Multi-agent planning by distributed constraint satisfaction

Pablo Castejón[1], Pedro Meseguer[2], and Eva Onaindía[1]

[1] Dept. de Sistemas Informáticos y Computación, UPV
Camino de la Vera s/n, 46022 Valencia, Spain,
pcastejon@dsic.upv.es, onaindia@dsic.upv.es,
[2] IIIA-CSIC, Campus de la UAB,
08193 Bellaterra, Spain
pedro@iiia.csic.es

Abstract. Multi-agent planning has received substantial attention in the last years. A few works have considered the translation into SAT terms, to be solved distributedly by an specialized SAT solver. Here, in the context of interleaving planning and coordination, we propose an approach that uses the ABT (asynchronous backtracking) algorithm to distributedly solve the propositional formulation of a multi-agent planning instance. This algorithm, presented for distributed constraint satisfaction, is generic and provides interesting privacy properties. We show some experimental results on some real-world instances coming from standard planning competitions.

1 Introduction

Multiagent Planning (MAP) is a relatively recent research field, in which multiple agents work together to solve a planning task that they are not able to solve by themselves, or to at least accomplish better by cooperating [9]. The distributed approach to MAP assumes various planning entities (agents) that synthesize plans and interact with each other to coordinate their local plans to solve the MAP task. Thus, planning and coordination are at the core of any MA planner.

A common approach in planning has been to translate the planning instance into a propositional formula, to be solved by an off-the-shelf SAT solver and the solution is retranslated into planning terms. To the best of our knowledge, the most recent SAT-based approach dealing with multi-agent distribution of the planning task is μ-SATPLAN [10], an extension of SATPLAN [12]. In this work, the task is distributed across agents by a goal allocation method and then it is solved by refinements of iterative local plans from each agent.

In this paper, we extend the previous model for problems with specialized agents and cooperative goals (so different agents have different capabilities but common goals). A common goal is a public proposition that must be satisfied at some point, therefore any agent could be responsible to achieve it. Thus, we adopt an interleaving planning and coordination method instead of goal allocation.

In addition, we consider ABT, a generic algorithm presented in the context of distributed constraint satisfaction, to solve the propositional logic translation of MAP instances. This algorithm provides some guarantees regarding privacy, keeping inside agents some information that could be considered sensible or reserved.

The structure of this paper is as follows. In Section 2 we summarize some planning concepts needed for the rest of the paper and the ABT algorithm. In Section 3 we provide a global view of our approach, detailing how we made the translation into propositional logic. In Section 4 we describe how ABT can solve the problem, with special emphasis into the privacy issues. We present some empirical results on real-world benchmarks in Section 5, and finally we conclude the paper in Section 6.

2 Background

In this section we present a brief review of some notions and the most relevant trends on SAT-based planning and multi-agent planning, as well as a description of the ABT algorithm.

2.1 Planning as Satisfiability

Planning as satisfiability was first introduced in [13] as an alternative to the classical heuristic search methods used to solve planning problems. The basic idea is to reduce the planning problem to a time horizon and synthesize a propositional formula that is satisfiable if and only if there is a plan that achieves all the goals within the bounded time. The propositional formula comprises the relations between the state variables at each time within the time horizon. Typically, this formula is expressed as a conjunctive normal form (CNF).

The primary advantage of SAT-based planning is that it finds the optimal plan regarding makespan, but the sequential search through time horizons may entail a weakness when dealing with large problems. Hence, efforts in SAT-based planning have been focused on improving the problem encoding, the search of parallel solutions and the use of heuristics to reduce the computation cost of satisfiability tests.

The encodings used to represent SAT problems have evolved mainly considering different restrictions about action parallelism. The $\forall$-step plans uses the GraphPlan's notion of parallelism that allows simultaneous actions if they are pairwise independent. This kind of encoding was used by BLACKBOX [14] and its successors. A more relaxed notion, $\exists$-step, considers simultaneous actions a set of actions that can be totally ordered so that an earlier action does not disable a later action or chage its effects. The latter method produces the smallest and generally most efficient encodings with asymptotically optimal linear size.

Most of the related work in SAT-based planning used general-purpose SAT-solvers, some used DPLL SAT-solvers [11], but during the past ten years the conflict-driven clause learning (CDCL) algorithm has become the most widely used resolution method along with some variable selection heuristic [17]. The use of these techniques allowed the latest SAT-based planners outperform some search-based planners on standard benchmarks.

2.2 Multi-agent Planning

MAP approaches can be classified according to the planning and coordination models they use. Some approaches, like MAPR [4], apply a pre-planning distribution, consisting in allocating the goals of the MAP task to the participating agents and then solving the individual planning tasks with a single-agent planner. Other approaches introduce a fully automated decomposition algorithm to break down a (single) planning task into subtasks with limited interaction; these subtasks are used to identify the underlying multi-agent nature of the planning task [8].

Plan-merging techniques have been popularized by various planning systems, from the iterative revision of the agents' local plans [18] to distributed coordination frameworks based on partial-order planning [7]. Plan merging is not an effective technique to attain cooperative goals (goals that require the collaboration of more than one agent) since this resolution scheme generally assumes that each agent is able to solve a subset of the task's goals by itself. However, some recent planners based on MA-STRIPS [5], a minimalist multi-agent extension of the STRIPS model, use plan merging to coordinate local plans of specialized agents (agents with different planning capabilities). This is the case of Planning First [16], where agents individually synthesize plans by using their private information and the local plans are then coordinated through a distributed Constraint Satisfaction Problem. In general, the focus of plan merging is placed on discovering the interaction points among agents through the shared public information.

A third group of MAP approaches can be classified as *interleaving planning and coordination*. MA-A* [15] is also a MA-STRIPS-based approach that performs a distributed A* search, guiding the procedure through admissible local heuristic functions, and the work in [3] formulates a privacy-preserving MAP model by adapting MA-A*. Unlike models that use local heuristic search, other approaches coordinate the agents' plans by using global distributed heuristics, which require a multi-agent design of the heuristic and a sophisticated communication machinery between agents. In contrast, the quality of the plans is usually much better than planners guided by local heuristics. FMAP is a fully-configurable distributed search procedure that efficiently solves tightly-coupled MAP tasks that have specialized agents and cooperative goals as well as loosely-coupled problems [19]. A later version of this planner that uses a novel multi-heuristic approach is presented in [20].

While almost all of aforementioned approaches are categorized as heuristic planning, it is worth mentioning the planner μ-SATPLAN [10], which extends a satisfiability-based planner to coordinate the agents' local plans by studying positive and negative interactions among them.

2.3 ABT: Asynchonous Backtracking

ABT (Asynchronous Backtracking) [21] was the pioneer algorithm to solve distributed constraint satisfaction problems. ABT is an asynchronous algorithm executed autonomously by each agent in the distributed constraint network, which takes its own decisions and informs of them to other agents, but no agent has to wait for decisions of other agents. Initially presented assuming one variable per agent, ABT computes a global

consistent solution or detects that no solution exists in finite time; it is correct and complete. Binary ABT (when constraints relate pairs of variables) considers that constraints are directed. A constraint causes a directed link between the two constrained agents: the value-sending agent (VSA), from which the link departs, and the constraint-evaluating agent (CEA), to which the link arrives. When the VSA makes an assignment, it informs to the CEA, which tries to find a consistent value. If it cannot, it sends back a message to cause backtracking in the VSA. To make the network cycle-free there is a total order among agents, which is followed by the directed links. [3]

ABT is executed on each agent. It maintains two main data structures: the agent view and the nogood store. If *self* is a generic agent, *self*'s agent view is the set of values that it believes to be assigned to agents connected to *self* by incoming links. The nogood store keeps the nogoods [4] received by *self* as justifications of inconsistent values. Agents exchange assignments and nogoods. ABT always accepts new assignments, updating the agent view accordingly. When a nogood is received, it is accepted if it is consistent with the agent view of *self*, otherwise it is discarded as obsolete. An accepted nogood is used to update the nogood list. When an agent cannot find any value consistent with its agent view, because of the original constraints or because of the received nogoods, a new nogood is generated from its agent view, and it is sent to the closest agent in the new nogood, causing backtracking. If *self* receives a nogood including another agent not connected with it, *self* requires to add a link from that agent to *self*. From this point on, a link from the other agent to *self* will exist, so *self* will receive the values taken by that agent. The process terminates when achieving quiescence (a solution has been found), or when the empty nogood is generated (the problem has no solution). ABT uses four types of messages:

1. $OK?(agent, value)$. It informs *agent* that *self* has taken *value* as value. It is sent to agents connected by outgoing links each time that *self* changes value.
2. $NGD(agent, ng)$. It informs *agent* that *ng* is a nogood. It is sent to the closest agent to *self* in *ng* w.r.t. the agent ordering, when *self* does not find a consistent value.
3. $ADL(agent)$. It asks *agent* to set a direct link to *self*.
4. $STP(agent)$. *self* notifies that there is no solution.

ABT operation is as follows. Concurrently, each agent takes a value not forbidden by any nogood and informs its lower priority agents via OK? messages. When an agent finds all its values forbidden by nogoods, a new nogood is created as the resolution of these nogoods, and it is sent to the closest agent mentioned in the new nogood, via a NGD message. The value of that agent is forgotten in the agent view, and the nogood store is updated accordingly. Then, the assignment process is repeated.

[3] For non-binary ABT, each clause has an *evaluating* agent, to which all other agents in the clause send their values. It is the last agent in the total ordering among the clause agents. It tries to satisfy the clause by choosing a value of its domain that is consistent with the values of the other agents. If no such value exists, it performs backtracking.

[4] A *nogood* is a conjunction of assignments that has been found inconsistent, so it cannot be satisfied by any solution. Nogoods are usually written in directed form: left-hand side $\Rightarrow$ right-hand side (abbr. `lhs` $\Rightarrow$ `rhs`), where `rhs` only contains the assignment of the deepest variable in the total ordering when finding the nogood. Assuming that $\neg(x = true \wedge y = false)$ is a nogood and y is the deepest variable, the directed nogood is $x = true \Rightarrow y \neq false$ [1].

3 SAT Encoding for Multi-Agent Planning

In this section we present the formalization of a MAP task, the particularization to SAT-based planning, and finally the method used to generate the SAT-encoding for each agent. We define a multi-agent planning task as in [20], where agents have limited knowledge of the planning task, and it is assumed that missing information is unknown to the agent. A world state is defined through a finite set of *state variables*, $\mathcal{V}$, each of which is associated to a finite domain, $\mathcal{D}_v$, of mutually exclusive values. Assigning a value d to a variable $v \in \mathcal{V}$ generates a *fluent*, a tuple of the form $\langle v, d\rangle$. A state S is defined as a finite set of fluents. An action $a \in \mathcal{A}$ is of the form $a = pre(a) \rightarrow eff(a)$, where $pre(a)$ and $eff(a)$ are sets of fluents representing the preconditions and effects of a. Executing an action a in a world state S leads to a new world state S' as a result of applying $eff(a)$. An effect of the form $\langle v, d\rangle$ updates S' w.r.t. S, replacing the fluent $\langle v, d'\rangle \in S$ by $\langle v, d\rangle$.

Definition 1. *A MAP task is a tuple $\mathcal{T}_{MAP} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A}\rangle$. $\mathcal{AG} = \{1, \ldots, n\}$ is a finite non-empty set of agents. $\mathcal{V} = \bigcup_{i\in\mathcal{AG}} \mathcal{V}^i$, where $\mathcal{V}^i$ is the set of state variables known to an agent i. $\mathcal{I} = \bigcup_{i\in\mathcal{AG}} \mathcal{I}^i$ is a set of fluents that defines the initial state of $\mathcal{T}_{MAP}$. Since specialized agents are allowed, they may only know a subset of $\mathcal{I}$; the initial states of two agents never contradict each other. $\mathcal{G}$ is a set of fluents defining the goals of $\mathcal{T}_{MAP}$. Finally, $\mathcal{A} = \bigcup_{i\in\mathcal{AG}} \mathcal{A}^i$ is the set of planning actions of the agents.*

A *solution plan* for $\mathcal{T}_{MAP}$ is a set of actions that achieves all the goals $\mathcal{G}$ of $\mathcal{T}_{MAP}$. In a MAP task with specialized agents, the planning capabilities of each agent are defined in a separate domain file. On the other hand, since agents have different knowledge of the task, this information is also encoded in a separate problem file. The distribution of the information sources of each agent is the most natural way to preserve privacy, but sharing information might be necessary to generate a consistent joint plan. Hence, assuming two different agents in $\mathcal{AG}$, i and j, we distinguish three levels of privacy:

$$
\begin{array}{ll}
\textit{public variable } (\mathcal{V}^i_{pu}): & \forall_{i,j}\ v \in \mathcal{V}^i \cap \mathcal{V}^j \\
\textit{private variable} \text{ to agent } i\ (\mathcal{V}^i_{pr}): & v \in \mathcal{V}^i \wedge \nexists j | v \in \mathcal{V}^j \\
\textit{shared variable} \text{ of agent } i \text{ with agent } j\ (\mathcal{V}^i_{sh}): & \bigwedge \begin{cases} v \in \mathcal{V}^i \cap \mathcal{V}^j \\ \exists a \in \mathcal{A}^i | \langle v, \cdot\rangle \in eff(a) \\ \nexists a' \in \mathcal{A}^j | \langle v, \cdot\rangle \in eff(a') \end{cases}
\end{array}
$$

Thus, the set of known variables of an agent i is $\mathcal{V}^i = \mathcal{V}^i_{pu} \cup \mathcal{V}^i_{pr} \cup \mathcal{V}^i_{sh}$, the union of the non-overlapping sets of variables according to their privacy. We adopt the STRIPS model of the multi-agent PDDL task definition used in the Competition of Distributed and Multi-Agent Planning (CoDMAP, http://agents.fel.cvut.cz/codmap/) and, therefore, the domain of each variable $v \in \mathcal{V}$ is reduced to $\mathcal{D}_v = \{d, \neg d\}$. Instead of using the factored representation of MAP tasks in CoDMAP, we used the unfactored representation, where a single information source is defined for all the agents. Thus, our aim is precisely to identify the planning task associated to each agent as in [8] and generate a planning graph equivalent to a Distributed Planning Graph [22]. Actions and propositions are defined using the following templates to better identify their owner:

- *(operator agent parameters)*: $a \in \mathcal{A}^{\text{agent}}$, an action performed by *agent*.
- *(property agent parameters)*: $v \in \mathcal{V}^{\text{agent}}$, a property of *agent*.
- *(property parameters)*: a property of an object in the domain (public proposition).

For instance, in the Logistics example shown in Figure 1 (instance `prob-4-0` from the ICAPS 2000 International Planning Competition http://ipc00.icaps-conference.org/), we consider trucks and planes as agents, therefore an action of the truck agent $tru1$ $(drive\text{-}truck\ tru1\ pos1\ apt1\ cit1)$ denotes that $tru1$ moves from position $pos1$ to the airport $apt1$ within city $cit1$. Similarly, a proposition owned by $tru1$ is $(at\ tru1\ pos1)$, representing that $tru1$ is at the position $pos1$. And, finally, $(at\ obj11\ pos1)$ represents the location of the package $obj11$ in the position $pos1$ as a public proposition.

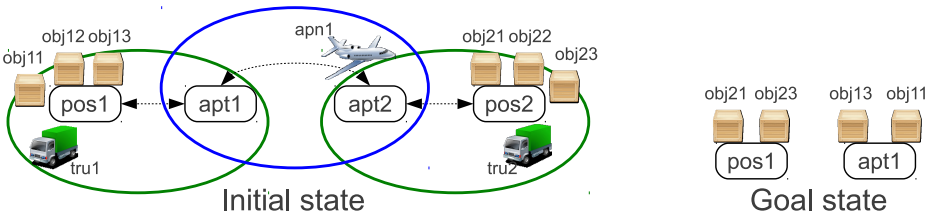

Fig. 1. Initial and goal states of the problem instance of the Logistics domain.

Our model follows the schema shown in Figure 2. The SAT-encoding of our approach is an extension of the Graphplan-based parallel encoding [12] and the proposal introduced in [2]. The compiler processes the domain and problem files and generates a global planning graph for a time horizon T. Each variable $v \in \mathcal{V}$ and action $a \in \mathcal{A}$ is replicated for each time $t \in \{0, \ldots, T\}$ at which it appears in the planning graph, and encoded into a CNF formula Φ_T such that Φ_T is satisfiable iff there is a solution plan within the time horizon T.

A planning agent handles multiple variables, but an ABT agent has a single variable only. Can we build a planning agent from several ABT agents? The answer is yes [6].The basic idea is that the planning agent $i \in \mathcal{AG}$ can be seen as a multitude of ABT agents (one per variable), working to satisfy the propositional formula. Since ABT with one variable per agent is correct and complete [21], the realization of the planning agent i via ABT agents is also correct and complete.

Since ABT requires a total order, each propositional variable is adequately numbered (see Section 4). Assuming that ABT uses a lexicographical ordering (a variable with lower index is located higher in the total order than a variable with greater index), each agent $i \in \mathcal{AG}$ is in charge of the propositional variables corresponding to its variables and actions, and it will receive $\Phi_T^i \subseteq \Phi_T$, i.e., the subset of clauses such that the highest numbered variable corresponds to one mapped variable from $\mathcal{V}^i \cup \mathcal{A}^i$.

When an agent i attempts to satisfy a formula Φ_T^i that involves a public variable with another agent, it is necessary some mechanism by which both agents are informed of a change in the variable. In order to do so, we replicate public variables for each agent, we add then equality constraints between replicas to each Φ_T^i, and the agents'

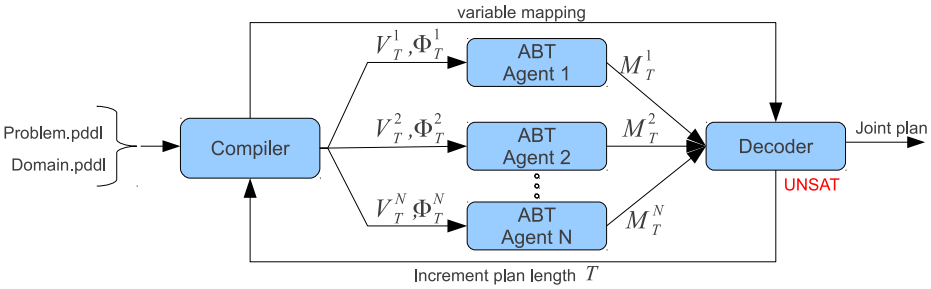

Fig. 2. Multi-agent sat-based planning schema. $V_T^1 \cap V_T^2 \cdots \cap V_T^n \neq \emptyset$ if planning agents have public variables (as the problem in Figure 1).

formulas are solved concurrently with a distributed SAT-solver (in this case ABT). We define $replace(\Phi, v, v')$ a function that replaces all the occurrences of variable v in the propositional formula Φ with the new variable v'. Variables and clauses are updated,

- $\mathcal{V}_{pu}^i = \mathcal{V}_{pu}^i \setminus \{v\} \bigcup_{j \neq i \in \mathcal{AG}} \{v_j\}$
- $\mathcal{V}_{sh}^i = \mathcal{V}_{sh}^i \cup \{v_i\}$
- $\Phi_T^i = replace(\Phi_T^i, v, v_i) \bigwedge_{j \neq i \in \mathcal{AG}} \{v_i \vee \neg v_j\} \wedge \{\neg v_i \vee v_j\}$

4 Resolution using ABT

According to the description of Section 3, each agent i receives the set of propositional variables $\mathcal{V}^i \cup \mathcal{A}^i$ that it controls and a formula CNF Φ_T^i, to be solved cooperatively with the rest of agents. This can be done as follows.

For a moment, let us consider the whole formula Φ. We know that, if there is a total order among the propositional variables of the formula, ABT is a correct and complete algorithm when executing on each variable of the formula. The total order can be easily obtained by ordering first the variables of agent 1, second the variables of agent 2, and so on. Inside each agent, the total order may be arbitrary (in the next paragraph we will require a particular order). Then, executing simple ABT on each propositional variable, we will obtain a solution of the considered instance (if it is satisfiable) or we will prove that no solution exists (if it is unsatisfiable). ABT processes exchange messages; inside the same agent, message exchange is implemented by shared memory, while between agents message exchange is done through a communication network.

If we require a particular order inside each agent, we can achieve a desirable property as explained next. If we order propositional variables coming from originally private variables last in the total order inside each agent, we can assure that during ABT execution the values of these propositional variables will not go out from each agent, as proved by the following theorem.

Theorem 1. *The values of propositional variables coming from originally private variables do not go out from its respective agents during ABT execution.*

Proof. The value of a propositional variable x is disseminated via $OK?$ messages, which are sent to other variables y below in the order, such that x appear with y in the same clause. Given that propositional variables coming from originally private variables do not appear in clauses with variables of other agents, the destination of these $OK?$ messages necessarily has to be the same agent.

Remains to be seen that x has not received an ADL message from a variable z of another agent located below x in the total order. Let us see that this is not possible. Let us assume that x has received such message from a variable z of another agent. How does z know the value of x? Since x has sent its value to variables in its own agent, it is impossible for z to know anything about x. So z cannot sent such message. □

In summary, with a simple total order, existing ABT is able to solve the MAP problem. In addition, with a particular total order inside each agent, we can prove that ABT will never disseminate values that should remain private inside each agent.

5 Implementation

As a proof of concept, we have run several experiments described in the following. All domains and problems are extracted from the CoDMAP as part of the DMAP workshop in the ICAPS'15. Domain files have been modified to the *STRIPS* version of PDDL as stated in Section 3. The SAT encoding has been done using BlackBox [14]. Table 1 shows the statistics gathered from several problems of three planning domains, Zenotravel, Driverlog and Logistics. The values correspond to the encoding generated for the minimum satisfiable makespan, where #A and #P are the total number of actions and the total number of propositions contained in the planning graph respectively; %Pu is the percentage of public variables within the associated set; #V is the total number of variables mapped to SAT; and finally #C is the total number of clauses encoded in CNF. We differentiate between the classical mono-agent encoding of the planning problem and the multi-agent version The mono-agent encoding is solved by an off-the-shelf SAT solver. In the multi-agent version $\mathcal{AG}$ refers to the number of participant agents.

Although ABT is able to solve completely the multi-agent planning instances reported, it is quite slow. We plan to improve our ABT-based implementation, in order to achieve empirical results closer to the performance reported by centralized ones. [5]

Table 2 shows the plan returned by our implementation for the logistics problem in Fig. 1. This is the plan of minimal makespan, as expected from an incrementally time horizon calculation. However, we can observe that the individual plans contain some *unnecessary* actions like, for instance, the first two consecutive reversible actions of agents $apn1$ and $tru1$. This is due to the ordering of the variables within the formulas and the absence of a planning heuristic that helps guide the search towards action minimization. As for future work, we intend to explore planning-specific variable selection strategies for SAT that have been successfully applied in single-agent planning [17].

[5] Typically, distributed asynchronous algorithms show lower performance than centralized ones because, in addition to the required computation, they have to face situations not considered by centralized approaches (they have to be ready to handle any asynchronous change).

Table 1. Planning problems' configuration and SAT-encoding statistics.

		Configuration			Classical		Multi-Agent		
Domain	Problem	Makespan	#A (%Pu)	#P (%Pu)	#V	#C	#V	#C	$\mathcal{AG}$
Zenotravel	pfile3	4	265 (0.22)	151 (0.56)	416	5276	560	6605	2
	pfile4	5	392 (0.22)	196 (0.58)	588	10847	786	12749	2
	pfile5	5	467 (0.18)	199 (0.59)	666	20235	869	22692	2
	pfile6	5	689 (0.14)	237 (0.55)	926	32402	1153	35988	2
Driverlog	pfile1	6	73 (0.70)	91 (0.74)	164	223	282	621	2
	pfile2	7	609 (0.34)	341 (0.76)	950	13936	1416	18159	2
	pfile3	7	564 (0.37)	340 (0.76)	904	11465	1372	15576	2
	pfile4	7	881 (0.25)	389 (0.71)	1270	32321	2264	48001	3
Logistics	prob-4-0	9	382 (0.31)	278 (0.53)	660	2585	1192	6055	3
	prob-5-0	9	484 (0.31)	347 (0.52)	831	3406	1495	7788	3
	prob-6-0	9	573 (0.31)	391 (0.54)	964	4471	1740	9853	3
	prob-7-0	12	1356 (0.27)	798 (0.53)	2154	19388	4545	45998	4

Table 2. Plan generated for the instance `prob-4-0` of the Logistics domain.

Time	Agent apn1	Agent tru1	Agent tru2
1	(fly-airplane apn1 apt2 apt1)	(drive-truck tru1 pos1 apt1 cit1)	(load-truck tru2 obj23 pos2) (load-truck tru2 obj21 pos2)
2	(fly-airplane apn1 apt1 apt2)	(drive-truck tru1 apt1 pos1 cit1)	(drive-truck tru2 pos2 apt2 cit2)
3	()	()	(unload-truck tru2 obj21 apt2) (unload-truck tru2 obj23 apt2)
4	(load-airplane apn1 obj23 apt2) (load-airplane apn1 obj21 apt2)	(load-truck tru1 obj11 pos1) (load-truck tru1 obj13 pos1)	()
5	(fly-airplane apn1 apt2 apt1)	(drive-truck tru1 pos1 apt1 cit1)	()
6	(unload-airplane apn1 obj23 apt1) (unload-airplane apn1 obj21 apt1)	(unload-truck tru1 obj11 apt1) (unload-truck tru1 obj13 apt1)	()
7	()	(load-truck tru1 obj23 apt1) (load-truck tru1 obj21 apt1)	()
8	()	(drive-truck tru1 apt1 pos1 cit1)	()
9	()	(unload-truck tru1 obj23 pos1) (unload-truck tru1 obj21 pos1)	()

6 Conclusion

We have presented an approach to solve MAP instances that, in the context of interleaving planning and coordination, considers its translation in a propositional formula, which is distributed among agents and solved asynchronously by the generic ABT algorithm (originally presented for solving distributed constraint satisfaction problems). Each planning agent implements a number of ABT processes, one per elementary propositional variable. Interestingly, an adequate ordering of variables allows this approach to assure privacy of the variables originally considered private in the initial PDDL formulation. Experimental results on some benchmark instances extracted from planning competitions show the feasibility of our approach.

References

1. A. B. Baker. The hazards of fancy backtracking. In *Proc. of AAAI-94*, pages 288–293, 1994.
2. M. Benedetti and L. Aiello. Sat-based cooperative planning: A proposal. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *LNCS*, pages 494–513. Springer Berlin Heidelberg, 2005.
3. A. Bonisoli, A. E. Gerevini, A. Saetti, and I. Serina. A privacy-preserving model for the multi-agent propositional planning problem. In *Proc. ECAI*, pages 973–974, 2014.
4. D. Borrajo. Multi-agent planning by plan reuse. In *Proc. AAMAS*, pages 1141–1142, 2013.
5. R. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proc. ICAPS*, pages 28–35, 2008.
6. D. Burke and K. Brown. Efficient handling complex local problems in distributed constraint optimisation. In *Proc. ECAI*, pages 701–702, 2006.
7. J. Cox and E. Durfee. Efficient and distributable methods for solving the multiagent plan coordination problem. *Multiagent and Grid Systems*, 5(4):373–408, 2009.
8. M. Crosby, M. Rovatsos, and R. Petrick. Automated agent decomposition for classical planning. In *Proc. ICAPS*, pages 46–54, 2013.
9. M. de Weerdt and B. Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009.
10. Y. Dimopoulos, M. A. Hashmi, and P. Moraitis. μ-satplan: Multi-agent planning as satisfiability. *Knowledge-Based Systems*, 29:54–62, 2012.
11. J. Hoffmann, C. P. Gomes, and B. Selman. Structure and problem hardness: Goal asymmetry and DPLL proofs in sat-based planning. *Logical Methods in Computer Science*, 3(1), 2007.
12. H. Kautz, D. Mcallester, and B. Selman. Encoding plans in propositional logic. pages 374–384, 1996.
13. H. Kautz and B. Selman. Planning as satisfiability. pages 359–363, 1992.
14. H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. pages 58–60, 1998.
15. R. Nissim and R. Brafman. Multi-agent A* for parallel and distributed systems. In *Proc. AAMAS*, pages 1265–1266, 2012.
16. R. Nissim, R. Brafman, and C. Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proc. AAMAS*, pages 1323–1330, 2010.
17. J. Rintanen. Planning as satisfiability: Heuristics. *Artif. Intell.*, 193:45–86, Dec. 2012.
18. H. Tonino, A. Bos, M. de Weerdt, and C. Witteveen. Plan coordination by revision in collective agent based systems. *Artificial Intelligence*, 142(2):121–145, 2002.
19. A. Torreño, E. Onaindia, and O. Sapena. FMAP: distributed cooperative multi-agent planning. *Appl. Intell.*, 41(2):606–626, 2014.
20. A. Torreño, O. Sapena, and E. Onaindia. Global heuristics for distributed cooperative multi-agent planning. In *Proc. ICAPS*, 2015.
21. M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Know. and Data Engin.*, pages 673–685, 1998.
22. J. F. Zhang, X. T. Nguyen, and R. Kowalczyk. Graph-based multiagent replanning algorithm. In *Proc. AAMAS'07*.