Self-interested Planning Agents using Plan Repair

Roman van der Krogt and **Mathijs de Weerdt**[*]
Transport Research Center Delft
Delft University of Technology
Delft, The Netherlands
{r.p.j.vanderkrogt | m.m.deweerdt}@ewi.tudelft.nl

Abstract

We present a novel approach to multiagent planning for self-interested agents. The main idea behind our approach is that multiagent planning systems should be built upon (single-agent) *plan repair* systems. In our system agents can exchange goals and subgoals through an auction, using their own heuristics or utility functions to determine when to auction and what to bid. Some experimental results for a logistics domain demonstrate empirically that this system supports the coordination of self-interested agents.

Introduction

Most of the interesting applications of planning involve more than one agent to plan for. Often these agents are self-interested and require some privacy concerning their plans and the dependencies of actions in their plans on other agents' actions. We propose a system in which *self-interested* agents can i) *construct their plans* themselves, ii) *coordinate* their actions during planning, and do so while iii) maintaining their *privacy*. With this system we take the challenge of negotiated distributed planning that "methods must be developed for adapting the various [existing] approaches in a way that is consistent with the resource-constrained nature of planning agents: planning should be a continuous, incremental process at both the individual and group level." (DesJardins *et al.* 2000).

Our idea is to combine a dynamic planning method for each agent with an auction for delegating (sub)tasks. However, to coordinate subtasks we should deal with inter-agent dependencies (Malone & Crowston 1994) to prevent deadlocks. Currently, multiagent planning methods manage inter-agent dependencies at a central place (Wilkins & Myers 1998), or by constructing and communicating a (partial) global plan (Decker & Li 2000; von Martial 1992). Obviously, in many applications, agents are not prepared to share this kind of information.

In our system, we have a number of agents that first concurrently plan for a single goal, after which they take part in an auction (if there is any) to exchange goals and subgoals. Then, they apply a plan repair technique to add another goal to their plan, and take part in an auction again. They continue to alternatingly perform these steps of adapting a plan using plan repair and taking part in an auction until a complete and valid plan is computed. When an agent gets a task assigned on which others depend, we use a heuristic that lets the agent schedule it early in its plan to prevent cyclic dependencies. Furthermore, we give the agents some high-level information about the services others can provide to reason about which subgoals they should auction.

As an example of a situation in which this type of planning is required, consider the following logistics problem. In this domain, a number of independent planning agents have to transport goods between different locations in different cities. Each of the agents is capable of only a select number of actions: for each of the cities, there is an agent that is capable of transporting goods *within* that city, using trucks. For transport *between* cities, only one agent can transport goods by air from one airport to another. Thus, for a typical transportation order, three agents have to work together: one to bring the goods from their current location to the airport in that city, one to transport the goods to another airport, and a third and final agent is required for the transport from that airport to the destination within that same city. As these agents are different companies, they are self-interested and competitive. However, they are willing to help each other, provided adequate compensation is offered.

In this paper we show how such companies can construct their plans individually, and meanwhile coordinate (some of) their actions while maintaining their privacy. In the next section we define an abstract version of this problem more precisely, and we show how a propositional plan repair method can be combined with a simple auction to deal with this problem. Also we present solutions to subproblems such as the prevention of deadlock, and dealing with agents that do nothing but accepting orders to sell them again. The given logistics problem is used to show the suitability of these ideas. Finally, in the discussion we summarize our findings, compare them with related work, and give away our ideas for further study.

A Multiagent Planner using Plan Repair

In this paper we propose a method that dynamically creates, coordinates, and repairs plans for agents that do not want to share crucial information. We base this work on the work of

[*]This research is supported by the Technology Foundation STW, applied science division of NWO, and the technology program of the Ministry of Economic Affairs.

propositional planning, see e.g. (Kambhampati 1997). We focus on problems that can be modeled as a *set of distinct propositional planning problems* $\Pi_a = \langle O_a, I_a, G_a \rangle$, one for each agent a. In such a problem the set O_a is the set of actions that the agent a can perform, I_a is the part of the initial state the agent can observe, and G_a are the goals to be achieved by the agent. The initial state is described by propositions, an action by its preconditions and effects, and goals, preconditions, and effects are all defined by conjunctions of literals. The problems of all agents are mutually distinct, meaning that we require that there are no agents able to perform actions using the same resources (i.e., described by the same propositions) and each agent has complete knowledge about its own problem. At first this may seem too restrictive, but in many domains agents (companies) that are not cooperative can indeed not use each other's resources. For some resources of general use where conflicts may occur (such as cross roads) we may introduce an additional agent to coordinate the use of such a resource.

Note that in a propositional planning problem there is usually no realistic model of the costs and duration of actions, nor of deadlines. Therefore, the length of the plan (i.e., the number of actions) is used as an indication of the costs.

To render the problem more manageable, we assume that all actions in the domain can be undone (are reversible), and that there are no goals that are inherently unattainable. This assumption ensures that in principle a solution can always be found. We also assume that agents do not break contracts, unless they really cannot hold to them, in which case they inform the other party immediately.

The above-mentioned assumptions help us to focus on the more interesting and more difficult problem of designing a system

- that only communicates offers and bids, but little else, and
- in which agents can auction (sub)tasks to other agents, while preventing
  - cyclic dependencies,
  - lazy agents, and deal with
  - decommitment of subtasks by subcontractors.

In the following section we lay out the design of such a system and we explain how to deal with problems that may occur when building it. The crux of our idea is quite simple: coordinate (single agent) plan repair systems through a task auction. To implement this idea, we suppose that we have a dynamic planner for such problems at our disposal, such as the Partial Order Plan Repair system (POPR) (van der Krogt & de Weerdt 2005), which is based on VHPOP (Younes & Simmons 2003). Although we use the same planning system for each of the agents in our discussion, we do not rely on the specifics of this planner for the coordination of the agents. This ensures that we truly simulate a situation in which each agent is free to choose its plan repair system, although these should be extended with a common communication module.[1] We assume that such a system includes a heuristic function $\mathcal{U}(P, \Pi, g)$ that, given a problem Π and a plan P, estimates the costs of adapting P to achieve another goal g. Usually such a system is only able to solve *single* instances of a propositional planning problem, not a combination of them. How to combine a number of these systems to form a multiagent planning system is the topic of this paper.

[1]Such a plan repair system can be derived from the planning system that the agent is currently employing by using the techniques of (van der Krogt & de Weerdt 2005).

Planning The first important decision made to achieve the properties described above is to process the goals one-by-one by a plan repair system, instead of in a single batch (as is usual in planning). This has a number of advantages: firstly, failure to add a goal to the plan immediately tells us which goal we should put up for auction, while when planning for a batch of goals fails, it is not immediately obvious which of the goals cannot be achieved. Secondly, we get regular moments at which we can easily make changes in the problem. Moreover, at these moments we have a valid plan that partially achieves our goals to base our decisions on. This means that we can make a more informed decision than if we would interrupt a regular planner at certain points. There is one disadvantage, however: we cannot as easily exploit positive interactions that may exist between goals. In the section on our experimental work, we shall come back to this issue.

We now describe the basic steps of this goal-by-goal planning approach. The process starts by taking the original planning problem Π, and creating a goal queue Q from it (containing all the goals that are to be solved in order to solve Π) as well as the problem Π_{PR}, initially identical to Π, but without goals. We use this problem Π_{PR} to keep track of the problem that we are trying to solve in the current iteration. Later it may contain additional goals that this agent has accepted from others, and it does not need to include all goals from Π (as some might not have been planned for yet, or are currently planned for by other agents). To plan a single goal g from Q the system performs the following steps:

1. It queries the planning heuristic $\mathcal{U}$ of the plan repair system to estimate the cost of adding g to the plan.
2. The heuristic may report that it cannot incorporate the goal, or that the costs of incorporating are so large that it is preferable to ask other agents for help. If this is the case, the agent passes this goal to the blackboard for auction. Otherwise, it removes it g from Q, adds it as a goal to Π_{PR}, and updates its plan for this new planning problem using the plan repair system.

These steps are interleaved with processing auctions (if any), as discussed hereafter. Once the goal queue is empty, each goal of the agent is either planned for in its own plan, or has been given to another agent (via the blackboard). From this point on, the agent stays active to respond to auctions until all other agents are finished as well.

Having described the basic planning loop, we turn to the multiagent specifics. First we discuss the auctioning of goals. Then, we describe a way with which we can, during the planning phase, decompose a goal into subgoals, some of which the agent might not be able to achieve itself. These subgoals then can also be auctioned.

Auctions As said, an agent planning a goal first consults its planning heuristic to discover whether it is advisable to plan for this goal itself. If it is not, or if it turns out after plan repair that the goal is unattainable, the agent will put the goal up for auction. For ease of implementation, this auction is currently run by a blackboard, but it can be distributed over the agents as well, of course. The blackboard keeps a list of auctions, and processes them one-by-one. This prevents additional difficulties that agents face when dealing with multiple simultaneous auctions (such as the "eager bidder" problem (Schillo, Kray, & Fischer 2002)). For each auction, the blackboard sends out request for bids. Note that we currently process the auctions in order of arrival. In the future, we might include a priority which determines the order of the auctions.

When an agent receives a request to bid on a goal, a heuristic is applied to discover whether this agent can incorporate the new goal in the current plan. If so, this also tells us what the estimated cost is of adapting the plan. This value is then sent as our bid for this goal. In the current system, we have chosen to allow the agents a single, sealed bid.

The blackboard waits for all bids, and selects the cheapest bid. The winner is awarded the goal, and receives payment equal to the second-lowest bid (Vickrey 1961).[2] Upon being awarded a goal g, the agent adds g to the front of its goal queue. This ensures that this goal is processed next, and that the agent can actually attain g by repairing its plan. If we allow other goals to be processed first, g might no longer be achievable. This would require decommitment of the agent, a situation that we would rather prevent. Only in the unlucky event that during plan repair it turns out that the heuristic was completely wrong, decommitment is performed as discussed on the next page.

Services Exchanging goals is necessary but not sufficient for a complete multiagent planning system, for it is often the case that a certain goal cannot be achieved by a single agent, but only through cooperation. For example, moving a package from one city to another in our example logistics domain requires three agents to work together. Hence, we also need to be able to decompose goals into subgoals that may have to be carried out by other agents. To continue our example, moving a package from one city to another decomposes into three subgoals: delivering the good to the airport in the source city, bringing the good to the airport in the destination city, and finally the delivery to the destination. Decomposition is not trivial, but fortunately, we can do some decomposition during the planning phase.

To perform decomposition during planning, agents need some knowledge on the actions (or groups of actions) that other agents can perform. We encode such knowledge as *services*. A *service* is a task that can be achieved by one or more other agents. It is not required to know *how* a task is achieved, nor is it required to know *who* can exactly achieve portions of a task. For example, in the distributed logistics domain, a trucking agent in a city might know that other agents can achieve the task to bring a certain package from other cities to the airport in his own city.

We model services as regular actions. To distinguish such actions from regular actions we refer to them as *external actions*. Like regular actions, external actions can be integrated in the plan during the planning phase to indicate that help from other agents is required. At the end of a planning iteration (in which an additional goal from the goal queue is planned for), the effects of new external actions are sent to the blackboard for auction. In this way, propositions can be "exchanged" between agents.

The complete planning loop Having described the features of the algorithm in isolation, we now end this section with the complete algorithm as we have used in our experiments. The algorithm is presented below, and starts with setting up some data structures, such as the goal queue Q, and the initial planning problem Π_{PR}. Then, in step 4, it tries to add a goal from the queue to the current plan P. At first, in step 4.2, we compute the heuristic value $\mathcal{U}(P, \Pi_{PR}, g)$ of establishing g with P. If this is estimated to cost more than the agent is willing to spend (with an unsatisfiable goal returning ∞), we send the goal to the blackboard for auction. Otherwise, we update the planning problem Π_{PR}, and compute the new plan. If this plan contains any external actions, the subgoals they satisfy are sent to the blackboard for auction. Having processed a goal from the queue (if any), we check whether a goal g' is currently being auctioned. If so, we compute our costs for it (using the heuristic $\mathcal{U}$ of the plan repair system), and send this as a bid to the blackboard. If our bid is winning, we add the goal g' to the front of our goal queue.

```
PLANNINGLOOP ($\Pi$)
Input: A problem $\Pi = \langle O, I, G \rangle$
begin
  1. Setup the goal queue $Q$ containing all goals from $\Pi$
  2. Create the initial problem description $\Pi_{PR} = \langle O, I, \emptyset \rangle$
  3. Create the initial (empty) plan $P$
  4. if $Q$ is not empty then
       4.1. pop goal $g$ from $Q$
       4.2. Estimate cost for this goal: $c = \mathcal{U}(P, \Pi_{PR}, g)$
       4.3. if the goal is too expensive then
              send $g$ to the blackboard for auction
       4.4. else
              4.4.1. Update problem: $\Pi_{PR} = \Pi_{PR} \cup \{g\}$
              4.4.2. Update plan: $P = PR(P, \Pi_{PR})$
              4.4.3. if $P$ contains external actions then
                       request results (subgoals) of these actions
                       via an auction (blackboard)
  5. if an auction is ongoing for a goal $g'$ then
       5.1. send bid (which is $\mathcal{U}(P, \Pi_{PR}, g')$)
       5.2. if goal is awarded then
              push $g'$ onto the front of $Q$
  6. goto step 4
end
```

[2] Note that with a repeated auction the main advantage of a Vickrey auction (that it is a dominant strategy to bid ones private value) is lost for agents that reason about future auctions. Other types of auctions are a topic for future study.

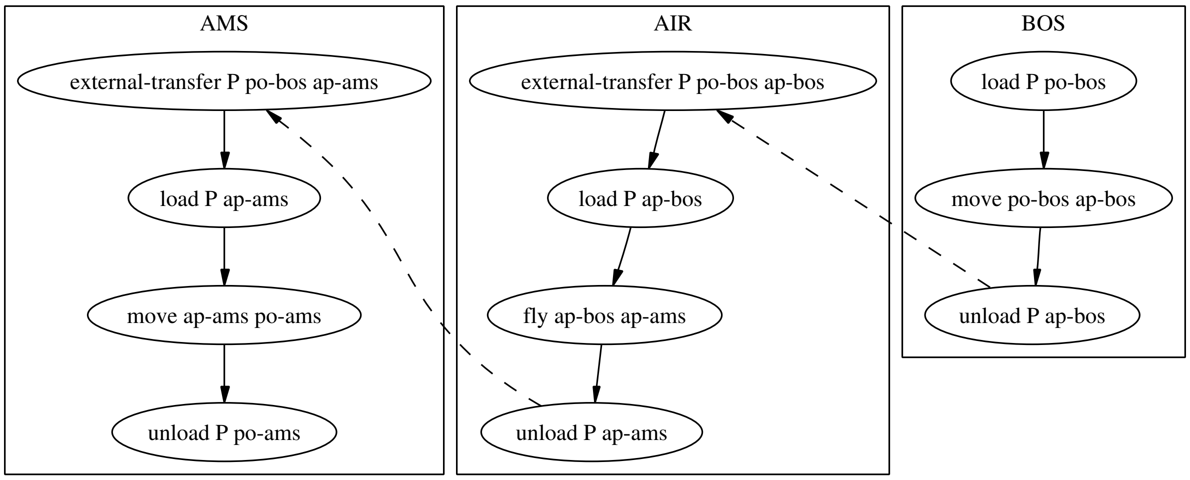

Figure 1: The multiagent plan for transporting package P from $po-bos$ to $po-ams$ using three agents: BOS, AIR, and AMS.

Note that besides the details mentioned in the algorithm a bit more bookkeeping is necessary. For example, we have to detect when everybody has finished planning for all their goals. Currently this is being recorded at the blackboard, where every agent can declare itself ready (and can remove that declaration when it accepts a new goal from the blackboard). Also, in step 4.4.3 where the subgoals from external actions are auctioned, we should only send those subgoals that were not present in the plan before. Finally, we observe that, currently, agents do not receive feedback on their goals that have been sent for auction. That is, if none of the other agents bids on a (sub) goal, the auctioneer will continue to periodically try to auction this goal, instead of reporting this to the original sender who should then try and find another solution. For the domains that we have used in our experiments, this is no problem, as there is always someone who can achieve the goal (the goals do not have time limits). For more complicated domains this does not hold, of course. We will come back to this issue in our discussion on future work.

Example. Suppose the following logistics problem. There are two cities, Amsterdam and Boston, each with an airport (denoted by $ap-ams$ and $ap-bos$, respectively). A package P is to be transported from the post office in Boston ($po-bos$) to the post office in Amsterdam ($po-ams$). In each city, there is a transportation company that uses trucks to transport goods within the city. Furthermore, there is an airline company that can transport goods between airports. In the domain of each agent one external action is present, called `external-transfer`, that describes that other agents are capable of transportation as well.

Initially, the goal (from $po-bos$ to $po-ams$) is given to the trucking agent in Amsterdam (we refer to this agent as AMS, the trucking agent in Boston is referred to as BOS, and the airline agent is denoted by AIR). AMS queries its heuristic and finds out that it can reach this goal. Using plan repair on the empty plan, it comes up with the following plan:

1. `external-transfer` P from $po-bos$ to $ap-ams$
2. `load` P at $ap-ams$
3. `move` from $ap-ams$ to $po-ams$
4. `unload` P at $po-ams$

The effect of the external action is a proposition $at(P, ap-ams)$, which AMS sends to the blackboard. In the following auction, BOS bids ∞ (it cannot reach this goal) and AIR bids 4. Hence, the goal is awarded to AIR, which creates the following plan for it:

1. `external-transfer` P from $po-bos$ to $ap-bos$
2. `load` P at $ap-bos$
3. `fly` from $ap-bos$ to $ap-ams$
4. `unload` P at $ap-ams$

This results in the goal $ap(P, ap-bos)$ being auctioned, which is subsequently won by and planned for by BOS. This completes the multiagent plan for delivering the package from Boston to Amsterdam using all three agents as shown in Figure 1.

Coordination problems

Whereas the former section gave an overview of the basics of our multiagent planning system, in practice one bumps into some additional difficulties that need to be solved. In this section we pay attention to three important issues that we encountered in building the system.

"Lazy" agents The first issue that we encountered in our initial experiments was that sometimes an agent accepted a goal from the blackboard, and then planned to use a service to have other agents satisfy the exact same goal. We called these agents "lazy agents", for they did not want to do some work themselves. In some experiments, this is a minor inefficiency, but in other problems two of such lazy agents were present who were continuously bouncing the same goal back-and-forth. As a solution we considered goals *tabu* for production by external actions. That is, when adapting the plan to include a goal g, no external actions may be planned that produce g. We adapted both the planning algorithm and the heuristic to honor these tabus.

Decommitment of bidders As indicated in the previous section, agents place their bids based on a heuristic estimate of the costs of changing their plans. Using a heuristic has two important repercussions: firstly, an agent may need to modify its plan in a far greater (and possibly more expensive) way than anticipated, and secondly, an agent may find that it cannot achieve the goal at all. When an agent discovers that it cannot actually satisfy the goal it has bid on, it indicates this in a message to the blackboard. The blackboard then re-auctions the goal, disregarding the bids of agents that have bid on the goal and rejected it before. Under the assumptions we made, there is always at least one agent capable of achieving the goal. Since bids of agents that have rejected a goal are disregarded, the goal will eventually be awarded to the agent that can satisfy the goal. For now, a decommitting agent pays (as a penalty) the cost difference between its own bid and the next one.[3]

Managing dependencies A distributed planning system, such as presented in the previous section, should ensure the validity of the proposed plans. For this, it is required that not only the individual plans are valid, but also that the combination of plans is valid. In particular, we should verify three conditions:

1. Actions in different plans may not interfere. As explained in the previous section, strongly autonomous agents, such as competitive companies, usually have distinct areas of control, or an additional agent can be introduced to ensure mutual exclusion of shared resources.
2. If an agent depends on another agent to provide a subgoal, another agent should actually provide this subgoal, and
3. the combined plans may not contain *cyclic dependencies*. That is, it may not be that an action a is (indirectly) dependent upon an action (of another agent) that is dependent on an effect of a.

The second condition is ensured by our assumption that all agents are sincere: when an agent promises to provide a subgoal, it will either do so, or inform the blackboard that it cannot. The last condition is the most difficult to guarantee, because in principle agents need to know the details of the other agents' plans to ensure this property. In existing solutions to prevent cyclic dependencies either a central facility is keeping track of dependencies (Wilkins & Myers 1998), or agents communicate to form a so-called partial global plan (Decker & Li 2000).

In our goal-by-goal approach, however, we can use a so-called backward planning heuristic. When an agent plans a task for someone else, it can prevent cycles from occurring without any additional communications by placing all actions (possibly including external actions) required for this task *before* all other actions in its plan. This heuristic depends for a great deal on the fact that only one goal is auctioned by the blackboard in a single iteration. Thus, only one agent can create a new inter-agent dependency at a time. If we ensure that this new dependency is not dependent upon previously existing dependencies, we prevent cycles from occurring. Any additional external actions inserted will be auctioned only after this part of the plan has been completed. Note that the need for this heuristic disappears when using a domain in which time is explicitly represented and a planner that can reason with time, since time attributes can be used to prevent cyclic dependencies.

Example. Consider the logistics problem of the previous example. Suppose that a second package P' would have to be transferred in the opposite direction, i.e., from the post office in Amsterdam ($po-ams$) to the post office in Boston ($po-bos$). BOS already had a plan for transporting P from $po-bos$ to $ap-bos$, but now it creates the following plan for this situation:

1. `external-transfer` P' from $po-ams$ to $ap-bos$
2. `load` P' at $ap-bos$
3. `move` from $ap-bos$ to $po-bos$
4. `unload` P' at $po-bos$
5. `load` P at $po-bos$
6. `move` from $po-bos$ to $ap-bos$
7. `unload` P at $ap-bos$

After the auctions for transporting P have been dealt with, AIR and BOS have computed the plans from the previous example. Then, the auction of $at(P', ap-bos)$ takes place, which is won by AIR. Due to the backward planning heuristic, AIR inserts the actions for this subgoal before its other actions:

1. `fly` from $ap-bos$ to $ap-ams$
2. `external-transfer` P' from $po-ams$ to $ap-ams$
3. `load` P' at $ap-ams$
4. `fly` from $ap-ams$ to $ap-bos$
5. `unload` P' at $ap-bos$
6. `external-transfer` P from $po-bos$ to $ap-bos$
7. `load` P at $ap-bos$
8. `fly` from $ap-bos$ to $ap-ams$
9. `unload` P at $ap-ams$

Thus, the added actions (related to P') do not have to wait for existing actions to finish. Hence, AIR cannot create a cyclic dependency. Had it tried to reuse part of its existing plan (like AMS did) by first waiting for the external action related to P (step 6), it would have created a cyclic dependency, because BOS firsts waits for the external action related to P' to finish (step 1).

Experimental Results

For our experiments we applied the method described in the previous section to the POPR plan repair system (van der Krogt & de Weerdt 2005), which is an adaptation of the VHPOP planner by (Younes & Simmons 2003). We used a series of benchmark problems from the AIPS competition (Bacchus *et al.* 2000) in the logistics domain that was used as an example before. We took a total of 11 problems, varying from 2 to 5 cities, and from 4 to 15 goals. The number of cities grows as the number of goals do: for a problem

[3]We are considering leveled-commitment contracting (Sandholm 2002) to enable strategic decommitting.

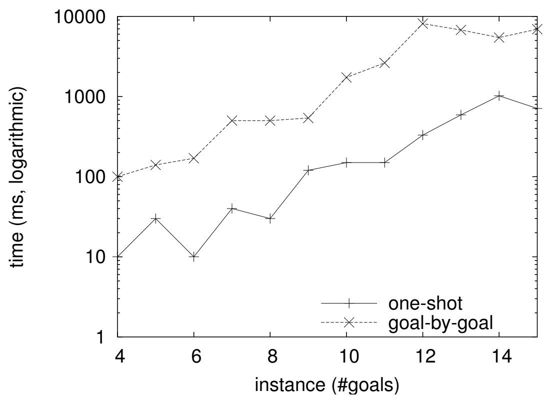

Figure 2: Run times of one-shot planning and goal-by-goal planning by a single agent.

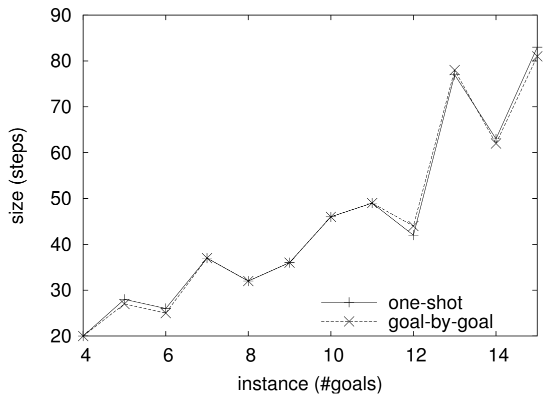

Figure 3: Plan lengths of one-shot planning and goal-by-goal planning by a single agent.

with n goals, $\left\lceil \frac{n}{3} \right\rceil$ cities are used. Each goal consists of the transport of a single package from one location to another. Transportation can be performed by truck (within a city), or by plane (between airports in different cities). Sometimes, the transportation orders are within a city, but for most orders, the destination city is different from the starting city. Because there are no deadlines in this domain, goals can eventually always be achieved.

Goal-by-Goal Planning

The first question that we posed is the following. In our current approach, we choose to plan goal-by-goal. That is, instead of considering all goals at once, in a single planning problem, we first plan for one goal, then add another, etc. The question is how this affects the quality of our solutions, and of course the speed with which we reach these solutions. To investigate this, we compared the plans produced by VHPOP (which plans in a single batch) and the plans produced by POPR, when given a series of plan repair problems in which the goals were added one by one.

Figures 2 and 3 show the runtime and plan quality respectively when goal-by-goal planning is compared with solving a single planning problem involving all goals. Although, in principle, a planning problem can be solved more efficiently by dividing it into subgoals (Korf 1987), we can see from Figure 2 that goal-by-goal planning takes quite some more time than solving a single planning instance. This is due to the fact that for each goal, a new planning problem is created, which invalidates a lot of the structures that were created before. For example, part of the heuristic that POPR inherits from VHPOP relies on a planning graph, which is currently created completely anew for each planning problem, whereas it can be reused when we solve multiple goals within a single problem. In principle, this could be mitigated by realizing that we are not solving *any* plan repair problem, but a specific one in which only a single goal is added. This would allow the unrefinement heuristic (which can be used to solve general plan repair problems) to reuse some of the existing structures that are not invalidated.[4] For our current system, we have not done so, however.

Concerning the quality of the plans, we can see from Figure 3 that one-shot planning for a single problem or goal-by-goal planning makes hardly any difference. This is to be expected, as VHPOP (also) uses a LIFO queue for its goal agenda and hence tries to completely satisfy one goal (including all subgoals that lead to this goal) before working on the next one.

Multiagent Planning

The more important part of our experiments obviously has to do with multiagent planning. In particular, we want to verify that our approach is feasible. The planning problems used in the previous section were translated into their multiagent counterparts by introducing a number of agents as follows: for each city, we introduced an agent that is capable of transport within that city (using trucks). One additional agent was given control over the airplanes, and is hence capable of inter-city transports. We used two types of domains: one for the inner-city agents and one for the inter-city agent. The former consisted of the usual `load`, `unload` and `move` actions, with which cargo can be loaded, transported and unloaded. In addition, we added an `external-transport` action that represents the knowledge of the other agents' capabilities. It specifies that any package in any location can be moved to the local airport by some means. The domain of the inter-city agent also consisted of four actions: `load`, `unload` and `fly` to transport goods from one airport to another, and an `external-transport` action specifying that other agents are capable of transporting goods between two locations within the same city.

These logistics problems were used to verify that our approach is feasible. The run time for three different cases can be seen in Figure 4. The first case is labelled *one-shot*. This shows the run time of the unmodified (central) VHPOP planner on the benchmark problem. In this case, we have a

[4]For example, the reason to regenerate the planning graph is that the initial state or the set of available actions might have been changed. Since this is not the case in our specific plan repair problems, the planning graph can be reused.

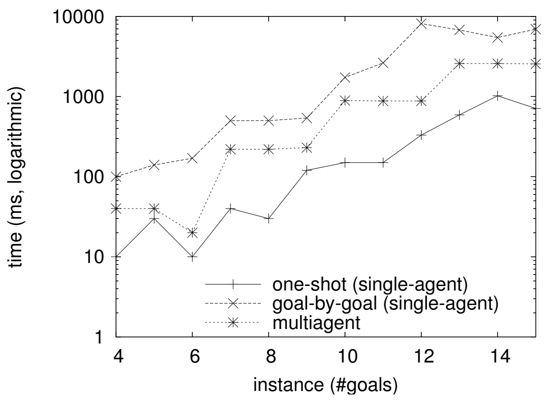

Figure 4: Run times of multiagent planning compared to single-agent planning. The time reported for the multiagent experiments is the time it took the slowest of the agents to compute its plan (the “make span”).

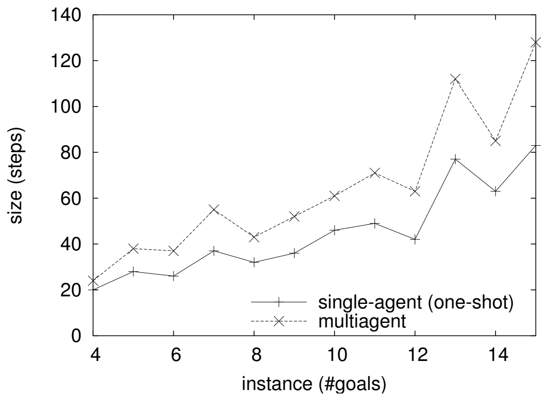

Figure 5: Plan lengths of single-agent one-shot planning, and the cumulative size of the multiagent plans.

single agent that plans for all trucks and airplanes. (Clearly, this is a hypothetical situation for a domain involving self-interested companies.) The second case, labelled *goal-by-goal* shows the amount of time it takes a single planner to create a plan for this problem when it uses a goal-by-goal approach. The third case is labelled *multiagent*. In this case, we used one planner to plan for the transport of goods in each of the cities (thus, for n cities, we used n planners) and a single planner for the planning of inter-city transportation orders (thus, a total of $n + 1$ planners for problems with n cities). As we can see, for these problems multiagent planning is considerably faster than planning centrally using a goal-by-goal approach, due to the fact that we can have different agents plan in parallel.[5] Notice that the differences are significant (as one might guess from the figure), as can be seen from the results of paired t-tests we performed:

		t	p
one-shot	goal-by-goal	-3.0756	< 0.02
one-shot	multiagent	-3.106	< 0.01
goal-by-goal	multiagent	2.7874	< 0.02

Besides run-time performance, plan quality is also important. Figure 5 shows the size of the resulting plans. As expected, the backward planning heuristic that we employ has a negative effect on the size of the plans, compared with a centralized solution. This is because it forces an ordering on the agents' plans that is stricter than necessary. As a result, the plans that we obtain are significantly bigger (a paired t-test results in t=-5.0344 and $p < 0.01$). This is the price one has to pay for not exchanging detailed information on the structure of the plans. An important question for future work is whether we can relax the ordering that is imposed by the heuristic a little, allowing us to reuse a part of the existing plan.

Discussion

In this paper we gave experimental evidence that self-interested agents can plan and coordinate their plans while only exchanging a very small amount of information. Our method should work with any plan repair algorithm, allowing agents to choose their own dynamic planner. We described how to use such an existing plan repair algorithm in a goal-by-goal setting and a simple auction, we showed how to prevent cyclic inter-agent dependencies, and how to deal with lazy agents and decommitment by a bidder that overshooted itself.

We studied the difference in both plan size and planning time between multiagent planning and single-agent planning. It turns out that our distributed approach produces longer plans than central solutions. This can be mainly attributed to our cycle-prevention heuristic, which is often too restrictive. However, it allows us to create valid multiagent plans without exchanging details about the plans, which is very important for self-interested agents.

The distribution of the planning problem in a multiagent planning system leads to an improvement of planning performance compared to a single-agent solving a planning problem goal-by-goal. We expect that for more realistic and more complicated domains the difference may be even larger, since agents can do a lot of work in parallel. Summarizing, from the experiments we conclude that it is indeed possible to use multiple single-agent plan repair systems to let self-interested agents plan for their goals individually, and request (or provide) help when necessary.

Related Work

This system for coordinating self-interested agents using propositional plan repair is unique in that we do not assume that the agents are *collaborating*. Agents may even be each

[5]For these easy problems a planning cycle takes about 5-10ms, while one communication takes about 40ms, because Linux schedules processes in slots of at least 10ms and communication uses at least 4 different processes. In realistic (i.e. complicated) domains the planning component is the dominating factor in the total run time. Therefore we focused on the time required for planning. The total time including communication is only slightly better than the single-agent goal-by-goal results for these simple problems.

other's competitors. Previous work on multiagent planning, although often more advanced in modeling problems realistically (by involving time constraints, minimizing costs, and efficient use of resources) assumes that the agents are collaborative. For example, in the Cougaar system (Kleinmann, Lazarus, & Tomlinson 2003) cooperative agents are coordinated by exchanging more and more details of their hierarchical plans until conflicts can be resolved (similar to (von Martial 1992)).

The Generalized Partial Global Planning (GPGP) method (Decker & Lesser 1992; Decker & Li 2000) describes a framework for distributedly constructing a (partial) global plan to be able to discover all kinds of potential conflicts. In GPGP agents exchange parts of their plans, so that each agent can build a partial global plan, containing the knowledge that this agent has of the other agents' plans. Using this partial global plan, the agent can detect possible positive and negative effects, and deal with them. To use GPGP, however, the agents need to trust each other with some of the details of their plans. Self-interested agents are not prepared to do this. A similar line of reasoning holds for most of the cooperative (often hierarchical) and mixed-initiative multiagent planning systems. Of these, the idea of planner-independent collaborative planning by Kim and Gratch (Kim & Gratch 2004) is particularly interesting in view of our idea for planner independence. They use such planners to solve small problems that can support the decision process of the user. In their situation there is no need for plan repair or cooperation.

Thirdly, in (Brenner 2003) a method using partially ordered temporal plans is proposed to solve multiagent planning problems in such a way that agents can ask others about the state of the world, who will (truthfully) answer as soon as possible. This work relaxes our assumption that agents have complete knowledge about the relevant part of the world, but in all of the above mentioned systems the agents are not self-interested.

Finally, we would like to compare our method to a multiagent planning approach based on the COMAS system (Cox, Elahi, & Cleereman 2003). In their approach each agent has one or more *unique* capabilities. Each agent can directly request such a 'specialist' when it needs its capability (based on knowledge about other agents' capabilities). The requesting agent is then sent a complete subplan that it can include in its plan. Besides the exchange of a lot more information than in our method (both beforehand and during planning), their system also takes a rather simple approach to preventing cyclic dependencies: they assume that actions that can possibly lead to cyclic dependencies (e.g. the load/unload pair of actions in logistics) can only be executed (and hence planned for) by a single agent.

Plan merging systems (Tsamardinos, Pollack, & Horty 2000; de Weerdt *et al.* 2003) can also be used by self-interested agents in order to coordinate their plans. In these systems, each agent builds its own plan, without exchanging information with the other agents. When all plans have been computed, limited information is exchanged to detect possible interactions. Clearly, these approaches are static and cannot be used to request help from other agents during the planning process.

Another line of research concerns the *reasoning* behind the creation of multiagent plans. Examples of this type of research are the work on *joint intentions* by Cohen and Levesque (1991) and the SharedPlans approach of Grosz and Kraus (1999). Although both these approaches focus on collaborative behaviour, some aspects are important to our work as well. Firstly, when one of our agents carries out an action to bring about a subgoal of another agent this can be seen as a particular type of joint intention. Secondly, we are considering a formalisation of our approach similar to the theory of elaborating multiagent plans as presented in the SharedPlans framework

Next to this work on coordinating multiagent plans, there is also a substantial body of work on *task allocation* for self-interested agents. For example using market mechanisms (Walsh & Wellman 1999), or using extensions of the contract-net protocol (Collins *et al.* 1998; Smith 1980). Ideas from this work may be used to improve the simple auction of our approach, for example to enable parallel or combinatorial auctions. Task (re)allocation, however, cannot completely be disconnected from planning. In our work we focus not so much on task allocation, but on coordinating the agents' *planning* and *plan repair* behavior (without the construction of a global set of constraints).

Future Work

Since our initial experiments showed promising results, we intend to continue this line of research towards a fully equipped multiagent planning system. Besides looking at improvements to our heuristic, one of the first things to do is to relax some of our assumptions to be able to tackle more advanced problems. First of all we would like to have a method to estimate the costs of external actions. Typically "external" actions are more expensive than your own actions. If all actions have costs, we can try to optimize costs instead of plan length. In most domains this may give more realistic solutions. For example, there may be two airports in a city, each serviced by a different airline company. Our current system cannot distinguish between the two options. When the costs of such external actions are known, the most efficient option can be chosen. Another important topic for future study is using a different type of auction and (de)committing mechanism (e.g. (Hoen & Poutré 2003; Sandholm 2002)) that matches the specific requirements of efficiently allocating sets of subtasks to self-interested planning agents.

Another important issue for further study is to give feedback to the agent on their auctioned goals. As we indicated in a previous section, the agents currently submit their auctions to the auctioneer, and assume they will be successfully auctioned. However, it may very well be that no other agent bids on a certain goal, in which case the agent submitting the auction should reconsider its plan, since its subgoals cannot be achieved. Also, in many applications, agents may need to deal with a very dynamic situation where actions may turn out to be disabled or planned goals may become useless. We would like to find an efficient coordination mechanism that can use the plan repair systems of the agents to remove parts of their plans that become irrelevant.

Furthermore, the algorithm for each agent is currently sequential: it processes a goal, then an auction, then a goal again, and so on. In the future we would like to have two independent subprocesses per agent taking care of each of these tasks. The same holds for the blackboard: it auctions goals one at a time, whereas we might want to have multiple simultaneous auctions, or smart heuristics for ordering the goals before auctioning. Finally, we would like to investigate whether exchanging just a tiny bit more information about the dependencies of actions (or for example making contracts that include time constraints) can lead to a more efficient plan and to more individuality by relaxing the heuristic of 'planning actions for others first in your plan'.

References

Bacchus, F.; Kautz, H.; Smith, D. E.; Long, D.; Geffner, H.; and Koehler, J. 2000. The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems Planning Competition. http://www.cs.toronto.edu/aips2000/.

Brenner, M. 2003. Multiagent planning with partially ordered temporal plans. In *Proceedings of the Doctorial Consortium of the International Conference on AI Planning and Scheduling*.

Cohen, P., and Levesque, H. 1991. Teamwork. *Nous* 25(4):487–512.

Collins, J.; Tsvetovatyy, M.; Gini, M.; and Mobasher, B. 1998. MAGNET: A multi-agent contracting system for plan execution. In *Proceeding of the Workshop on Artificial Intelligence and Manufacturing (SIGMAN-98)*.

Cox, M. T.; Elahi, M. M.; and Cleereman, K. 2003. A distributed planning approach using multiagent goal transformations. In *Fourteenth Midwest Artificial Intelligence and Cognitive Sciences Conference*, 18–23.

de Weerdt, M. M.; Bos, A.; Tonino, J.; and Witteveen, C. 2003. A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence, special issue on Computational Logic in Multi-Agent Systems* 37(1–2):93–130.

Decker, K. S., and Lesser, V. R. 1992. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems* 1(2):319–346.

Decker, K. S., and Li, J. 2000. Coordinating mutually exclusive resources using gpgp. *Autonomous Agents and Multi-Agent Systems* 3(2):113–157.

DesJardins, M. E.; Durfee, E. H.; Ortiz, C. L.; and Wolverton, M. J. 2000. A survey of research in distributed, continual planning. *AI Magazine* 20(4):13–22.

Grosz, B. J., and Kraus, S. 1999. The evolution of SharedPlans. In Rao, A., and Wooldridge, M. J., eds., *Foundations and Theories of Rational Agency*. Dordrecht, The Netherlands: Kluwer Academic Publishers. 227–262.

Hoen, P.J., t., and Poutré, J.A., L. 2003. A decommitment strategy in a competitive multi-agent transportation setting. In *Proceedings of the AAMAS-03 Workshop on Agent Mediated Electronic Commerce V: Designing Mechanisms and Systems*, volume 3048 of *Lecture Notes on Artificial Intelligence*.

Kambhampati, S. 1997. Refinement planning as a unifying framework for plan synthesis. *AI Magazine* 18(2):67–97.

Kim, H.-S., and Gratch, J. 2004. A planner-independent collaborative planning assistant. In *Proceedings of the Third International Conference on Autonomous Agents and Multi-Agent Systems*, 764–771.

Kleinmann, K.; Lazarus, R.; and Tomlinson, R. 2003. An infrastructure for adaptive control of multi-agent systems. In *IEEE Int. Conf. on Integration of Knowledge Intensive Multi-Agent Systems*, 230–236.

Korf, R. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.

Malone, T. W., and Crowston, K. 1994. The interdisciplinary study of coordination. *ACM Computing Surveys* 21(1):87–119.

Sandholm, T. W. 2002. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* 135(1–2):1–54.

Schillo, M.; Kray, C.; and Fischer, K. 2002. The eager bidder problem: A fundamental problem of DAI and selected solutions. In *Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems*, 599–606. ACM Press.

Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C-29(12):1104–1113.

Tsamardinos, I.; Pollack, M. E.; and Horty, J. F. 2000. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, 264–272. Menlo Park, CA: AAAI Press.

van der Krogt, R., and de Weerdt, M. 2005. Plan repair as an extension of planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-05)*.

Vickrey, W. 1961. Computer speculation, auctions, and competitive sealed tenders. *Journal of Finance* 16:8–37.

von Martial, F. 1992. *Coordinating Plans of Autonomous Agents*, volume 610 of *Lecture Notes on Artificial Intelligence*. Berlin: Springer Verlag.

Walsh, W. E., and Wellman, M. P. 1999. 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)*, 325–332. An extended version of this paper is also available.

Wilkins, D., and Myers, K. 1998. A multiagent planning architecture. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, 154–162. Menlo Park, CA: AAAI Press. Also available as a technical report.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of AI Research* 20:405–430.