Abstract

Multi-agent planning is a fundamental problem in multi-agent systems that has acquired a variety of meanings in the relative literature. In this paper we focus on a setting where multiple agents with complementary capabilities cooperate in order to generate non-conflicting plans that achieve their respective goals. We study two situations. In the first, the agents are able to achieve their subgoals by themselves, but they need to find a coordinated course of action that avoids harmful interactions. In the second situation, some agents may ask the assistance of others in order to achieve their goals. We formalize the two problems in a more general way than in previous works, and present algorithms for their solution. These algorithms are based on an underlying classical planner which is used by the agents to generate their individual plans, but also to find plans that are consistent with those of the other agents. The procedures generate optimal plans under the plan length criterion, but they can be adapted to other criteria as well. The central role that has been given to the classical planning algorithm, can be seen as an attempt to establish a stronger link between classical and multi-agent planning.

1 Introduction

Multi-agent coordination and cooperation are important issues in the multi-agent field. Several works have been proposed in the literature covering different aspects of the problem of coordinating the plans of several agents operating in the same environment. These include situations where plan generation is distributed (see e.g.[9], [19]), meaning that several agents with complementary specialties have to cooperate in order to produce a complex plan that none of them could generate alone, situations where planning is centralized and plan execution is distributed (see e.g. [3], [10]) meaning that a multi-agent plan is produced in a centralized fashion and then it is decomposed into sub-plans that are assigned to different agents and may be executed in parallel, and finally situations where both planning and execution are distributed (see e.g. [5], [4], [6], [1], [17], [16], [15]) meaning that several agents interact in order to generate plans that then will be carried out by themselves in order to achieve individual or common goals. However, only few works (see e.g. [1], [3], [7]) tackle aspects of the cooperation problem in the context of multi-agent planning and more especially the problem where an agent might need the assistance of another agent in order to achieve his goal.

In this paper we study both problems of coordination and cooperation of multiple agents. In particular, we consider two scenarios that we believe cover an important number of applications. In the first, agents have individual (private) goals that they can achieve by themselves. The agents can generate and execute their plans independently. However, as they operate in the same environment, conflicts may arise. Therefore, they need to coordinate their course of action in order to avoid harmful interactions. We will call this situation multi-agent coordination. In the second scenario, which is a special case of multi-agent cooperation, an agent can ask some other agent to establish preconditions of actions that appear in his plan. We will call this situation multi-agent assistance.

This work aims at tackling the above problems by utilizing a classical planning algorithm. We envisage the use of multi-agent planning in application domains where an agent needs to plan in two different situations. In the first, the agent is in a situation where its actions are known to be completely independent of the actions of all other agents. In the second situation the actions of the agent can affect and be affected by the action of the other agent. In the current literature these two situations require different treatment. In the first case a classical planning can solve the associated planning problem, whereas in the second case some additional algorithm or mechanism has the responsibility of co-
ordinating the actions of the agents. In this paper we tackle both situations with a classical planner.

More specifically, we define formally the two problems of coordination and cooperation, and propose branch and bound algorithms for their solution. The domain theories of the agents are represented in the STRIPS language, and the solution algorithms rely on SATPLAN [11] for plan generation, one of the most successful systems for optimal classical planning [8]. The agents generate individual plans using SATPLAN, which they send to the other agents, that, again using SATPLAN, attempt to expand to a joint plan. The procedure iterates over all individual plans until an optimal, wrt plan length, joint plan is found. When a new joint plan is generated, its length serves as an upper bound in the search for better plans. Apart from plan length, our procedures can be adapted to other plan quality criteria, provided that the underlying classical planner can generate optimal plans wrt these criteria. Finally, our algorithms define the way agents communicate through the exchange of messages.

On a high level, an important difference of our approach from other works, is that it handles both coordination and cooperation in a uniform way. Moreover, the use of a classical planner, brings recent and future advances in classical planning into multi-agent planning. On a more technical level, and as far as coordination is concerned, the works closer to ours are [5] and [6]. Although there are similarities to the techniques presented in [5], our approach tackles a more general problem. The main difference is that the input to the algorithm of [5] is a fixed combination of individual plans that need to be coordinated, whereas our coordination procedure generates all such possible combinations from which it selects an optimal one. On the other hand, the problem we tackle here bares a resemblance to the one addressed in [6]. However, our solution to the problem presents certain advantages. Firstly, our algorithm finds optimal solutions, which is not the case for [6] when there are “positive” interactions between the actions. Secondly, if optimality is not required, our algorithm can terminate as soon as a consistent solution of acceptable quality is found, or a time limit is reached, which is not the case for [6], where only the optimal solution (assuming no positive interactions between actions) is returned when the process terminates. It worths noting that the idea of using satisfiability for solving the closely related problem of plan merging has been considered in [14]. However, in plan merging no new actions can be added in the new plan, i.e., an action can not belong to the final plan if it does not appear in some of the plans that are merged. Moreover, [14] does not study the parallel encoding of planning into satisfiability that we investigate here.

In a few words, our approach is an extension of techniques similar to those presented in [5] to the more general problem studied in [6], with the additional advantage of an efficient underlying classical planner.

The rest of the paper is organized as follows. Section 2 presents how the planning as satisfiability framework can be used and extended to meet the needs of a multi-agent setting. Sections 3 and 4 define cooperation and assistance, and present algorithms for their solution. Section 5 discusses future work and concludes.

## 2 Propositional Satisfiability based Planning

We assume that the agents’ planning domain theories are described in the STRIPS language, and denote by $D_\alpha$ the set of actions that appear in the domain theory of agent $\alpha$. To generate their plans the agents use the SATPLAN system [11]. The rationale behind choosing the propositional satisfiability approach to planning is twofold. First, it is one of the most computationally effective approaches to optimal (wrt plan length) STRIPS planning [8, 13]. Second, it can be easily extended to accommodate the needs of our multi-agent planning scenarios.

We assume that the reader is familiar with the propositional satisfiability encoding of STRIPS planning. Here we recall very briefly the basics of SATPLAN approach to planning. First, a plan length $k$ is assumed, and the planning problem is translated into a propositional theory (set of clauses). If the resulting satisfiability problem is solvable, a plan of length $k$ is extracted from the model that has been found. Otherwise, the plan length is set to $k + 1$ and the procedures iterates.

Among the several ways to transform a planning problem into satisfiability one, we adopt the Graphplan-based parallel encoding [11]. The facts of the (fully specified) initial state and the final state are translated into literals. The propositional theory also contains clauses that constraint actions to imply their preconditions, and fluents to imply the disjunction of all actions that have these fluents in their add-effects. Finally, conflicting actions are declared as mutual exclusive through suitable binary clauses that are added to the theory. For a description of the latest version of SATPLAN, please refer to [13].

The SATPLAN procedure is invoked through the call $\text{ComputePlan}(T, G, L, C, P)$, where $T$ is a CNF theory that includes the agent’s domain theory and initial state, $G$ is the set of goals of the agent, $L$ is an upper bound on the length of the generated plan (ie. if $l(P)$ is the length of the generated plan, $l(P) < L$ holds), and $C$ is a (possibly empty) set of additional constraints that are taken into account by the planner in the plan generation phase. The planner returns a plan $P$ that satisfies these constraints or reports failure if such a plan does not exist, by returning $\text{fail}$ in the argument $P$. Moreover, the procedure $\text{ComputeNewPlan}$ is the same as $\text{ComputePlan}$, with the difference that each time it is invoked, it returns a new plan different than those
generated by the previous \textit{ComputeNewPlan} calls.
In the following we assume that a plan is a set of temporal propositions of the form \( A(t) \), where \( A \) is an action and \( t \) is a time point, meaning that action \( A \) executes at time \( t \). If \( D \) is a domain theory, \( I \) an initial state, \( P \) a plan and \( G \) a set of goals, the notation \( P \models_{D, I} G \) denotes that \( P \) is a plan for goal \( G \) in the domain \( D \) with initial state \( I \), under the standard STRIPS semantics. When there is no possibility for confusion, we simply write \( P \models G \).

For the purposes of multi-agent assistance we slightly extend the STRIPS language to accommodate the representational needs of cooperation between the agents. In the extended language, the preconditions \( \text{prec}(A) \) of an action \( A \) is the union of the sets \( \text{norm}_{\text{prec}}(A) \) and \( \text{extern}_{\text{prec}}(A) \). The set \( \text{norm}_{\text{prec}}(A) \) contains normal action preconditions, i.e., fluents that must hold before the execution of the action for that action to succeed. The elements of the set \( \text{extern}_{\text{prec}}(A) \) are fluents that the agent may request some other agent to bring about in the world. Therefore, the agent can assume that these fluents will be true in the world when needed, and ignore them during planning. In the context of the SATPLAN framework this means that the agent plans by taking into account only the propositions in the sets \( \text{norm}_{\text{prec}} \) and ignores \( \text{extern}_{\text{prec}} \). The call to the planner now becomes \( \text{ComputePlan}(T, G, L, C, < P, R >) \) where instead of returning a plan \( P \), a pair \( < P, R > \) is returned, where \( R = \{ p(t) | A(t) \in P \text{ and } p \in \text{extern}_{\text{prec}}(A) \} \). We call the elements of the set \( R \), \textit{assistance requests}. The example below presents a case of multi-agent assistance.

\textbf{Example 1} Consider the domain theory \( D_A \) of agent \( A \) that includes the operators \( \text{pickup}(X, L) \) and \( \text{putdown}(X, L) \) for picking up and putting down an object \( X \) at location \( L \) respectively, with the usual preconditions and effects. Moreover, \( D_A \) contains the action \( \text{move}(X, Y) \) for moving from location \( X \) to location \( Y \), with preconditions \( \text{norm}_{\text{prec}}(\text{move}(X, Y)) = \{ \text{at}(X, Y) \} \) and \( \text{extern}_{\text{prec}}(\text{move}(X, Y)) = \{ \text{opendoor}(X, Y) \} \), and the usual effects. The external precondition \( \text{opendoor}(X, Y) \) means that the agent expects that some other agent will open the doors for him. Assume the initial state \( I_A = \{ \text{at}(l_1), \text{at}(l_2) \} \) and the set of goals \( G_A = \{ \text{at}(\text{obj}, l_2) \} \). A plan for this problem is \( P_A = \langle \{ \text{pickup}(\text{obj}, l_1, 0), \text{move}(l_1, l_2, 1), \text{putdown}(\text{obj}, l_2, 2) \}, \{ \text{opendoor}(l_1, l_2, 1) \} \rangle \). Note that for this plan to succeed, some other agent will bring about \( \text{opendoor}(l_1, l_2) \) at time \( I \).

Note that the domain theory of an agent may contain different versions of the same action, say \( A_1 \) and \( A_2 \), that differ only in the elements they contain in their sets \( \text{norm}_{\text{prec}} \) and \( \text{extern}_{\text{prec}} \), i.e., \( \text{prec}(A_1) = \text{prec}(A_2) \) but \( \text{norm}_{\text{prec}}(A_1) \neq \text{norm}_{\text{prec}}(A_2) \).

\section{Multi-Agent Coordination}

In a multi-agent coordination scenario, a number of agents need to generate individual plans that achieve their respective goals and are not in conflict with each other. We restrict ourselves to the case of two agents, \( \alpha \) and \( \beta \), and study a multi-agent coordination scenario that is defined by the following characteristics:

- Each agent is able to achieve his goals by himself. This distinguishes coordination from cooperation, which is discussed in the next section. Moreover, agents have different capabilities. In the most extreme case, the effects of the actions of the agents are disjoint.
- Plan length is the criterion for evaluating the quality of both the individual and the joint plans, with preference given to the joint plan length. Agents seek to minimize the length of the joint plan, even in the case where this leads to non-optimal individual plans. The algorithm that we develop in the following can accept other optimization criteria as well, without major modifications, but here we concentrate on plan length as it is one of the most commonly accepted domain independent plan quality criteria.

The coordination problem is defined formally as follows.

\textbf{Definition 1} (Coordination Problem). Given two agents \( \alpha \) and \( \beta \) with goals \( G_\alpha \) and \( G_\beta \), initial states \( I_\alpha \) and \( I_\beta \), and sets of actions \( D_\alpha \) and \( D_\beta \) respectively, find a pair of plans \( (P_\alpha, P_\beta) \) such that

- \( P_\alpha \models_{D_\alpha, I_\alpha} G_\alpha \) and \( P_\beta \models_{D_\beta, I_\beta} G_\beta \)
- \( P_\alpha \) and \( P_\beta \) are non-conflicting

We refer to the plans \( P_\alpha \) and \( P_\beta \) as individual plans, and to the pair \( (P_\alpha, P_\beta) \) as joint plan. Moreover, we use the term joint plan to also refer to the plan \( P_\alpha \cup P_\beta \). Observe that if \( (P_\alpha, P_\beta) \) is a solution to the coordination problem, then \( P_\alpha \cup P_\beta \models D_\alpha \cup D_\beta, I_\alpha \cup I_\beta \ G_\alpha \cup G_\beta \). The plan length of a joint plan \( (P_\alpha, P_\beta) \) is defined as \( \max(l(P_\alpha), l(P_\beta)) \).

The problem we investigate in this section is how to generate an optimal solution \( (P_\alpha, P_\beta) \) wrt plan length in a distributed manner, where each agent generates its individual plan using SATPLAN. A solution \( (P_\alpha, P_\beta) \) to a coordination problem is optimal wrt plan length if there is no other solution \( (P'_\alpha, P'_\beta) \) to the problem such that \( \max(l(P'_\alpha), l(P'_\beta)) < \max(l(P_\alpha), l(P_\beta)) \). This notion of “distributed” optimality is weaker than plan length optimality that can achieved by a centralized planner as demonstrated in the following simple example.
Example 2 Consider two agents $\alpha$ and $\beta$ with domain theories that contain the actions $D_\alpha = \{A_1, A_2, A_3, A_4\}$ and $D_\beta = \{B_1, B_2, B_3, B_4\}$ with the following preconditions and (positive) effects: $\text{prec}(A_1) = \{a_1\}$, $\text{eff}(A_1) = \{p_1\}$, $\text{prec}(A_2) = \{p_1\}$, $\text{eff}(A_2) = \{p_2\}$, $\text{prec}(A_3) = \{p_2\}$, $\text{eff}(A_3) = \{q_2\}$, $\text{prec}(B_1) = \{a_1\}$, $\text{eff}(B_1) = \{q_1\}$, $\text{prec}(B_2) = \{q_1\}$, $\text{eff}(B_2) = \{q_2\}$, $\text{prec}(B_3) = \{q_2\}$, $\text{eff}(B_3) = \{q_3\}$, $\text{prec}(B_4) = \{a_1\}$, $\text{eff}(B_4) = \{p_2\}$. Assume that $I_\alpha = I_\beta = \{a_1\}$ and $G_\alpha = \{p_3\}$, $G_\beta = \{q_3\}$. The optimal distributed solution to this problem is $(P_\alpha, P_\beta)$ with $P_\alpha = \{A_1(0), A_2(1), A_3(2)\}$ and $P_\beta = \{B_1(0), B_2(1), B_3(2)\}$ and plan length 3. Notice however that the optimal solution that can be found by a centralized planner to the problem with initial state $I = I_\alpha$, goal $G_\alpha \cup G_\beta$ and actions $D_\alpha \cup D_\beta$, is $P = \{A_4(0), B_4(0), A_3(1), B_3(1)\}$ which is shorter than the optimal solution to the corresponding coordination problem.

Therefore, our approach to optimality can be seen as an "opportunistic" one, in the sense that agents plan for their goals independently, but they may use actions of other agents in their plans if these actions contribute to the achievement of their own goals.

As it is well known, there can be "negative" (i.e. conflicts) and "positive" (i.e. redundant actions) interactions between the actions of the plans of different agents. The "negative" interactions come from two different sources that are discussed below. The "positive" interactions, and the way that they are taken into account, are discussed in section 3.1.

- **Causal link threatening.** This conflict is well known in the context of partial order planning [18]. Let $A_1(t_1)$ and $A_2(t_2)$ be two actions of a plan $P$ such that $t_1 < t_2$ and $A_1(t_1)$ is the latest action of the plan $P$ that adds the precondition $p$ of action $A_2(t_2)$. Then, we say that there is causal link between time points $t_1$ and $t_2$ related to $p$, denoted by the triple $(t_1, t_2, p)$. Furthermore, if $p$ is a precondition of action $A(t)$, $p$ appears in the initial state, and there is no action in plan $P$ that adds $p$ and is executed at some time point $t' < t$, then there is a causal link $(0, t, p)$ in $P$. Moreover, if $A(t)$ is the last action that adds a goal $g$, there exists a causal link $(t, t_{fin}, g)$, where $t_{fin}$ is the plan length.

Finally, if $p$ is a proposition that belongs both to the initial and the final state of planning problem, and there is no action in plan $P$ that adds $p$, then $P$ contains the causal link $(0, t_{fin}, p)$. An action $A(t)$ threatens the causal link $(t_1, t_2, p)$ if $t_1 \leq t \leq t_2$ and $A$ deletes $p$.

- **Parallel actions interference.** This conflict was introduced in Graphplan [2]. Two actions interfere if they are executed in parallel and one deletes the preconditions or the add effects of the other.

Given a plan $P$ and a set of actions $S$, we define the set of constraints $C_P$ that when satisfied by a plan $P'$ guarantee that $P$ and $P'$ are non-conflicting. The set $C_P$ has the form of negated action occurrence literals that represent actions that must not be included in the plan $P'$. Formally, the set is defined as $C_P = \{\neg A(t) | A \in S \land A(t)$ threatens a causal link $(t_1, t_2, p)$ of $P \} \cup \{\neg A(t) | A \in S \land t < t' \}$ where $t'$ is the initial time point of $A(t)$.

3.1 The Coordination Algorithm

In the coordination algorithm we present in this section, we assume two agents $\alpha$ and $\beta$ with goals $G_\alpha$ and $G_\beta$, domain theories $D_\alpha$ and $D_\beta$ and initial states $I_\alpha$ and $I_\beta$ respectively. To solve the coordination problem as specified in definition 1, each agent uses the SATPLAN algorithm for plan generation, and exchanges messages with the other agent. To simplify the presentation, we assume that all joint plans are of different length.

The SATPLAN system is employed in two, slightly different, ways. First it is called, say by agent $\alpha$, to compute a new plan $P_\alpha$ that achieves his goals $G_\alpha$, without taking into account possible conflicts with the plan of agent $\beta$. This task is carried out by the call $\text{ComputePlan}(T_\alpha, G_\alpha, \emptyset, P_\alpha)$, where $L$ is the length of the best current joint plan. The individual plan $P_\alpha$ is sent to agent $\beta$ as a candidate sub-plan of a new joint plan that must be better (wrt plan length) than the best joint plan found so far. Similarly, agent $\alpha$ receives candidate sub-plans from agent $\beta$. Upon processing a proposed plan $P_\beta$, agent $\alpha$ invokes the planning algorithm by calling $\text{ComputePlan}(T_\alpha \cup T'_\beta \cup P_\beta, G_\alpha, L, C_{P_\alpha} \cap P_\beta)$, that returns the best plan $P_\alpha$ that achieves the goals $G_\alpha$ and has length that is shorter than $L$, the length of the best current joint plan. The CNF input theory to the SATPLAN procedure is the union of the set $T_\alpha$, which encodes the agent’s domain theory and initial state, and the sets $T'_\beta$ and $P_\beta$ which account for possible "positive" interactions between $P_\beta$ and the plan that agent $\alpha$ will generate. The set $T'_\beta$ contains (the CNF representation of) implications of the form $A(t) \rightarrow e_i(t+1) \land e_j(t+1) \land \ldots \land e_n(t+1)$ for each action $A$ in $D_\beta$, positive effect $e_i$ of $A$, $1 \leq i \leq n$, and time point $t$ in the plan horizon. The set $P_\beta$ is defined as $P_\beta = \{A(t) | A(t) \in P_\beta \} \cup \{\neg A(t) | A \in D_\beta \land A(t) \not\in P_\beta\}$. The sub-theory $T'_\beta \cup P'_\beta$ entails all add effects of actions that are executed by agent $\beta$, and therefore they do not need to be re-established by agent $\alpha$. 


Moreover, the plan $P_a$ returned by the call $\text{ComputePlan}(T_a \cup T'_b \cup P'_b, G_a, L, C_{P_a}^D, P_a)$, satisfies the constraints $C_{P_a}^D$, which means that the two plans $P_a$ and $P_b$ are non-conflicting, therefore $P_a \cup P_b \models \Delta_{D_a \cup D_b, \alpha \cup \beta} G_a \cup G_b$. Furthermore, the joint plan $(P_a, P_b)$ (equivalently $P_a \cup P_b$) becomes the best current joint plan.

It is important to discuss a complication that may arise the first time, say agent $\alpha$, receives a plan from agent $\beta$. Agent $\alpha$ invokes SATPLAN by calling $\text{ComputePlan}(T_a \cup T'_b \cup P'_b, G_a, L, C_{P_a}^D, P_a)$, with the plan length parameter $L$ having the value $\infty$, since no joint plan has been computed so far. Assume that there is no plan that achieves the goals $G_\alpha$, while satisfying all the constraints of $C_{P_a}^D$. Since SATPLAN has no means to determine that the problem is unsolvable, it will run into an infinite sequence of unsuccessful attempts to generate a plan. One practical way to overcome this problem is to impose a limit to the computation time that SATPLAN can use each time it is called by $\text{ComputePlan}$. If this timeout is reached, $\text{ComputePlan}$ returns failure. Another way is to combine SATPLAN and GRAPHPLAN as this is done in BLACKBOX [12], and use GRAPHPLAN’s mechanism for determining insolvability. We are currently investigating the possibility of developing a more efficient method for determining insolvability due to constraints imposed by the plans of other agents. In the following we assume that SATPLAN returns failure on unsolvable problems.

The following example presents a case of insolvability as discussed above, and illustrates the parameters of the $\text{ComputePlan}$ procedure.

**Example 3** Consider blocks world domain instance involving two agents $A$ and $B$ with initial state $I = \{ \text{on}(a, \text{table}), \text{on}(b, \text{table}), \text{on}(c, b), \text{on}(d, \text{table}) \}$ and goals $G_A = \{ \text{on}(a, c) \}$ and $G_B = \{ \text{on}(b, d) \}$ respectively. Assume that the only operator in the domain of agent $A$ is $\text{moveA}(X, Y, Z)$ for moving block $X$ from location $Z$ to location $Y$, with the usual preconditions and effects. Similarly, agent $B$ has the operator $\text{moveB}(X, Y, Z)$.

Assume that the call $\text{ComputePlan}(T_A, G_A, L, \emptyset, P_A)$ returns the plan $P_A = \{ \text{moveA}(a, c, \text{table}, 0) \}$ and $\text{ComputePlan}(T_B, G_B, L, \emptyset, P_B)$ returns the plan $P_B = \{ \text{moveB}(c, a, b, 0), \text{moveB}(b, d, \text{table}, 1) \}$.

Consider first the input parameters $T'_A$, $P'_A$ and $C_{P'_A}^D$ of the call $\text{ComputePlan}(T_B \cup T'_A \cup P'_A, G_B, L, C_{P'_A}^D, P_B)$. The set $T'_A$ contains the ground instances of the propositions $\text{moveA}(X, Y, Z, T) \rightarrow \text{clear}(Z, T+1) \land \text{on}(X, Y, T+1)$, whereas $P'_A$ contains the proposition $\neg \text{moveA}(a, c, \text{table}, 0)$ and the set of propositions $\neg \text{moveA}(X, Y, Z, T)$ for all other combinations of values to the variables $X, Y, Z, T$. The set $C_{P'_A}^D$ contains the propositions $\neg \text{moveB}(a, Y, \text{table}, 0)$, $\neg \text{moveB}(Y, a, Z, 0)$, $\neg \text{moveB}(Y, c, Z, 0)$ and $\neg \text{moveB}(a, Y, c, 0)$, because these actions delete a precondition or an add effect of $\text{moveA}(a, c, \text{table}, 0)$. Additionally, $C_{P'_A}^D$ contains the propositions whose preconditions or add effects are deleted by $\text{moveA}(a, c, \text{table}, 0)$, i.e. $\neg \text{moveB}(c, Y, Z, 0)$, $\neg \text{moveB}(Y, c, Z, 0)$, $\neg \text{moveB}(a, Y, \text{table}, 0)$, $\neg \text{moveB}(Y, Z, c, 0)$, and $\neg \text{moveA}(a, \text{table}, 0)$.

Furthermore, since $\text{moveA}(a, c, \text{table}, 0)$ is the last action that adds the goal on(a, c), any action that threatens the causal link $(1, t_{\text{fin}}(a, c))$ is excluded from the plan of agent $B$. Therefore, $C_{P'_A}^D$ contains the propositions $\neg \text{moveB}(a, Y, c, T)$, for $1 \leq T \leq t_{\text{fin}}$. Notice there is no plan that achieves the goals of agent $B$ and satisfies all the constraints of $C_{P'_A}^D$. Therefore, failure is returned.

Consider now agent $A$ that receives plan $P_B$ from agent $B$. The call to the $\text{ComputePlan}$ procedure, with its parameters suitably instantiated, will return the plan $\{ \text{moveA}(c, \text{table}, a, 1), \text{moveA}(a, c, \text{table}, 2) \}$.

The agents exchange messages of the form $(P_1, P_2)$, where $P_1$ and $P_2$ are (possibly empty) individual plans. Messages sent from one agent to the other are placed in the receiving agent’s incoming message queue and are processed in a FIFO manner. The coordination algorithm, the main body of which is presented in detail in figure 1 and refers to agent $B$, describes how these messages are processed by the agents. The messages can be of three different types, each carrying a different meaning. They are either of the type $(P_1, P_2)$, or $(P_1, \emptyset)$, or $(\emptyset, \emptyset)$, where $P_1$ and $P_2$ are non-empty plans. The meaning of each of these messages, and the reaction of the agents to these messages, are described in the following.

Before moving to the main body of the coordination algorithm (figure 1), the agents go through a phase in which the algorithm's variables and data structures are initialized. Moreover, each agent sends a message of the form $(P, \emptyset)$, where $P$ is the (optimal) plan generated by the call $\text{ComputePlan}(T, G, \infty, \emptyset, P)$, where $T$ and $G$ are the agent’s domain theory and goals respectively.

Each incoming message is processed by the coordination algorithm in a way that depends on its type. A message of the form $(P, \emptyset)$ that appears in the queue of an agent, means that the other agent proposes $P$ as a candidate sub-plan of a joint plan. The receiving agent will check, by invoking the planning algorithm as explained earlier, if he can generate a plan $P'$ that achieves his own goals and is consistent (non-conflicting) with $P$. An additional requirement is that the length of the joint plan $P \cup P'$, defined as $\max(l(P), l(P'))$, is shorter than the best joint plan. If this is the case, the agent sends the message $(P, P')$ to the other agent, meaning that $P$ can be part of an improved joint plan $(P, P')$. If the agent receives the message $(P, \emptyset)$ fails to find a plan as
specified above, he sends the message \((P, \text{fail})\), indicating that \(P\) cannot be part of a better joint plan.

A message of the form \((P_1, P_2)\), with \(P_1 \neq \emptyset\) and \(P_2 \neq \emptyset\), in the incoming queue of an agent is a reply to an earlier message of his, where he proposed the plan \(P_1\) to the other agent. Upon processing such a message, an agent deletes the entry \(P_1\) from the queue \(\text{sentbox}\), the structure that stores the, yet unanswered, proposals (i.e. plans) he sends to the other agent. As explained earlier, if \(P_2 \neq \text{fail}\) and the proposal (i.e. plan \(P_2\)) leads to an improved joint plan \((P_1, P_2)\), the variables \(P_{\text{best}}\) and \(l_{\text{best}}\) are updated accordingly. Then, the agent attempts to generate a new sub-plan with length shorter than \(l_{\text{best}}\). If such a sub-plan exists, he sends it to the other agent and appends it to the queue \(\text{sentbox}\). Otherwise, he sends the message \((\emptyset, \emptyset)\), indicating that there are no shorter individual plans. Thus, upon receiving a message \((\emptyset, \emptyset)\), an agent sets his expect variable to false, meaning he does not expect any further proposals.

The coordination algorithm is a branch and bound one where each agent generates individual plans that may improve the joint plan length. The algorithm, which is able to generate optimal solutions with respect to the joint plan length, terminates when the condition \((\text{sentbox} = \emptyset) \land \text{(not continue)} \land \text{(not expect)}\) becomes true. In such a case, the agent has received replies to all the sub-plans that he has proposed, he has no other plan to propose, and he does not expect any further proposals from the other agent.

Note that the algorithm can be modified so that it does not require agents to exchange complete plans, but only information that is necessary for the computation of the set \(G_{\text{best}}\). We do not discuss further this important issue.

The proposed coordination algorithm is sound and generates optimal solutions, as stated formally in the next proposition.

Proposition 2 The coordination algorithm always terminates. If the input coordination problem is solvable, the algorithm generates a joint plan of minimum length.

The following example illustrates the working of the algorithm.

Example 4 Consider two agents \(\alpha\) and \(\beta\) with a set of plans \(\{P_1^\alpha, P_2^\alpha, P_3^\alpha\}\) and \(\{P_1^\beta, P_2^\beta, P_3^\beta\}\) respectively for achieving their respective goals \(G_\alpha\) and \(G_\beta\). Assume that \(\text{len}(P_1^\alpha) = 4\), \(\text{len}(P_2^\alpha) = 8\), \(\text{len}(P_3^\alpha) = 9\) and \(\text{len}(P_3^\beta) = 6\), \(\text{len}(P_2^\beta) = 10\), \(\text{len}(P_3^\beta) = 11\). Assume that the pairs of non-conflicting plans are \((P_1^\alpha, P_3^\beta)\), \((P_2^\alpha, P_3^\beta)\), \((P_2^\alpha, P_3^\beta)\), \((P_3^\alpha, P_1^\beta)\), \((P_3^\alpha, P_2^\beta)\), \((P_3^\alpha, P_3^\beta)\).

In the initialization phase the agents generate their optimal individual plans \(P_1^\alpha\) and \(P_3^\beta\) and initialize their variables and data structures. Then, they exchange the messages \((P_1^\alpha, \emptyset)\) and \((P_3^\beta, \emptyset)\) and put \(P_1^\alpha\) and \(P_3^\beta\) in their respective \(\text{sentbox}\).

\[\textbf{while true do} \\
\text{get\_incom\_message}(P_\alpha, P) \\
\text{if } P_\alpha \neq \emptyset \text{ and } P = \emptyset \text{ then} \\
\text{ComputePlan}(T_B \cup T_\alpha \cup P_\alpha, G_B, l_{\text{best}}, \text{sentbox}(P_\alpha)) \\
\text{if } P_B \neq \text{fail} \text{ and } \max(l(P_\alpha), l(P_B)) < l_{\text{best}} \text{ then} \\
\text{send message } (P_\alpha, P_B) \\
\text{l}_{\text{best}} := \max(l(P_\alpha), l(P_B)), P_{\text{best}} := (P_\alpha, P_B) \\
\text{else} \\
\text{send message } (P_\alpha, \text{fail}) \\
\text{if } (\text{sentbox} = \emptyset) \text{ and } (\text{not continue}) \text{ and } (\text{not expect}) \text{ then} \\
\text{exit}(P_{\text{best}}) \\
\text{else if } P_\alpha \neq \emptyset \text{ and } P \neq \emptyset \text{ then} \\
\text{delete } P_\alpha \text{ from } \text{sentbox} \\
\text{if } P \neq \text{fail} \text{ and } \max(l(P_\alpha), l(P)) < l_{\text{best}} \text{ then} \\
\text{l}_{\text{best}} := \max(l(P_\alpha), l(P)), P_{\text{best}} := (P_\alpha, P) \\
\text{if continue then} \\
\text{ComputeNewPlan}(T_B, G_B, l_{\text{best}}, \emptyset, P_B) \\
\text{if } P_B \neq \text{fail} \text{ then} \\
\text{send message } (P_B, \emptyset) \\
\text{add } P_B \text{ to } \text{sentbox} \\
\text{else} \\
\text{send message } (\emptyset, \emptyset) \\
\text{continue} := \text{false} \\
\text{if } (\text{sentbox} = \emptyset) \text{ and } (\text{not continue}) \text{ and } (\text{not expect}) \text{ then} \\
\text{exit}(P_{\text{best}}) \\
\text{else } (P_\alpha = \emptyset \land P = \emptyset) \\
\text{if } (\text{sentbox} = \emptyset) \text{ and } (\text{not continue}) \text{ then exit}(P_{\text{best}}) \\
\text{else} \text{expect} := \text{false} \]

Figure 1: Coordination Algorithm

When agent \(\alpha\) processes the message \((P_1^\beta, \emptyset)\), he executes \(\text{ComputePlan}(T_\alpha \cup T_\beta \cup P_1^\beta, G_\alpha, \infty, \text{sentbox}(P_\alpha))\), which returns the plan \(P_3^\alpha\). Agent \(\alpha\) sets \(l_{\text{best}} = 9\), \(P_{\text{best}} = (P_3^\alpha, P_1^\beta)\), and sends the message \((P_1^\beta, P_3^\alpha)\) to agent \(\beta\). Similarly, agent \(\beta\) invokes \(\text{ComputePlan}\), which returns the plan \(P_3^\beta\), sets \(l_{\text{best}} = 11\), \(P_{\text{best}} = (P_1^\alpha, P_3^\beta)\), and sends the message \((P_1^\alpha, P_3^\beta)\).

When agent \(\alpha\) processes the message \((P_1^\alpha, P_3^\beta)\), he first deletes \(P_3^\alpha\) from \(\text{sentbox}\), which becomes empty. He then checks the condition \(\max(l(P_1^\alpha), l(P_3^\beta)) < l_{\text{best}}\), which is false, calls \(\text{ComputeNewPlan}(T_\alpha, G_\alpha, 9, \emptyset, P_2^\alpha)\), sends the message \((P_2^\alpha, \emptyset)\), and adds \(P_2^\alpha\) to \(\text{sentbox}\). Agent \(\beta\) gets the message \((P_1^\beta, P_3^\alpha)\) and sets \(l_{\text{best}} = 9\), \(P_{\text{best}} = (P_3^\alpha, P_1^\beta)\), and \(\text{sentbox} = \emptyset\). He then invokes \(\text{ComputeNewPlan}\), which returns failure (there exists no plan shorter than \(l_{\text{best}}\)), sends the message \((\emptyset, \emptyset)\), and sets continue to false. Agent \(\alpha\) gets the message \((\emptyset, \emptyset)\) and sets expect to false.
Agent $\beta$ processes message $(P_2^\beta, \emptyset)$ and sends the message $(P_2^\alpha, \text{fail})$. Agent $\alpha$ receives the message, replies with $(\emptyset, \emptyset)$, and exits with the plan $(P_3^\alpha, P_3^\beta)$. Upon receiving the message $(\emptyset, \emptyset)$, agent $\beta$ exits the coordination algorithm as well, with the same joint plan as $\alpha$.

4 Multi-Agent Assistance

As in the coordination case, in assistance two agents $a$ and $b$ need to achieve their goals with non-conflicting individual plans. The difference in this situation is that one or both agents may request the other agent to achieve specific subgoals that will enable the requesting agent to attain his own goals. When an agent receives such a request, he attempts to generate a plan that, except for his own goals, also achieves the requesting agent’s subgoals. The assistance problem can be defined formally as follows.

Definition 3 (Assistance Problem). Given two agents $\alpha$ and $\beta$ with goals $G_\alpha$ and $G_\beta$, initial states $I_\alpha$ and $I_\beta$ and sets of actions $D_\alpha$ and $D_\beta$ respectively, find a pair of plans $(P_\alpha, P_\beta)$ such that

- $P_\alpha = P_\alpha^\alpha \cup P_\alpha^\beta$ and $P_\beta = P_\beta^\beta \cup P_\beta^\alpha$
- $P_\alpha^\alpha \cup P_\beta^\beta \models_{D_\alpha \cup D_\beta, I_\alpha \cup I_\beta} G_\alpha$ and $P_\beta^\beta \cup P_\alpha^\beta \models_{D_\alpha \cup D_\beta, I_\alpha \cup I_\beta} G_\beta$
- if $P_\alpha^\alpha \models_{D_\alpha, I_\alpha} G_\alpha$ then $P_\alpha^\beta = \emptyset$, and if $P_\beta^\beta \models_{D_\beta, I_\beta} G_\beta$ then $P_\beta^\alpha = \emptyset$
- if $c(t) \in P_\alpha$ then $c \in D_\alpha$, and if $c(t) \in P_\beta$ then $c \in D_\beta$
- $P_\alpha$ and $P_\beta$ are non-conflicting

Note that coordination is a special case of assistance, where $P_\beta^\beta$ and $P_\alpha^\beta$ are empty. Also, we may impose the additional requirement that in an assistance scenario one or both agents are not able to achieve their goals by themselves, in other words $\neg \exists P \text{ s.t. } P \models_{D, I} G$, where $D$, $I$ and $G$ are the domain theory, the initial state and the goal of the agent. However, we do not enforce this restriction. Indeed, it can be the case that an agent can achieve some of his subgoals by executing his own actions, but he may seek the assistance of other agents as this may lead to better quality (in our case shorter) plans.

The assistance algorithm for agent $\beta$ is given in figure 2. It is very similar to the coordination algorithm, with the main differences being in the invocation of the classical planner and the form of the messages that the agents exchange. In the assistance algorithm agents generate their individual plans by invoking the algorithm $\text{ComputePlan}(T, G, l, \emptyset, <P, R>)$ which given the CNF encoding $T$ of the domain theory (represented in the extended STRIPS language described in section 2), a set of goals $G$ and a bound $l$, generates a plan $P$ together with a (possibly empty) set of assistance requests $R$. The plan $P$ succeeds only if the propositions of the set $R$ are true at their specified time points. This more general form of generated plans necessitates a slightly more complex form of messages. The messages are now of the form $(<P, R>, P')$, where $P$ and $P'$ are plans and $R$ is a set of assistance requests. When an agent generates a new individual plan $<P, R>$, he sends out the message $(<P, R>, \emptyset)$.

As in the coordination case, each incoming message is processed by the assistance algorithm of the receiving agent in a way that depends on its type. A message of the form $(<P_\alpha, R_\alpha>, \emptyset)$ in the incoming message queue of agent $\beta$, is interpreted as a request to search for a plan that is consistent with $P_\alpha$ and achieves $R_\alpha$. Upon processing such a message, agent $\beta$ invokes the $\text{SATPLAN}$ algorithm by calling $\text{ComputePlan}(T_\beta \cup T_\alpha \cup P_\alpha, G_\beta \cup R_\alpha, l_{\text{best}}, c_{P_\beta}^{D_\beta}, <P_\beta, \emptyset>)$. The empty set in the last parameter $<P_\beta, \emptyset>$ of the call, enforces the generation of a plan that does not contain assistance requests. This means that agent $\beta$ must achieve his goals without the assistance of agent $\alpha$. In a more general version of the assistance algorithm, one could allow a call of the form $\text{ComputePlan}(T_\beta \cup T_\alpha \cup P_\alpha, G_\beta \cup R_\alpha, l_{\text{best}}, c_{P_\beta}^{D_\beta}, <P_\beta, R_\beta>)$, where agent $\beta$ may reply to agent $\alpha$ with a plan $P_\beta$ but also a set $R_\beta$ of assistance requests. In the general case, we may end up with a situation of ”nested assistance”, where an agent can reply to an assistance request with a new assistance request. Such a situation terminates successfully if one of the agents achieves his goals without the need for further assistance. However, handling the general case requires more complicated data structures, and it is not discussed further.

All other message types are processed by the assistance procedure in a manner similar to the way they are handled by the coordination algorithm.

5 Future Work and Conclusions

In this paper we studied some aspects of the problems of coordination and cooperation of multiple agents that have individual goals and operate in the same environment. We formalized the coordination problem in a general way, and modelled a special case of multi-agent cooperation called assistance. For both problems we presented algorithms that rely on a classical planner and generate optimal solutions. The multi-agent assistance algorithm we propose is an innovative approach to dealing with the problem of mutual assistance among agents with complementary capabilities, whereas our coordination procedure presents certain advantages over previous approaches.

There are several lines for future work. The proposed
Figure 2: Assistance Algorithm

| while true do |
| get incoming message(< P_A, R_A >, P) |
| if P_A ≠ ∅ and P ≠ ∅ then |
| ComputePlan(T_B ∪ T'_A ∪ P_A, GB ∪ R_A, l_{best}, C_{P,A}^{D3}, < P_B, ∅ >) |
| if P_B ≠ fail and max(P_A, P_B) < l_{best} then |
| send message(< P_A, R_A >, P_B) |
| l_{best} := max(l(P_A), l(P_B)), P_{best} := (P_A, P_B) |
| else |
| send message(< P_A, R_A >, fail) |
| if (sentbox = ∅) and (not continue) and (not expect) then exit(P_{best}) |
| else if P_A ≠ ∅ and P ≠ ∅ then |
| delete < P_A, R_A > from sentbox |
| if P ≠ fail and max(l(P_A), l(P)) < l_{best} then |
| l_{best} := max(l(P_A), l(P)), P_{best} := (P_A, P) |
| if continue then |
| ComputeNewPlan(T_B, G_B, l_{best}, ∅, < P_B, R_B >) |
| if P_B ≠ fail then |
| send message(< P_B, R_B >, ∅) |
| add < P_B, R_B > to sentbox |
| else |
| send message(< ∅, ∅ >, ∅) |
| continue= false |
| if (sentbox = ∅) and (not continue) and (not expect) then exit(P_{best}) |
| else (ie. P_A = ∅ ∧ P = ∅) |
| if (sentbox = ∅) and (not continue) then exit(P_{best}) |
| else expect=false |

algorithms must be evaluated experimentally, and possible computational inefficiencies must be addressed. There seems to be room for improvement by combining the classical planning based algorithms we presented with techniques from multi-agent plan merging as those presented eg. in [16] and [5]. Another direction of future study concerns the extension of the proposed framework to other problems from multi-agent planning, such as action synchronization and interleaved planning and execution.

As a concluding remark, we reiterate that this work can be seen as a first step towards establishing a framework where different multi-agent planning problems can be studied in the light of recent advances in classical planning.

References