Coordination through Plan Repair

Roman van der Krogt and Mathijs de Weerdt[⋆]

Delft University of Technology
PO Box 5031, 2600 GA Delft, The Netherlands
{R.P.J.vanderKrogt,M.M.deWeerdt}@ewi.tudelft.nl

Abstract. In most practical situations, agents need to continuously improve or repair their plans. In a multiagent system agents also need to coordinate their plans. Consequently, we need methods such that agents in a multiagent system can construct, coordinate, and repair their plans. In this paper we focus on the problem of coordinating plans without exchanging explicit information on dependencies, or having to construct a global set of constraints. Our approach is to combine a propositional plan repair algorithm for each agent with a blackboard that auctions subgoals on behalf of the agents. Both the details of a first construction and some initial experimental results are discussed.

1 Introduction

In most application domains for distributed artificial intelligence, several autonomous agents (e.g., companies or personal assistants) each have their own goals and therefore need to plan, to coordinate their actions, to deal with uncertainty, and to interleave all this with plan execution [1, 2]. To complicate matters even more, many agents are self-interested and require some privacy concerning their plans and the dependencies of actions in their plans on other agents' actions. In fact, multiagent planning methods are often preferred above mature central planners exactly because of privacy reasons. Our goal is a system in which *self-interested* agents can *(i)* construct and *repair plans*, *(ii) coordinate* their actions, and do so while *(iii)* maintaining their *privacy*.

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 [3] to prevent deadlocks. Currently, multiagent planning methods manage inter-agent dependencies at a central place [4], or by constructing and communicating a (partial) global plan [5, 6]. Besides coordinating tasks, we also have to *allocate* tasks to different agents. In contrast to existing work on task and role allocation where task allocation is seen as one static problem (such as [7, 8]), the set of subtasks that has to be allocated in our setting arises from the planning produced by the agents. Planned subtasks that agents cannot carry out themselves need to be reallocated. The planning in its turn is of course influenced by the existing task allocation.

Our approach raises a number of questions. If agents do not want to construct a set of global constraints, how can we then guarantee that there will be no deadlocks

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

upon execution of their plans? How can agents integrate the process of auctioning or bidding with the process of plan construction or adaptation? How to decompose a task into subtasks to auction to other agents? And what happens if agents discover that their assumptions about the initial state of their own actions turn out to be invalid? These questions will be addressed in this paper, resulting in a multiagent plan repair system for self-interested agents. More specifically, we let agents schedule tasks on which others depend early in their plans to prevent cyclic dependencies, we let them alternatingly plan for one goal and bid on a task, we give them some high-level information about the capabilities of others, and we allow agents to come back on their commitment of accepting a task.

Such a system can be used, for example, to plan for logistics problems. In many logistic domains, some of the transport orders are only known in the nick of time, and, as we all know, traffic is often unpredictable. Consequently, plans need to be revised all the time. Furthermore, often a significant cost reduction can arise when transportation companies coordinate their actions well. For example, a company (agent) may assign a subtask to another either because the other can do it much more efficiently (because it is already in the neighbourhood), or because the first one cannot perform the task at all. Clearly, these companies are also each other's competitor, so they simply refuse to exchange more information than just a request for or an accept of a subtask.

In the next sections, we describe the problem dealt with in this paper, and we show how a propositional plan repair method can be combined with a simple auction to solve it. We also present solutions to subproblems such as the prevention of deadlock. We illustrate our method using a logistic planning domain with one company that can transport packages by air between airports, and several other companies that can only transport packages within cities. Each of these companies has several transportation orders involving other cities, and thus other companies.

2 The multiagent plan repair problem

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 propositional planning, see e.g. [9]. 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 [9].

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 need to introduce an additional agent to coordinate the use of such a resource. This agent then can provide this resource on request as a service to other agents.

We model the *plan repair* problem by assuming that at some point during or after planning for each agent a its problem Π_a is replaced by a slightly different problem Π'_a. This new problem includes information about added or removed propositions in the initial state I'_a, a changed set of actions O'_a, and/or a changed set of goals G'_a.

Note that in this first approach we used a propositional plan repair method without a 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 currently 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.

These assumptions help us to focus on the more interesting and more difficult problem of designing a system that *(i)* only communicates offers and bids, but little else, *(ii)* can dynamically add, delete, or modify goals, actions, and initial propositions, update its plan accordingly, and meanwhile *(iii)* can auction (sub)tasks to other agents, while preventing cyclic dependencies. 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.

3 Multiagent plan repair

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 a plan repair (PR) system [10]. We require 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 adding actions 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.

Goal-by-goal Planning Our first important contribution is the idea 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 gives us an indication 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 when we would interrupt a regular planner at certain points. There are two disadvantages, however: firstly, 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. Secondly, this goal-by-goal plan repair usually takes more time than constructing a one-shot plan for all goals. Especially when goals strongly interact, the plan repair method we use may eventually reconstruct the whole plan for all goals if necessary.

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. To plan a single goal g from Q the system performs the following steps: *(i)* It queries the planning heuristic $\mathcal{U}$ of the plan repair system to estimate the cost of adding g to the plan, and *(ii)* the heuristic may report that it cannot incorporate the goal, or that the costs of incorporating are so large that it prefers asking other agents for help. If this is the case, the agent tries to auction this goal. Otherwise, it removes it g from Q, adds it as a goal to Π_{PR}, and updates its plan for this new planning problem.

These steps are interleaved with responding to auctions (if any), as discussed later in this section. 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. From this point on, the agent continues to respond to auctions until all other agents are finished as well.

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 turns out that the goal is unattainable, the agent will put the goal up for auction. Agents may also put subgoals up for auction, when they create a plan based on some abstract knowledge of the capabilities of other agents. We model this abstract knowledge by *external actions* in the domain of the agent. External actions are specified like regular actions and describe what (a group of) other agents can achieve. If they are included in a plan, the agent knows that it has to find other agents to carry out the task for him. The effects of the included external actions are sent to the blackboard for auction (as described below).

In the case of auctioned subgoals there is an additional issue we have to take care of: 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. 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 [4], or agent communicate to form a so-called partial global plan [5]. In our goal-by-goal approach, however, we can use the following property. 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.

Plan repair Plan repair is accomplished as follows. After every planning iteration, we observe the world to detect any differences. If the world has changed we record this in our plan by adding a special (virtual) action that reflects the change. Thus, for a changed initial condition, an action is added that transforms the new initial condition into the original initial condition that we expected. These actions are called *gap* actions, for they record the gaps that are present in the current plan.

To repair a failure, we can now remove a gap action from the plan, and hand the resulting plan over to the plan repair system. This detects that the plan has preconditions that are not met and repairs it. As a result of the repair, external actions may be removed from or added to the plan. In the latter case, they are auctioned as if they were the result of a planning step. In the former case, we notify the agent that satisfies the (now unnecessary) subgoal for us, so that it can remove the actions it had planned in order to achieve the subgoal. These actions may include external actions as well. These external actions are now no longer required, in which case the agents that have planned for these tasks are notified also. In this way, a whole chain of dependencies can be removed if necessary.

Auctions We use a blackboard to keep track of a list of auctions, and process them one-by-one. This prevents additional difficulties that agents face when dealing with multiple simultaneous auctions (such as the "eager bidder" problem [11]). For each auction, the blackboard sends out the request for bids.

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 [12].[1] Upon being awarded a goal g, the agent adds g to the front of its goal queue. This ensures that this goal is processed immediately, and that the agent can actually attain g by repairing its plan. It also ensures that at any time there is at most one goal for another agent in the goal queue. 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. Sometimes, however, the heuristic is wrong, and the bidder discovers that it cannot actually satisfy the goal it has bid on. In this situation, the blackboard is notified which re-auctions the goal, disregarding the bids of agents that have bid on the goal and rejected it before. For now, a decommitting agent pays (as a penalty) the cost difference between its own bid and the next one. For the other type of decommitment, where an agent does not require an 'external action' by another agent any more (as described in a previous section), we currently do not issue penalties.[2]

The complete planning loop Having described the features of the algorithm in isolation, we now end this section with the complete algorithm as we use in our experiments. The algorithm (see below) starts with setting up the 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

[1] 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.

[2] We are considering leveled-commitment contracting [13] to enable strategic decommitting.

$\mathcal{U}(P, \Pi_{PR}, g)$ of establishing g with P. If this is estimated to cost more than acceptable (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 cost for it (using the heuristic $\mathcal{U}$ of the PR 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.

```
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 $c$ 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 new external actions then
                         request results (subgoals) of these actions via an auction (blackboard)
  5. elseif $P$ contains gap actions then
       5.1. Update plan: $P = PR(P, \Pi_{PR})$
       5.2. Notify the blackboard of any subgoals that are no longer required
       5.3. if $P$ contains new external actions then
              request results (subgoals) of these actions via an auction (blackboard)

  6. if the environment has changed then
       6.1. Update $P$ with gap actions reflecting the changes
       6.2. Update $\Pi_{PR}$ to reflect the new situation
  7. if an auction is ongoing for a goal $g'$ then
       7.1. send bid (which is $\mathcal{U}(P, \Pi_{PR}, g')$)
       7.2. if goal is awarded then
              push $g'$ onto the front of $Q$
  8. if not all agents have completed executing their plan then
       8.1. goto step 4
end
```

4 Experimental results

As a proof-of-concept, we tested our system with multiagent versions of the logistics problems of the plan repair benchmark problems [14]. This set consists of variations on the same base problem, and the goal is to repair the plan for the base problem in order to make it a valid plan for the variation. The variations consist of additional goals, changed goals, and changed initial situations (including the removal of resources such as airplanes). These problems were converted into a multiagent problem by creating an agent for each of the cities, capable of making deliveries within that city, and an additional agent that handles the inter-city transport using airplanes.

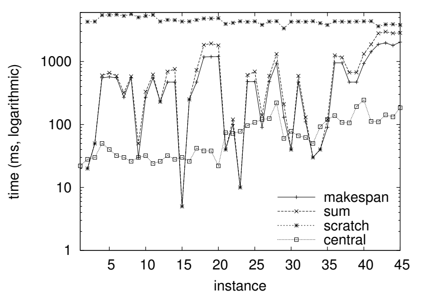
To evaluate the run-time performance of our system, we used the benchmark set to obtain three different figures. The first is the *makespan* of our repair algorithm, i.e. the amount of time it takes the slowest of the agents to adjust its plan to the new situation. The second is the *sum* of the CPU time consumed by the agents. The third figure that we computed, is the makespan of *planning from scratch* with our system, i.e. the time it takes to distributedly compute a plan. These three figures are compared with the performance of the (central) single-agent plan repair method we have adopted [10]. Note that we report CPU times, implying that delay due to communications is not measured (waiting for incoming messages takes virtually no CPU time). However, it is reasonable to assume that for most realistic problems the CPU time involved in finding a solution is much higher than the communication delays.

The runtimes that were obtained on the multiagent variant of the Logistics benchmark set, can be seen in Figure 1. As one can observe, plan repair of a multiagent plan is quite a bit faster than planning from scratch. This can be expected because fewer goals have to be planned and fewer (sub)goals have to be auctioned: on average about 17 goals are auctioned when planning from scratch, compared to about 2 when performing plan repair. Furthermore, we can observe that the single-agent system is faster than the distributed system, although for a number of problems the difference between the makespan and the single-agent system is negligible. This increase can be attributed for a great deal to the goal-by-goal and failure-by-failure approach that we take.

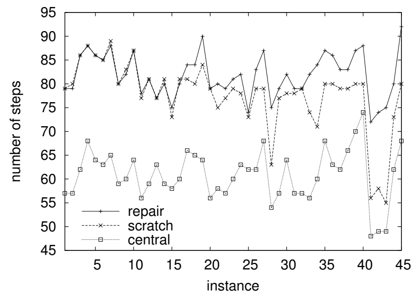
Besides run-time performance, plan quality is also important. The heuristic of planning actions for others early in the plan has a negative effect on the size of the plans, compared with a centralized solution. This is because it forces an ordering on the agents' plans that is stricter than necessary. As a result, the plans that we obtain are often bigger. On average, our plans are 20% larger than plans computed by a single-agent planner (which consist of about 60-70 actions). This is the price we have to pay for not exchanging detailed information on the structure of the plans. The difference in plan size between plans distributedly produced from scratch and plans that have been repaired is significant.

The table below summarizes the results and shows the significance values obtained from a pairwise t-test.

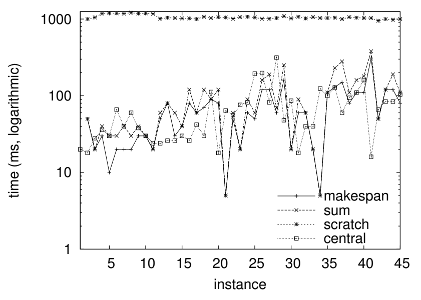
set	issue	Multiagent plan repair	distributed, from scratch	t	p
A	performance (ms)	$\mu = 581.4, \sigma = 564.9$	$\mu = 4378.2, \sigma = 624.5$	-26.0	< 0.01
	quality (steps)	$\mu = 81.8, \sigma = 4.6$	$\mu = 77.7, \sigma = 7.4$	5.3	< 0.01
B	performance (ms)	$\mu = 68.2, \sigma = 55.0$	$\mu = 1056.9, \sigma = 67.0$	-65.3	< 0.01
	quality (steps)	$\mu = 59.6, \sigma = 4.8$	$\mu = 57.6, \sigma = 4.5$	3.0	< 0.01
C	performance (ms)	$\mu = 126.8, \sigma = 99.8$	$\mu = 1101.6, \sigma = 76.8$	-42.5	< 0.01
	quality (steps)	$\mu = 77.5, \sigma = 3.6$	$\mu = 74.2, \sigma = 4.4$	5.4	< 0.01



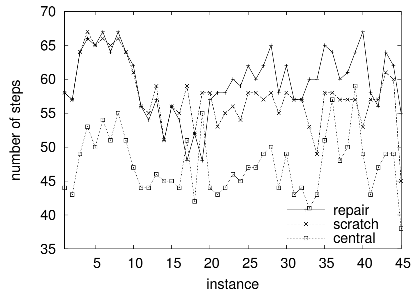
a. Performance problem set A



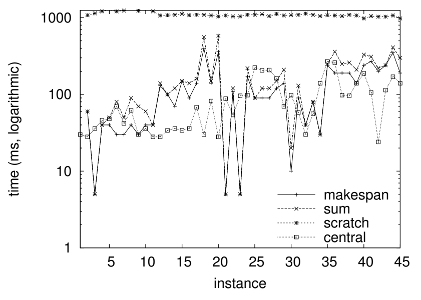
b. Plan quality problem set A



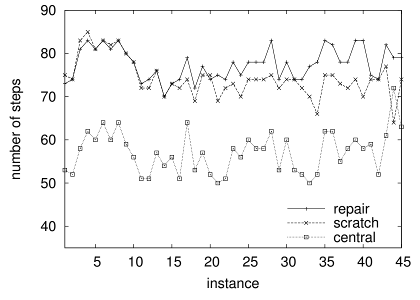
c. Performance problem set B



d. Plan quality problem set B



e. Performance problem set C



f. Plan quality problem set C

Fig. 1. Experimental results

5 Discussion

In this paper we gave experimental evidence that, in a simplified setup, self-interested agents that have a distinct set of resources can coordinate and repair their plans while only exchanging a very small amount of information. We used an existing plan repair algorithm in a goal-by-goal setting and a simple auction, and we showed how to prevent cyclic inter-agent dependencies, and how to deal with lazy agents and decommitment.

We studied the difference in both plan size and planning time between multiagent plan repair and multiagent replanning. It turns out that although in some occasions completely replanning leads to slightly shorter plans, it usually takes much longer to reach those. Another advantage of plan repair is that inter-agent dependencies do not change that much compared to replanning from scratch, so there is much less costs for decommitting, which is especially important in real-life applications. Our distributed approach produces bigger 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. We believe that our results are also applicable to systems with more advanced market mechanisms and strategies (such as discussed below), as plan repair is just as useful in those situations. Note, however, that this system is not yet a fully continual planning system. Studying the interaction between plan repair and plan execution in a multiagent system is still a topic for future research.

This system for coordinating self-interested agents using propositional plan repair is unique in that we integrate planning and coordination without assuming that the agents are *collaborating*. Agents may even be each 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 [15] cooperative agents are coordinated by exchanging more and more details of their hierarchical plans until conflicts can be resolved (similar to [6]). The General Partial Global Planning (GPGP) method [5] describes a framework for distributedly constructing a (partial) global plan to be able to discover all kinds of potential conflicts. Finally, in [16] 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. His 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.

There is, however, a substantial body of work on *task allocation* for self-interested agents. For example using market mechanisms [17], or using extensions of the contract-net protocol [18, 19]. The main difference between the work in this area and ours, is that we do not consider a set of tasks given beforehand. Instead, the set of tasks that has to be allocated arises from the results of the planning activities of the agents. In our view, task (re)allocation, cannot be disconnected from planning. Nevertheless, ideas from the work in this area may be used to improve upon our approach. For example, we might replace the simple auction by a parallel or combinatorial auction. In particular, the results presented by Hunsberger [8] are interesting. He describes how a group of autonomous agents that encounter an opportunity to collaborate on some group activity, can decide whether to commit to doing that activity, using a combinatorial auction. Although in his approach a single given set of tasks is to be performed, it might be extended to a more dynamic situation, such as we consider. Also, the work on *role (re)allocation* and *team formations* (e.g. [20]) may lead to improvements in performance and solution quality, by recognizing the different roles involved. Finally, in temporal domains, the control of the temporal network may be distributed using techniques presented in [21].

Besides looking at improvements of our goal-by-goal heuristic, we would like to test our method in other domains and study the conditions that define when this approach is feasible. Furthermore, we intend to try to relax our assumptions to be able to tackle more advanced problems. Most importantly, we would like to have a method to estimate the costs of external actions. Typically those 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. Another important topic for future study is using a different type of auction and (de)committing mechanism [22, 13] to allow backtracking over different agents. We intend to study the applicability of such a technique in relation to the specific requirements of the problem domain of these self-interested planning agents.

References

1. DesJardins, M., Durfee, E., Ortiz, C., Wolverton, M.: A survey of research in distributed, continual planning. AI Magazine **20** (2000) 13–22
2. Pollack, M., Horty, J.: There's more to life than making plans: Plan management in dynamic, multi-agent environments. AI Magazine **20** (1999) 71–84
3. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. ACM Computing Surveys **21** (1994) 87–119
4. Wilkins, D., Myers, K.: A multiagent planning architecture. In: Proc. of the 4th Int. Conf. on AI Planning Systems. (1998) 154–162
5. Decker, K., Li, J.: Coordinating mutually exclusive resources using GPGP. Autonomous Agents and Multi-Agent Systems **3** (2000) 113–157
6. von Martial, F.: Coordinating Plans of Autonomous Agents. Volume 610 of Lecture Notes on AI. (1992)
7. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial Intelligence **101** (1998) 165–200
8. Hunsberger, L., Grosz, B.J.: A combinatorial auction for collaborative planning. In: Proc. Int. Conf. on Multi-Agent Systems. (2000) 151–158
9. Kambhampati, S.: Refinement planning as a unifying framework for plan synthesis. AI Magazine **18** (1997) 67–97
10. van der Krogt, R., de Weerdt, M.: Plan repair as an extension of planning. In: Proc. of the Int. Conf. on Automated Planning and Scheduling. (2005)
11. Schillo, M., Kray, C., Fischer, K.: The eager bidder problem: A fundamental problem of DAI and selected solutions. In: Proc. of the 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems. (2002) 599–606
12. Vickrey, W.: Computer speculation, auctions, and competitive sealed tenders. Journal of Finance **16** (1961) 8–37
13. Sandholm, T., Lesser, V.: Leveled-commitment contracting: a backtracking instrument for multiagent systems. AI Magazine **23** (2002) 89–100

14. Gerevini, A., Serina, I.: Fast plan adaptation through planning graphs: Local and systematic search techniques. In: Proc. of the Fifth Int. Conf. on AI Planning Systems. (2000) 112–121
15. Kleinmann, K., Lazarus, R., Tomlinson, R.: An infrastructure for adaptive control of multi-agent systems. In: IEEE Int. Conf. on Integration of Knowledge Intensive Multi-Agent Systems. (2003) 230–236
16. Brenner, M.: Multiagent planning with partially ordered temporal plans. Technical Report 190, Universität Freiburg (2003)
17. Walsh, W., Wellman: A market protocol for decentralized task allocation and scheduling with hierarchical dependencies. In: Proc. of the 3rd Int. Conf. on Multi-Agent Systems. (1999) 325–332
18. Collins, J., Tsvetovatyy, M., Gini, M., Mobasher, B.: MAGNET: A multi-agent contracting system for plan execution. In: Proc. of SIGMAN. (1998)
19. Smith, R.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers **C-29** (1980) 1104–1113
20. Nair, R., Tambe, M., Marsella, S.: Role allocation and reallocation in multiagent teams: towards a practical analysis. In: Proc. of the 2nd Int. Joint Conf. on Autonomous agents and multiagent systems. (2003) 552–559
21. Hunsberger, L.: Distributing the control of a temporal network among multiple agents. In: Proc. of the 2nd Int. Conf. on Autonomous Agents and Multi-Agent Systems. (2003) 899–906
22. Hoen, P.J., t., Poutré, J.A., L.: A decommitment strategy in a competitive multi-agent transportation setting. In: Proc. of the AAMAS-03 Workshop on Agent Mediated Electronic Commerce V. Volume 3048 of Lecture Notes on Artificial Intelligence. (2003)