A Multiagent Planning Algorithm with Joint Actions

Satyendra Singh Chouhan and Rajdeep Niyogi

Abstract In this paper, we consider multiagent planning with joint actions—that refer to the same action being performed concurrently by a group of agents. There are few works that study specification of joint actions by extending PDDL. Since there are no multiagent planners that can handle joint actions, we propose a multiagent planning algorithm, which is capable of handling joint actions. In a multiagent setting, each agent has a different capability. The proposed algorithm obtains the number of agents involved in a joint action based on the capability of the individual agents. We have implemented our algorithm and compared its efficiency with some state-of-the-art classical planners. The results show that when the problem size increases, our algorithm can solve such problems whereas it cannot be solved by the classical planners.

Keywords Multiagent planning · Joint action · PDDL

1 Introduction

Multiagent system refers to a system of agents working together to reason, plan, solve problems, and make decision Here an agent can be an intelligent system, process, robot, sensor [1]. Multiagent system is applicable to various domains such as supply chain management, robotics, games, military operation, logistics, and security [2–4].

S.S. Chouhan (✉) · R. Niyogi
Department of Computer Science and Engineering,
Indian Institute of Technology Roorkee, Roorkee 247667, India
e-mail: satycdec@iitr.ernet.in

R. Niyogi
e-mail: rajdpfec@iitr.ernet.in

S. Das et al. (eds.), *Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA) 2015*, Advances in Intelligent Systems and Computing 404, DOI 10.1007/978-81-322-2695-6_59

In recent years, several models and algorithms have been developed in multiagent planning (MAP). A multiagent planning problem is given a set of initial states, goal states, and actions of the agents, to find plans that transform the initial state to goal state of each agent. Typically, a multiagent plan consists of concurrent actions [5, 6]. In some domains, the problem may be decomposed into subproblems and each subproblem is assigned to a different agent [7, 8]. Unlike classical planning (single agent), very few state-of-the-art multiagent planning systems exists. A recent development is FMAP [9].

Concurrent actions come naturally in multiagent planning problems. For example, consider a two-agent domain [6] where both agents are in a room R1 and want to move into the room R2 through a single doorway. The agents have the operators 'G' to go through the door and 'W' to wait. Then possible concurrent actions would be (G, G), (G, W), (W, G), and (W, W). However, a successful plan would be obtained when one agent waits and allows the other agent to pass through the gate. Therefore, coordination is needed between the actions of different agents [10]. Concurrent interacting actions are said to consistent if there is no adverse effect of actions on to each other. Concurrent actions may have negative interaction also.

Joint actions can be considered as a special case of concurrent actions. In some planning domains, an action should be performed jointly by more than one agent to get a desired effect. For example, in order to lift a heavy table, two agents have to lift the opposite sides of the table simultaneously, i.e., both agents have to perform the same action on a particular object at the same time [11]. In this paper, we focus on multiagent planning problems involving joint actions.

Joint actions in multiagent planning raise two main issues: specification and planning. In [11], PDDL [12] is extended for specifying joint actions by including concurrent action constraints in action schema. Another approach to handle joint action is proposed in [13]. In this work, agents are encoded as parameters in action schema and necessary preconditions are added in the precondition list. However, this representation can only specify positive interaction. However, there is no multiagent planning system that can handle joint actions.

In this paper, we make an attempt in this direction to come up with a multiagent planning system that handles joint actions. For this, we propose a multiagent planning algorithm in Sect. 3. The rest of the paper is structured as follows. Section 2 presents specification of joint actions. Section 4 presents the implementation results. Conclusions are given in Sect. 5.

2 Specification of Joint Action

We discuss some approaches that extend the planning domain definition language (PDDL) to represent joint actions. PDDL is widely used to formally describe planning problems. A basic syntax of action in PDDL is as follows:

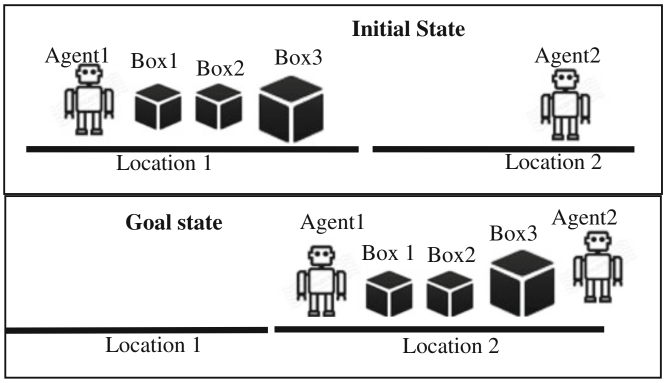


Fig. 1 A Box-pushing domain with possible configuration

```
:action <action_name>
:parameter <parameter_list>
:precondition <precondition list>
:effect <effect list>
```

The above syntax is extended in [11] to introduce concurrent interacting actions. Two main issues that are handled include: (i) 'who' is performing the action and (ii) 'what other actions' should (not) be performed concurrently.

The first issue is handled by including an agent as a parameter in the parameter list of an action. The second issue is dealt by adding a 'concurrent' action list in the action schema. In the concurrent action list, all the actions that are to be performed concurrently are mentioned.

Consider a box-pushing domain with varying size of boxes. Some boxes are heavy; a single agent cannot move it. Hence, more than one agent has to perform the action 'push' simultaneously to move heavy boxes, i.e., a joint action should be performed to achieve the desired effect. Figure 1 shows a simple box-pushing domain with two agent and three boxes. PDDL syntax of the action 'push' is given below:

```
:action push
:parameter (?a1-agent ?b -box ?l- location)
:precondition (and(at ?a1 ?l) (at ?b ?l))
:concurrent (and (push ?a2 ?b ?l)(not(= ?a1 ?a2))
:effect (not (at ?b ?l))
```

Other representation of joint action is proposed in [14]. In this representation, agents are specified in the parameter list and joint action preconditions are mentioned in the precondition list. There is no such additional syntax or special tag to represent concurrency.

```
:joint action push
:parameter (?a1-agent ?a2-agent ?b-box ?l-location)
:precondition (and(at ?a1 ?l)(at ?a2 ?l)(at ?b ?l)
  (not(=?a1 ?a2)))
:effect (not(at ?b ?l))
```

Joint action representation with concurrent action list can also specify the negative interaction of an action, i.e., we can also specify the actions that should not be performed concurrently. In this work, joint action is handled at the planning level. Therefore, there is no requirement of explicit representation of joint actions. To handle joint actions, we introduce the notion of capabilities of agents.

Multiagent classical planning can easily be extended to support agent's capabilities. It is possible to handle the joint action scenario by comparing the required capability of an action at a particular state with the capability of agent performing at that time. Consider the example, Fig. 1, to move a heavy box that may involve at least two agents. The number of agents involved in the joint actions depends on the capability of the individual agents. So the number of agents need not be decided *a priori*.

3 Algorithm for Multiagent Planning with Joint Actions

In this section, we present the brief overview of the proposed multiagent planning algorithm. Input to the planner is given by a PDDL input problem file. Input file contains the domain description that includes the number of agents, set of actions agent can perform, and other objects in the domain. It also specifies the capabilities of each agent. Each action is associated with the required capability to perform the action. Planner agent uses a heuristic forward search to search in state space.

3.1 Description of the Algorithm

The algorithm maintains an individual state of each agent and a global state. For each agent 'i' it maintains an open list of states that needed to be explored and closed list of states that are already explored. For each agent it selects an action from the applicable action list of current state. Applicable action list is ordered based on heuristic values. Here the heuristic value is taken as number of different propositions in the current state and goal state. After selecting actions for all agents, it merges the effects of all agents' selected actions to the current global state. The algorithm eventually stops when the current global state is same as the goal state. The algorithm is given below.

Pseudocode of the algorithm

Algorithm: MA-planner

```
1.  current state = initial state
2.  While (goal is not achieved)
3.      for each agent i
4.          If (! Agent_busy)
5.              App_act_list_i = Applicable_Action(current state, Actionlist_i)
6.              Opt_Act_i = get_optimal_action(App_act_list_i, current state, goal
                state)
7.          else
8.              Opt_Act_i = No_op
9.              Add Opt_Act_i into the MA-plan
10.             If (Check status(agent_i) then set agent_i = free;
11.         end if
12.     end for
13.   current_state = merge_effect (Current_state, Opt_Act_1, Opt_Act_2 ....., Opt_Act_k)
14. end while
```

In the algorithm, planner extracts the applicable action list for agent i using **Applicable_Action**(.) function. Applicable action list contains all the actions that can agent apply at current state (step 5). Form the Applicable action list, optimal action is selected based on heuristic value using **get_optimal_action**(.) (step 6). Pseudocode for **get_optimal_action**(.) is given below.

Algorithm: get_optimal_action (*App_act_list*, *current state, goal state*)

```
1.  for each action act in App_act_list,
2.      calculate heuristic values
3.  end for
4.  sort App_act_list according to heuristic values
5.  for each act in sorted_App_act_list
6.  If (!conflict(act))
7.          If (coordinated_capbility_act == 0)
8.                  opt_act = act;
9.                  coordinated_capbility_act = act_req_cap - agent_Cap
10.             If (coordinated_capbility_act > 0)
11.                 set agent = busy;
12.             end if
13.             return opt_act
14.         else
15.             opt_act = act
16.             coordinated_capbility_act =agent_Cap – act_req_cap
17.             return opt_act
18.         end if
19. end if
20.  end for
21. return no -op
```

In the above algorithm, for each action in applicable action list of agent i heuristic value is calculated (steps 1–3). Applicable action list is sorted according to the heuristic values step (4). For each action from the sorted applicable action list, algorithm checks whether this action can be added in MA-plan or not. If so then optimal action is return otherwise no-op action is return steps (5–21).

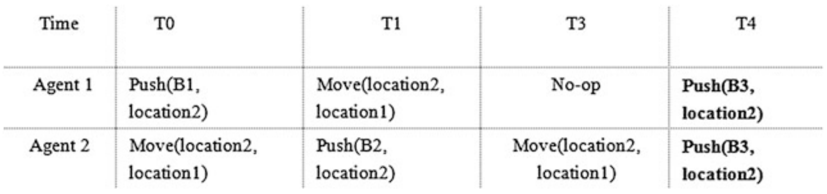
3.2 Specification of Multiagent Plan

Planner agent generates the parallel sequence of actions for multiple agents, i.e., at particular state a set of action $A = \{a_1, a_2, a_3.. . a_n\}$ is selected. Where a_1 is an action of agent 1, a_2 is an action of agent 2, and so on (possibly included no-op action). Sequence of the action sets $A_1;...; A_k$ is called a multiagent plan. This MA-plan is consistent and coordinated with respect to all agents. There can be more than one possible MA-plan for same problem instance. The planner outputs one possible MA-plan for the problem and it may be the case that it is not optimal.

For example, consider the initial state of and goal state of the box-pushing domain as shown in Fig. 1. Objective is to move all the boxes to the location2. Here box3 is heavy therefore can be move if both agents jointly perform the push action at same time. Figure 2 shows one of the possible MA-plan structures for the configuration shown in Fig. 1.

In the MA-plan (Fig. 2), agent1 is waiting at time T3 for agent2 to perform the joint action. At time T4, agent1 and agent2 performs the joint action ‘push’. The plan length of the MA-plan is four. For various problem instances summarized results shown in Table 1.

Here the multiagent plan would be the sequence of these concurrent actions set of agent1 and agent2, i.e., {Push(B1, location2), Move (location2, location1)}; {Move (location2, location1), Push (B2, location2)};{No-op, Move (location2, location1)}; {Push(B3, location2), Push (B3, location2)}, where the elements in a set are executed concurrently or jointly.



Time	T0	T1	T3	T4
Agent 1	Push(B1, location2)	Move(location2, location1)	No-op	**Push(B3, location2)**
Agent 2	Move(location2, location1)	Push(B2, location2)	Move(location2, location1)	**Push(B3, location2)**

Fig. 2 A possible MA-plan structure for the configuration shown in Fig. 1

Table 1 Experimental results in Box-pushing domain and extended block world domain

	No. of agents	Problem size (s, m, l)	Solved?			MA-plan length		
			BB	PP	FF	BB	PP	FF
Box-pushing domain	2	(4, 2, 0)	Y	Y	Y	7	8	11
	2	(5, 3, 0)	Y	Y	N	7	12	–
	2	(6, 2, 0)	Y	Y	N	9	12	–
	3	(6, 2, 1)	Y	Y	N	9	9	–
	3	(8, 4, 2)	N	Y	N	–	17	–
Extended-block world domain	2	(3, 0, 0)	Y	Y	Y	4	4	4
	2	(2, 1, 0)	Y	Y	Y	6	6	6
	2	(2, 2, 0)	N	Y	N	–	6	–
	3	(1, 2, 1)	N	Y	N	–	8	–
	3	(1, 1, 2)	N	Y	N	–	7	–

[a]*BB* Blackbox planner, *PP* Proposed planning system, *FF* Fast Forward planner *N* No, *Y* Yes

4 Implementation Results

We have implemented the forward search algorithm with joint action using Java. Experiments were run on the Intel core i5 with 2.53 GHz machine with 4 GB of RAM. We ran our algorithm on two domains: box-pushing domain and extended block world domain. Box-pushing domain is explained in Sect. 2.

In the block-world domain, the goal is to find a plan to rearrange blocks from the initial configuration to the goal configuration. Blocks can be put on the table or on top of each other. A block can be moved only if it is the top block; additionally, only one block can be moved at a time. It is a classical planning problem. In our experiment, we considered block world domain with multiple agents and different sizes of blocks. In this experiment, light box can be moved or pickup by single agent. However to move heavy boxes more than agent should pick up the box jointly.

We have compared the performance of our algorithm with some classical planners like black box (BB) [15] and fast forward (FF) planner [14]. Black box and fast forward takes planning problems defined in STRIPS or PDDL. We have described the domains using the joint action specification technique suggested in [14]. With some modifications in PDDL, these planners (FF and BB) can support joint action up to a certain level. Table 1 shows the implementation results.

In the table, column1 shows the number of agents in the system. Column2 shows the problem size. The notation (s, m, l) is the number of small, medium, large boxes present in the problem, respectively. A small box needs one agent to push it. A medium box requires two agents simultaneously (jointly). A large box needs more than two agents simultaneously. Column3 shows whether the centralized planner was able to generate a solution plan. Column4 shows the MA-plan length.

Experimental results show that problems having more than two agents and large objects cannot be solved by classical planners, as in the (8, 4, 2) case. It is because these classical planner search in the joint state space of all agents, which becomes exponentially large with the problem size.

5 Conclusion and Future Work

In this paper, we considered a multiagent planning problem where multiple agents with different capabilities may be involved in performing some actions. We have developed a planner that can generate such plans. We have compared the performance of our planner with some existing state-of-the-art planners on some benchmark domains. Experimental results show that some problem instances cannot be solved by the classical planners. However, the proposed planner can easily handle such problem instances. There are few multiagent planners available; none of these planners can handle joint actions. Thus, our work is a first attempt toward the development of such planners. As part of future work, we would like to develop a distributed planner capable of handling joint actions.

References

1. Parker, L.E.: Distributed intelligence: overview of the field and its application to multi-robot systems. J. Phys. Agents **2**(1), 5–14 (2008)
2. Marcello C., Pecora F., Andreasson, H., Uras, T., Koenig, S.: Integrated motion planning and coordination for industrial vehicles. In: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS) (2014)
3. Sara B., Fox, M., Long, D.: Planning the behaviour of low-cost quadcopters for surveillance missions. In: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS) (2014)
4. Wu, F., Zilberstein, S., Chen, X.: Online planning for multi-agent systems with bounded communication. Artif. Intell. **175**, 487–511 (2011)
5. Brenner, M., Nebel, B.: Continual planning and acting in dynamic multi-agent environments. In: Proceedings of Autonomous Agent Multi-agent System, pp. 297–331 (2009)
6. Bowling, M., Jensen, R., Veloso, M.: A formalization of equilibria for multi agent planning, In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 1460–1462 (2003)
7. Brafman, R.I., Domshlak, C.: From one to many: planning for loosely coupled multi-agent systems, In: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS), pp. 28–35 (2008)
8. Ephrati, E., Rrosenschein, J.S.: Divide and conquer in multi-agent planning. In: Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 375–380 (1994)
9. Torreño, A., Onaindia E., ÓscarSapena: FMAP: distributed cooperative multi-agent planning. Appl. Intell. **41**, 606–626 (2014)
10. Larbi, R.B., Konieczny, S., Marquis, P.: Extending classical planning to the multi-agent case: a game-theoretic approach. In: ECSQARU. Lecture Notes in Computer Science, vol. 4724, pp. 731–742. Springer (2007)

11. Boutilier, C., Brafman, R.I.: Planning with concurrent interacting actions. J. Artif. Intell. Res. (JAIR), **14**, 105–136 (2001)
12. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. (JAIR) **20**, 61–124 (2003)
13. Brafman, R.I., Zoran, U.: Distributed heuristic forward search for multi-agent system. In: Proceedings of 2nd Distributed and Multi-agent Planning Workshop (ICAPS DMAP), pp. 1–7 (2014)
14. Hoffmann, J., Nebel, B.: The FF planning system: fast plan generation through heuristic search. J.Artif. Intell. Res. (JAIR) **14**, 253–302 (2001)
15. Kautz, H., Selman, B.: BLACKBOX: a new approach to the application of theorem proving to problem solving. In: AIPS98 Workshop on Planning as Combinatorial Search (1998)