Deterministic Multiagent Planning Techniques: Experimental Comparison (Short paper)

Karel Durkota[1] and **Antonín Komenda**[2]
karel.durkota@gmail.com, komenda@agents.fel.cvut.cz
[1]Faculty of Electrical Engineering, Czech Technical University in Prague
[2]Dept. of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract

Deterministic domain-independent planning techniques for multiagent systems stem from principles of classical planning. Three most recently studied approaches comprise (i) DisCSP+Planning utilizing Distributed Constraint Satisfaction Problem solving for coordination of the agents and individual planning using local search, (ii) multiagent adaptation of A* with local heuristics and (iii) distribution of the GraphPlan approach based on merging of planning graphs.

In this work, we summarize the principles of these three approaches and describe a novel implementation and optimization of the multiagent GraphPlan approach. We experimentally validate the influence of parametrization of the inner extraction phase of individual plans and compare the best results with the former two multiagent planning techniques.

Introduction

The problem of multiagent planning as defined in (Brafman and Domshlak 2008) is similarly important as classical planning, as it can provide generally usable techniques for intelligent agents, which are required to cooperatively come up with distributed plans. Recently the research community proposed both theoretical treatments and implementations of such distributed multiagent planning (DMAP) techniques.

Similarly to the classical planning, the agents in DMAP cooperatively search for the local sequences of actions, which after execution transform the world from an initial state to a common goal state. The local sequences of actions—the local plans—has to interleave appropriately, as each particular agent cannot possibly solve the problem on its own, but have to base its own actions on the results of actions of the other agents. Furthermore, the agents are motivated to communicate as few information as possible not to put load on the other agents if it is not needed.

Three recently theoretically treated approaches for DMAP are (i) multiagent planning utilizing a solver for Distributed Constrain Satisfaction Problems (DisCSP) for the coordination part and a classical planning for the individual plans denoted as DisCSP+Planning (Brafman and Domshlak 2008), (ii) extension of A* for multiagent systems coined Multiagent Distributed A*, (MA-A*) (Nissim and Brafman 2012a; 2012b) and (iii) Distributed Planning through Graph Merging (DPGM) (Pellier 2010) which uses principally the same factorization scheme for separation of parts of the original planning to more agents as in the previous approaches, defined originally in (Brafman and Domshlak 2008) together with the MA-STRIPS formalization.

First two approaches, namely DisCSP+Planning and MA-A*, were already implemented and experimentally validated. Works describing the implementation and experiments are for DisCSP+Planning (Nissim, Brafman, and Domshlak 2010) and for MA-A* the original papers (Nissim and Brafman 2012a; 2012b). However, according to our knowledge, the Pellier's approach was not implemented and experimentally verified yet. Therefore, our initial focus in context of this work was the implementation of the approach described by Pellier and comparing it with the other two approaches. Since this comparison was not done yet, it was not clear if the GraphPlan (Blum and Furst 1997) approach could be viable in multiagent setting, although the underlying approach in classical planning was outperformed already at (IPC 2004). Especially as in the multiagent setting the communication complexity can be of much more importance than the computational complexity.

Multiagent planning

Planning in a multiagent (MA) systems is by (Brafman and Domshlak 2008) a search for a plan for each agent, assuming that agents have to cooperate in order to reach a global goal. Formally, problem for a set of k agents $AG = \{ag_i\}_{i=1}^{k}$ is given by a quadruple $\Pi = \langle P, \{A_{ag_i}\}_{i=1}^{k}, I, G\rangle$, where P is a finite set of propositions describing facts holding in the world; $I \subseteq P$ is a set of propositions that hold in the initial state; $G \subseteq P$ is a set of propositions that must hold in a goal state; and A_{ag_i} is a set of actions that an agent ag_i can perform. Each action has a standard STRIPS syntax, i.e., $a = \langle pre(a), add(a), del(a)\rangle$, where $pre(a), add(a), del(a) \subseteq P$ and $add(a) \cap del(a) = \emptyset$. An action a can be performed only in a state $s \subseteq P$, which the propositions form $pre(a)$ hold in. Performing an action a will add to the state s propositions from $add(a)$ and remove the propositions from $del(a)$.

DisCSP+Planning-based planner The algorithm from (Nissim, Brafman, and Domshlak 2010) can be

described as two interleaving components, a *coordination component* and an *individual planning component*; both of which require the separation of the public and individual actions of each agent. An action a of an agent ag_i is public if there exists an action b of an agent ag_j, $i \neq j$, such that $(\text{pre}(a) \cup \text{add}(a) \cup \text{del}(a)) \cap (\text{pre}(b) \cup \text{add}(b) \cup \text{del}(b)) \neq \emptyset$, otherwise the action is considered individual. This separation defines the multiagent problem factorization.

The *coordination component* deals only with the public actions. It searches for a sequence of the interaction points between the agents' plans. For a given length of the public part of the plan δ, it tries to assign different public actions of each agent in the different time-steps so that the global goal is satisfied. This is the interaction part, so the resulting plan of each agent has to satisfy the requirements put by the rest of the agents and vice versa. These requirements are described in form of *coordination constraints* in the inner DisCSP problem. Solving this problem effectively means solving the multiagent planning problem. If some agent or the team as a whole can not solve the DisCSP, then δ, the length of the coordination public part of the plan is increased by one and the whole process is repeated.

The *individual planning component* forms the other type of constraints for the DisCSP process encoding the requirement of the local parts of the plan. The *individual planning constraints* limits usage of the public actions in the coordination part of the plan such that the gaps between them can be filled by sequences of individual actions of the agents.

Multiagent A* The algorithm proposed in (Nissim and Brafman 2012b) is inspired by the well-known A* algorithm. Similarly to centralized A* the Multiagent Distributed A* (MA-A*) maintains *open lists* for all agents, that keep track of the so far unvisited states, and *closed lists*, that keep track of the already visited states. Each agent also uses a local heuristic to decide which state from the open list it should expand as next. As stated in the paper, each agent can use different heuristics.

In the MA-A*, similarly to the DisCSP+Planning, it is firstly necessary to separate every agents' public and individual actions. The algorithm runs simultaneously for each agent. During the search, the agents send messages to each other to distribute the search at the points, where the other agents can follow. Effectively, it means the messages are sent only for states achieved by public actions. Each such message consists of a state s, its cost value $g_{ag_i}(s)$ and a heuristic estimate $h_{ag_i}(s)$. In decoupled problems, this principle allows distribution of the knowledge about the entire search space among the agents. When an agent receives a message with a state s, it decides either to: visit the state (by adding it to its *open list*), update its knowledge about s, or discard it if it knows better path to the state.

The search terminates if an agent expands a state which is compatible with the goal set G and the resulting plan is ensured to be globally optimal.

Distributed Planning through Graph Merging The algorithm DPGM presented in (Pellier 2010) uses as the main data structure a planning graph together with the distributed versions of algorithms for its building and for extraction of the resulting plan. The distributed extraction consists of a individual CSP and a distributed coordination mechanism. The planner as a whole can be described by following five phases: *global goal decomposition*, *expansion*, *planning graph merging*, *individual plan extraction*, *coordination*. The result is in form of a *coordinated individual solution plan*.

In the first phase, the *global goal decomposition*, each agent creates an individual goal, i.e., a subset of the propositions from the global goal that it can reach. Proposition $p \in G$ is in the individual goal G_{ag_i} of an agent ag_i if exists an action $a \in A_{ag_i}$ such that $p \in add(a)$. If any proposition from the global goal cannot be assigned to any agent, the problem has no solution.

In the next two phases, the *expansion* and the *planning graph merging*, every agent builds an individual planning graph via GraphPlan algorithm (Blum and Furst 1997). Firstly, every agent builds a new layer in their planning graphs. Afterward, relevant actions from the new layer are shared among the agents. The shared actions are included into their respective planning graphs. An action $a_{ag_i} \in A_{ag_i}$ is *relevant* to an agent ag_j in two cases: (i) $pre(a_{ag_i})$ contains a proposition that another action $a_{ag_j} \in A_{ag_j}$ uses in $del(a_{ag_j})$ or $add(a_{ag_j})$, then the action a_{ag_i} *promotes* the action a_{ag_j} or (ii) $del(a_{ag_i})$ contains a proposition that another action $a_{ag_j} \in A_{ag_j}$ uses in $pre(a_{ag_j})$ or $add(a_{ag_j})$, then the action a_{ag_i} *threats* the action a_{ag_j}. The algorithm alternates between the *expansions*, where all the agents build new layers in their planning graphs and *planning graph merging*, where all the agents share their actions until all agents reach their individual goals in their planning graphs or the fixed point is reached.

In the *individual plan extraction* phase, each agent extracts plan(s) from its planning graph. In the centralized GraphPlan algorithm, this is done by compilation of the problem into a CSP and solved by a CSP solver. Each result from the solver is then one resulting plan as Kambhampati showed in (Kambhampati 2000).

In the last two *coordination* phases, before an agent ag_i generates an individual plan, it includes *requirement* and *commitment constraints*—induced by the other agents' plans—into its individual plan extraction CSP problem. The *requirement constraints* are couples (a, l) which describe an action a has to be performed in l-th layer. As $req(\pi_{ag_i})$, we will denote a set of requirement constraints induced by the current partial plan π_{ag_i} of the agent ag_i. A partial plan π_{ag_i} in this phase contains finished parts from agents $1, \ldots, i$ and future actions required by the agent ag_i (and possibly from previous agents) for the following agents $i+1, \ldots, k$. The *commitment constraints* are again couples (a, l) denoting that no action b, which is in *mutex* with action a, can be performed in l-th layer. Let $com(\pi_{ag_i})$ be a set of the commitment constraints induced by the current partial plan π_{ag_i}.

A couple $c = (com(\pi_{ag_i}), req(\pi_{ag_i}))$, besides representing the partial plan π_{ag_i} in form of the action commitments and requirements, describes which agents have already contributed to the plan π_{ag_i} with their individual plans (the performers of the actions in $com(\pi_{ag_i})$) and which agents have not (the performers of the actions in $req(\pi_{ag_i})$ but not in

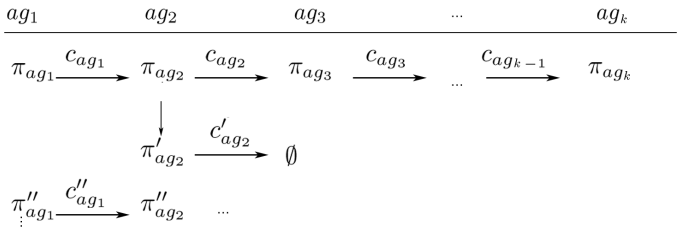

Figure 1: The DPGM plan search process.

$com(\pi_{ag_i})$). Such common partial plan in form of requirement couple c is passed from one agent to another. Each agent extends it with its individual plan and possibly new requirements for the following agents. After c passes all the agents, it contains a global plan consisting of the individual plans of all the agents.

Assume the agents are ordered $ag_1, ag_2, \ldots, ag_k$, as illustrated in Figure 1. The phase starts with an agent ag_1 which generates its first plan π_{ag_1}—lower index indicates the owner of the individual plan. The agent computes the commitment and requirement constraints, denoted as $c_{ag_1} = (com(\pi_{ag_1}), req(\pi_{ag_1}))$ and sends them to the next agent ag_2. ag_2 includes constraints c_{ag_1} into its CSP problem as additional constraints and generates a plan π_{ag_2} which is compatible with the plan π_{ag_1}. Next, ag_2 derives new constraints $c_{ag_2} = (com(\pi_{ag_1}) \cup com(\pi_{ag_2}), (req(\pi_{ag_1}) \cup req(\pi_{ag_2}))$, and passes them to the agent ag_3, and so on. In Figure 1, the search would actually end with generating ag_k's plan π_{ag_k} which illustrates the ideal way of the solution, since no agent had to generate a variance of its plan more than once to fulfill all the requirements from previous agents. If agent ag_3 could not generate plan π_{ag_3}, algorithm would backtrack to agent ag_2, who would generate an alternative plan π'_{ag_2}, as shown in the figure. If all agents ran out of plans, algorithm return to *expansion* and *planning graph merging*, and whole search repeats again.

Implementation

In (Pellier 2010), the author introduces theoretically the algorithm, yet no experiments were carried out to verify its efficiency. Besides the implementation of the algorithm in Java programming language, we included some implementation improvements to speed up the algorithm.

Constraint cache To reduce the search tree, as illustrated in Figure 1, each agent memorizes the constraints it passed to the next agents. If the constraints caused one of the following agents is not able to generate any plan, the requesting agent do not require such constraints for the next agents any more.

For an instance depicted in Figure 1, at point when agent ag_2 generates plan π_{ag_2} together with its constraints c_{ag_2}, he memorizes the constraints c_{ag_2} before passing them to ag_3. Let us assume that the constraints caused ag_3 to be unable to generate any individual plan depicted as $\emptyset$. The agent ag_2 memorizes this information and introduces new constraint into his CSP assignment that will eliminate generating of plans that restricts the agent ag_3 in future as c_{ag_2} did.

For an instance, $c_{ag_2} = (\{\}, \{(a_{ag_3}, 1), (a_{ag_4}, 2)\})$, where $(a_{ag_3}, 1)$ restricts ag_3 to perform action a_{ag_3} in first layer and $(a_{ag_4}, 2)$ restricts ag_4 to perform action a_{ag_4} in second layer. Since ag_3 could not find any plan that satisfies restriction $(a_{ag_3}, 1)$, agent ag_2 should not generate such plans. Thus, ag_2 introduces new constraint into its CSP assignment that will eliminate plans having action a_{ag_3} in first layer. This will cause agent ag_2 to eliminate generating of the plans that agent ag_3 cannot satisfy.

Ordering of agents Ordering of the agents turned out to be crucial for the DPGM algorithm to work efficiently. It is necessary to separate agents that have an individual goal from those that have no goal, but whose cooperation might be required somewhere during the plan. Let AG_g be a set of all agents having an individual goal and let AG_s be a set of all agents without their own goals. As the plan search progresses, the actual agents' ordering dynamically changes. The ordering starts with a random agent ag_i from the set AG_g (the set cannot be empty at this point; if it was, we have no goal, therefore the problem would be unsolvable). Afterward, a plan π_{ag_i} is generated together with constraints c_{ag_i}. Next, agent ag_j is selected by looking which cooperation is required in c_{ag_i}. If there are more such agents, they are prioritized from AG_g over the agents from AG_s. After generating new π_{ag_j} and c_{ag_j}, the process is repeated. If c_{ag_j} has no requirements on the other agents at any point, although AG_g or AG_s are not empty yet, it means that there exists a goal reaching plan without cooperation of the agents remaining in AG_g and AG_s sets. Notice that this may happen even for AG_g, although we said that these are the agents with goals, if a goal proposition can be reached by more than one agent and an agent outside AG_g has reached it.

Removing unnecessary actions Since we used pure PDDL parser to read the domain and the problem, we could end up with useless actions. For example, in the simplest LOGISTICS problem these actions are following: (drive car1 place1 place1) or (fly plane1 airport1 airport1). These actions do not change the state in any way. Preconditions of an action (drive car1 place1 place1) are (at car1 place1), a positive effect is (at car1 place1) and a negative effect is (at car1 place1). A result of this actions is that the car will remain in the same place (so does the action (fly plane1 airport1 airport1) with the plane). These actions can be generalized as $a = \langle pre(a), add(a), del(a) \rangle$, where $add(a) = del(a)$. Moreover, actions (as STRIPS defines them) have a requirement $add(a) \cap del(a) = \emptyset$, but a pure PDDL parser cannot hold this requirement while parsing. Hence, these actions had to be removed by the algorithm.

Experiments

The experiments were carried out on five different domains, where three originated from the International Planning Competition benchmarks adapted for the multiagent planning: ROVERS, LOGISTICS, and SATELLITES. The additional two are: LINEAR LOGISTICS (one package has to be transported step-wise by all agents in a chain) and DECONFLICTION

domain-agents	Minion	Minion srf	Choco	comm.
rover-a2	8.9s	3.8s	11.7s	69kB
rover-a3	6.6s	20.3s	17.4s	234kB
log-a4	0.9s	0.3s	1.2s	34kB
log-a6	0.7s	0.6s	1.3s	136kB
log-lin-a6	0.5s	0.5s	0.3s	167kB
log-lin-a8	0.7s	0.7s	0.5s	417kB
log-lin-a10	0.9s	0.9s	0.7s	849kB
log-lin-a15	1.6s	1.6s	1.8s	2.9MB
deconf-a2	–	1.3s	–	18kB
deconf-a3	0.2s	0.2s	0.1s	13kB
satellite-a6	1.6s	1.5s	4.6s	266kB
satellite-a8	5.0s	4.3s	24.8s	793kB
satellite-a10	14.3s	12.7s	101s	1.8MB

Table 1: Comparison of CSP solvers used in DPGM. The dash – means that the time or memory limit was exceeded.

(robots on a grid are tasked to switch its positions with opposite ones, not colliding with each other). Each domain was tested on several problems with various numbers of agents. All experiments were run on 8-core processor at 3.6GHz with 2.5GB limit on memory and 10 minutes time limit. We used time and communicated bytes as metrics for the comparison of the algorithms.

Comparison of used CSP solvers in DPGM

As DPGM algorithm uses CSP solver for the local plan extraction, we experimented which solver would serve the best. Two CSP solvers were tested: Choco CSP Solver[1] and Minion CSP Solver[2]. Choco solver was tested with its basic setting, while Minion was tested with and without smallest-ratio-first (*srf*) variable order. Table 1 shows the times DPGM took to solve the problems using certain CSP solvers and settings. Although Minion solver showed to be sometimes unstable and did not return any result over the longer period of time, DPGM was faster with it than with the Choco solver. Another Minion's disadvantage is that if a problem is unsolvable, it is inefficiently detected, since it has to go through all the possibilities in the search space. Last column (comm.) in Table 1 shows the communicated bytes among the agents. As the CSP solver is used only for local extraction of a plan, the numbers are the same for all the solvers. In the deconf-a3 problem, even the number of agents is higher than in deconf-a2, the results are better. This is caused by the particular problem instance, where the agents in the a2 case has to pass by each other, and therefore the solution is found not before 4th layer, however in a3 the agents only rotates and therefore the solution is found in 2nd layer.

Comparison of the multiagent planners

In the final experiment, the DPGM algorithm was compared to other two cited algorithms. Table 2 shows the results. As the *srf* setting of Minion showed the best results—

[1]http://www.emn.fr/z-info/choco-solver/

[2]http://minion.sourceforge.net/

domain-agents	DPGM	DisCSP+Pl.	MA-A*
rover-a2	3.8s/69kB	1.4s/0.8kB	22.4s/52kB
rover-a3	20.3s/234kB	7.9s/1.7kB	230s/2.5MB
rover-a4	–	62.3s/3.1kB	–
log-a4	0.3s/34kB	0.6/15kB	0.8s/77kB
log-a6	0.6s/136kB	38.5/7.1MB	2.0s/320kB
log-lin-a6	0.5s/167kB	–	1.7s/87kB
log-lin-a8	0.7s/417kB	–	4.7s/254kB
log-lin-a10	0.9s/849kB	–	15.4s/589kB
log-lin-a15	1.6s/2.9MB	–	217s/4.2MB
deconf-a2	1.3s/18kB	N/A	0.9s/15.3kB
deconf-a3	0.2s/13kB	N/A	1.2s/187kB
deconf-a4	–	N/A	3.8s/2.1MB
satellite-a6	1.5s/266kB	4.4/6.5kB	7.4s/270kB
satellite-a8	4.3s/793kB	–	37.5s/964kB
satellite-a10	12.7s/1.8MB	–	189s/2.5MB

Table 2: Results for DPGM, DisCSP+Planning and MA-A* with set-additive heuristic. The dash – means the time or memory limit was exceeded. N/A means the planner did not return a sound plan.

especially because of its ability to solve most of the presented problems—we chose it for comparison with the other algorithms.

The results show the DPGM to be efficient in decoupled domains which are rather combinatorially easy (LOGISTICS, LINEAR LOGISTICS and SATELLITES). The DisCSP+Planning is efficient in problems which are combinatorially hard from perspective of individual planning (ROVERS), as the internally used planner is highly efficient FastForward. The used implementation of MA-A* with set-additive heuristics was most effective in highly coupled domains (DECONFLICTION).

Final remarks

DPGM showed its strength based on efficient factorization of the problems. However problems as DECONFLICTION, which are coupled and require high combinatorial search, DPGM solves rather inefficiently if at all. An issue that hindered the algorithm was the order how CSP generated the plans. For instance, the first plan that the agent ag_1 generated in the 4th layer of the deconf-a4 problem, consisted of its own actions, leading him to the goal. Additionally, agent generated requirements for the agent ag_2 to prevent future collisions. However, the requirements were unreasonable: instead of requiring one action that would suffice for agent ag_1 to avoid the collision with agent ag_2, he built a whole plan for the agent ag_2. And since ag_1 did not know the ag_2's goal, the plan was usually invalid. We tried to avoid this, by stating to minimize requirements put on other agents in the CSP solver. This approach helped to lower the number of the constraints; however, the solution of such CSP became combinatorially more complex and therefore did not bring much of improvement in the efficiency. Deeper study of these phenomenons remains for future work.

Acknowledgments This work was supported by Czech Science Foundation (GACR) under grant no. 13-22125S.

References

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial intelligence* 90(1):281–300.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of ICAPS'08*, 28–35. AAAI.

IPC. 2004. International planning competition (IPC) – Results. http://www.tzi.de/ edelkamp/ipc-4/results.html.

Kambhampati, S. 2000. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in Graphplan. *J. Artif. Intell. Res. (JAIR)* 12:1–34.

Nissim, R., and Brafman, R. 2012a. Multi-agent A* for parallel and distributed systems. In *Proceedings of 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1265–1266.

Nissim, R., and Brafman, R. 2012b. Multi-agent A* for parallel and distributed systems. 43–51.

Nissim, R.; Brafman, R. I.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of AAMAS'10*, 1323–1330. IFAAMAS.

Pellier, D. 2010. Distributed planning through graph merging. In Filipe, J.; Fred, A. L. N.; and Sharp, B., eds., *Proceedings of ICAART'10*, volume 2, 128–134. IFAAMAS.