Cooperative heuristic multi-agent planning

Mathijs de Weerdt, Hans Tonino and Cees Witteveen

Delft University of Technology
Faculty of Information Technology and Systems
TRAIL
P.O.Box 356, 2600 AJ Delft, The Netherlands
e-mail: {M.M.deWeerdt,J.F.M.Tonino,C.Witteveen}@its.tudelft.nl

Abstract

In a previous BNAIC paper [10] we introduced a resource-based framework for representing plans. Using this framework we developed a polynomial algorithm for *plan merging*, a process in which agents combine their plans to save actions and resources. In this paper we will use the framework to study *cooperative heuristic multi-agent planning*. During the construction of their plans, the agents use a heuristic function inspired by the FF planner ([3]). At any time in the process of planning the agents may exchange available resources, or they may request an agent to produce a missing resource. The latter feature is enabled by an advertising mechanism: Any agent communicates to the other agents which resources it can, in principle, produce using a *plan scheme* stored in its knowledge base. The planning algorithm presented in this paper is sound but not complete: It might fail to find a plan in cases a valid plan does exist.

1 Introduction

Multi-agent planning is a very active research area, as can be derived from overviews of multi-agent planning approaches by, e.g., Durfee [1] and Mali [5]. However, most of the research in this field focuses on either finding an assignment of tasks to the participating agents (cf. [4, 7, 8]) or on cooperation once complete plans have been constructed, called *plan merging* [2, 9, 12]. In this paper we will consider an extension of this latter work where planning and cooperation is fully integrated: *cooperative heuristic multi-agent planning.*

In previous work [6, 10, 11], we have developed a framework for multi-agent planning using an explicit representation of resources, skills and goals in a multi-agent environment. This framework was used in a plan merging algorithm, where complete plans were merged. In this paper, the agents have to actually compute a plan from scratch. They do so by employing a *heuristic function* to guide the search process. This procedure is obtained from the FF planner [3]. Our contribution is to use this procedure in the setting of cooperative planning. At any time during the planning process, the agents may exchange unused resources by selling them to or buying them from each other. Another extension is that agents may *advertise* resources they do not actually have at a certain moment, but know how to produce these by using a plan scheme. So, if during a planning process an agent needs a

certain resource, it can either try to buy the missing resource from an agent, or request an agent which advertises that particular kind of resource. In the latter case, the advertising agent will answer whether it is able to produce the requested resource. As a side effect of this mechanism, inactive agents can be triggered by other agents to become active and produce resources for them. The procedure described above is not complete: Due to the heuristic function used and the fact it doesn't back-track, it might fail to find a solution in cases one does exist.

As already mentioned, we will use our action and resource planning formalism (ARPF) (see, e.g., [9, 10, 11]).[1] Therefore, in Section 2 we will briefly summarize this framework. Then, in Section 3 we will present our cooperative heuristic multi-agent planning algorithm. We conclude the paper with a discussion and some ideas for future work.

2 The action resource planning formalism (ARPF)

The framework we will discuss is built up from three primitive notions: resources, actions and goals. Resources are the objects consumed and produced by actions, while goals are descriptions of resources we want to have.

Resources and resource schemes A (ground) *resource* is denoted by a ground atomic formula $p(t_1 : s_1, t_2 : s_2, \ldots, t_n : s_n)$ in some language $\mathcal{L}$. The terms s_j are the attribute types of the resource type p with attribute *values* t_j. As an example, take the resource $airplane(747 : id, AMS : loc)$, referring to an object of type airplane with identification 747 at location Amsterdam. *Resource schemes* are non-ground atomic formulas used to denote arbitrary resources that are *ground instances* of it. For example, $airplane(x : id, AMS : loc)$ is a resource scheme denoting an arbitrary airplane at Amsterdam.

If Var and $Term$ denote the sets of all variables and all terms, respectively, then a resource r is said to be an *instance* of the resource scheme rs if a ground substitution $\theta : Var \rightarrow Term$ exists such that $rs\theta = r$. If we have a *set* Rs of resource schemes, we would also like to know whether a given set R of resources *satisfies* Rs, i.e., whether R is a typical set of resources of the schemes denoted by Rs. Here we have to be careful: it is not sufficient to have a substitution θ such that $Rs\theta = R$, since an individual resource cannot be used to satisfy more than one resource scheme. So, for each $rs \in Rs$ we have to find a single resource $r \in R$ as an instance of rs, such that each resource is used at most once. This is accomplished by using *resource-identity preserving* ground substitutions, ip-ground substitutions for short: A substitution θ is said to be *ip-ground* w.r.t. Rs, if $rs_1 \neq rs_2$ implies $rs_1\theta \neq rs_2\theta$ for every $rs_1, rs_2 \in Rs$.[2]

A set of resources R *satisfies* a set Rs of resource schemes, denoted by $R \models Rs$, if there exists a substitution θ ip-ground w.r.t. Rs such that $Rs\theta \subseteq R$. To indicate the substitution θ explicitly, we write $R \models_\theta Rs$.

[1] Note that in the original work 'actions' were called 'skills'. We have chosen to rename these, because of their closeness to the 'actions' used in the AI planning literature.

[2] Hence, an ip-ground substitution w.r.t. Rs gives rise to an injection $Rs \rightarrow R$. Note that if θ is an ip-substitution then $|Rs| = |Rs\theta|$.

Goals and actions In general, an agent does not care about obtaining a specific (unique) resource g as a goal, but only about a resource having some specific properties. So, we will conceive an individual goal g as a resource *scheme*. This allows us to use $R \models Gs$ to express that the resource set R satisfies the set of goal schemes Gs.[3] Suppose we are given some set of resources R and we want to obtain some set of resources R' satisfying a given set of goal schemes Gs. To transform the set R into this set R', we introduce *actions* as (elementary) resource consumption and resource production processes. Such an action is a rule of the form $d : Rs' \leftarrow Rs$. Here, d is the name of the action (deed), and Rs' is a set of resource schemes containing only variables that occur in the resource scheme Rs, i.e., $var(Rs') \subseteq var(Rs)$.[4]

Intuitively, the meaning of an action d is as follows: whenever there is a set of resources R and an ip-ground substitution θ such that $R \models_\theta Rs$, the elements $Rs\theta$ of the set R are consumed by d and a set of resources $Rs'\theta$ will be produced. We denote by $in(d) = Rs$ the set of input resource schemes and by $out(d) = Rs'$ the set of output resource schemes of d. If θ is known we say that $d\theta$ is an *instantiation* of an action, that is a rule $R' \leftarrow R$ where both R and R' are sets of resources, not sets of resource schemes.

For some applications it is very important to take the costs of actions into account. In a multi-agent setting, these costs are used to determine the price of resources that are sold. We denote the costs of an action d by $costs(d) \in \mathbb{Z}$ and we also overload this function to denote the costs of a resource: $costs(r)$.

Let D be some set of actions. We say that R' *can be produced from* R *using* D, written as $R \vdash_D R'$, if there exist an action $d : Rs' \leftarrow Rs$ in D, resource sets R and R', and a substitution θ ip-ground w.r.t. $Rs \cup Rs'$, such that *(i)* $R \models_\theta Rs$, and *(ii)* $R' = (R - Rs\theta) \cup Rs'\theta$, i.e., the resources $Rs\theta$ consumed by d are removed from R, while the resources $Rs'\theta$ produced by d are added to the remaining set of resources. The reflexive-transitive reduction of $\vdash_D$ is denoted by $\vdash_D^*$.

Plans Most of the time, we will have to combine actions to produce resources that satisfy a certain goal scheme. More specifically, such actions will have to be combined in a partial order, allowing input resources to be transformed to a set of output resources satisfying the goals specified. Such a partially ordered set of instantiations of actions, transforming input resources into output resources is called a *plan*.[5] It turns out to be very useful to define the underlying graph $gr(P)$ of a plan P.

Definition 1 *Let P be a plan containing a set D of instantiated actions. Let $R = \bigcup_{d \in D} (in(d) \cup out(d))$ be the set of resources occurring in P. Then, the* plan graph *$gr(P)$ of P is the bipartite directed acyclic graph $(D \cup R, A)$, where $A = \{(r,d) \mid r \in in(d),\ d \in D\} \cup \{(d,r) \mid r \in out(d),\ d \in D\}$, i.e., the set of all arcs connecting resources to actions using or producing these resources. Furthermore,*

[3] In general, we might want goals to meet certain constraints. How this can be handled using ARPF is discussed elsewhere, e.g., in [11].

[4] That is, every action is *range-restricted*.

[5] For formal details, we refer to [11].

the set of input resources $in(P)$ *of plan* P *is defined by* $\{r \mid \nexists d \in D \cdot (d, r) \in A\}$ *and the set of output resources* $out(P)$ *by* $\{r \mid \nexists d \in D \cdot (r, d) \in A\}$.

In the sequel, we will simply identify a plan with its plan graph. Let R be some (initial) set of ground resources, and let Gs be a goal scheme. We say that P is a *plan* for Gs using R if *(i)* $in(P) \subseteq R$, i.e., the set of input resources of P is contained in the initial set, and *(ii)* $out(P) \models Gs$, i.e., the set of output resources of P satisfies the goal scheme Gs.

3 Forward heuristic cooperative planning

This section describes an algorithm that facilitates cooperative planning in a multi-agent system, based on the ARPF. We use a forward heuristic planner for each agent. This heuristic is similar to the one used in Fast Forward (FF, see [3]). We assume the agents are able to communicate for cooperation, but each agent is free to decide to what extent. Communication is implemented by offering superfluous resources to each other, and by offering resources that one agent is prepared to produce especially for another (services).

Suppose that we have a set of agents, and that each agent has some initial resources, a goal scheme, a set of actions to represent the abilities of the agent, a set of plan schemes that represent services (plans to produce resources for other agents), and a way of communicating. The problem to be solved by multi-agent planning is to find a plan for each agent to satisfy all its goals, starting from the initial resources, and to determine additional goals for each agent to represent the selling of a resource to another.

Planning We propose the following method. The plan is constructed bottom-up: Starting with the initial set of resources, actions are added to the plan. Once an action has been added it may not be removed again. This form of planning is called *forward planning*. In the multi-agent environment forward planning ensures that agents will not have to withdraw previously made commitments. The disadvantage of forward planning is that sometimes the agent may end up in a 'dead end', i.e., that it has constructed a plan that can not (be extended to) attain its goals.

The choice of which action to add next is made using a heuristic function. The heuristic function determines the cost of achieving the goal from the current state. A *state* is the result of a partial plan P, i.e., the set of resources $R = out(P)$ that has not been consumed by actions. To determine the heuristic value of a state, we use the approach proposed by Hoffmann et al. [3]. We assume each action also reproduces each of its input resources. In other words, the actions do not consume their input. Under this assumption a next action towards attaining the goals can be chosen in polynomial time. The sequence chosen within the heuristic is called a *heuristic plan*. Formally, the heuristic can be defined as follows:

Definition 2 *The heuristic value of a state* R *to attain a set of goals* Gs *using*

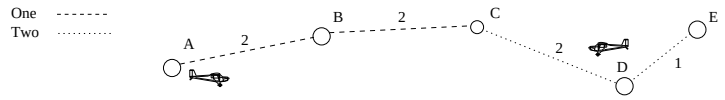

Figure 1: A situation with two agents, One and Two, that are ready to transport light weight goods and persons. Each agent has its own transport network.

the actions D that can be done by the agent is defined as

$$h(D,R,G) = \begin{cases} 0 & \text{if } R \models Gs \\ \infty & \text{if } D = \emptyset \\ \min_{\{d \mid d \in D,\, in(d) \subseteq R\}} (cost(d) & \\ \qquad +h(D - \{d\}, R \cup out(d), Gs)) & \text{otherwise} \end{cases}$$

To obtain this minimum a sequence of actions $d \in D$ is chosen. This sequence is called the heuristic plan*, denoted by $P^{h(D,R,G)}$.*

Selling resources Each agent may offer resources for sale on a blackboard. These offers are updated each time an action has been added to its plan. We allow agents only to offer two kinds of resources: First, resources ($\in R$) , which they already have produced and don't expect to need themselves, that is, resources that are not used in their own heuristic plan. And second, resources that represent certain services. The agent should be able to produce these resources on request, for example by using one of its plan schemes.

To determine the prices of resources offered, we need to develop a price mechanism. The price should be related to the costs of producing the resource, and it may be lower for resources that are superfluous than for services.

If another agent requests a resource, a goal is added to produce this resource. If the resource has already been produced, the link between this resource and the goal can directly be made for example by the introduction of a sell action in the plan. Otherwise, the requesting agent will have to wait for the resource until it occurs in the state of the partial plan.

Buy actions On the requesting side, agents can use resources offered by others by the introduction of a *buy action* for each resource that is offered. These actions can be used in the planning process in the same way as the usual actions.

Definition 3 *A* buy action *for a resource r is an action that consumes nothing, has the costs of the resource r and produces r. So $cost(buy_r) = cost(r)$.*

$$buy_r : \{r\} \leftarrow \emptyset$$

A formal specification of the method described above can be found in the CooperativePlanning Algorithm.[6]

[6]Note that it may happen that several agents would like to buy the same offered resource. To deal with these kinds of situations, we propose the following protocol. The agent whose request

Algorithm CooperativePlanning (A)

Input: *A set of agents, A, and for each agent $a \in A$ its goals G_a, its initial resources R_a, its possible actions D_a, and a set of plan schemes PS_a.*

Output: *For each agent a, its goals $G'_a \supseteq G_a$ and a plan P_a to achieve G'_a.*

begin
  for each $a \in A$ **pardo**
    $G'_a := G_a$, $P_a := (R_a, \emptyset)$, $h_a := h(D_a, R_a, G'_a)$, and $P_a^h := P^{h(D_a, R_a, G'_a)}$
  repeat
    for each $a \in A$ **pardo**
      1. **if** $out(P_a) \not\models G'_a$ **then**
           $d :=$ PlanStep(P_a, P_a^h, G'_a, h_a)
           if d is valid, actually add d to the plan P_a, otherwise exit
      2. *determine the set of resources $F_a \subset out(P_a)$ that are not used in P_a^h and offer them on the blackboard*
      3. *select the plan schemes PS_a that can be executed using F_a and offer all output resources on the blackboard*
  until *all plans P_a satisfy G'_a*
end

PlanStep (P_a, P_a^h, G'_a, h_a)
  $h'_a = \infty$ *; select an action d from P_a^h such that $in(d) \subseteq out(P_a)$*
  while $h'_a >= h_a$ *and d is valid* **do**
    1. $R'_a :=$ *result of applying action d to R_a*
    2. $h'_a := h(D_a, R'_a, G'_a)$ and $P_a'^h := P^{h(D_a, R'_a, G'_a)}$
    3. **if** *d is a buy action and $h'_a < h_a$* **then**
         if *requesting the corresponding resource failed* **then**
           $h'_a := \infty$
    4. *select an action d from P_a^h such that $in(d) \subseteq out(P_a)$*
  $P_a^h := P_a'^h$ *and* $h_a = h'_a$
  return d

Example The following example illustrates how the CooperativePlanning Algorithm should work. Assume we have three companies: One and Two each have a small airplane (with capacity 2) and rights to land at certain airports (see Figure 1) and Three is a travel agency without any airplanes, but it does have many customers that wish to travel. We suppose that One has a goal to get a passenger from B to C, and Three has also one goal: to get a passenger from B to E. Two has nothing to do, but offers some services.

One is perfectly able to reach its goal by having its airplane fly from A to C. By the exchange of free resources, Three can buy capacity from B to C, but the

has first arrived at the seller's, receives the resource. The other agent receives a message (failed) that the resource is not available anymore and proceeds with its PlanStep.

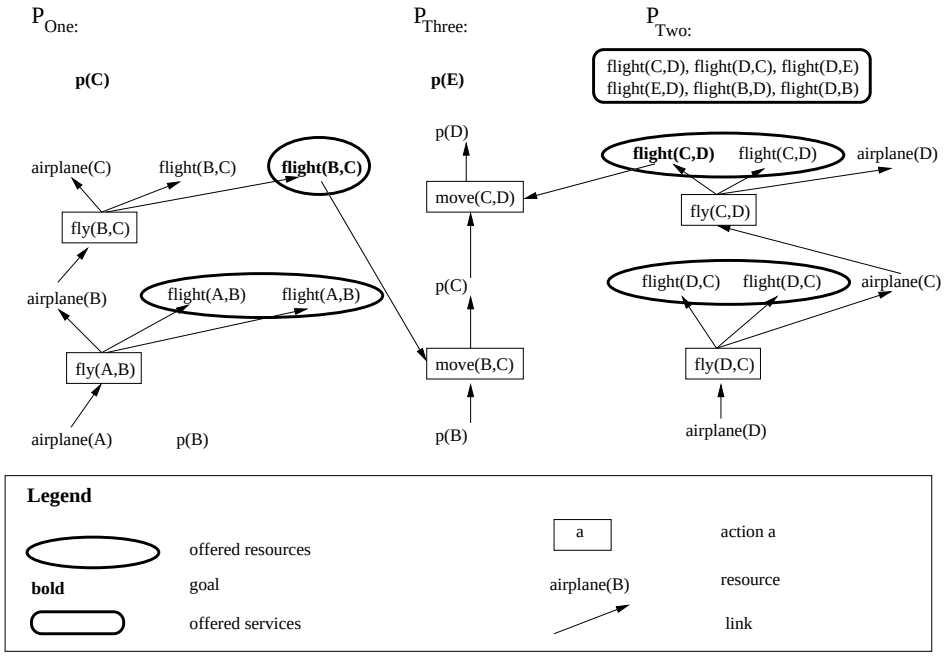

Figure 2: The partial plans of agents One, Two and Three during the planning process. Furthermore, this figure shows the resources and services that are offered by each agent.

passenger won't be able to reach E by this exchange alone. Fortunately, company Two offers many services, such as CD, DC, ED and DE. By buying for example CD and DE even Three is able to attain its goal. Intermediate states of the plans during the planning of agents One, Two and Three can be found in Figure 2. As can be seen from this example, agent Two is activated by agent Three, that otherwise would not be able to attain its goals.[7] This is the main contribution of the algorithm in this paper to the plan merging algorithm [10].

4 Discussion and future work

In this work we proposed a framework for multi-agent planning based on the exchange of resources. We showed how cooperation can be integrated with planning. The proposed method was illustrated by an example. This work is an extension of the plan merging approach [10], where resources could be exchanged after each agent had constructed a plan for itself. The advantage over plan merging is twofold: Firstly, agents are not restricted anymore to plans where they produce all resources they need by themselves (which sometimes may be impossible), but can

[7] In this example we have omitted the costs of all offered resources due to lack of space. We have also omitted the identity strings of the resources. Each resource in the figure is unique.

use resources created by others right away. Secondly, agents that are able to produce certain resources, but do not have any plans exploiting them, can be activated to do so, when prompted by other agents.

To thoroughly support the proposed method, we need to implement the algorithm and test it using real test cases. Right now we are working on an implementation. We still need to develop a solid cost mechanism to allow the use of the proposed algorithm even in competitive environments. Further future work is to show how to deal with resources with attributes with large or infinite domains. These resources introduce additional difficulties such as many or even an infinite number of possible ground actions. This problem may be solved by creating plan schemes with variables and constraints on these variables instead of ground plans (see also [11]). Constraint programming should then be used to determine applicable actions in each step of the planning algorithm.

References

[1] E. H. Durfee. Distributed problem solving and planning. In G. Weiß, editor, *A Modern Approach to Distributed Artificial Intelligence*, chapter 3. Massachusetts Institute of Technology, 1999.

[2] D. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence Journal*, 57(2–3):143–182, 1992.

[3] J. Hoffmann and B. Nebel. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of AI Research*, 12:338–386, 2000. The FF Planning System: Fast Plan Generation Through Heuristic Search.

[4] L. Hunsberger and B. J. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-00)*, pages 151–158. IEEE Computer Society Press, 2000.

[5] A. Mali and S. Kambhampati. Distributed planning. In *The Encyclopaedia of Distributed Computing*. Kluwer Academic Publishers, 1999.

[6] B.-J. Moree, A. Bos, H. Tonino, and C. Witteveen. Cooperation by iterated plan revision. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-00)*, 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] W. Walsh and M. Wellman. A market protocol for decentralized task allocation and scheduling with hierarchical dependencies. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 325–332, 1999.

[9] M. M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. Fusion of plans in a framework with constraints. In *Proceedings of the ISCS Conference on Intelligent Systems and Applications (ISA-00)*, pages 393–399, 2000.

[10] M. M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. Multi-agent cooperation in a planning framework. In *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference (BNAIC-00)*, pages 53–60, 2000.

[11] M. M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A resource logic for multi-agent plan merging. *Accepted by the Annals of Mathematics and Artificial Intelligence, special issue on Computational Logic on Multi-Agent Systems*, 2001.

[12] Q. Yang, D. S. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(4), November 1992.