Coordinating Autonomous Planners

Adriaan ter Mors*, Jeroen Valk* and Cees Witteveen*†
* Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands
Email: {a.w.termors, j.m.valk, c.witteveen}@ewi.tudelft.nl
† Center for Mathematics and Computer Science (CWI),
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Email: c.witteveen@cwi.nl

***Abstract*— We present a framework for coordinating autonomous planning agents. Together, these agents have to achieve a set of interdependent (elementary) tasks. Each of the agents receives a unique subset of tasks to achieve, but an agent may be dependent on other agents completing (some of) their tasks first.**

Each of the agents needs to make a plan in order to achieve its set of subtasks. Task dependencies between tasks assigned to different agents, however, prevent them from making plans independently. Therefore, in order to guarantee planning autonomy, we need a pre-planning coordination method. We introduce a coordination method that can be used to guarantee a solution to the joint planning problem whatever individual plans the agents may have constructed independently.

As a byproduct of our research, we show how this method can be applied to (re)use existing planning tools in a multi-agent context, by solving a multi-modal logistic planning problem by coordinating several autonomous vehicle routing planners.

Keywords: coordination, multi-agent systems, autonomous agents, planning, approximation algorithms.

I. Introduction

The problem we want to discuss in this paper applies whenever there is a number of autonomous actors accepting a joint task that creates dependencies between these actors. A simple example is a supply chain where goods have to be transported by several autonomous agents each having their own business policy, planning software and billing systems. Clearly, to manage such dependencies between agents created by the joint task, they need to be coordinated in order to guarantee that a joint plan will be constructed. Coordination, however, could easily result in restricting the *planning autonomy* of the agents, i.e., the freedom of the agents to decide how to perform their tasks. In cases where agents try to maximize their planning autonomy such restrictions might simply be unacceptable. For example, if the agents have competitive relationships, they might be reluctant to share details of their plans with other agents and do not want to be interfered during planning their own activities.

This motivates the following coordination problem: How to guarantee that the joint task is planned well if we lack information about the details of the plans developed by the participating agents?

In this paper we analyze this problem and discuss a coordination method that (*i*) guarantees the planning autonomy of each agent, (*ii*) at the same time ensures the existence of a joint plan respecting each of the individual agent plans, and (*iii*) does not require knowledge on which plans will be developed by each of the agents.

In our setup, we assume that there exists a joint task $\mathcal{T}$ consisting of a partially ordered set T of elementary tasks t allocated to a number of agents. Each agent has received a (non-overlapping) subset of tasks and has to make a plan satisfying the given partial order. Order dependencies might exist not only between tasks given to the same agent, but also between tasks given to different agents, thereby inducing inter-agent dependencies. The approach we take to solve the coordination problem is to take a *pre-planning* approach to coordination: prior to planning, we try to *minimally* constrain the agents' tasks in such a way that agents can subsequently plan their part of the joint task without taking into account the plans of the other agents. The result of this approach is the development of algorithms and protocols that ensure coordination while guaranteeing completely autonomous planning to each of the agents.

This set-up differs from common multi-agent-coordination approaches to coordination in planning. For instance, in Ephrati's research [1], planning and coordination steps are interleaved. In the (G)PGP framework

(e.g. [2], [3]), planning and coordination is an iterative process, with plans of various levels of abstraction being exchanged between agents to achieve efficient co-operation. In both approaches, the agents' planning method must be adapted to allow for coordination of the planning process. In our pre-planning coordination approach, however, the coordination takes place at the task level and occurs independently from the planning process. Since we separate the coordination method from the planning processes, our approach allows existing single-agent planning software to be *reused* in a multi-agent planning context.

The structure of this paper is as follows. In Section II, we present our framework for multi-agent planning and coordination and we identify the coordination problem. In Section III, we briefly discuss the complexity of the coordination problem and show its relation to some problems in graph theory. In Section IV, we present a distributed coordination protocol that allows agents to achieve coordination while remaining autonomous in the planning process. In Section V, we will show how this coordination protocol can be used to solve multi-agent planning problems using (existing) single-agent planning systems, and we will present some results we achieved in solving instances of multi-modal logistic benchmark problems. Section VI concludes the paper by discussing the results and identifying areas for future work.

II. A Task-Oriented Multi-agent Planning Framework

We consider problems that have to be solved by several autonomous agents, each having their own capabilities and using their own (planning) tools. Problems we have in mind consist of a set T of *interrelated elementary tasks*. Such an *elementary task* t, or simply *task*, is a unit of work that can be performed by a single agent. A task t_1 depends on another task t_2 if there exists a *precedence constraint* between them: If a task t_2 is preceded by task t_1, denoted by $t_1 \prec t_2$, the execution of t_2 may not start until t_1 has finished; for example, achieving t_1 results in creating resources needed to perform t_2. Such a set of interdependent tasks is called a *composite task*. A composite task $\mathcal{T} = (T, \prec)$ therefore is a non-empty set of tasks $T = \{t_1, \dots t_m\}$, partially ordered by a set of precedence constraints $\prec \subseteq T \times T$. This composite task must be performed by a set $\mathcal{A} = \{A_1, \dots A_n\}$ of autonomous planning agents.

We assume that the tasks in T have been assigned to the agents in $\mathcal{A}$ by some (surjective) task assignment $f : T \rightarrow \mathcal{A}$ thereby inducing a *partitioning* $\mathbf{T} = \{T_1, \dots, T_n\}$ of T, where $T_i = \{t_j \in T \mid f(t_j) = A_i\}$ denotes the set of tasks allocated to agent A_i. As a result of this task assignment A_i also inherits the precedence constraints that apply to T_i, i.e., the set $\prec_i = \prec \cap (T_i \times T_i)$. Together these sets $\prec_i$ constitute the set $\prec_{intra} = \bigcup_{i=1}^{n} \prec_i$ of *intra-agent* constraints, while the remaining set of constraints $\prec_{inter} = \prec \setminus \prec_{intra}$ constitutes the set of *inter-agent* constraints. Note that each agent A_i is responsible for achieving the (composite) subtask $(T_i, \prec_i)$.

A. Tasks, Plans and Refinements

To achieve the composite task $(T_i, \prec_i)$, agent A_i has to make a *plan* using its own favorite planning tool [1]. We distinguish between the *abstract plan* and the *concrete plan* of an individual agent A_i. The concrete plan of A_i is the direct output of the agent's planning system containing detailed information about actions to be performed and resources needed. The abstract plan is the translation of the concrete plan in terms of the set of tasks $t \in T_i$ to be performed and their dependency (precedence) relations. Since every feasible plan has to specify a partial order of actions to be performed, an abstract plan for $(T_i, \prec_i)$ is simply a partial order $(T_i, \prec_{p_i})$. Clearly, such an abstract plan satisfies the composite task $(T_i, \prec_i)$ iff $\prec_{p_i}$ *refines* $\prec_i$, i.e., $\prec_i \subseteq \prec_{p_i}$.

Example 2.1: Suppose that an agent is instructed to deliver packages p_1 and p_2 from a location A to location B and from B to C, respectively. That is, he is allocated the composite task $\mathcal{T} = (\{t_1, t_2\}, \emptyset)$, with t_1 : pickup p_1 at A and deliver it at B, and t_2 : pickup p_2 at B and deliver it at C. Since the agent is assumed to be autonomous, it can decide for itself the best plan to accomplish the tasks. Suppose the agent makes the following *concrete* plan: First, it will travel from its current location X to A. Then it will pickup p_1 and it will go to location B delivering package p_1. Hereafter the agent will pickup p_2 in B and will take a detour via Y to C in order to have lunch. It will stop in C to deliver p_2 and then finally will return home to X.

From a coordination point of view, several pieces of information are not of interest, namely: the exact location from which the agents starts, going to Y for lunch, and returning home to X. The only relevant information derived from the concrete plan occurs in the *abstract* plan specifying that the agent will perform t_1 before t_2.

[1] In general, planning to achieve such a set of elementary tasks constitutes a non-trivial problem. For example, even making an optimal plan for a set of unrelated pickup-delivery orders constitutes an NP-hard problem.

Thus, the agent's abstract plan is the partial order $(T, \prec_p)$ with $t_1 \prec_p t_2$. Indeed, this plan satisfies $(T, \emptyset)$, because $\prec_p$ clearly refines $\emptyset$.

To check whether a concrete plan can be coordinated with the plan of another agent, we only need the information contained in its abstract plan.[2] Hence, from now on we only use abstract plans of agents and assume that they are feasible, i.e., simple refinements of the composite tasks given to the agents.

B. The Coordination Problem

A coordination algorithm or protocol for autonomous planning agents should ensure that, after receiving its part T_i of the joint task $\mathcal{T}$, (*i*) each autonomous agent A_i is allowed to construct its plan independently from the other agents, (*ii*) there is a simple way to combine their plans into a joint plan, while respecting each individual plan[3], and (*iii*) both these objectives should be achieved irrespective of the choice of the plans constructed by the individual agents.

Clearly, a simple task allocation alone does not always guarantee the existence of such a feasible joint plan:

Example 2.2: Consider the composite task depicted in Figure 1, where the tasks t_1 and t_4 are allocated to agent A_1 and the tasks t_2 and t_3 are allocated to agent A_2. Precedence relations are represented by arcs: e.g. the arc e_1 between t_1 and t_2 represents the precedence relation $t_1 \prec t_2$. The arcs e_1 and e_2 together constitute the set of precedences (so the set of intra-agent precedences is empty). The dotted arc r_1 represents a feasible *refinement* for agent A_1: since there is no precedence relation between t_1 and t_4, agent A_1 might come up with a plan where t_4 is executed prior to t_1. Similarly, agent A_2 might decide to introduce the refinement r_2, by planning to execute t_2 before t_3. Each of these plans is a perfectly feasible plan meeting the intra-agent constraints. However, if both refinements r_1 and r_2 are made, then combining these plans into a joint plan that respects them both results in an infeasible joint plan, due to the existence of a cycle $e_1 - r_2 - e_2 - r_1$. At run-time, such a cycle in a joint plan would cause a deadlock, since it requires e.g. t_1 to be executed before t_2, and also t_2 to be executed before t_1.

It can be easily shown that the only possibility[4] to ensure a solution to the coordination problem is to add, prior to planning, a set $\Delta = \bigcup \Delta_i$ of additional constraints to the set of precedence constraints $\prec$. More precisely, each set Δ_i has to be added to each set of precedence constraints $\prec_i$ such that the following conditions are satisfied:

1) for all i, $(T_i, \prec_i \cup \Delta_i)$ constitutes a composite task, i.e., $\prec_i \cup \Delta_i$ is a partial order refining $\prec_i$;
2) for every conceivable set $p_{\mathcal{A}} = \{p_1, \ldots, p_n\}$ of individual plans of agents where each $p_i = (T_i, \prec_{p_i})$ is a plan for $(T_i, \prec_i \cup \Delta_i)$, the structure $P = (\bigcup_{i=1}^{n} T_i, \prec \cup \prec_{p_1} \cup \cdots \cup \prec_{p_n})$ is a partially ordered set, i.e., P is a feasible joint plan that respects the individual plans and refines the composite task $(T, \prec)$.

The *coordination problem* then is, given a composite task $(T, \prec)$ and a partitioning $\mathbf{T} = T_1, \ldots, T_n$ of T, to find a *minimum* set $\Delta = \{\Delta_1, \ldots, \Delta_n\}$ of additional precedence constraints to ensure a feasible joint plan whatever feasible abstract plans might be constructed by the individual agents.

Example 2.3: Continuing the previous example, in Figure 1, the set $\Delta = \{\{c_1\}, \emptyset\}$ constitutes a minimal *coordination set* in which agent A_1 receives constraint c_1 and agent A_2 receives no constraints. Note that, due to the constraint c_1, no subsequent refinements can create a cycle in the precedence relation. That is, after adding constraint c_1, every feasible individual plan each of the agents might construct can always be combined into a feasible joint plan.

Fig. 1. A composite task, augmented two potential refinements r_1 and r_2 and a potential constraint c_1.

III. The Complexity of the Coordination Problem and Relations to Problems in Graph Theory

Elsewhere [**?**], we have shown that the problem to decide whether or not a given set Δ of coordination arcs is a solution to the coordination problem, is co-NP-complete. As a result, the decision variant of the coordination problem[5] turns out to be Σ_2^p-complete[6]. A complexity analysis also showed the following results: (i) if each agent has only two tasks to achieve, the decision variant of the coordination problem is NP-complete;

[2]We assume no dependencies between the agents other than the precedence constraints between their tasks.

[3]That is, there should be no need for additional (re)planning for any agent.

[4]If we require that planning and coordination be separated.

[5]This is the problem to decide whether there exists a coordination set of size K or less.

[6]Both proofs rely on a reduction from the path with forbidden pairs (PWFP) problem.

(ii) if each agent has four tasks to achieve, the problem to decide whether $\Delta = \emptyset$ is a solution to the coordination problem (this is the coordination *detection* problem) is co-NP-complete and the coordination problem itself is in Σ_2^p; (iii) if each agent has at least 8 tasks to achieve, the coordination problem is Σ_2^p-complete.

Due to its complexity, it is very unlikely that the coordination problem can be solved in reasonable time. In this section we will use a reduction to the minimum subset feedback arc set (SFAS) problem[7] to find an *approximate* solution to the coordination problem. The minimum subset feedback arc set (SFAS) problem is an extensively studied minimization problem [4], and fast $\mathcal{O}(\log|V|\log\log|V|)$-approximations have been developed [5], [4]. By reducing the coordination problem to the SFAS problem we could use these approximation algorithms to solve the coordination problem.

Briefly, this reduction is based on the following idea: A solution to the coordination problem consists in finding additional constraints Δ such that (whatever intra-agent refinement arcs are added) the resulting graph cannot become cyclic. So why not construct a graph that already contains all possible intra-agent refinements and then select a feedback arc set F consisting of intra-agent arcs. If we *invert* all arcs in F then adding this set F^{-1} should break every such cycle. To make a coordination set, the inverted feedback arc set only needs to break cycles that involve more than one agent; cycles within an agent are already taken care of by that agent (because we assume agent plans to be acyclic). So we need to map coordination instances to minimum *subset* feedback arc set instances.

Specifically, the transformation from the coordination problem to SFAS is as follows: given a coordination instance $(\mathcal{T}, (T_1, \ldots, T_n))$, construct an SFAS instance (V, A, B) such that

1) $V = T$;
2) $A^+ = [\prec \cup \hat{\Delta}_1 \cup \cdots \cup \hat{\Delta}_n]^+$, with $\hat{\Delta}_i = (T_i \times T_i) \setminus (\prec_i \cup \prec_i^{-1})$ the set of possible refinements for agent A_i;
3) $B = \prec_{inter}$, i.e., only those cycles are considered that intersect the set of inter-agent dependencies.

To obtain a coordination set, we compute a feedback arc set F of the above SFAS instance, and then we invert the arcs in F to obtain a solution $\Delta = F^{-1}$. We can easily show that if a subset-minimal feedback arc set F has been found, then the corresponding $\Delta = F^{-1}$ is a solution of the original coordination instance [**?**].[8]

Unfortunately, although this approach is sound, i.e., every solution found by this reduction constitutes a solution to the coordination problem, it has two obvious disadvantages: First of all, the reduction may sometimes be too constraining[9]. Figure 2 shows an instance where an SFAS solution will constrain the agent's planning freedom more than necessary: one of the dashed (refinement) arcs will be placed in a feedback arc set, even though the instance is already coordinated: the only way a cyclic joint plan can be created is if agent A_2 chooses refinements r_1 and r_2. But such a refinement cannot occur, because it would create a cycle in A_2's plan. As a result, the $\mathcal{O}(\log(|V|)\log\log(|V|))$-ratio for SFAS is not inherited by the coordination problem. Secondly, the approximation algorithms applied to solve the coordination problem are *centralized* algorithms, leaving no opportunity for the agents to influence the coordination process.

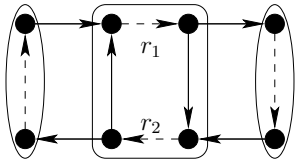

Fig. 2. Despite the presence of an inter-agent cycle, this instance is coordinated.

IV. A Distributed Coordination Algorithm

To allow agents to influence the coordination process we will now present a *distributed* algorithm to solve the coordination problem in which an agent receives additional precedence constraints (a subset of the coordination set) only if it decides to accept these constraints.

The algorithm is based on the following idea: in constructing a plan for $(T_i, \prec_i)$, each agent A_i can safely start to make a plan for a subset of tasks $T_i^1 \subseteq T_i$ that are not dependent (via inter-agent constraints $\prec_{inter}$) upon tasks assigned to other agents. To determine whether a given task in T_i depends on a task assigned to another agent, we let the agents use a common *blackboard*. This blackboard stores the inter-agent dependency relations.

[7]The problem given a digraph $G = (V, A)$, and a subset $B \subseteq A$ to find a minimum subset $F \subseteq A$ that intersects every cycle in G that contains at least one arc of B.

[8]Only arcs in $\{\hat{\Delta}_1, \ldots, \hat{\Delta}_n\}$ may be considered for placement in F, since arcs in $\prec$ may not be inverted.

[9]This is not surprising, because, assuming $P \neq NP$, the coordination problem is outside the class NPO. Minimum SFAS, however, is known to belong to NPO.

Once each agent A_i has selected such a subset T_i^1 (which may be empty), the agent informs the blackboard that the tasks t in T_i^1 can be removed from the set T_i, together with all precedence constraints in $\prec$ that involve task t. Then the agents iteratively start a new round where each agent A_i again selects a subset T_i^2 of the set of remaining tasks not dependent on tasks assigned to other agents. These rounds continue until after, say, $k \leq |T|$ rounds the set of remaining tasks is empty. As a result, the original task T_i of agent A_i is partitioned into a set of k possibly empty subsets $T_i^1, \ldots, T_i^k$. Now the coordination set Δ_i for agent A_i is constructed as follows: first, all empty subsets T_i^j in $\{T_i^j\}_{j=1}^k$ are removed. Next, for every $j = 1, \ldots, k-1$ and for every pair of tasks t, t': if $t \in T_i^j$ and $t' \in T_i^{j+1}$, the precedence constraint (t, t') is added to Δ_i.

To obtain these sets T_i^j, each agent A_i executes the following algorithmic scheme:

Algorithm 1

Input: *a composite task $(T_i, \prec_i)$ assigned to agent A_i*
Output: *a linearly ordered partition $(T_i^1, \ldots, T_i^k)$ of T_i*

begin
1. *let* $k := 0$
2. **while** $T_i \neq \emptyset$ **do**
   2.1. $k := k + 1$
   2.2. *ask the blackboard for the subset $F_i \subseteq T_i$ of tasks that are* prerequisite-free, *i.e., those tasks t in T_i for which there exists no $t' \in T_j$, $j \neq i$, such that $t' \prec t$*
   2.3. T_i^k := *Select_SubsetFrom(F_i)*
   2.4. **if** $T_i^k \neq \emptyset$ **then**
      2.4.1. *Let* $T_i = T_i - T_i^k$
      2.4.2. *Send the set T_i^k to the blackboard*
   2.5. **else**
      2.5.1. $k := k - 1$
3. *return* $(T_i^1, \ldots, T_i^k)$

end

In each round of the algorithm, agent A_i splits off a (possibly empty) set of tasks T_i^k from T_i. In Step 2.2, agent A_i asks the blackoard to compute the set F_i of tasks in T_i that are free of prerequisites. In Step 2.3, A_i makes a decision which subset T_i^k of F^i it will split off in this round. Note that if no deadlock occurs, each agent A_i is able to determine its subset of coordination arcs. Observe that if all agents repeatedly decide to 'split off' the empty set, i.e., they wait until they can choose a subtask T_i^j as large as possible, one or more agent processes might deadlock.

Recall that tasks in a block T_i^j must precede all tasks in the block T_i^{j+1}. In fact, the set Δ_i of additional constraints for agent A_i is specified by:

$$\Delta_i = \bigcup_{j=1}^{k-1} T_i^j \times T_i^{j+1}$$

Thus, the more blocks the partitioning contains, the more restrictions are placed on the set of tasks to be performed. Since more precedence restrictions also restrict the freedom of planning of an agent, and thereby reduce the possibility to find cheaper plans, an agent might be reluctant to split its set of tasks too soon. Instead, it might prefer to postpone splitting off a block until F_i is sufficiently large. If possible, an agent would thus prefer to adopt a *lazy* partitioning strategy, that is: an agent would choose $T_i^j = \emptyset$ in every round of Algorithm 1, unless $F_i = T_i$, in which case it chooses $T_i^j = T_i$. Clearly though, if all agents are lazy (and $\prec_{inter}$ is non-empty), then the algorithm deadlocks.

To ensure coordination, more cooperative strategies are needed. The opposite of a lazy agent is a *diligent* agent: in every round of Algorithm 1, a diligent agent chooses $T_i^j = F_i$. If all agents are diligent, the algorithm never deadlocks, and a coordination solution is found. Moreover, we can give some guarantee with regard to the worst-case performance of the partitioning, as it is not hard to see that if all agents are diligent, then the depth of each agent's partitioning (the value k for $T_i^1, \ldots T_i^k$) is at most the minimum of $|T_i|$ and the depth of the partial order of the complete task $\mathcal{T}$.

In some cases, when a subset of the agents are lazy, and the rest of the agents are diligent, we can still guarantee that Algorithm 1 runs deadlock-free. The dependencies between agents can be visualized in the *agent dependency graph*. The agent dependency graph is a directed graph $G = (V, E)$, where $V = \mathcal{A}$ (the set of agents), and $e = (A_i, A_j) \in E$ iff there exist tasks t and t' such that $t \in A_i$, $t' \in A_j$, and $t \prec t'$. If the agent dependency graph restricted to the set of lazy agents is acyclic, then Algorithm 1 will not deadlock.

The worst-case performance of the coordination-by-partitioning approach, measured in terms of the cost of the resulting plan, is bounded by the depth of the partitioning each agent creates for his set of tasks. If an agent A_i splits his set of tasks T_i into k segments $T_i^1, \ldots, T_i^k$, then the cost of the combined plan for all k segments can be at most k times as high as a plan for his unconstrained set of tasks T_i. As an example, consider

the case where an agent only has two tasks t_1 and t_2. If there exists no precedence constraint between t_1 and t_2, then we might imagine that an agent can execute both tasks in parallel. By introducing the constraint $t_1 \prec t_2$, the tasks can no longer be executed in parallel, so the cost of the plan may increase — but with no more than the cost of the original plan, since at most all work done on t_1 must be redone for t_2.

In case all agents are diligent, this means that no agent can have a worst-case plan cost higher than d times his optimal plan, with d the depth of partial order of the composite task $\mathcal{T}$.

V. Application to coordination in multi-modal Logistics

Our approach enables existing stand-alone planners to be used in solving planning problems that require the joint effort of several planners that are dependent upon each other. In this section we will show how our coordination method can be used to compose such a multi-agent planning system for logistic planning.

The logistics domain comprises transportation of packages in an infrastructure consisting of airplanes, trucks, cities and locations. Each city consists of a set of locations, one of which is the city's airport. Within a city, trucks can transport packages; between cities, airplanes can transport packages. The aim is to make a plan for picking up a set of packages at their source location and to deliver them at their destination location (which may be located in a different city).

Intra-city orders, i.e., the transportation of packages with source and destination in the same city, can be carried out by a single truck. Inter-city orders, however, may require cooperation among airplanes and trucks. For example, suppose a package must be transported from location A in city C_x to location B in city C_y: a truck will drive the package from A to the airport of city C_x, then a plane will fly the package to the airport of city C_y, and finally a truck will deliver the package to location B.

In the world's second planning & scheduling competition, hosted by the AIPS'00 conference [6], several planning domains were to test the competitors' planning systems. All AIPS'00 competitors use a centralized approach in which one planning system computes the actions for all vehicles in the infrastructure. As a result, solving the larger instances requires a complex computational problem to be solved. Most of the systems gave up on the larger-sized problem instances. Only a few planners were able to solve all instances.

We applied the coordination method specified by Algorithm 1 to these logistic problems. The airplanes were configured as one coalition using a lazy strategy and each truck adopted a diligent strategy. This approach allows single-agent planners to be reused in the multi-agent planning context; we assigned very simple route planners to both the trucks and the plane agents and applied the coordination approach to allow these planners to plan concurrently and independently from each other. The airplanes' lazy strategy guarantees that optimal airplane plans can be found. But surprisingly, in this benchmark set, even though trucks adopt the diligent strategy, they are still allowed to execute their optimal plans. The result is that if the airplane coalition and the trucks use optimal solvers for their local planning problems, then the resulting joint plan will also be optimal.

The local planning problems are of such a small size that the whole coordination approach including all local planning activities runs extremely fast. Figure 3 shows the cpu time needed by our coordination approach compared to the top three hand-tailored planning systems of the competition. Our cpu times were obtained using optimal local solvers for the smaller instances; for the larger instances, we used local approximations for the route planning problems. In spite of using local approximations, we still outperformed the AIPS competitors in terms of plan quality, as shown in Figure 4. Although TALplanner and SHOP produce plans of comparable quality, System-R produces considerably longer plans, and its results do not fit in the graph of Figure 4.

VI. Related and Future Work

Except for the multi-agent planning methods already mentioned in the introduction, our approach is also closely related to the work of Shehory and Kraus [7] where they consider a set of tasks with possibly a set of precedence constraints that have to be distributed among (coalitions) of agents capable of performing certain subsets of tasks. However, an important difference with our approach is that Shehory and Kraus assume that performing these tasks requires no planning activity.[10] Hence, coordination can be achieved simply by assigning tasks to agents. Therefore, their approach is geared towards finding the best allocation of tasks to agents. In our approach, however, we assume that a set of interrelated tasks has already been allocated to agents and we concentrate on methods to ensure coordination

[10] More accurately, they assume no difference between planning for one task and planning for two or more tasks.

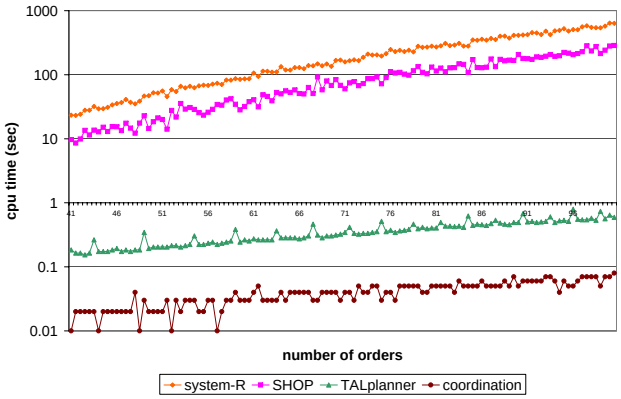

Fig. 3. CPU times required by the hand-tailored planning systems to solve the additional track of logistic problems.

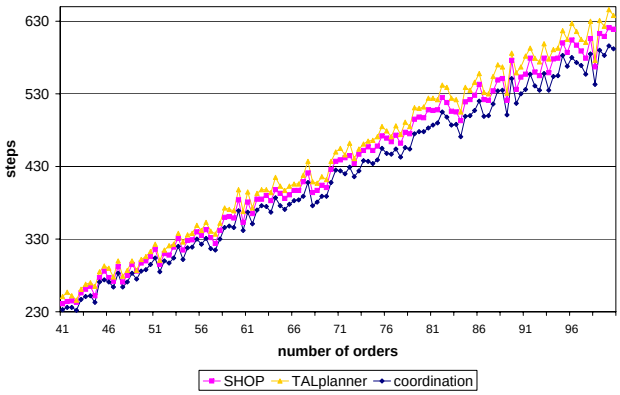

Fig. 4. The number of steps (actions) required by the hand-tailored planning systems to solve the additional track of logistic problems.

while leaving the agents completely free in the planning of their set of tasks.

Regarding future work, more work needs to be done on characterizing stable strategies for planning agents. For the logistic benchmark problems of the AIPS'00, we have identified stable agent strategies for Algorithm 1: a truck will choose the diligent strategy, a plane will choose the lazy strategy, allowing each agent to compute its optimal plan. For general composite tasks, or even for general multi-modal logistic instances, it is likely that we have to come up with negotiation mechanisms in order to establish stability. Rosenschein and Zlotkin [8] summarize the challenge faced by designers of such negotiation mechanisms: the negotiation protocol must induce agents to choose strategies that are globally desirable, yet individually rational. In a coordination-by-partitioning algorithm such as Algorithm 1, a globally desirable strategy must ensure that the algorithm terminates, and consequently returns a coordination set. An individually rational strategy must make it worthwhile for an agent to accept additional precedence constraints.

REFERENCES

[1] E. Ephrati and J. S. Rosenschein, "Multi-agent planning as the process of merging distributed sub-plans," in *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence (DAI-93)*, May 1993, pp. 115–129. [Online]. Available: http://www.cs.huji.ac.il/labs/dai/papers.html

[2] E. H. Durfee and V. R. Lesser, "Partial global planning: a coordination framework for distributed hypothesis formation," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 21, no. 5, pp. 1167–1183, 1991. [Online]. Available: http://citeseer.nj.nec.com/durfee91partial.html

[3] K. S. Decker and V. R. Lesser, "Designing a family of coordination algorithms," in *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence (DAI-94)*, 1994, pp. 65–84. [Online]. Available: http://citeseer.nj.nec.com/decker95designing.html

[4] G. Even, J. Naor, B. Schieber, and M. Sudan, "Approximating minimum feedback sets and multicuts in directed graphs," *Algorithmica*, vol. 20, no. 2, 1998.

[5] P. Seymour, "Packing directed circuits fractionally," *Combinatorics*, vol. 15, 1995.

[6] F. Bacchus, "Aips'00 planning competition (artificial planning and scheduling 2000)," *"AI" Magazine*, 2001.

[7] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artificial Intelligence*, 1998.

[8] J. S. Rosenschein and G. Zlotkin, *Rules of encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.