Efficient Auction Based Coordination for Distributed Multi-Agent Planning in Temporal Domains Using Resource Abstraction

Andreas Hertle and Bernhard Nebel *

University of Freiburg, Department of Computer Science, 79110 Freiburg, Germany.

Abstract. Recent advances in mobile robotics and AI promise to revolutionize industrial production. As autonomous robots are able to solve more complex tasks, the difficulty of integrating various robot skills and coordinating groups of robots increases dramatically. Domain independent planning promises a possible solution. For single robot systems a number of successful demonstrations can be found in scientific literature. However our experiences at the RoboCup Logistics League in 2017 highlighted a severe lack in plan quality when coordinating multiple robots. In this work we demonstrate how out of the box temporal planning systems can be employed to increase plan quality for temporal multi-robot tasks. An abstract plan is generated first and sub-tasks in the plan are auctioned off to robots, which in turn employ planning to solve these tasks and compute bids. We evaluate our approach on two planning domains and find significant improvements in solution coverage and plan quality.

1 Introduction

Recent advances in robotics and AI promise to revolutionize industrial production. Gone will be static assembly lines and hardwired robots. Instead autonomous mobile robots will transport parts for assembly to the right workstation at the right time to assemble an individualized product for a specific customer. At least that is the dream of various manufacturing companies around the globe. To ensure that production runs without interruptions around the clock, these robots will need strong planning capabilities. The challenges for such a planning system stem from making plans with concurrent processes and multiple agents, deadlines and external events.

The Planning and Execution Competition for Logistics Robots in Simulation (PExC) [6] addresses these problems and provide a test-bed for for experimenting

* This work was supported by the PACMAN project within the HYBRIS research group (NE 623/13-1). This work was also supported by the DFG grant EXC1086 BrainLinks-BrainTools to the University of Freiburg, Germany.

with different methods for solving these problems, abstracting away from real robots. It is a simulation environment based on the RoboCup Logistics League (see Fig. 1).

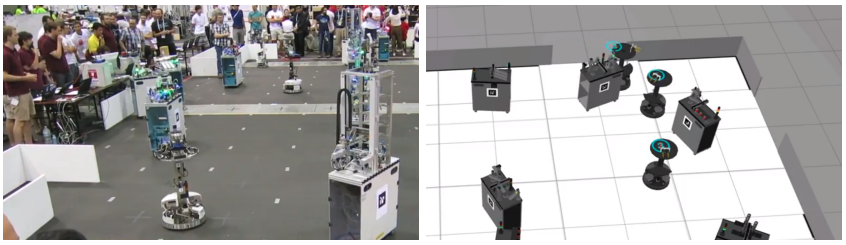
Our aim was to demonstrate that current planner technology is mature enough to be used in such an environment. As it turned, however, this is far from the the truth. We employed the temporal planners POPF [1] and TFD [2], which seem like a good fit for these kinds of planning tasks, as time and duration of processes are modeled explicitly. It turned out that it is not possible to use them in reliable way. While they both can plan for one robot, two or more robots are beyond the reach. If one requires optimality in makespan, then the planners took too long, meaning using up much of the time reserved for planning and execution. If one chooses to use greedy plan generation, then the plans result often in assigning most of the work to just one robot.

In this paper we show how planning in temporal domains with multiple agents can be improved to find plans with lower makespan and find solutions for bigger problems. The key is to abstract resources, in this case robots, away, and plan for the simplified instance. After that the plan is refined using a contract-net protocol approach for the planning agents.

The rest of the paper is structured as follows: After giving some background information in section 2, we present our approach in section 3. The experimental evaluation can be found in section 4. Section 5 discusses related work.

2 Temporal PDDL

The planning domain definition language (PDDL) was developed as an attempt to standardize Artificial Intelligence Planning. Since its inception in 1998 more features were added to represent planning tasks with numerical functions, non-deterministic outcomes and temporal actions. Due to international planning competitions a number of well tested planning systems are available. We are



Finals 2017 (1) Simcomp 2017 (2)

Fig. 1. In the RoboCup Logistics League competition three autonomous robots must coordiante efficiently to solve production tasks. On the left (1): finals of the RCLL competition in 2017 between teams Carologistics and GRIPS. On the right (2): planning track of the simulation competition.

interested in finding plans for multiple physical robots or systems. Any number of processes could be happening simultaneous and considering various duration during the planning process is crucial to finding good plans. For this reason we require a planning system capable of temporal planning as defined in PDDL 2.1.

In PDDL a planning task is defined by a domain and a problem file. The domain defines what types, predicates and actions are possible and how they interact. The actions in a domain describe how the state can transition during planning. Each actions has typed arguments that specify which objects are relevant for this action. For temporal planning actions have a start event and an end event separated by the duration of the action. The conditions of an action determine when an action is *applicable* and the effects how the state changes when the action is *applied*. Conditions can refer either to the start, the end or the open interval between them. Effects take place either at the start or the end of an action.

The problem specifies the current situation and the goal condition. The current situation is specified as a set of objects and initial values for relations between them. For temporal planning future events can be specified as *timed initial literals*. These events encode a value change for a predicate or function to happen at a specific time in the future. Our approach makes extensive use of timed initial literals as way to integrate actions from previous plans into the planning process.

Solutions to temporal planning tasks are temporal plans consisting of a list of actions, where each action starts at a certain timestamp and has a specific duration.

3 Task Auction Planning

Our goals are twofold: we want to reduce complexity during the planning process, thus increasing the chance to find a valid plan, and we want to minimize makespans of plans by achieving a better plan parallelization when planning for multiple agents. Our approach decomposes a planning task for multiple agents into multiple simpler planning tasks for each agent. First we solve an abstract planning problem by removing agents form the planning problem and hiding some complex interactions in the planning domain. Once an abstract plan is found, a central agent acts as auctioneer in order to distribute tasks between the other agents, where a task is derived from an action in the abstract plan. Each agent can compute plans for offered tasks and submit bids based on the time it takes this agent to achieve the task goal. The auctioneer chooses from the valid plans for each task and continues to offer the next set of tasks until all tasks have valid plans from one agent.

Another way to look at this is to consider the resources used by the agents. The abstract plan coordinates shared resources between the agents. Each agent in turn uses its own resources to achieve a single step of the abstract plan, while unaware of the other agents and their resources. Our approach is applicable in planning domains that do not require concurrency to be solved. Usually prob-

lems in such domains could be solved by a single agent without help. However efficiency can be greatly increased when multiple agents participate.

3.1 PDDL Domain Creation

In this section we show how to convert an existing temporal PDDL domain to a task- and an agent domain. To ensure compatibility between the domains, we make no changes to types, predicates or functions, but focus solely on the actions. We expect the temporal domain to be modeled in the following way: A certain type represents the agents, the *agent-type*. Some actions in the domain are modeled to represent activities performed by the agents, we call them the *agent-actions*. They can be recognized by having an *agent-type* as parameter. Other actions represent processes in the environment and are not specific to an agent, we call them the *environment-actions*. Those do not have *agent-type* objects as parameter.

First we discuss how to construct the *task-domain*. The intend is to identify typical sequences of actions that are performed by the same single agent. That chain of actions could to be replaced with a single macro action. This reduces the branching factor during planning. A macro can be created by gathering the effects of each action and either add it to the start or the end effect of the macro action. Some effects might cancel each other; it is up to the domain designer to determine which effects are essential for the macro action. The same careful consideration is necessary to select which action conditions to add to macro action. In the final step the agent is removed from the macro, meaning the parameter of *agent-type* and all predicates or functions in the conditions and effects of the macro that refer to the agent. Once a macro action for each task is created it is also necessary to add the *environment-actions* from the temporal domain to ensure that the domain is complete.

Next we discuss the purpose of the *agent-domain*. The agents are supposed to solve each offered task. However they must not interfere with other unrelated tasks. For this reason it is helpful to remove all *environment-actions* from the agent domain. Thus, the agent domain is intentionally incomplete: It is not possible to solve the whole problem with the agent domain. However it contains all actions necessary to allow an agent to solve each offered task.

3.2 Combining and Rescheduling Plans

In our approach we combine and reschedule plans. In a *valid* temporal plan each action is *applicable* at its start time. When looking through effects of previous actions in the plan we can determine which events made the action applicable. The action then can be moved to the time of the latest of the events it *depends* on. If an action does not *depend* on any earlier event it can be moved to the beginning of the plan. When appending actions from another plan, we insert the action at the end of the plan (after the last event) and verify applicability. Then the action can then be rescheduled to the earliest time as described above.

Algorithm 1 Agent: state update and bidding

```
1:  state ← state_init, Events ← ∅, plan_agent ← ∅, Proposals ← ∅
2:  while ⊤ do
3:      Assignments, Events_new, Tasks ← receive()        ▷ Receive from auctioneer
4:      a ← find_assigned_to_agent(Assignments)
5:      state, plan_agent ← apply(Plans[a.task])          ▷ Retrieve and apply plan
6:      Events ← Events ∪ Events_new
7:      for all t ∈ Tasks do
8:          plan ← make_plan(state, Events, t)            ▷ Call PDDL planner
9:          if plan solves t then
10:             Plans[t] ← plan                           ▷ Store plan
11:             make_bid_and_send(plan)                   ▷ Send plan to auctioneer
12:         end if
13:     end for
14: end while
```

3.3 Solving and Bidding for Sub-tasks

In this section we discuss the planning process from an agent's point of view. When an agent receives a task offer it needs to find a plan for the task. Once a plan is found the agent determines the point in time when it could start working on the task and when the task will be finished and submits the plan and those two timestamps as a bid for the task. Then the agent may continue computing solutions for alternative tasks and await the reply from the auctioneer. Algorithm 1 shows a simplified overview of the bidding process. In the actual implementation the communication takes place asynchronously and interrupts the planning process if the situation has changed.

Initially, the agent's current state could be supplied via PDDL file. During the planning process the current state can change from two sources. Once an agent won a bid for a task the current state is updated with the agent's actions by applying the plan that was proposed for the task as showed on line 5. Applying a plan also increases the timestamp of the current state by the makespan of the plan. The other source of changes comes from external events during the planning process, i.e. when other agents interact with the environment as showed on line 6. These external events do not advance the time of the current state. Instead, external events are represented in as timed initial literals, that will happen at a certain time in the future of the current state.

A task is communicated to the agent in the form of a PDDL action definition from the task-domain. The goal for a task can be derived form the effects of the action; this happens in the **make_plan** function on line 8. This is possible because both the tasks- and the agent-domain allow for the same predicates and functions. Thus the effects of the task-action applied to the current state of the agent define the goal for the task. However most planning systems are unable plans for negated goal predicates, so negated effects have to be omitted from the goal conjunction. If necessary complimentary predicates can be added to the PDDL domains such that goals for each possible task are sufficiently specified.

Now that the goal and the current state is known, a temporal planner can search for a solution. If no plan is found the agent is unable to solve this task. If a plan is available the agent can make a bid for the task. The bid consist of the plan and two timestamps. The former indicates when the agent will be able to start working on the task and the latter when the agent will presumably finish the task. The timestamps are useful for the auctioneer to determine which agent to assign a task to.

At some point the auctioneer publishes the next announcement consisting of which agent was assigned which task, what events are going to happen as a consequence and a new set of tasks to solve as showed on line 3. If a task was awarded to the agent, the agent applies the corresponding plan to the current state. The auctioneer also includes a list of future events in the announcement. These events represent the changes to the environment, possibly from actions of other agents. Each event consist of a timestamp and a set of effects. In case their timestamp is earlier than the time of the current state, the events need to be applied in the correct order. Later events are added as timed initial literals to the current state. Once the current state is updated the agent is ready to search for solutions to newly available tasks.

3.4 Decomposing and Auctioning of Sub-tasks

The auctioneer works with two plans: an abstract plan and a combined plan. The abstract plan determines which tasks can be offered to the agents. Once the agents submit bids for some tasks, the auctioneer can chose which bids provide the best value. These plans submitted by the agents are then integrated into the combined plan. This ensures that plans submitted by the agents are free of conflicts. The agents are notified of their assigned tasks. Then the process continues with a search for a new abstract plan. In the end the resulting combined plan is a solution to the original planning problem. Algorithm 2 shows a simplified overview of the process. In the actual implementation the communication takes place asynchronously.

The initial problem could be supplied via PDDL file and with the task-domain a temporal planner can search for the abstract plan as showed in line 3. Once a plan is found, the auctioneer determines which actions in the plan can be offered as tasks to the other agents. As discussed in section 3.1, some actions in the plan are intended as tasks for agents to solve while other model aspects of the environment. The temporal plan needs to be analyzed (line 7) to determine which action depends on previous actions in the plan as discussed in 3.2. The following rules determine which actions can be offered:

1. A task-action without dependencies can be offered to the agents.
2. An environment-action without dependencies on other actions is *executable*.
3. An environment-action where all dependencies are executable is also executable.
4. A task-action where all dependencies are executable can be offered.

Algorithm 2 Auctioneer: abstract plan and offering sub-tasks

1: $state \leftarrow state_{init}, Events \leftarrow \varnothing, Proposed \leftarrow \varnothing, plan_{comb} \leftarrow \varnothing$
2: **while** $\top$ **do**
3: $plan_{abs} \leftarrow$ **make_abstract_plan**$(state, Events)$ ▷ Call PDDL planner
4: **if** $|plan_{abs}| = 0$ **then**
5: **return** $plan_{comb}$
6: **end if**
7: $Actions_{env}, Tasks \leftarrow$ **determine_executable_prefix**$(plan_{abs})$
8: $state, plan_{comb} \leftarrow$ **apply**$(Actions_{env})$
9: $Events \leftarrow$ **extract_events**$(plan_{comb})$
10: **offer_tasks_and_wait**$(Assignments, Events, Tasks)$ ▷ Send to agents
11: $Proposals \leftarrow$ **receive**() ▷ Receive from agents
12: $Assignments \leftarrow$ **assign**$(Proposals)$
13: **for all** $a \in Assignments$ **do**
14: $state, plan_{comb} \leftarrow$ **apply**$(a.plan)$
15: **end for**
16: **end while**

All *executable* environment-actions form the abstract plan are appended to the combined plan as showed on line 8.

In order to solve tasks, the agents need to know what events are scheduled to happen. However they do not need to know the details of the other agents actions, only the changes they impose on the environment. These events are derived from the effects in the combined plan by removing all agent-specific predicates and functions (line 9).

Once a set of tasks has been offered the auctioneer waits for bids from the agents as showed on line 10. A bid from an agent consist of the plan for the task and the timestamps when the agent will be able to begin and achieve the tasks. An agent can bid on any number of tasks simultaneously. However the agent can only execute one task at a time, thus bidding on multiple tasks provides alternatives for the auctioneer to choose from.

Our approach does not specify or expect a certain number of agents. This offers great flexibility, as agents can join the planning process at any time or leave it provided they completed all tasks they committed to. However when waiting for solutions from agents it is difficult for the auctioneer to determine how long to wait for alternatives. Besides naive greedy strategies we implemented two alternatives:

- *Just-in-time assignment*: The decision is delayed until one the bidding agents needs to start working for this task as indicated by the *starting* timestamp of the bid.
- *Delayed batch assignment*: If there are a lot of simultaneous tasks available, it might take too long to wait for solutions for every task before assigning the winning agents. Once at least one solution is received the auctioneer delays the decision by a fixed duration and then performs a batch assignment.

In the literature the Hungarian method is recommended for optimal assignment of tasks to agents. However, since we do not have a matching problem between robots and tasks, robots can take on more than one task, the method does not work here.

We expect the *Just-in-time assignment* to perform best on physical systems. With this strategy the agents have the maximum amount of time to investigate possible alternative solutions without waiting or delaying the execution of the plan. Also, the agents would be more flexible since they do not commit to certain tasks ahead of time. For benchmark purposes this is impractical however, since the planning process would be prolonged roughly by the makespan of the plan and the planning timeout for our benchmarks is by far lower than the makespans. Thus for the benchmarks in this paper we make assignments based on the *Delayed batch* strategy.

Once an assignment is chosen, the auctioneer integrates the plans submitted by the agents into a combined plan as showed on line 14. Then the auctioneer computes a new abstract plan and continues to offer tasks to agents until an empty abstract plan is found, which signifies that the goal has been achieved.

4 Experimental Evaluation

We evaluated our approach on numerous planning tasks from two domains. Three planner configurations were used for the evaluation:

1. *POPF* is a forwards-chaining temporal planner [1]. Its name is based on the fact that it incorporates ideas from partial-order planning. During search, when applying an action to a state, it seeks to introduce only the ordering constraints needed to resolve threats, rather than insisting the new action occurs after all of those already in the plan. Its implementation is built on that for the planner COLIN, and it retains the ability to handle domains with linear continuous numeric effects.
2. *Temporal Fast Downward* is a temporal planning system that successfully participated in the temporal satisficing track of the 6th International Planning Competition 2008. The algorithms used in TFD are described in the ICAPS 2009 paper [2]. TFD is based on the Fast Downward planning system [3] and uses an adaptation of the context-enhanced additive heuristic to guide the search in the temporal state space induced by the given planning problem.
3. *Temporal Fast Downward Sequential Reschedule*. In this configuration the TFD-SR will search for purely sequential plans without taking advantage of concurrent actions. Once a plan is found it will be rescheduled to take advantage of concurrency. This usually increases planning efficiency allowing to solve bigger planning tasks.

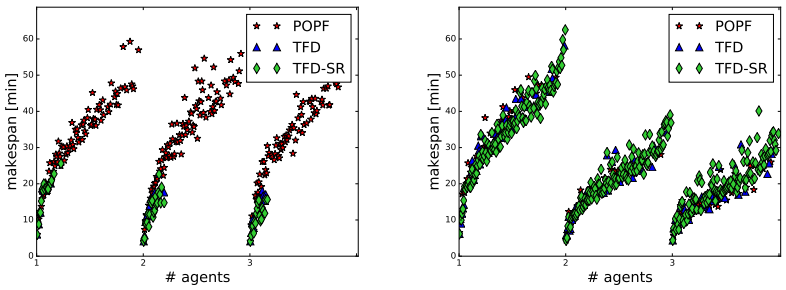
We run each temporal planner configuration as a base line. For our auction based approach we also run all three planner configurations for the auctioneer. For the agents we found that POPF greatly outperformed TFD. The cause for

this is likely a costly analysis before the search for a plan starts, where the analysis time is significantly greater than the following search time. For the agents that have to search for many short plans this is highly disadvantageous. Thus for all experiments the agents were planning with POPF. Finally, each plan is validated with *VAL* [4] to verify correctness.

The benchmarks were run on one machine with a Intel Core i7-3930K CPU at 3.2 GHz and 24 GB of memory. The baseline planning configurations run on a single thread, while the auction planning configurations use one thread per agent and one thread for the auctioneer. Each planning instance has a time limit of 15 minutes. In the results we compare expected execution time, that is makespan of the plan plus time until the first action is known. For the baseline that means total planning time and for our approach that means time until the first round of assignments is announced.

4.1 RoboCup Logistics League Domain

This domain was created for the participation in the planning track of RoboCup Logistics League competition. In the competition, three robots are tasked to assemble a number of products in an automated factory. A product consist of a base, zero to three rings and a cap. Each piece of the product has a certain color and the order of rings does matter. There are six specialized stations each capable of performing a certain step in the assembly. Some assembly steps require additional material that has to be brought to the station before the step can be performed. The robots can transport the workpieces between stations. The exact makeup of the ordered products are not known in advance, instead they are comunicated during the production. The decision which products to assemble before the deadline and coordinating the three robots most efficiently is key for performing well in the competition.



Baseline (1) Auction (2)

Fig. 2. Benchmark results in the RCLL domain. The problem set is evaluated with one, two and three agents. The lower the makespan, the better the plan result. On the left the baseline is shown. On the right the auction based task assignment is shown.

Table 1. Number of solved instance out of 125 for the RCLL domain with 1–3 agents

	Baseline			Auction		
# agents	1	2	3	1	2	3
POPF	85	90	78	52	40	50
TFD	11	20	12	58	44	47
TFD-SR	17	23	19	123	120	114

In this domain we have modeled most aspects of the competition. However for this benchmark the products to assemble are known at the start and there are no deadlines for finishing them. The agents can perform the following actions: *move* from one station to another, *pickup* a product from a station, *prepare* a station to receive a product and *insert* a product into a station. For the tasks-domain we replaced the agent actions with a number of *task-transport-product* actions; one for each station type. Usually the agents find plans in the form *move*, *pickup*, *move*, *prepare*, *insert* when solving a transport task.

We generated 125 problem instances with five products of varying complexity, the simplest requireing 4 and the most complex 10 production steps. Each problem is solved by one, two and three agents. The results can be seen in table 1 and figure 2.

The baseline results show that both TFD variants can only solve few problems with low complexity. POPF can solve half of the problems, however the makespan for plans for two and three agents are as high as for one agent. Thus POPF is not able take advantage of multiple agents. The auction task assignment results show that TFD-SR is able solve most problems. TFD solves significantly more problems compared to the baseline. POPF solves only one third of the problems, less than in the baseline configuration. In many cases POPF is unable to find an initial plan in the task-domain within the timelimit. For all three planners the makespan for plans with two and three agents is significant lower than with one agent, showing better utilization of multiple agents.

4.2 Transport

For the second experiment we employ the well known transpot domain, where a set of packages need to be delivered to individual destinations by a number of trucks. Trucks can move along a network of roads of different lengths. Each truck can load a certain number of packages at the same time ranging from 2 to 4. Each package is initially found at some location and needs to be transported to its destination location.

The agents can perform the following actions: *move* from one location to a neighbouring location in the road graph, *pick-up* a package at a certain location if below maximum capacity and *drop* a transported package at a certain location. For the task-domain we replaced the agent actions with a *task-pickup-package* and a *task-deliver-package* action. This results in simple task plans, where each package is first picked up at its location and then dropped at its destination.

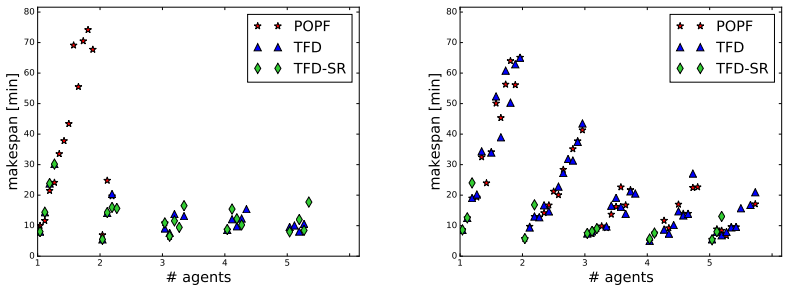
Table 2. Number of solved instances out of 13 for the transport domain for 1–5 agents

	Baseline					Auction				
# agents	1	2	3	4	5	1	2	3	4	5
POPF	12	3	2	1	0	13	12	10	9	7
TFD	4	3	4	5	4	12	11	9	8	9
TFD-SR	4	4	5	4	4	3	2	3	2	3

Usually the agents find plans in the form *move*, *move*, ..., *move*, *pickup* for the pickup tasks. Similar plans are found for the drop-tasks, however only the agent that picked up the package before can solve this task. Usually the planner can easily determine whether a deliver-task can be solved. Furthermore, if an agent tries to solve a pickup task while carrying the maximum number of packages, no valid plan will be found. It is intended that the agent solves a deliver task for one of the packages it carries. However it is difficult for the planner to determine that a pickup task is impossible, usually the planner searches until timeout. Thus for this domain we use the low planning timeout of 1 second for the agents to reduce time wasted on unsolvable tasks.

We generated a road network for two cities with ten locations each. Travel time within a city is low and travel time between cities is considerably higher. We sampled random locations for between 3 and 40 packages in increments of 3 for a total of 13 problem instances. Each problem is solved by between 1 and 5 agents. The results are shown in table 2 and figure 3.

The baseline results show that both TFD variants can only solve problems with few packages. POPF is able to solve all problems with one agent, but is unable to find plans with multiple agents. The auction task assignment results show that TFD-CR can solve only few problems; in most cases no initial task



Baseline (1) Auction (2)

Fig. 3. Benchmark results in the transport domain. The problem set is evaluated with one to five agents. The lower the makespan, the better the plan result. On the left the base line is shown. On the right the auction based task assignment is shown.

plan can be found. Since TFD-CR searches for sequential plans, we assume that the search heuristic is confused by the high amount of simultaneous applicable pickup tasks of equal cost. On the other hand POPF and TFD are able to solve most problems with any number of agents.

5 Related work

The work closest to ours is the work by Niemüller and colleagues, who describe an architecture based on ASP [8]. They do not use a temporal planner but compile the planning problem into ASP and then only plan a few steps ahead. As they can show, this is an effective and efficient way to address the RCLL planning and execution problem.

Our approach instead is based on abstraction techniques, an approach that goes back a long way [7]. The particular kind of abstraction that we used can be called resource abstraction. This has also been employed before to speed up planning and to increase the number of tasks that could be executed in parallel in the RealPlan system [10]. However, in this case, no temporal planning was involved.

Coordination of agents using announcements and bidding is a technique often used in multi-agent systems [9]. In our context with planning agents, it is very similar to the architecture used in the elevator control designed by Koehler and Ottiger [5].

6 Conclusions

We showed how planning in temporal multi-agent domain can be enhanced by abstracting resource away. A central auctioneer offers tasks related to these resources to agents to be solved individually. The agents propose their solutions and the auctioneer chooses which solutions fit together best and assembles them into a combined plan. Our experiments show that compared to baseline temporal planning our approach can solve bigger problems and the resulting plans have significant lower makespan. The next step in the development will be to deploy our approach on physical robots or in simulations, where plan execution and monitoring could pose additional challenges. In addition, we also aim at automating the process of abstracting the resources away and construct the planning instances for them that are solved individually.

References

1. Coles, A.J., Coles, A.I., Fox, M., Long, D.: Forward-chaining partial-order planning. In: Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10) (May 2010)
2. Eyerich, P., Mattmüller, R., Röger, G.: Using the context-enhanced additive heuristic for temporal and numeric planning. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009 (2009)

3. Helmert, M.: The fast downward planning system. J. Artif. Intell. Res. **26**, 191–246 (2006)
4. Howey, R., Long, D., Fox, M.: Validating plans with exogenous events. In: Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group (2004)
5. Koehler, J., Ottiger, D.: An ai-based approach to destination control in elevators. AI Magazine **23**(3), 59–78 (2002)
6. Niemueller, T., Karpas, E., Vaquero, T., Timmons, E.: Planning competition for logistics robots in simulation. In: WS on Planning and Robotics (PlanRob) at Int. Conf. on Automated Planning and Scheduling (ICAPS) (2016)
7. Sacerdoti, E.D.: Planning in a hierarchy of abstraction spaces. Artif. Intell. **5**(2), 115–135 (1974)
8. Schpers, B., Niemueller, T., Lakemeyer, G., Gebser, M., Schaub, T.: Asp-based time-bounded planning for logistics robots. In: Proceedings of the Twentyeightth International Conference on Automated Planning and Scheduling (ICAPS-18) (2018)
9. Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Trans. Computers **29**(12), 1104–1113 (1980)
10. Srivastava, B., Kambhampati, S., Do, M.B.: Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. Artif. Intell. **131**(1-2), 73–134 (2001)