figure 7. A unidirectional ring network of four agents.

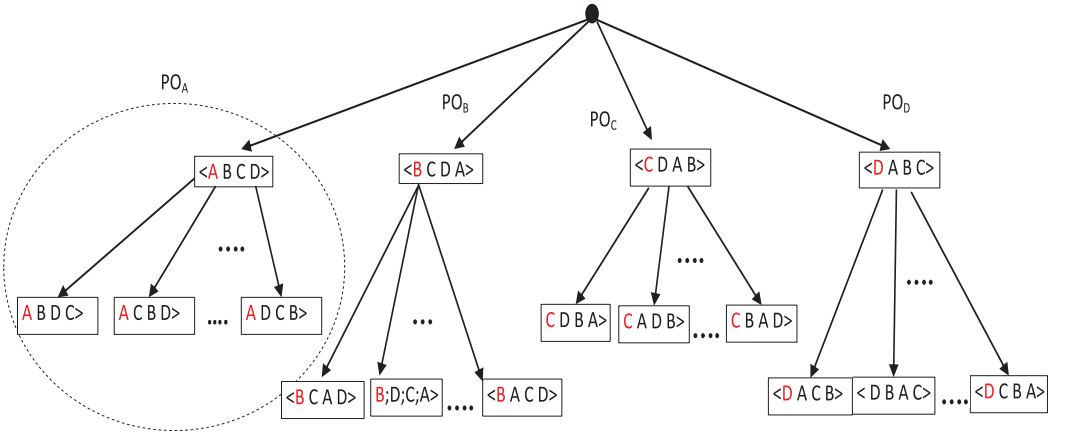


Figure 8. The possible priority orders for four agents.

In Figure 8, PO_A represents the possible priority orders with A as the initiator agent. These are $\langle A\ B\ C\ D\rangle$, $\langle A\ B\ D\ C\rangle$, $\langle A\ C\ B\ D\rangle$, $\langle A\ C\ D\ B\rangle$, $\langle A\ D\ B\ C\rangle$, and $\langle A\ D\ C\ B\rangle$. Similarly for agent B, C and D as initiator, there would be also six possible priority orders.

Assume that A is the initiator agent and the priority order $\langle A\ B\ C\ D\rangle$ fails to give a solution ($MA-plan$). Without loss of generality, assume that agent D cannot restructure its plan w.r.t to the plan of higher priority agents. Then the following cases are possible.

Case 1 Plan of agent D is conflicting with the plan of initiator agent (i.e. A), such that D fails to restructure its plan w.r.t to the plan of A.
Case 2 Plan of D is conflicting with the plan of a higher priority agent except initiator agent, i.e. B or C.
Case 3 Agent D fails to restructure its plan w.r.t to the plan of all the higher priority agents together.

Action for Case 1: The given priority order $\langle A\ B\ C\ D\rangle$ fails, therefore, all the priority orders (having A as initiator agent) will also fail because the plan of agent A will remain same in all the combinations. Thus agent D will never be able to restructure its plan. Therefore, we need not check all the remaining priority orders ($\langle A\ B\ D\ C\rangle$, $\langle A\ C\ B\ D\rangle$, $\langle A\ C\ D\ B\rangle$, $\langle A\ D\ B\ C\rangle$, and $\langle A\ D\ C\ B\rangle$).
Action for Case 2: In case 2, assume that agent B's plan is conflicting with the plan of D. In this case, all the priority orders where B has higher priority than D such as: $\langle A\ B\ D\ C\rangle$ and $\langle A\ C\ B\ D\rangle$ will also fail (as in Case 1). Here, $\langle A\ D\ B\ C\rangle$, $\langle A\ D\ C\ B\rangle$, and $\langle A\ C\ D\ B\rangle$ may give the solution. However, we can ignore them because they will be checked in the other priority orders with different initiator agent (given in Example 4).

Consider the priority order $\langle A\ D\ B\ C\rangle$. It is equivalent to the priority order $\langle D\ A\ B\ C\rangle$ (with D as the initiator agent) because here also D has higher priority than B and this order can give a solution. Similarly, the other two priority orders $\langle A\ D\ C\ B\rangle$ and $\langle A\ C\ D\ B\rangle$ will also be covered by the priority orders having C and D as the initiators. Thus if $\langle A\ B\ C\ D\rangle$ fails we need not check any of the priority orders having A as the initiator agent.

Action for Case 3: In case 3, higher priority agents A, B, and C are together creating conflict for D such that agent D cannot restructure its plan. However, the plan of each agent (A, B, C) may not be conflicting with the plan of D. Here $\langle A\ C\ B\ D\rangle$ will definitely fail, since A, C, B precede D. The priority orders $\langle A\ B\ D\ C\rangle$, $\langle A\ D\ B\ C\rangle$, $\langle A\ C\ D\ B\rangle$, and $\langle A\ C\ B\ D\rangle$ may give a solution. These orders can be covered by checking the priority orders, $\langle B\ C\ D\ A\rangle$, $\langle C\ D\ A\ B\rangle$, and $\langle D\ A\ B\ C\rangle$ (as in Case 2).

Thus, we have observed that if the priority order $\langle A\ B\ C\ D\rangle$ fails to give a solution, we need not check all the priority orders where agent A is the initiator agent. The same is true for all the priority orders with B, C, and D as initiator agents. Therefore, we need to check only 4 priority orders that start with different initiator agents. For example, $\langle A\ B\ C\ D\rangle$, $\langle B\ C\ D\ A\rangle$, $\langle C\ D\ A\ B\rangle$, and $\langle D\ A\ B\ C\rangle$.

In general, if there are n agents, we need to check n priority orders having different initiator agents placed in a ring topology to cover all the $n!$ priority order combinations.

Proposition 5.1: *Given n agents (in a unidirectional ring topology), arranged in the ring such that A_1 is at node 1, A_2 is at node 2, . . . , and A_n is at node n. Then, n possible priority orders are:*

$$\langle A_1\ A_2\ A_3 \ldots A_{n-1}\ A_n\rangle \quad (1)$$

$$\langle A_2\ A_3\ A_4 \ldots A_n\ A_1\rangle$$

$$\vdots \quad (2)$$

$$\langle A_n\ A_1\ A_2 \ldots A_{n-2}\ A_{n-1}\rangle \quad (n)$$

In order to find whether a MAPP solution exists, it suffices to check n priority orders given in (1) to (n).

***Proof*:** Assume that priority order, given in (1), fails to give a solution. Then the following cases will occur.

Case 1: Suppose that there exists an agent A_i, where $i \in \{2 \ldots n\}$ such that plan of agent A_i is conflicting with the plan of initiator agent(A_1) and A_i fails to restructure its plan.

Since A_1 is the initiator agent, therefore, plan of A_1 will not be changed by any agent that precedes A_i and follows A_1. Thus we need not check all the remaining $(n-1)!$ priority orders with A_1 as the initiator agent.

Case 2: Suppose that there exists an agent A_i, where $i \in \{2 \ldots n\}$ such that plan of agent A_i is conflicting with the plan of a higher priority agent A_j but not the initiator (A_1). In this case, those priority orders where A_j precede A_i will also fail (as in Case 1). The remaining priority orders where A_i precede A_j will be covered in (2) to (n). Thus we need not check all the priority orders with A_1 as the initiator agent.

Case 3: Suppose that there exists an agent A_i, where $i \in \{2 \ldots n\}$ such that plan of agent A_i is conflicting with the plans of all the higher priority agents combined together. In this case, the priority orders where all the higher priority agents precede A_i will also fail (as in Case 1). The remaining priority orders will be covered in (2) to (n). Thus we need not check all the priority orders with A_1 as initiator agent.

In all the three cases, if priority order (1) fails to give a solution then we can discard the remaining $(n-1)!$ priority orders with A_1 as the initiator. The same argument holds for other priority orders (2) to (n). Hence, the priority orders (1) to (n) each with different initiator agent are sufficient to cover all the $n!$ possible priority orders. □

5.2. DiMPP algorithm

In this algorithm, the phase (i) and phase (ii) are same as in DMAPP (Section 3). Recall that in phase (i) of DMAPP, each agent computes its individual path. In phase (ii), distributed priority decision-making

is done. After phase (ii), each agent knows the number of agents in the system and an initial priority order of the agents for phase (iii). In the proposed work if the number of agents in the system is known *a priori*, phase (ii) is not required; phase (iii) of DiMPP can start with any agent as the initiator. However, we assume that the number of agents are not known *a priori*.

In phase (iii) the underlying network topology for message passing is a unidirectional ring consisting of n nodes, numbered 1 to n in the clockwise direction. Counting is modulo n. Agents are ordered according to their priority, i.e. agent at node 1 is the highest priority agent and the agent at node n is the lowest priority agent. The data structures used in Algorithm 1 (phase (iii)) are shown in Table 2.

Agent_list[*i*] is set to either 0 or 1. Initially set to 0. The value of *Agent_list*[*i*] indicates that the agent at node *i* was an initiator agent or not (in the plan restructuring phase).

Algorithm 1: Phase (iii) plan restructuring phase of DiMPP

```
1  send function for agent j (send_msg)
2  if status = success or failure then
3  |  send status to all the agents and terminate;
4  else
5  |  send msg⟨agent_list, MA − plan⟩ to its clockwise neighbour k;
6  end
```

```
1  receive function for agent k
2  if status = success or failure then
3  |  terminate;
4  end
5  if MA − plan contains k's plan then
6  |  status := success;
7  else
8  |  if MA − plan ≠ ∅ then
9  |  |  if CONFLICT(P_k, MA − plan) then
10 |  |  |  P'_k := RESTRUCTURE-PLAN (P_k , MA − Plan)
11 |  |  |  if P'_k ≠ null then
12 |  |  |  |  update MA − plan with P'_k;
13 |  |  |  else if agent_list[k] = 0 then
14 |  |  |  |  Init_agent := k; agent_list[Init_agent] := 1;
15 |  |  |  |  MA − plan := ∅;
16 |  |  |  |  update MA − plan with P_k;
17 |  |  |  else if there exists an agent l s.t. agent_list[l] = 0 then
18 |  |  |  |  MA − plan := ∅;
19 |  |  |  else
20 |  |  |  |  status := failure;
21 |  |  |  end
22 |  |  else
23 |  |  |  update MA − plan with P_k;
24 |  |  end
25 |  else
26 |  |  Init_agent := k;
27 |  |  update MA − plan with P_k;
28 |  end
29 end
```

Algorithm description. Let A_1 be the initiator agent for starting the planning restructuring phase without loss of generality. However, any agent can be the initiator. There are two functions used in Algorithm 1: **CONFLICT (**P_k**,** *MA* − *plan*) and **RESTRUCTURE-PLAN (**P_k**,** *MA* − *Plan*)**. The CONFLICT(.) function, checks the conflicts (Section 4.3.1) between received *MA* − *plan* and plan of agent k (P_k). It returns true if conflicts are found otherwise returns false. RESTRUCTURE-PLAN(.) function, tries to

Table 2. Data structures and variables used in DiMPP.

Notation	Meaning
agent_list[1 . . . *n*]	An array list of size *n*: *Agent_list*[*i*] is either 0 or 1 and *i* is the node number in the ring
msg	Token message to send in the ring: it contains *MA* − *plan* and *agent_list*
Init_agent	Initiator agent: Highest priority agent that will trigger the plan restructuring phase
MA − *Plan*[1 . . . *n*][1 . . . *m*]	A cell (*i*, *k*) of *MA* − *Plan* contains the *k*th action of agent *i*
Node_ID	An integer representing the node number in the ring

resolve all the conflicts between plan P_k and *MA* − *plan* (Section 4.3.1). It recursively modifies the plan P_k until all the conflicts are resolved. If it succeeds then, it will return the modified plan P'_k otherwise it will return null. Now Algorithm 1 works as follows.

Initiator agent triggers the plan restructuring phase by updating *MA* − *plan* with its own plan and set it's *Node_ID* in the *agent_list* to 1. Thereafter, it sends a message to its neighbour. When an agent receives the message then the following cases can occur:

(1) *If MA-plan contains receiving agent's plan* (receive function, step 6). It means that a solution has been found.
(2) *If MA-plan is empty* (receive function, steps 26 and 27). The receiving agent makes itself the initiator and updates *MA* − *plan* with its own plan.
(3.1) *If MA-plan is not empty* (receive function, step 8). The receiving agent's plan is not conflicting with the *MA* − *plan* (receive function, step 23). The receiving agent updates *MA* − *plan* with its own plan.
(3.2) *If MA-plan is not empty* (receive function, step 8). If the receiving agent is able to restructure its plan (receive function, step 12). The receiving agent updates *MA* − *plan* with its restructured plan.
(3.3) *If MA-plan is not empty* (receive function, step 8). If the receiving agent is not able to restructure its plan and it has not been the initiator (receive function, steps 14 and 15). The receiving agent makes itself the initiator, first resets *MA* − *plan* and then updates *MA* − *plan* with its own plan.
(3.4) *If MA-plan is not empty* (receive function, step 8). If the receiving agent is not able to restructure its plan and it has already been the initiator and there exists an agent who has not been the initiator (receive function, step 18). The receiving agent resets *MA* − *plan*.
(3.5) *If MA-plan is not empty* (receive function, step 8). If the receiving agent is not able to restructure its plan and all the agents have been the initiator (receive function, step 20). It means that there does not exist a solution.
(4) *If status is success or failure* (receive function, step 3). The receiving agent terminates.

5.2.1. Properties of DiMPP algorithm

Lemma 5.2: *Plan restructuring algorithm of DiMPP alway terminates.*

***Proof*:** In Algorithm 1, there are two cases when algorithm terminates.
Case 1: (success) when agent receives its own plan (Algorithm 1 (receive function, steps 5 and 6)). It means that *MA* − *plan* is successfully updated by all the agents ($n - 1$) and the message contains the *MA* − *plan* is received by the initiator. In the receive function each instruction takes finite amount of time. In addition, agent receives its plan in a finite number of round (i.e. $n - 1$). Thus, in case of success, plan restructuring goes for at most *n* iteration for a priority order. Finally, receiving agent sends the success message to all the agents and plan restructuring process is terminated (Algorithm 1 (send function, steps 2 and 3)).
Case 2: (failure) Algorithm maintains a data structure called *Agent_list*. The value of *Agent_list*[*k*] indicates that the agent at node *k* was an initiator agent or not (Algorithm 1 (receive function, steps 13–18)). If *Agent_list*[*i*] =1 for all the agents then, it means that every agent acted as the initiator and solution is not found. Now, the size of the *Agent_list* is *n* (finite). Each agent can be the initiator agent

for at most one time. In worst case, for each priority order (out of n possible orders) algorithm takes at most n iteration to find out that plan restructuring fails. Therefore, in case of failure, algorithm returns fails after at most n^2 iterations ((Algorithm 1 (receive function, steps 19 and 20))).

Hence, the Algorithm 1 will always terminate. □

Lemma 5.3: *MA-plan obtained by the restructuring phase is correct.*

***Proof*:** For $n = 1$ (basis of induction), $MA - plan$ contains only one plan from highest priority agents. Since there is only one plan, therefore it is coordinated and valid plan. Let the lemma be true for the *ith* agent in the priority. To prove that if the $(i + 1)$th agent in the priority finds a plan then it is correct. For this, we need the following cases.

Case 1: The plan of the $(i + 1)$th agent in the priority is not conflicting with the remaining plans. In this case, no restructuring is to be done. So the overall plan is correct.

Case 2: The plan of the $(i + 1)$th agent in the priority is conflicting with the remaining plans. The single agent path planning algorithm(A^*) computes a conflict free plan by looking at all the other plans and static obstacles. Thus the, overall plan is correct. However, if the restructuring phase obtains a plan it is correct. □

Completeness of DiMPP algorithm is determined by phase (i) and phase (iii). Phase (i) is complete since it uses A* algorithm which is known to be complete. Now, the completeness of the phase (iii) is given as follows.

Lemma 5.4: *(Completeness). If there exist a solution for MAPP then Algorithm 1 will find it.*

***Proof*:** The algorithm assumes that the agents are in a unidirectional ring topology, arranged in the ring such that A_1 is at node 1, A_2 is at node 2, . . . , and A_n is at node n.

From the receive function in Algorithm 1, it follows that each agent can be the initiator exactly once (Algorithm 1 (receive function, steps 13 and 14)). Thus the algorithm will consider only n priority orders as given in Proposition 5.1. From Proposition 5.1, we obtain that, in order to find whether a MAPP solution exists, it suffices to check n priority orders. Thus if the algorithm cannot find a solution it implies that there does not exist a MAPP solution.

Thus phase (iii) is complete. Hence DiMPP is complete as well. □

Lemma 5.5: *The time complexity of Algorithm 1 is* $O(n^2T)$.

***Proof*:** In Algorithm 1, every agent can be an initiator (receive function, steps 13–18). Thus there can be maximum n initiators. Now for each initiator agent there can be a maximum of $(n - 1)$ plan restructuring (receive function, steps 9 and 10). Each restructuring takes $O(T)$ time by A* algorithm (receive function, steps 10). Hence the total time will be $n(n - 1)T$, i.e. $O(n^2T)$. □

Lemma 5.6: *The message complexity of Algorithm 1 is* $n(n - 1)$.

***Proof*:** In Algorithm 1, for an initial priority order, at most $(n - 1)$ messages will be sent between the agents. Algorithm 1 will check at most n priority orders (receive function steps 13–18). Thus the total number of messages sent will be $n(n - 1)$. □

6. Implementation details

In order to evaluate the performance of DiMPP, we have performed experiments on three benchmark path planning domains, namely, (1) brc202d (2) den520d and (3) ost003d from game Dragon Age: origins (DAO) (Figure 9) (Sturtevant, 2012). These maps are available as a benchmark planning domain for MAPP (Sturtevant, 2012). We have also performed experiments on a 8×8 4-connected grid domain. This section is organised as follows. First, we provide some information on the DiMPP implementation and experimental settings. Next, we show the comparison of DiMPP with existing algorithms in a grid domain. Then, we show the performance of DiMPP on DAO domains.

(a) (b) (c)

Figure 9. DOA MAPS: (a) brc202d, (b) den520d, (c) ost003d.

6.1. Implementation and experimental set-up

There are different variants of MAPP problem. In Sharon et al. (2015a), the following assumptions are made:

(1) All the agents can simultaneously perform the '*move*' and '*wait*' actions.
(2) Agents can follow each other.
(3) Agents never disappear from the goal state.

In order to compare the performance of DiMPP with the existing algorithms we also make the above assumptions (Sharon et al., 2015a).

In phase (i) of DiMPP, each agent computes a path corresponding to its initial-goal states. Each agent uses the A* algorithm with shortest distance heuristics, i.e. Manhattan distance. In Sharon et al. (2015a), an admissible heuristic SIC (sum of individual cost) is used for MA-A*, EPEA*, and CBS algorithms. SIC is the sum of all the Manhattan distance heuristic of all the agents. For example, at a particular state, if the Manhattan distance to the goal of an agent 'A_1' is 4, 'A_2' is 5 then the total SIC value will be $4+5=9$. Therefore, SIC value of an agent is same as the Manhattan distance.

We have implemented the DiMPP algorithm in JAVA using NetBeans 8.0.2 IDE. Experiments were performed on a workstation with 2.53GHz processor with 4 GB of RAM. Simulation of the overall implementation was done using Java threads (*Thread Class*). Each agent (thread) runs the search algorithm to compute its plan (Phase 1). We have used an inbuilt semaphore class of JAVA (*java.util.concurrent.Semaphore*) to ensure synchronisation between agents (Phase 2). Synchronisation between all the three phases was done using valid checkpoints. The experiments were run 10 times for each instance and the average running time is computed.

6.2. Comparison with centralised algorithms

In this section, we first we discuss the performance of DiMPP on 8×8 four-connected grid domain. Next we present the results on DAO maps.

6.2.1. 8 × 8 4-connected grid domain

In 8×8 4-connected grid domain, we performed the experiments on same problem instances for all the algorithms (DiMPP, MA-A*, EPEA* and CBS). We give the results with the number of agents (n) ranging from 3 to 15. The time limit is set to five minutes for the all algorithms. Table 3 shows the performance of the algorithms. In the table, some values of the results are rounded off, thus it is shown

Table 3. Evaluation of DiMPP on 8 × 8 grid domain.

	DiMPP		MA-A*		EPEA*		CBS	
#agents	#node	run time	#node	runtime	#node	runtime	#node	runtime
3	98	1.6	513	3.6	**15**	**0.6**	500	7
4	148	2.3	4650	74.3	**25**	**1**	1072	14
5	183	3.4	24,844	1307.7	**35**	**1**	2436	32
6	212	3.4	165,390	410,68	**39**	**4**	1399	20
7	**291**	**7.523**	–	–	88	15	4111	60
8	**263**	**5.476**	–	–	293	70	8910	148
9	**299**	**6.167**	–	–	1053	>400	>40,000	879
10	**376**	**18.751**	–	–	2373	>1000	>110,000	2429
11	**415**	**17.555**	–	–	7923	>8000	>330,000	7712
12	**490**	**18.515**	–	–	13,178	>10,000	>460,000	12,363
13	**590**	**46.868**	–	–	14,989	>15,000	>560,000	16,481
14	**605**	**50.042**	–	–	13,897	>15,000	>750,000	24,441
15	**670**	**55.001**	–	–	22,967	>30,000	>850,000	30,509

'-' Denotes problem instances not solved within the time limits.
The best results achieved by the algorithms are marked as bold.

Table 4. Evaluation of DiMPP on DOA maps.

	DiMPP		EPEA*		CBS	
#agents	Cost	Runtime (ms)	Cost	Runtime (ms)	Cost	Runtime (ms)
brc202d map						
5	666.6	1712.8	705.07	7320	666.7	**1664**
10	1287.2	4664.7	1269.88	22,554	1286.5	**5318**
15	1868.5	8773.4	1866.6	>47,000	1866.6	**8681**
20	2628.9	**13,969**	2799	>116,600	2625.8	>30,000
25	3515	**18,372.8**	3439.4	>195,260	3439.4	>66,000
30	3732	**21402.1**	3725.9	>254,000	3725.9	>89,000
35	4492.8	**26,168**	4078.75	>455,000	4449.72	>150,000
den520d map						
5	677.12	447.18	701.86	223	681.1	**218**
10	1303.2	1018	1269.36	1099	1269.36	**555**
15	1857.8	1314.09	1895.81	>1100	1895.81	**1672**
20	2757.7	3208.2	2674.25	>4700	2674.25	**1708**
25	3583	2953.63	3546.22	>7600	3483.8	3046
30	4079	**3204.55**	4208	>62,000	3947.33	7754
35	4947.25	**3734.5**	4975.4	>65,000	4886.9	>40,000
40	5665.3	**4408.6**	5664.1	>81,000	5664.16	>45,000
ost003d map						
5	529	**145.55**	542.6	470	542.6	220
10	1113.4	**345.2**	1110.4	>8100	1110.4	935
15	1669	**518.88**	1612.1	>8900	1612.1	>1400
20	2141.55	**658.11**	2180.444444	>29,500	2180.44	>3000
25	2777.71	**946.01**	2741.444444	>122,000	2741.44	>59,000
30	3212.55	**850.88**	3172.857143	>162,200	3172.85	>69,000
35	3617	**1016.8**	4172.2	>73,085	3948.77	>73,000

The best results achieved by the algorithms are marked as bold.

with greater than '>' notation. The notation '>' implies that the actual results' values are greater than the value shown in Table 3. The following observations are made from the results.

(1) When $n \leq 6$, EPEA* outperforms the other algorithms. The number of nodes generated by EPEA* is very less compared to DiMPP, MA-A*, and CBS.
(2) When $n \geq 7$, the state space of MA-A* increases exponentially and fails to solve the instances within the time limits. Similarly, the state space of EPEA* and CBS also increases exponentially. DiMPP, on the other hand scales well as the number of agents increases.

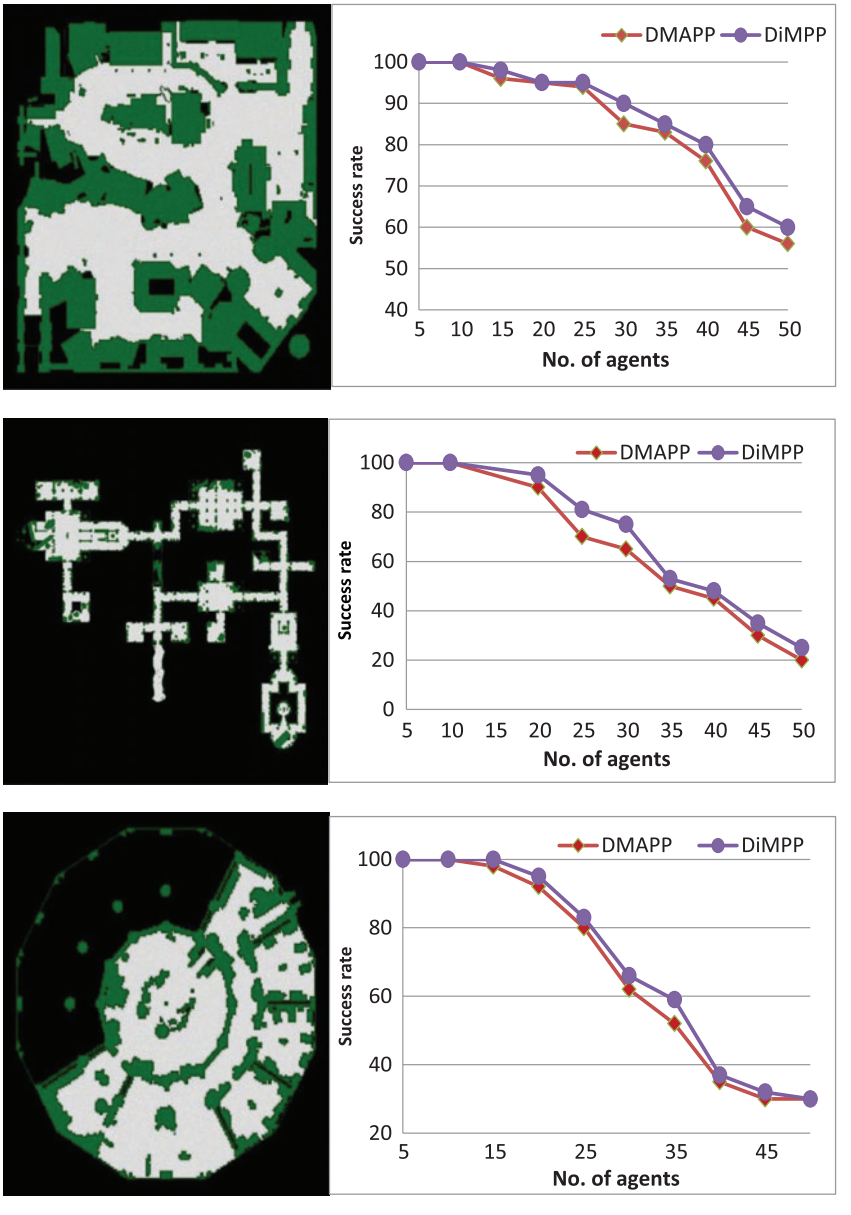


Figure 10. Success rate of DiMPP and DMAPP for different DAO maps.

Explanation

DiMPP computes the individual paths in the local state space of agents. Whereas, other algorithms (except CBS) maintains a joint state space. Though CBS computes the individual paths at the low level, but at the high level it explores every possible combination of constraints to resolve the conflicts between agents. Due to this, the state space of CBS increases exponentially with the increase in the number of agents (and conflicts).

In DiMPP, each agent uses the A* algorithm and there is an additional overhead of communication time. Thus, for problems of small size, time taken and the number of nodes expanded by DiMPP are higher than that by EPEA*. However, as the number of agents increases (6 onwards), the joint state space of EPEA* increases exponentially whereas DiMPP increases polynomially. Specifically,

- When $n = 6$, runtime of EPEA* is for 39 nodes. Whereas in DiMPP, runtime is proportional to #node/n which equals to 212/6 ($\approx$35) nodes. Thus, when $n = 6$, the corresponding runtimes are 3.4 s (for $\approx$35 nodes in DiMPP) and 4 s (for 39 nodes in EPEA*).
- When $n = 7$, the number of nodes in DiMPP is higher than that of EPEA* because the joint state space of EPEA* is still smaller than the total number of nodes from the state space of all the agents in DiMPP. However, as the number of agents increases, state space of EPEA* increases exponentially.

The results of Table 3 is important because it shows that as the size of problem increase, a distributed algorithm performs better than a centralised algorithm.

6.2.2. Experiments on DAO maps

We have performed experiments on three benchmark problem from the dragon Age: Origins game. Table 4 shows the performance of DiMPP on brc202d, den520d and ost003 maps. Since the MA-A* cannot handle the problem instances when the number of agents exceeds six, so we give the comparison of DiMPP with EPEA* and CBS.

Results show (Table 4) that for small size problems runtime of DiMPP is higher than EPEA* and CBS. It is because of the extra overhead of priority decision-making process and having three phases in the algorithm. However, as the problem size increases (number of agents) the runtime of EPEA* and CBS increases exponentially with the number of agents. The runtime increases linearly with the number of agents in DiMPP.

6.3. Comparative analysis of DiMPP with DMAPP

We have performed experiments on three benchmark problem from the dragon Age: Origins game. We have performed 1000 experiments on instances of each DAO maps and have compared the success rate of DMAPP and DiMPP. We set a time limit of five minutes to solve a problem instance.

Figure 10 shows the number of problems solved (in percentage) on brc202d, den520d and ost003d maps. All three maps are different in terms of number of obstacles, bottlenecks and open spaces. In den 520d, there are large open spaces and no bottlenecks. In brc202d, there are very few open spaces and many narrow corridors. In ost003d maps, there are few bottlenecks and few open spaces. Results show that success rate of DiMPP is higher than DMAPP.

7. Conclusion

In this paper, we have developed a novel distributed and complete multi-agent path planning algorithm (DiMPP) that uses a priority-based decoupled approach. Our result is significant since most priority-based decoupled approaches, in the literature, use centralised algorithms and are incomplete. The novelty of our algorithm in terms of completeness comes from the strategy adopted in the restructuring phase (Algorithm 1) which considers n priority orders (the total number of possible priority orders is $n!$). We have proved that n priority orders suffice to determine whether or not a solution exists.

We have validated our algorithm using several benchmark domains. The experimental results reveal that DiMPP is able to solve more problem instances then DMAPP. We have also compared the performance of DiMPP with some existing centralised algorithms. The experimental results show that for large problem instances involving several agents, the runtime of DiMPP is orders of magnitude less than that of the other algorithms.

Disclosure statement

The authors declare that they have no conflict of interest.

References

Bernardini, S., Fox, M., & Long, D. (2014). Planning the behaviour of low-cost quadcopters for surveillance missions. In *ICAPS* (pp. 445–453). Portsmouth, NH.

Chouhan, S. S., & Niyogi, R. (2015). DMAPP: A distributed multi-agent path planning algorithm. LNAI, Vol. 9457. In *28th Australasian Joint Conference on Artificial Intelligence* (pp. 123–135). Canberra, Australia.

Cirillo, M., Pecora, F., Andreasson, H., Uras, T., & Koenig, S. (2014). Integrated motion planning and coordination for industrial vehicles. In *ICAPS* (pp. 463–471). Portsmouth, NH.

De Weerdt, M. M., & Clement, B. (2009). Introduction to planning in multiagent systems. *Multiagent and Grid Systems, 5*, 345–355.

Erdmann, M., & Lozano-Perez, T. (1987). On multiple moving objects. *Algorithmica, 2*, 477–521.

Felner, A., Goldenberg, M., Sharon, G., Stern, R., Beja, T., Sturtevant, N. R., ... Holte, R. (2012). Partial-expansion A* with selective node generation. In *AAAI* (pp. 180–181). Toronto, Ontario.

Ferner, C., Wagner, G., & Choset, H. (2013). Odrm* optimal multirobot path planning in low dimensional search spaces. In *ICRA* (pp. 3854–3859). Karlsruhe, Germany.

Geramifard, A., Chubak, P., & Bulitko, V. (2006). Biased cost pathfinding. In *AIIDE* (pp. 112–114). Marina Del Rey, California.

Goldenberg, M., Felner, A., Stern, R., Sharon, G., Sturtevant, N. R., Holte, R. C., & Schaeffer, J. (2014). Enhanced partial expansion A*. *Journal of Artificial Intelligence Research, 50*, 141–187.

Hopcroft, J. E., Schwartz, J. T., & Sharir, M. (1984). On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the warehouseman's problem. *International Journal of Robotics Research, 3*, 76–88.

Khorshid, M. M., Holte, R. C., & Sturtevant, N. R. (2011). A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *SOCS* (pp. 76–83). Barcelona, Spain.

Liu, S., Sun, D., & Zhu, C. (2014). A dynamic priority based path planning for cooperation of multiple mobile robots in formation forming. *Robotics and Computer-Integrated Manufacturing, 30*, 589–596.

Masehian, E., & Nejad, A. (2009). Solvability of multi agent motion planning problems on trees. In *IEEE International Conference on Intelligent Agents and Systems* (pp. 5936–5941). St. Louis, MO.

Parker, L. E. (2008). Distributed intelligence: Overview of the field and its application in multi-robot systems. *Journal of Physical Agents, 2*, 5–14.

Peasgood, M., Clark, C. M., & McPhee, J. (2008). A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics, 24*, 283–292.

Regele, R., & Levi, P. (2006). Cooperative multi-robot path planning by heuristic priority adjustment. In *International Conference on Intelligent Robots and Systems* (pp. 5954–5959). Beijing, China.

Röger, G., & Helmert, M. (2012). Non-optimal multi-agent pathfinding is solved (since 1984). In *SOCS* (pp. 173–174). Niagara falls, Canada.

Ryan, M. (2006). Multi-robot path planning with subgraphs. In *Proceedings of the 19th Australasian Conference on Robotics and Automation* (pp. 1–8). Auckland, New Zealand.

Ryan, M. R. (2007). Graph decomposition for efficient multi-robot path planning. In *IJCAI* (pp. 2003–2008). Hyderabad, India.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015a). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence, 219*, 40–66.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015b). Conflict-based search for optimal multi-agent pathfinding. In *AAAI* (pp. 563–569). Austin, TX.

Silver, D. (2005). Cooperative pathfinding. In *AIIDE* (pp. 117–122). Marina Del Rey.

Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems. In *AAAI* (pp. 28–29). Atlanta, GA.

Sturtevant, N. (2012). Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games, 4*, 144–148.

Sturtevant, N. R., & Buro, M. (2006). Improving collaborative pathfinding using map abstraction. In *AIIDE* (pp. 80–85). Marina del Rey.

Surynek, P. (2009). A novel approach to path planning for multiple robots in bi-connected graphs. In *ICRA* (pp. 3613–3619). Kobe, Japan.

Ter Mors, A. W., Witteveen, C., Zutt, J., & Kuipers, F. A. (2010). Context-aware route planning. In *German Conference on Multiagent System Technologies* (pp. 138–149). Berlin, Germany.

Van Den Berg, J. P., & Overmars, M. H. (2005). Prioritized motion planning for multiple robots. In *IROS* (pp. 430–435). Edmonton, Canada.

Wagner, G., & Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artificial Intelligence, 219*, 1–24.

Wagner, G., & Choset, H. (2011). M*: A complete multirobot path planning algorithm with performance bounds. In *IROS* (pp. 3260–3267). San Francisco, California.

Wang, K.-H. C., & Botea, A. (2011). Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research, 42*, 55–90.

Wang, K.-H. C., & Botea, A. (2009). Tractable multi-agent path planning on grid maps. In *IJCAI* (pp. 1870–1875). Pasadena, California.

Wilt, C., & Botea, A. (2014). Spatially distributed multiagent path planning. In *ICAPS* (pp. 332–340). Portsmouth, NH.
Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative autonomous vehicles in warehouses. *AI magazine, 29*, 9–19.
Yu, W., Peng, J., & Zhang, X. (2014). A prioritized path planning algorithm for mmrs. In *33rd Chinese Control Conference (CCC)* (pp. 966–971). Nanjing, China.