Multi-Agent Coordination and Cooperation through Classical Planning

Yannis Dimopoulos
Dept. of Computer Science
University of Cyprus
CY-1678 Nicosia, Cyprus
yannis@cs.ucy.ac.cy

Pavlos Moraitis
Dept. of Mathematics and Computer Science
René Descartes University
75270 Paris Cedex 06, France
pavlos@math-info.univ-paris5.fr

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 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. 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, covering different aspects of the problem of coordinating the plans of several agents operating in the same environment. These include scenarios where plan generation is distributed, where planning is centralized and plan execution is distributed, or where both planning and execution are distributed. However, only few works (e.g. [2], [5]) tackle aspects of the cooperation problem in the context of multi-agent planning.

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*.

We define formally the two problems, 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` [6] for plan generation. 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 planner generates 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 [3] and [4]. Space limitations do not allow a comparison here.

2 Propositional Satisfiability based Planning

We assume that the agents' planning domain theories are described in the STRIPS language, and denote by D_α the

set of actions that appear in the domain theory of agent α. To generate their plans the agents use the SATPLAN system [6]. 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. 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. Among the several ways to transform a planning problem into a satisfiability one, we adopt the Graphplan-based parallel encoding [6].

The SATPLAN procedure is invoked through the call $ComputePlan(T, G, L, C, P)$, 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 no such plan exists, by returning $fail$ in the argument P. Moreover, the procedure $ComputeNewPlan$ is the same as $ComputePlan$, with the difference that each time it is invoked, it returns a new plan different than those generated by the previous $ComputeNewPlan$ 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 $prec(A)$ of an action A is the union of the sets $norm\_prec(A)$ and $extern\_prec(A)$. The set $norm\_prec(A)$ contains normal action preconditions, ie. fluents that must hold before the execution of an action for that action to succeed. The elements of the set $extern\_prec(A)$ 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 $norm\_prec$ and ignores $extern\_prec$. The call to the planner now becomes $ComputePlan(T, G, L, C, < P, R >)$ where instead of returning a plan P, a pair $< P, R >$ is returned, where $R = \{p(t) | A(t) \in P \text{ and } p \in extern\_prec(A)\}$. We call the elements of the set R, *assistance requests*.

Note that the domain theory of an agent may contain different versions of the same action A, say A_1 and A_2, that differ only in the elements they contain in their sets $norm\_prec$ and $extern\_prec$, ie. $prec(A_1) = prec(A_2)$ but $norm\_prec(A_1) \neq norm\_prec(A_2)$.

3 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, α and β, and study a multi-agent coordination scenario that is defined by two main characteristics. The first is that each agent is able to achieve his goals by himself. This distinguishes coordination from cooperation, which is discussed in the next section. The second characteristic is that 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. Therefore, agents seek to minimize the length of the joint plan, even in the case where this leads to non-optimal individual plans.

The coordination problem is defined formally as follows.

Definition 1 (Coordination Problem). *Given two agents α and β with goals G_α and G_β, initial states I_α and I_β, and sets of actions D_α and D_β respectively, find a pair of plans (P_α, P_β) such that:*
- *$P_\alpha \models_{D_\alpha, I_\alpha} G_\alpha$ and $P_\beta \models_{D_\beta, I_\beta} G_\beta$*
- *P_α and P_β are non-conflicting*

We refer to the plans P_α and P_β as *individual* plans, and to the pair (P_α, P_β) as *joint* plan. Moreover, we use the term joint plan to also refer to the plan $P_\alpha \cup P_b$. Observe that if (P_α, P_β) 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_α, P_β) 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_α, P_β) wrt plan length in a *distributed* manner, where each agent generates its individual plan using SATPLAN. A solution (P_α, P_β) to a coordination problem is optimal wrt plan length if there is no other solution (P'_α, P'_β) to the problem such that $max(l(P'_\alpha), l(P'_\beta)) < max(l(P_\alpha), l(P_\beta))$. We note that this notion of "distributed" optimality is weaker than plan length optimality that can achieved by a centralized planner.

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.

The first source of negative interaction is *causal link threatening*, and is well known in the context of partial order planning. 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_1 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.

The second kind of negative interaction between actions is *parallel actions interference*, and it was introduced in Graphplan [1]. 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^S that when satisfied by a plan P' guarantee that P and P' are non-conflicting. The set C_P^S has the form of negated action occurrence literals that represent actions that must not be included in the plan P'. It is defined formally as $C_P^S = \{\neg A(t) | A \in S$ and $A(t)$ threatens a causal link (t_1, t_2, p) of $P\} \cup \{\neg A(t) | A \in S$ and there is some action $A'(t)$ in P such that $A(t)$ deletes a precondition or add effect of $A'(t)$, or $A'(t)$ deletes a precondition or add effect of $A(t)\}$.

3.1 The Coordination Algorithm

In the coordination algorithm we present in this section, we assume two agents α and β with goals G_α and G_β, domain theories D_α and D_β and initial states I_α and I_β 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 α, to compute a new plan P_α that achieves his goals G_α, without taking into account possible conflicts with the plan of agent β. This task is carried out by the call $ComputePlan(T_\alpha, G_\alpha, L, \emptyset, P_\alpha)$, where L is the length of the best current joint plan. The individual plan P_α is sent to agent β 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 α receives candidate sub-plans from agent β. Upon processing a proposed plan P_β, agent α invokes the planning algorithm by calling $ComputePlan(T_\alpha \cup T'_\beta \cup P'_\beta, G_\alpha, L, C_{P_\beta}^{D_\alpha}, P_\alpha)$, that returns the best plan P_α that achieves the goals G_α 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_α, which encodes the agent's domain theory and initial state, and the sets T'_β and P'_β which account for possible "positive" interactions between P_β and the plan that agent α will generate. The set T'_β contains (the CNF representation of) implications of the form $A(t) \rightarrow e_1(t+1) \wedge e_2(t+1) \wedge \ldots \wedge e_n(t+1)$ for each action A in D_β, positive effect e_i of A, $1 \leq i \leq n$, and time point t in the plan horizon. The set P'_β is defined as $P'_\beta = \{A(t) | A(t) \in P_\beta\} \cup \{\neg A(t) | A \in D_\beta$ and $A(t) \notin P_\beta\}$. The sub-theory $T'_\beta \cup P'_\beta$ entails all add effects of actions that are executed by agent β, and therefore they do not need to be re-established by agent α.

Moreover, the plan P_α returned by the call $ComputePlan(T_\alpha \cup T'_\beta \cup P'_\beta, G_\alpha, L, C_{P_\beta}^{D_\alpha}, P_\alpha)$, satisfies the constraints $C_{P_\beta}^{D_\alpha}$, which means that the two plans P_α and P_β are non-conflicting, therefore $P_\alpha \cup P_\beta \models_{D_\alpha \cup D_\beta, I_\alpha \cup I_\beta} G_\alpha \cup G_\beta$. Furthermore, the joint plan (P_α, P_β) (equivalently $P_\alpha \cup P_\beta$) becomes the best current joint plan.

To simplify the discussion we assume that when an agent, say α, receives the *first* plan from agent β and invokes SATPLAN by calling $ComputePlan(T_\alpha \cup T'_\beta \cup P'_\beta, G_\alpha, \infty, C_{P_\beta}^{D_\alpha}, P_\alpha)$, the call always succeeds and returns a plan that achieves the goals G_α, while satisfying all the constraints of $C_{P_\beta}^{D_\alpha}$.

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 β, 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 $ComputePlan(T, G, \infty, \emptyset, P)$, 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, whether there exists 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 such a plan exists, 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 that receives the message $(P, \emptyset)$ fails to find a plan as specified above, he sends the message $(P, fail)$, 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 $sentbox$, the structure that stores the, yet unanswered, proposals (i.e. plans) he sends to the other agent. As explained earlier, if $P_2 \neq fail$ and the proposal (i.e. plan P_2) leads to an improved joint plan (P_1, P_2), the variables P_{best} and l_{best} are updated accordingly. Then, the agent attempts to generate a new sub-plan with length shorter than l_{best}. If such a sub-plan exists, he sends it to the other agent and appends it to the queue $sentbox$. 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 $(sentbox = \emptyset) \wedge (not\ continue) \wedge (not\ expect)$ 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 does not expect any further proposals from the other agent.

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.*

while $true$ **do**
get_incom_message(P_A, P)
if $P_A \neq \emptyset$ and $P = \emptyset$ **then**
  $ComputePlan(T_B \cup T'_A \cup P'_A, G_B, l_{best}, C^{D_\beta}_{P_A}, P_B)$
  if $P_B \neq fail$ and $max(l(P_A), l(P_B)) < l_{best}$ **then**
    send message (P_A, P_B)
    $l_{best} := max(l(P_A), l(P_B))$; $P_{best} := (P_A, P_B)$
  else send message $(P_A, fail)$
  if $(sentbox = \emptyset)$ and $(not\ continue)$ and $(not\ expect)$
    then exit(P_{best})
else
if $P_A \neq \emptyset$ and $P \neq \emptyset$ **then**
  delete P_A from $sentbox$
  if $P \neq fail$ 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**
    $ComputeNewPlan(T_B, G_B, l_{best}, \emptyset, P_B)$
    if $P_B \neq fail$ **then**
      send message $(P_B, \emptyset)$; add P_B to $sentbox$
    else
      send message $(\emptyset, \emptyset)$; continue:= false
  if $(sentbox = \emptyset)$ and $(not\ continue)$ and $(not\ expect)$
    then exit(P_{best})
else (ie. $P_A = \emptyset \wedge P = \emptyset$)
  if $(sentbox = \emptyset)$ and $(not\ continue)$ **then exit**(P_{best})
  else $expect$=false

Figure 1: Coordination Algorithm

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 α and β with goals G_α and G_β, initial states I_α and I_β and sets of actions D_α and D_β respectively, find a pair of plans (P_α, P_β) such that:*
• $P_\alpha = P^\alpha_\alpha \cup P^\beta_\alpha$ *and* $P_\beta = P^\beta_\beta \cup P^\alpha_\beta$
• $P^\alpha_\alpha \cup P^\alpha_\beta \models_{D_\alpha \cup D_\beta, I_\alpha \cup I_\beta} G_\alpha$ *and* $P^\beta_\beta \cup P^\beta_\alpha \models_{D_\alpha \cup D_\beta, I_\alpha \cup I_\beta} G_\beta$
• *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_α *and* P_β *are non-conflicting*

Note that coordination is a special case of assistance, where P_α^β and P_β^α are empty. The assistance algorithm for agent β 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 $ComputePlan(T, G, l, \emptyset, < P, R >)$ 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 β, is interpreted as a request to search for a plan that is consistent with P_α and achieves R_α. Upon processing such a message, agent β invokes the SATPLAN algorithm by calling $ComputePlan(T_\beta \cup T'_\alpha \cup P'_\alpha, G_\beta \cup R_\alpha, l_{best}, C_{P_\alpha}^{D_\beta}, < P_\beta, \emptyset >)$. 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 β must achieve his goals without the assistance of agent α. In a more general version of the assistance algorithm, one could allow a call of the form $ComputePlan(T_\beta \cup T'_\alpha \cup P'_\alpha, G_\beta \cup R_\alpha, l_{best}, C_{P_\alpha}^{D_\beta}, < P_\beta, R_\beta >)$, where agent β may reply to agent α with a plan P_β but also a set R_β 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.

while $true$ **do**
get_incom_message$(< P_A, R_A >, P)$
if $P_A \neq \emptyset$ and $P = \emptyset$ **then**
  $ComputePlan(T_B \cup T'_A \cup P'_A, G_B \cup R_A, l_{best}, C_{P_A}^{D_\beta}, < P_B, \emptyset >)$
  if $P_B \neq fail$ 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 $(sentbox = \emptyset)$ and $(not\ continue)$ and $(not\ expect)$
    then exit(P_{best})
else
if $P_A \neq \emptyset$ and $P \neq \emptyset$ **then**
  delete $< P_A, R_A >$ from $sentbox$
  if $P \neq fail$ 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**
    $ComputeNewPlan(T_B, G_B, l_{best}, \emptyset, < P_B, R_B >)$
    if $P_B \neq fail$ **then**
      send message $(< P_B, R_B >, \emptyset)$
      add $< P_B, R_B >$ to $sentbox$
    else
      send message $(< \emptyset, \emptyset >, \emptyset)$; continue= false
  if $(sentbox = \emptyset)$ and $(not\ continue)$ and $(not\ expect)$
    then exit(P_{best})
else (ie. $P_A = \emptyset \wedge P = \emptyset$)
  if $(sentbox = \emptyset)$ and $(not\ continue)$ **then exit**(P_{best})
  else $expect$=false

Figure 2: Assistance Algorithm

5 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 can be seen as a step towards establishing a stronger link between classical and multi-agent planning.

References

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence Journal*, 90(1-2), 1997.

[2] G. Boutilier and R. Brafman. Partial-order planning with concurrent interacting actions. *JAIR*, 14, 2001.

[3] J. Cox and E. Durfee. An efficient algorithm for multiagent plan coordination. In *AAMAS05*, 2005.

[4] E. Ephrati and J. Rosenschein. Divide and conquer in multi-agent planning. In *AAAI94*, 1994.

[5] A. El Fallah-Seghrouchni and S. Haddad. A recursive model for distributed planning. In *ICMAS96*, 1996.

[6] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *KR96*, 1996.