Fusion of Plans in a Framework with Constraints

Mathijs de Weerdt André Bos Hans Tonino Cees Witteveen

Delft University of Technology
Zuidplantsoen 4, 2628 BZ Delft, The Netherlands
email: {M.M.deWeerdt, A.Bos, J.F.M.Tonino, C.Witteveen}@ITS.TUDelft.NL

Abstract

The promise of multi-agent systems is that multiple agents can solve problems more efficiently than single agents can. In this paper we propose a method to implement cooperation between agents in the planning phase, in order to achieve more cost-effective solutions than without cooperation. Two results are presented: First of all, we introduce a flexible framework to describe cooperation in a planning process. This framework allows us to specify the basic tasks that have to be performed in order to satisfy a set of goals, and to specify constraints to model deadlines, capacity constraints, speed limits, conflicting situations, *et cetera.* Furthermore, we present a polynomial fusion algorithm to implement cooperation between multiple agents.

1 Introduction

As agents are autonomous systems, they plan their actions in relative isolation from the rest of the agent society. Often it is much more cost effective to cooperate by taking advantage of the available resources of other agents. As an agent has only a local span of control, it is not clear how to exploit these resources during the joint plan construction phase. In this paper, we propose a solution to this problem by taking the following approach: First, each individual agent is assumed to construct or to select a plan from a plan library, without taking into account the activities of other agents. Next, a (sub)group of agents investigates whether a cooperative approach leads to a more cost effective plan by taking advantage of so-called *free resources* of each agent's activities. The creation of such a joint plan is called a fusion.

This method is related to work by Wooldridge and Jennings [9] and Shehory and Kraus [7] on coalition formation. Whereas the former concentrates on the modeling of the mental state of cooperating agents, and the latter on polynomial algorithms for task allocation, we focus on the details of *combining* plans. As described in [3], coalition formation in general is NP-hard, but we will show that for benevolent agents, a (suboptimal) fusion can be found in polynomial time. Part of this work, concerning only time constraints and a preliminary version of the framework, has been published recently [2, 6, 8].

To give an intuitive idea of the cooperation processes we have in mind, we present the following taxi scheduling example.

Example 1 Assume two taxis (taxi 1 and taxi 2), transporting three passengers (see Figure 1): Taxi 1 takes passenger 1 from A to B, passenger 2 from C to D, and taxi 2 takes passenger 3 from D to E. We assume both taxis are available from time 3 on ($taxi(1, A, 3)$ and $taxi(2, F, 3)$), passenger 1 at time 1 ($p(1, A, 1)$), passenger 2 at time 2 ($p(2, C, 2)$), and passenger 3 at time 1 ($p(3, D, 1)$). We use the following notations for the resources passenger and taxi: $p(3, D, 1)$ to describe passenger 3 at location D from time 1 on and $taxi(2, F, 3)$ to describe taxi 2 at location F from time 3 on. ■

In this paper we show a way to model these plans and an algorithm to create a combined plan that is more efficient. For example, taxi 2 could pick up passenger 2 as well, so taxi 1 won't have to drive all the way to E, but just to B. This example shows that by cooperation the taxi-agents can realize all initial goals with less production (viz. transportation) costs. Such a reduction can be realized by exchanging *goals* and/or by exchanging necessary *resources.*

The following notions play a central role in this analysis: *(i)* elementary production processes, in our framework called *skills* of an agent, that constitute

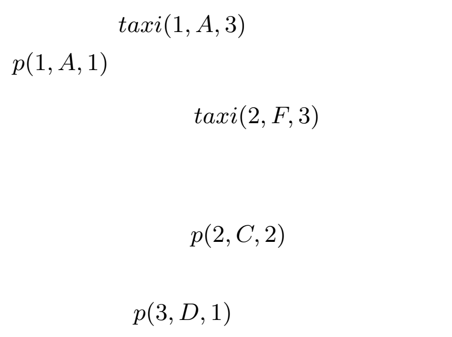

Figure 1: Overview of the locations and the initial situation. The dashed arrows represented the goals of the passengers.

the building blocks for *production plans*, *(ii)* the *resources* needed to "fire" a skill, *(iii)* the *goals* to be realized, and *(iv) attributes* to specify constraints on the goals and the skills and to define production properties of the resources.

In the next section we present an overview of the framework [1].

2 The resource-skill-plan formalism

In this section we propose a framework to describe plans and operations on plans (like fusion). We start by giving the building blocks: resources, goals to achieve, and the skills to combine the resources to obtain these goals. After that we define a plan in terms of these building blocks.

2.1 Resources, goals, and skills

We consider the production of a set of products from a set of resources. A product itself can be a resource for another product. Therefore, every object, whether used or produced, is called a resource.

A resource r will be identified by its *type* (a predicate symbol) and the set of *values* of all its *attributes*, like, e.g., the time the resource is available, its capacity, an identifier *id* to refer to a real world object, a maximum speed, current speed, current location, weight, etc. Resources can be used exactly once, but may result in new resources that refer to the same type of objects in the real world (but with different properties). We assume an enumerable set $\mathcal{R}$ of all resources.

We use predicate symbols like *taxi* and *plane* to identify the type of a resource $r \in \mathcal{R}$, denoted by $type(r)$. The arity n of a predicate symbol refers to the number of attributes needed to fully specify an individual resource. Attributes a_i are (inductively) specified by means of *value-terms*, being either *constants* belonging to the domain $Dom(a_i)$ of the a_i, *variables* occurring in a set $Var(a_i)$, or *composite terms* of the form $f(t_1, \dots t_m)$ where f is an m-ary function symbol and $t_1, \dots t_m$ are value-terms. A value-term t_i not containing any variable is called a *ground value term*. The set of value terms associated with a_i is $Term(a_i)$.

If $a_1, \dots a_n$ are (all) the attributes associated with a predicate symbol p, and $v_1, \dots v_n$ are ground value terms associated with them respectively, then $r = p^n(v_1 : p_1, v_2 : p_2, \dots, v_n : p_n)$ is called a *completely instantiated resource* or, abbreviated, a *resource*. An example of a resource is a taxi containing two other resources (passengers) p_1 and p_2:

$$\begin{array}{ll} taxi(& \text{AX-20-BT} : id,\ 4 : cap, \text{Amsterdam} : loc, \\ & [0, \rightarrow>: time,\ 2 : left\_cap,\ 0 : speed, \\ & 120 : max\_speed,\ \{p_1, p_2\} : contents\ ) \end{array}$$

The following attributes are used here: *id* to identify the taxi, *cap* to specify the total capacity (number of passengers) for this taxi, *loc* to specify the current location, *time* gives the complete interval the resource can be used, $left\_cap$ defines the left capacity of the taxi, *max_speed* the maximum speed and *speed* the current speed, and *contents* the current contents of the taxi. Let A denote the set of all attributes distinguished. We will ensure $Var(a_i) \cap Var(a_j) = \emptyset$ for every $a_i \neq a_j$. The set $Var = \bigcup_{a \in A} Var(a)$ denotes the set of all variables. Likewise, we will use $Term$ as the set of all value terms associated with the set of all attributes. A substitution θ then is a finite set of pairs of the form $x_i = t_i$ with $x_i \in Var$ and $t_i \in Term$ satisfying the normal conditions[2]. If T is a set of terms, $T\theta$ denotes the result of applying θ to every $t \in T$. Besides completely instantiated resources, we also have to consider incompletely spec-

[1]A simpler variant of this framework, without the ability to specify attributes and constraints over attributes, was proposed in [6] and further developed in [8].

[2]That is: *(i)* variables and terms match, i.e., if $x_i \in Var(a)$, then $t_i \in Term(a)$; *(ii)* $i \neq j$ implies $x_i \not\equiv x_j$; *(iii)* x_i does not occur as a variable in t_i.

ified resources or *resource schemes*. We distinguish two sorts:

- *Basic resource schemes* of the form $r = p^n(v_{i_1} : a_{i_1}, \dots, v_{i_m} : a_{i_m})$ where *(i)* p is a predicate symbol of arity n; *(ii)* $1 \leq i_j \neq i_k \leq n$ for $j \neq k$, with $j, k = 1, \dots, m$; *(iii)* v_{i_j} is a value term (not necessarily ground) of attribute a_i. Basic resource schemes therefore are able to characterize *sets* of resources sharing the same predicate symbol.
- *Extended resource schemes*, to represent arbitrary subsets of instantiated resources, consisting of a *set* RS of basic resource schemes together with a set of constraints C, being formulas using variables occurring in this set RS. A resource r is said to belong to an extended resource scheme (RS, C) if there exists a ground substitution θ such that $r \in RS\theta$ and every element of $C\theta$ evaluates to true. Extended resource schemes are used to define skills.

We will use skills to represent elementary resource production processes. In general, a skill s can be applied to some set R of given resources and produces as output some new set R' of resources where $R' \cap R = \emptyset$, i.e. skills consume their inputs completely ensuring that every individual resource can be used only once.

Definition 1 A skill s is a rule of the form $RS' \leftarrow RS, C$ such that *(i)* $RS' = \{rs'_1, \dots, rs'_m\}$ is a set of basic resource schemes, and *(ii)* (RS, C) is an extended resource scheme. $Out(s) = RS'$ is called the set of output schemes of s, $In(s) = RS$ is the set of input schemes, and $C(s) = C$ is the set of skill constraints.

Example 2 To give two concrete examples, take the following skills:

$$\begin{array}{l}
\textbf{drive} : taxi(i : id, y : loc, t + t(x,y) : time), \\
\qquad ride(x : from, y : to, 2 : cap, t : time) \\
\quad \leftarrow taxi(i : id, x : loc, t : time) \qquad \{\ \ \} \\
\textbf{travel} : p(i : id, y : loc, t_2 + t(x,y) : time), \\
\qquad ride(x : from, y : to, c - 1 : cap, t_2 : time) \\
\quad \leftarrow p(i : id, x : loc, t_1 : time), \\
\qquad ride(x : from, y : to, c : cap, t_2 : time) \\
\qquad\qquad \{t_1 \leq t_2, c \geq 1\}
\end{array}$$

To explain the last skill, **travel**: For a passenger to travel from location x to location y, two resources are necessary to get a resource $p(i : id, y : loc, t_2 + t(x,y) : time)$. First of all the passenger i should be at location x at a certain time t_1 and secondly, a ride (produced by a taxi) should be available at a time $t_2 \geq t_1$ and with capacity at least 1. The time the passenger arrives at y is calculated using a pre-defined time-distance matrix $t(x,y)$. ▪

Definition 2 [Produced from] Let S be a set of skills, R_1 and R_2 be sets of resources. We say that R_2 *can immediately be produced* from R_1 *using* S, abbreviated by $R_1 \vdash_S R_2$, if there is a skill $s \in S$ and a ground substitution θ such that *(i)* $In(s)\theta \subseteq R_1$, i.e., specific resource instances of the input of the skill occur in R_1; *(ii)* $R_2 = (R_1 - In(s)\theta) \cup Out(s)\theta$, i.e., the specific input instances are deleted from the original set of resources R_1, while the corresponding output instances of s are added to the remaining resources. We say that R_2 *is produced from* R_1 *using* S if $R_1 \vdash^*_S R_2$ holds, where $\vdash^*_S$ denotes the reflexive, transitive closure of $\vdash_S$.

Skills are applied to realize a *goal scheme* GS. A goal scheme is a special kind of extended resource set: i.e. a set of basic resource schemes $G = \{g_1, \dots, g_m\}$ (basic goals) together with a set of constraints C using variables occurring in G. We say that a given set of resources *satisfies* a goal scheme $GS = (G, C)$ using some ground substitution θ, abbreviated as $R \models_\theta GS$, if *(i)* $G\theta \subseteq R$ and *(ii)* every constraint in $C\theta$ evaluates to true. It is now easy to express that an agent having resources R and skills S is able to realize a goal scheme GS:

Definition 3 [Goal realizability] A goal scheme GS is *realizable* from a set of resources $R \subseteq \mathcal{R}$ using skills $S \subseteq \mathcal{S}$ and ground substitution θ, if there exists some set of resources R' such that *(i)* R' can be produced from R using S: $R \vdash^*_S R'$ and *(ii)* R' satisfies the goal scheme GS using θ: $R' \models_\theta GS$. A goal scheme GS is said to be *realizable* from R using S, if there exists a ground substitution θ such that GS is *realizable* from R using S and θ.

An example of a goal in this context is "passenger 2 should be at D before time 6". This can be represented in a goal scheme as follows: $(p(2, D, t), t \leq 6)$.

2.2 Plans

We assume that each agent already has some kind of plan, for example generated by a planning system such as Blackbox [5] or GraphPlan [1]. We represent a plan by a bi-partite Directed Acyclic Graph (DAG) $P = \langle N_R \cup N_S, E\rangle$, where $N_R \subseteq \mathcal{R}$ denotes a set of resource nodes, N_S a set of skill nodes n_s where $s \in \mathcal{S}$, and E a set of arcs. Note that different skill nodes may refer to different applications of the same skill: the notation "n_s" means that n_s is a skill node denoting an application of skill s. For any two nodes $r \in N_R$ and $n_s \in N_S$, $(r, n_s) \in E$ means that resource r is used by an application of skill s, and $(n_s, r) \in E$ means that resource r is produced by an application of s. Mind also that each resource can be consumed only once.

We will use the following notational conventions for subsets of nodes in a DAG $P = \langle N_R \cup N_S, E\rangle$: the set of input resources of P will be denoted by $In(P) = \{r \in N_R \mid d^-(r) = 0\}$[3], whereas $Out(P) = \{r \in N_R \mid d^+(r) = 0\}$ will refer to the set of final products of P.

Definition 4 [Plan] Let S be a set of skills, and GS a goal scheme. A *plan* P for GS using S and ground substitution θ, is a bi-partite DAG $P = \langle N_R \cup N_S, E\rangle$, such that

1. $N_R \subseteq \mathcal{R}$ ($\mathcal{R}$ being the set of all resources),
2. $Out(P) \models_\theta GS$,
3. if $n_s \in N_S$, then $\{r \mid (n_s, r) \in E\} = Out(s)\theta$, $\{r \mid (r, n_s) \in E\} = In(s)\theta$, and $C(s)\theta$ evaluates to true for the ground substitution θ,
4. if $r \in N_R$, then $d^+(r) \leq 1$ and $d^-(r) \leq 1$ (resources are used at most once).

The following observation is immediate:

Observation 1 *If P is a plan for GS using S and θ then $In(P)$ realizes GS using S and θ. If such a θ exists, we say P* realizes *GS using S.*

Example 3 Consider taxi 2 as described in Example 1. Its plan is to pickup passenger 3 at D and drive to E, where passenger 3 gets out. This plan can be modeled in this formalism by applying skills **drive** and **travel** from Example 2. The plan, including the needed initial resources, is shown in Figure 2 (with an applied substitution θ). The initial resources are the bottom nodes of this DAG. The applied-skill nodes are represented by small boxes.

[3] $d^-(n)$ denotes the *in-degree* of node n; likewise, $d^+(n)$ denotes the *out-degree* of n.

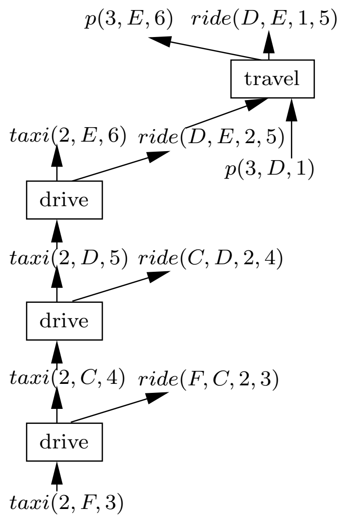

Figure 2: The plan for taxi 2 as described in Example 3.

There are more ways to model such activities in this formalism, but as we will see in Section 3, fusion only works when free (unused) resources are available. That is why we model a drive of a taxi as the generation of free resources. ■

3 Plan improvement and fusion

Until now we have only paid attention to individual plans. The formalism as described in the previous section can also be used to describe certain cooperation processes. In this section, we will look at the *fusion* process, where agents are willing to share all their resources and products in order to improve overall performance. The fusion algorithm is based on the notion of plan improvement: Skills will be removed if the resources that the skills produce –needed for goal satisfaction– can be acquired in another way (e.g., these resources are by-products of other skill applications). Fusing agents can help each other by exchanging their resources. In this section, we present a polynomial algorithm for plan improvement that can be used to model fusion.

Previous to starting the fusion, the agents should have their own correct, individual plans. The improvement algorithm inspects all skills of all plans and removes them, if possible, one by one. As these skill removals do not necessarily lead to a globally optimal plan, this procedure is a local optimization. A skill can be removed if for each output resource r that is needed by another skill or as a goal, a substitute can be found. To denote that resource r is needed (directly or indirectly) for goal satisfaction, we use the predicate $needed(r)$. This predicate is implemented by a boolean variable. Initially this variable is defined true if the resource is used within a plan and false otherwise. The variable is only changed by the function RE-ASSIGN_RESOURCES for resources that were needed by the (now removed) skill *olds*. The function FIND_SUBST checks each resource that is not needed for goal satisfaction whether it can be used as a substitute for an output resource r of a skill considered to be removed. The attributes of the substitution resource r_2 must satisfy all constraints imposed by the plan. These constraints are checked by the function SAT_CONSTRAINTS. Furthermore, this function verifies that r_2 is not dependent on r in the current plan. Now we see that if there are no free resources (all resources are *needed*), fusion does not improve the solution.

The procedure RE-ASSIGN_RESOURCES replaces all used output resources of the removed skill by their replacements found by the function FIND_SUBST. As the procedure is rather straightforward the details are omitted here. The loop structure of OPTIMIZE needs some more explanation: the outer loop, implemented by a recursive call to OPTIMIZE, ensures that skills are removed until no skill to be removed can be found. The inner loop (step 2) specifies that all skills have to be checked for removal until one is found.

Fusion is based on the same plan improvement algorithm. Fusing agents are interested in an improvement in performance of the whole set of cooperating agents, and not in individual plan improvement. Therefore, we may use the plan improvement algorithm on the joint set of individual plans. We will denote the finite set of agents cooperating in a fusion by $\mathcal{A}$. The joint plan $P_{\mathcal{A}}$ must be such that all individual goals of the agents still can be realized. In this paper, the fusion process is modeled as follows: First, all individual plans are combined as a forest of individual plans, and, second, the joint plan is optimized by the described algorithm. The following propositions can be easily verified:[4]

```
FUSE($\mathcal{A}$)
  1. $plan = \cup_{A_i \in \mathcal{A}} P_{A_i}$
  2. OPTIMIZE(plan)

FIND_SUBST ( $r, s$ )
  for each resource $r_2$ do
      if $r_2 \neq r$
          $\wedge(\neg needed(r_2) \vee r_2 \in In(s))$
          $\wedge \neg r_2 \in Out(s)$
          $\wedge type(r_2) = type(r)$
          $\wedge$ SAT_CONSTRAINTS($r_2, r$)
       then
          return true
  return false

OPTIMIZE(plan)
  1. s_found = false
  2. for each skill $s$ until s_found do
         s_found = true
         for each $r \in Out(s)$ do
             if $needed(r)$ then
                 s_found = s_found $\wedge$
                   FIND_SUBST($r, s$)
         olds = s
  3. if s_found then
         RE-ASSIGN_RESOURCES(olds)
         DELETE(olds)
         OPTIMIZE(plan)
```

Proposition 1 *Given a finite set of m agents $\mathcal{A}$ with for each agent i a goal scheme (G_{A_i}, C_{A_i}) and a plan P_{A_i}, the fusion algorithm will always find a plan $P_{\mathcal{A}}$ that realizes $(\cup_{A_i \in \mathcal{A}} G_{A_i}, \cup_{A_i \in \mathcal{A}} C_{A_i})$ using all available skills.*

Since we start with correct, individual plans, it is always possible to construct a joined plan, being nothing more than a union of these individual plans.

With respect to the complexity, let $n = \sum_{1 \leq i \leq m} ||P_{Ai}||$ denote the sum of the sizes[5] of the plans of all agents in $\mathcal{A}$. The worst-case time complexity of SAT_CONSTRAINTS is $O(n)$, because all consequences of using r_2 instead of r have to be propagated through the plan: whether r_2 was dependent on r and all constraints can be checked in linear time, since all attributes are known (i.e. if we store previously evaluated attributes and constraints in the plan).

[4]The proofs are easy but somewhat tedious and, due to lack of space, omitted here.

[5]The size $||P||$ of a plan P equals the size of the DAG representing P, i.e., the number of vertices plus the number of arcs in the plan.

This function is executed for each free resource, so the worst-case time complexity of FIND_SUBST is $O(n^2)$. This function is called at most $O(n^2)$ times, since each time a skill is removed at most $O(n)$ resources may have been searched for and in total at most $O(n)$ skills can be removed.

Proposition 2 *Fusion of agents with total plan size n can be performed in $O(n^4)$-time.*

A slightly simpler variant of this algorithm (including only time constraints) is implemented in C in the following set-up. First a translation is done of the set of skills to a STRIPS-like Blackbox [4, 5] input. Then each agent runs an instance of Blackbox to produce a plan. These plans are fused using the described algorithm. To evaluate our algorithm, we compared the results of this process to Blackbox solving the problem globally. Some preliminary tests showed us that solving problems for the agents separately, followed by our fusion process is about 25% faster than solving the problem globally, and returns a better solution (in terms of the number of skills used) as well. The better running time is exactly what we expected: Blackbox has an exponential time complexity, while our algorithm is polynomial. In the following example the fusion of two plans using this algorithm is demonstrated.

Example 4 Suppose the two taxis as described in Example 1 would like to cooperate. They decide to fuse, i.e. they join their complete plans, including all unused resources. The individual plan for taxi 2 is displayed in Figure 2, the individual plan for taxi 1 looks almost the same, except it does have one travel more, because it transports two passengers. Once combined, the OPTIMIZE algorithm tries to remove skills. The unused *ride* resources produced by taxi 2 can be used as substitutes in the plan originally belonging to taxi 1. This leads to the joint plan displayed in Figure 3. In this joint plan, taxi 1 only drives from A to B and taxi 2 transports passenger 2. ∎

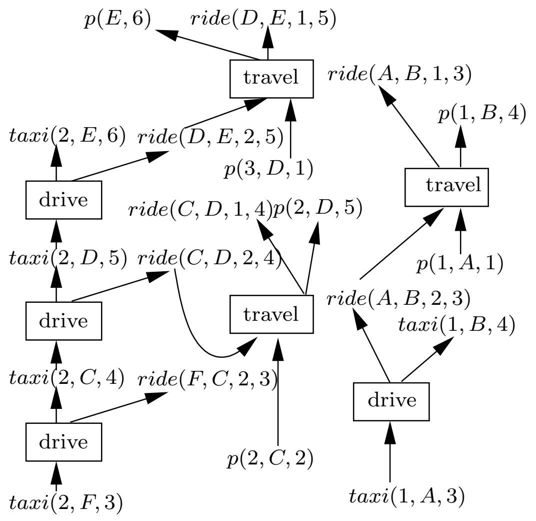

Figure 3: The joint plan, see Example 4.

4 Conclusions and future work

One of the unique features of multi-agent systems is that individual agents can *cooperate* in order to achieve their goals. One reason to cooperate is that agents cannot realize their goals individually; another reason is that cooperation leads to a more efficient means to realize their goals. In this paper, we have concentrated on the latter.

We described a computational framework, consisting of resources and skills, to model cooperation processes between different agents. Central in this framework is that we model *side products* explicitly, so that other agents can exploit these unused resources. Furthermore, it is possible to specify attributes of resources and constraints on these attributes in skill definitions. This allows us to deal with durations and deadlines, capacity constraints, traffic limits, conflicting situations, costs, and many more types of constraints.

The second main result of this paper is a novel algorithm that *fuses* the individual plans of a set of agents centrally in polynomial time, such that the joint plan includes less skills.

We think our results are very promising, but there are still some issues we will look at in the near future: First of all a future version should include a more detailed notion of costs, such that the optimization of plans will be based on the cost properties of resources and skills. Secondly, when only simple constraints are used, the fusion algorithm can use some form of constraint propagation of goal constraints to check possible substitute resources more efficiently. Also, sometimes, a substitute resource can be modified slightly to fit in by changing some attributes in the plan only slightly (for example, a ride resource may be produced somewhat later to be able to pick up another passenger as well). We will also pursue research on the following topics: dealing with plan fragments, creating robust plans (and plans having alternatives built in or leaving some details to be filled in at run-time), and some form of replanning.

Acknowledgements

Mathijs de Weerdt and André Bos are supported by the research program Seamless Multimodal Mobility (SMM), and André Bos by the research program Freight Transport Automation and Multimodality (FTAM) as well. Both programs are carried out within the TRAIL research school for Transport, Infrastructure and Logistics.

References

[1] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[2] M.M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A plan fusion algorithm for multi-agent systems. In *Proceedings of the Workshop on Computational Logic in Multi-Agent Systems (CLIMA-00)*, pages 56–65, 2000.

[3] M. d'Inverno, M. Luck, and M. Wooldridge. Cooperation structures. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan*, pages 600–605, 1997.

[4] R. E. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208, 1971.

[5] H. Kautz and B Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the workshop on planning as combinatorial search, held in conjunction with AIPS'98*, Pittsburgh, PA, 1998.

[6] B.-J. Moree, A. Bos, H. Tonino, and C. Witteveen. Cooperation by iterated plan revision. In *Proceedings of the ICMAS 2000*, 2000.

[7] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, May 1998.

[8] H. Tonino, A. Bos, and C. Witteveen. Replanning by revision in collective agent based systems. Technical Report PDS-2000-004, Delft University of Technology, May 2000.

[9] M. Wooldridge and N.R. Jennings. The cooperative problem solving process. *Journal of Logic & Computation*, 9(4), 1999.