Resource Based Multi Agent Plan Merging: framework and application

Mathijs de Weerdt, Roman van der Krogt*, Cees Witteveen

Delft University of Technology, P.O.Box 5031, 2600 GA Delft, The Netherlands
{m.m.deweerdt,r.p.j.vanderkrogt,c.witteveen}@ewi.tudelft.nl

Abstract We discuss a resource-based planning framework where agents are able to merge plans by exchanging resources. In this framework, plans are specified as structured objects composed of resource consuming and resource producing processes (actions). A plan itself can also be conceived as a process consuming input resources and producing output resources. A plan can be improved if we can remove actions from it while maintaining goal realizability. We describe a reduction property that specifies how one agent can improve its plan by using (free) resources from another agent in such a way that goal realizability is preserved. The plan-merging algorithm we use to specify plan merging in a multi-agent context is an iterative, distributed, any-time application of this reduction property. The performance of this algorithm has been evaluated using a planning data set obtained from a taxi company. The quality of the algorithm is measured by the decrease of the total distance driven by all taxis. By allowing passengers to share rides, we create a trade-off between the additional travel time of passengers and the total drive distance. Allowing passengers to be a few minutes later at their destination and share rides, a significant improvement of the plans can be obtained (from 5% up to 30% reduction of the taxi driving distance).

1 Introduction

Many complex coordination problems can be modeled as multi-agent planning problems, i.e., a (distributed) set of interrelated AI-planning problems. Examples of such problems are the coordination of transport organizations, armies, and manufacturing processes, to name just a few. A common characteristic of such problems is the need for managing dependencies, in particular *resource dependencies*. Such dependencies are created by resource sharing, resource transaction, resource consumption and resource production processes.

Existing planning frameworks specify plans as means of changing a state of the world into another state and specify such a state by an unstructured set of (atomic) facts relevant to the planning problem. This approach makes them

* Mathijs de Weerdt is supported by the Seamless Multimodal Mobility (SMM) research program and Roman van der Krogt is supported by the Freight Transport Automation and Multimodality (FTAM) research program. Both research programs are carried out within the research school for Transport, Infrastructure and Logistics.

less suitable to deal with resource dependencies in multi-agent problems, since modeling a world consisting of resources requires a more structured (resource-oriented) approach than modeling a world as a collection of facts.

In this paper we discuss such a resource oriented approach to deal with multi-planning problems. This action resource framework (ARF) consists of *resource facts* and *actions*. A resource fact aggregates the attributes of an (exchangeable) object (resource) and specifies all the properties (attributes) of such an object at a specific time into a single predicate. Actions are rules to transform a set of resource facts.

Using the ARF-approach, we introduce a *plan merging* method to solve multi-agent planning problems. This plan merging method requires each planner to be capable of finding a solution for its own planning problem, e.g., by using an existing refinement planning method [1]. After the plan construction phase, the planners try to cooperate by exchanging resources in order to reduce costs. For example, if two taxis have each planned to bring one passenger from the railway station to the hospital at about the same time, plan merging might result in a plan driving two passengers in one taxi.

We show that using resources as the atomic objects of a planning problem and actions as resource-consuming and resource-producing processes appears to work very well not only in specifying a plan merging algorithm, but also in solving a practical multi-agent planning problem as coordinating taxi plans.

This paper is organized as follows: In the next section we take a closer look at the action resource framework. Then we show how this framework can be used to design and specify an algorithm to improve plans. In Section 3 we discuss the plan merging problem and its solution using this framework. Then we present experimental results obtained from using this plan merging algorithm on a data set of the daily operations of taxi company. We finish by discussing related work and possible extensions to the plan merging algorithm.

2 The action resource framework

The *Action Resource Framework* (ARF) is an improved version of the resource-skill formalism [2,3].[1] This framework has the following properties. Firstly, the planning problem is specified by two sets of *resources* (one set specifying the set of resources available, the other the set of resources to be achieved). Secondly, actions are processes that consume and produce resources. Thirdly, a plan is defined by a set of actions and an (acyclic) dependency relation between these actions.

2.1 Actions and resources

In the ARF the two basic notions in planning, i.e., *resource (facts)* and *actions* are described using a many-sorted logic. Goals and plans are derived notions that are defined using resources and actions.

[1] The ARF and the resource-skill formalism use a subset of linear logic [4] to specify facts and actions.

A *resource fact* is the concise description of an object (resource) that is relevant to an agent with respect to the planning problem at hand. Such a resource is either a physical object such as a taxi or a passenger, or an abstract conceptual notion such as an opportunity to travel, i.e., a ride.[2] Syntactically, an atomic resource fact is denoted by a *predicate name* together with a complete specification of its *attributes*, together with their *values*. The predicate name serves to indicate the *type* of resource mentioned in the fact (e.g., a taxi or a passenger). If *taxi* is a resource type, having an attribute location *loc* and an attribute *num*, then taxi(8 : *num*, A : *loc*) is an atomic resource fact describing taxi number 8 in A. Since similar resources (i.e., resources with the same type and the same values for the attributes) may occur multiple times in the same plan, we add to each resource a unique *occurrence identifier* of sort *identity*, i.e., a special index that is used exclusively to distinguish between different occurrences of a resource in a plan. A resource of type t with i as (the value of its) occurrence identifier is denoted as $t_i(\ldots)$.

Values of attributes may be ground (i.e., constant), but may also be variables or functions. In the latter case, a resource fact describes a *set* of ground resource facts (instances) of the same resource type. For example, the following resource refers to all taxis in location A: taxi(n : *num*, A : *loc*).

To specify one specific ground resource fact denoted by such a *general resource fact*, we introduce the notion of a *substitution*. A substitution θ replaces variables[3] occurring in a resource r by terms of the appropriate sort. We write $r\theta$ to denote the resource r' that results from replacing the variables occurring in r according to θ. If R is a set of general resources, $R\theta$ is a shorthand for $\{r\theta \mid r \in R\}$.

A set of *goals* G is specified by a set of general resources $G = \{g_1, \ldots, g_n\}$. We say that G is *satisfied* by a given set of resources R, abbreviated $R \models G$, if there exists a ground substitution θ such that $G\theta \subseteq R$, i.e., there is a set of ground instances of the goals that is provided by the resources in R. Two resources r_1 and r_2 are called *equivalent*, denoted by $r_1 \equiv r_2$, when they are equal except for the value of their occurrence attribute.

Resource facts are used to model the state of the world (as far as it is relevant) by enumerating the set of resource facts that are true at a certain time point. Possible *transitions* from one state to another are described by *actions*. An action is a basic process that consumes and produces resources. An action o has a set of input resources $in(o)$ that it consumes, and a set of output resources $out(o)$ it produces. Furthermore, an action may contain a specification $param(o)$ of some variables that occur in the set of output resources as *parameters* of the action. To ensure that each output resource *out* is uniquely defined, it may only contain variables $var(out)$ that already occur in the input resources or in the set of the parameters. This is formally defined as follows.

---

[2] Abusing language, we use the notions of a resource and a resource fact interchangeably.

[3] with the exception of occurrence identifiers

Definition 1. *An* action *specification is a formula*
$o_{id}(x_1, \ldots, x_n) : \varphi_{\{in_{id,-i}|1\leq i\leq m\}} \Rightarrow \varphi_{\{out_{id,i}|1\leq i\leq k\}}$ *where*

1. *id is a variable of sort identity,*
2. $x_1, \ldots, x_n$ *are the parameters of the action,*
3. $\varphi_{\{in_{id,-i}|1\leq i\leq m\}}$ *is a conjunction of general (input) resources* $in_{id,-i}(\ldots)$*,*
4. $\varphi_{\{out_{id,i}|1\leq i\leq k\}}$ *is a conjunction of general (output) resources* $out_{id,i}(\ldots)$*, such that for all* $1 \leq i \leq k$*,* $var(out_{id,i}) \subseteq \bigcup_j var(in_{id,j}) \cup \{x_l \mid 1 \leq l \leq n\}$*.*

Remark 1. The value $(id, -i)$ of the occurrence id for the i-th input resource is inherited from the value of the occurrence identifier of the action o in which its is used.[4] Analogously the value (id, i) of output resources is constructed. As can be seen later on, this mechanism provides us with unique occurrence labels of resources in plans.

Example 1. Consider the following action

$$\begin{aligned}\mathbf{move}_{id}(y : loc) : \mathrm{taxi}_{(id,-1)}(n : num,\, x : loc) \Rightarrow \\ \mathrm{ride}_{(id,1)}(x : loc,\, y : loc) \wedge \mathrm{taxi}_{(id,2)}(n : num,\, y : loc)\end{aligned}$$

This action specifies how a taxi may move from a source location x to a destination y and requires the taxi to be present at the source x. It "produces" a taxi at the destination and the ride opportunity (for passengers) to travel with this taxi from x to y.

Let $\mathcal{O}$ be a finite set of action specifications. The set O is said to be a set of (concrete) actions over $\mathcal{O}$ if (i) every action occurring in O is obtained from an action $o_{id}()$ in $\mathcal{O}$ by substituting a unique ground value v for its occurrence identifier id and (ii) variables in actions are standardized apart, i.e., for each $o \neq o' \in O$ we have $\mathrm{var}(o) \neq \mathrm{var}(o')$.

A concrete action $o \in O$ can be applied to a set of (ground) resources R if there exists a ground substitution θ such that $in(o)\theta \subseteq R$. Application of this action to R results in consuming the set $in(o)\theta$ of input resources while producing the set $out(o)\theta$. The result of o applied to R (under θ) therefore is a resource transformation: starting with R, the set $R \setminus in(o)\theta \cup out(o)\theta$ is produced.

The set $res(O)$ is the set of all resources mentioned in the input $in(o)$ or output $out(o)$ of actions $o \in O$. The set $cons(O)$ specifies the set of all resources consumed using actions in O: $cons(O) = \bigcup_{o\in O} in(o)$, while the set of resources produced equals $prod(O) = \bigcup_{o\in O} out(o)$.

2.2 Plans

In general, a single action applied to an initial set of resources is not sufficient to achieve a desired state. Often, a set of actions have to be applied in some partial order to produce the desired effect. A specification of only the ordering

[4] Like resources, the same action can be used at different places in a plan. To distinguish these occurrences, we use occurrence identifiers for actions, too.

of actions, however, in general is not sufficient to describe a plan in sufficient detail. We also need to specify for each consumed resource, which produced resource it is *dependent* upon. Such a partially ordered set of actions with a specification of the dependency relation between resources is called a *plan*. To specify how actions are interrelated, we use the notion of a *dependency function*.

Definition 2. *Let O be a set of (concrete) actions. A dependency function is an injective function $d : cons(O) \to prod(O) \cup \{\bot\}$ specifying (in a unique way) for each resource r to be consumed which resource r' produced by another action is used to provide r (or $\bot$ if r is not produced by an action in O, i.e., r is an input resource).*

Sometimes we need the inverse d^{-1} of d, which is defined as follows: for every $r' \in prod(O)$ such that $d(r) = r'$, $d^{-1}(r') = r$, and for every $r' \in prod(O)$ not occurring in $\mathrm{ran}(d)$, $d^{-1}(r') = \bot$.

As mentioned before, plans are composed of partially ordered actions. Since a dependency function d specifies an immediate dependency of input resources of an action on output resources of another action, d can only specify a *valid* dependency if (i) the resources involved are equivalent and (ii) d generates a partial ordering of the actions in O.

The first requirement is met if there exists a substitution θ such that for two resources r and r', $d(r) = r'$ implies $r\theta \equiv r'\theta$, that is θ is a *unifier* for every pair of resources $(r, d(r))$. In particular, we are looking at a *most general unifier* (mgu) θ with this property.

The second condition requires that there are no loops in the dependency relation between actions generated by d: we say that o directly depends on o', abbreviated as $o' \ll_d o$, if resources $r \in in(o)$ and $r' \in out(o')$ exist such that $d(r) = r'$. Let $<_d = \ll_d^+$ be the transitive closure of $\ll_d$. Then the second condition simply requires $<_d$ to be a (strict) partial order on O.

Now a plan can be defined as follows.

Definition 3. *A* plan *P over a set of actions O is a triple $P = (O, d, \theta)$ where d is a dependency function specifying dependencies between equivalent resources and generating a partial order $<_d$ on O, while θ is the mgu of all dependency pairs $(r, d(r))$ with $r, d(r) \in res(O)$. The empty plan $(\emptyset, \emptyset, \emptyset)$ is denoted by $\lightning$.*

Given a plan $P = (O, d, \theta)$, the set $In(P) = \{r\theta \mid r \in cons(O), d(r) = \bot\}$ is the set of input resources of P, i.e., the set of resources not depending on other resources in the plan. Analogously, the set $Out(P) = \{r\theta \mid r \in prod(O), d^{-1}(r) = \bot\}$ of output resources of P is the set of resources that are not consumed by actions in the plan. Furthermore, we define $prod(P) = prod(O)\theta$ and $cons(P) = cons(O)\theta$. Note that resources from $prod(P)$ may be consumed by other plans to produce resources.

A *planning problem* $\Pi = (\mathcal{O}, I, G)$ is given by a set of ground initially available resources I, a set of goal resources G, and a set of action specifications $\mathcal{O}$ that can be used for this problem domain to transform resources. A *solution*

to such a planning problem Π is a plan $P = (O, d, \theta)$ where O is a set of concrete actions over $\mathcal{O}$ and there exists a substitution τ such that $In(P)\tau \subseteq I$ and $Out(P)\tau \models G$.

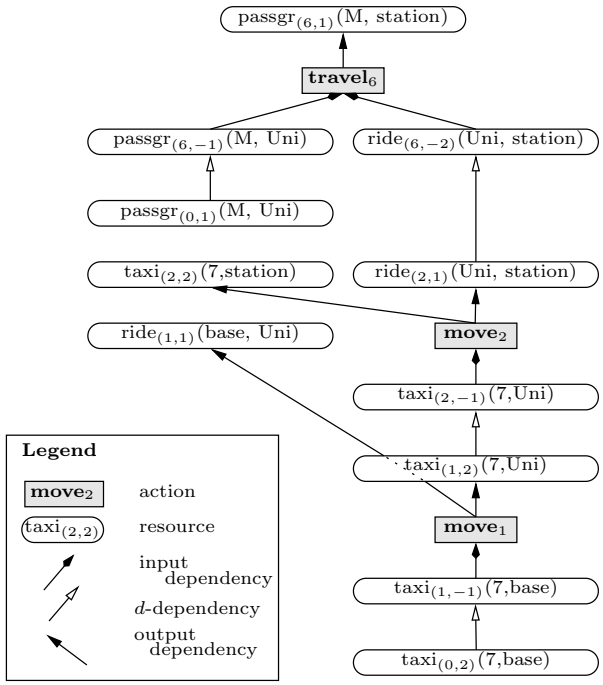

Figure 1. Dependencies between the resources and actions.

Example 2. Consider the following taxi planning problem $\Pi = (\mathcal{O}, I, G)$. The set of action specifications is $\mathcal{O}$ contains **move** action specifications analogously to the **move** action in Example 1, and the following **travel** action specifications:

$$\begin{aligned}\textbf{travel}_{id} : \text{passgr}_{(id,-1)}(n : name,\ x : loc) \wedge \text{ride}_{(id,-2)}(x : loc,\ y : loc) \Rightarrow \\ \text{passgr}_{(id,1)}(n : name,\ y : loc)\end{aligned}$$

The initial state I is defined as

$$I = \left\{\text{passgr}_{(0,1)}(\text{M} : name,\ \text{Uni} : loc),\ \text{taxi}_{(0,2)}(7 : num,\ \text{base} : loc)\right\}.$$

The goal G is specified as the single resource $\text{passgr}_{(t,i)}(\text{M} : name,\ \text{station} : loc)$. To solve this problem, a ground plan[5] $P = (O, d, \theta)$ is devised, where

[5] A plan $P = (O, d, \theta)$ is ground if every resource mentioned in it is a ground resource.

– the set of concrete actions O consists of two instances of the **move** action and an instance of the **travel** action,
– d is the relation as given by Figure 1 (including the dependency on the resources of the initial state), and
– θ is defined as the composition of the substitutions given in Table 1.

It can be easily seen that d is a valid dependency function, that $In(P) \subseteq I$ and since $\text{passgr}_{(6,1)}(\text{M} : name, \text{station} : loc) \in Out(P)$, $Out(P) \models G$. Hence, P is a solution to Π.

Table 1. The definitions of the substitutions for the three actions in Figure 1.

variable	value
x_1	base
y_1	Uni
x_2	Uni
y_2	station
n_6	M
x_6	Uni
y_6	station

2.3 The plan reduction property

We assume that associated with each concrete action $o \in O$ there is a positive integer value $cost(o)$, representing the cost of executing o. Agents, once having conceived a plan $P = (O, d, \theta)$ will aim at reducing the cost $cost(P) = \Sigma_{o \in O} cost(o)$ of their plan, while maintaining goal-realizability. To reduce the costs of a plan an agent therefore might try to remove some action o from its plan, by replacing every output resource r of o that is used in the plan, i.e. $d^{-1}(r) \neq \bot$, by another other, freely available resource r' occurring in the current plan or by an input resource of o. The resources r' that are not used in a plan to attain the goals are called *free resources* and are defined as follows:

Definition 4. *Given a plan $P = (O, d, \theta)$ that is a solution to a problem $\Pi = (\mathcal{O}, I, G)$, and a substitution θ_G such that $G\theta_G \subseteq Out(P)$, the set of* free resources *is defined as $Free(P, \Pi) = (I \setminus In(P)) \cup (Out(P) \setminus G\theta_G)$.*

If, using $Free(P, \Pi) \cup in(o)\theta$, every output resource can be replaced such that a cheaper valid plan P' can be obtained, we say that P is *reduced* by removing o.

Note that if o cannot be removed, a nonempty subset of its (used) output resources cannot be replaced. We call this set the request set $RequestSet(P, \Pi, o)$ belonging to o and note that it contains the set $\{r \mid r \in out(o),\ d^{-1}(r) \neq \bot\} \setminus (Free(P, \Pi) \cup in(o))$. [6] Viewed from a multi-agent perspective, these request

[6] The exact specification of the request set is more involved and requires the check for dependencies of free resources upon used output resources of o.

sets can be used for resource transactions between agents: if free resources are available from other agents, we can show that a successful reduction of the plan P exists if these free resources are sufficient to attain *RequestSet*:

Theorem 1. *Given a set of agents A, for each agent $i \in A$ a problem Π_i and a plan P_i that solves this problem, the action o in the plan P_j of agent j can be removed if*

$$\bigcup_{i \in A \setminus \{j\}} \mathit{Free}(P_i, \Pi_i) \models \mathit{RequestSet}(P_j, \Pi_j, o).$$

A proof of this theorem can be found in [5]. This reduction property is used in the plan merging algorithm to be discussed next.

3 Plan merging

The principal idea behind plan merging is that agents can *reduce the cost* of their own plan, i.e., they can remove actions for which the results (the resources produced by these actions) are *readily available* at other agents. To facilitate the exchange of resource facts, we assume that a trusted third party acts as the *auctioneer*.

The plan merging algorithm (Algorithm 1) works as follows: The auctioneer announces when the agents can start the plan merging, thereby announcing some minimum allowed cost reduction value. All agents deposit their request sets with this auctioneer. Each request set corresponds with the removal of an action from an agent's plan and contains a set of resource facts the agent needs to remove the particular action. Furthermore, the request contains a *cost reduction value* defined by the difference in costs between the old plan and the resulting plan if the exchange would succeed.

The (greedy) auctioneer deals with the request with the highest potential cost reduction first. We assume that all the agents honestly announce their cost reduction values. Right before each auction round starts, the requesting agent (a_i) is asked for the specific set of resource facts that has to be replaced by resource facts of other agents – this set is called the *RequestSet*. This set is not necessarily equal to the set in the initial request, since other exchanges influence the availability of resource facts for the agents. Next, the set of requested resource facts is sent to each agent, except to a_i. The agents return all their free resource facts for which there is an equivalent one in the request set *RequestSet*, and include the price of each of their offered resource facts. When all bids (collected in R') are collected by the auctioneer, it selects for each requested resource fact the cheapest bid.

If for each resource fact in *RequestSet* a replacement can be found, the requesting agent a_i may remove the corresponding action(s). Furthermore, we have to add *dependencies* between the providing agents and the initial resource facts for the requesting agent. At the end of each successful exchange each involved agent has to update the cost reduction values of all of their requests, because this

Algorithm 1 PLAN_MERGING(A)

1. *auctioneer broadcasts minimum allowed cost reduction.*
2. *auctioneer retrieves requests with their cost reduction from all agents A.*
3. **while** *some requests left* **do**
   - 3.1. *get the request with the highest cost reduction.*
   - 3.2. *ask the requesting agent a_i for the required resource facts RequestSet.*
   - 3.3. **for each** *agent $a_j \in A \setminus \{a_i\}$* **do**
     - 3.3.1. *ask a_j for free res. facts equivalent to RequestSet.*
     - 3.3.2. *add these resource facts to R'.*
   - 3.4. **if** *$R' \supseteq RequestSet$* **then**
     - 3.4.1. *let $R'' \subseteq R'$ be the cheapest set that satisfies RequestSet.*
     - 3.4.2. *add for each $r \in RequestSet$ the corresponding dependency to R''.*
     - 3.4.3. *remove as much actions as possible from the plan P_i of a_i.*
     - 3.4.4. *for each involved agent, update the cost reduction of all requests.*

value can change as the agent can now have more or less resource facts available. One could repeat this process until none of the auctions has been successful.

The plan-merging algorithm is an *any-time* algorithm, because it can be stopped at any moment. If the algorithm is stopped, it still returns an improved set of agent plans, because it uses a greedy policy, i.e., dealing with the requests with the largest potential cost reduction first. Algorithm 1 can be shown to have a worst-case time complexity of $O(n^2)$ where n is the number of actions of the plans of all agents involved in plan merging [2].

4 Experimental results

After the formal analysis of the plan-merging algorithm, three questions remained which we tried to answer by doing experiments.

- Concerning the *performance* of the plan merging algorithm, we are interested in the practical aspects of the time complexity. For example, is the quadratic worst-case complexity applicable also to real data and what is the size of doable instances of plan merging problems.
- Concerning the *applicability*, we would like to establish experimentally whether plan merging can be used in realistic cases to obtain a non-negligible cost reduction.
- Finally, we would like to explore the effect of the *greedy approach* in real applications to get an idea about the relation between the cumulative improvement versus the available run time.

The ideal data set to use for these experiments would consist of the schedules of several taxi companies in the same region and for the same time that are prepared

to have their passengers share taxis. Such information is very hard to obtain. Instead, we succeeded in retrieving a data set from one large taxi company.[7] We created a hypothetical experiment for multiple organizations given a centrally created schedule by the company's call center as follows: The schedule for 35 taxis from this one taxi company were used to represent separate plans for 35 *virtual* taxi companies. These plans were merged using the plan merging algorithm. We assume that the results for merging the plans of these virtual taxi companies give a reliable indication of a situation where real taxi companies are (co)operating.

The received schedule contains about 600 rides a day (600 move actions for the taxis and 600 travel actions) of 35 taxis exactly as they were planned in the months January and February 2002. This information includes a taxi number, (start and end) date and time, number of passengers and the origin and the destination location of each order. From this information we constructed an action resource plan for each of the taxis, and we tried to find improvements over these plans using the plan merging algorithm.

We assume that picking up a passenger along an already planned route does not invalidate the existing plan of a taxi. Furthermore, we assume that in this domain the costs of a plan equal the sum of the costs of the move actions where the cost of each move action equals the distance driven the taxi during this action. Consequently, our goal is to reduce this distance by exchanging orders among taxis (agents). Therefore, one of the resource facts describes a passenger and its attributes (destination, preferred pickup time, etc). The most important resource fact, however, describes the ride of a taxi and models the possibility for passengers to travel from one location to another with a taxi. This is the only type of resource fact that is exchanged between taxis and is involved in plan merging.

Remark 2. If a customer is transported by taxi A instead of by taxi B the additional distance and time needed by taxi A is estimated using the euclidean distance measure and an analysis of all drives as extracted from the data set. In principle, the (ground) plan merging algorithm is only useful if one taxi drives *exactly* past the pick-up and the delivery location of a passenger assigned to another taxi at *exactly* the right time (equivalent resources). This almost never happens. Therefore, we adapt the following notion of equivalence: we allow an exchange if the passenger's estimated arrival time is not increased by more than Δt minutes and the detour length of taxi A is less than the distance reduction of taxi B.

4.1 Run-time analysis

First we analyze the running time of plan merging. Given the worst-case quadratic upperbound, we expect a quadratic run-time complexity to occur since the number of exchangeable resources is about quadratic in the number of actions. To test this hypothesis, we run the algorithm with a fixed Δt of 3, 6, and 15 minutes and a fixed day on a number of plans varying from 2 to 35. Each test is

[7] Taxi Zeevang, Purmerend, The Netherlands.

performed 20 times on a randomly selected set of taxis. For each run we store the total number of actions of all involved plans and the run time on a 1GHz Pentium processor. The results of these runs can easily be fitted with a quadratic function (about $2 \cdot 10^{-5} n^2$ to $4 \cdot 10^{-5} n^2$), as can be seen in Figure 2. The standard error of these fits is very small, as can be seen in the first column of Table 2. Performing these experiments we observed that it takes less than a minute to merge the plans for one day of all 35 taxis (1200 actions).

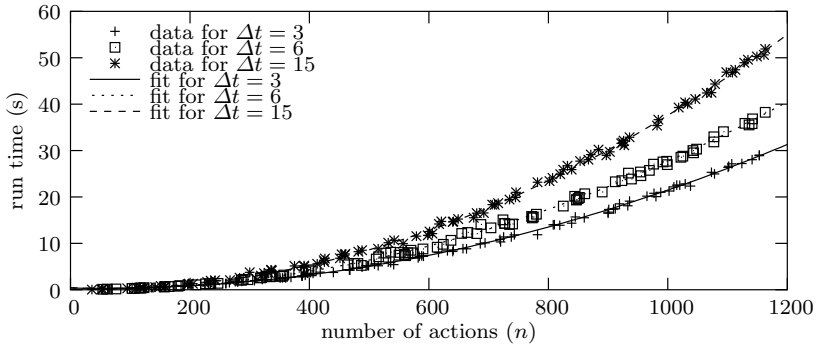

Figure 2. The run time versus the number of actions for three different values for Δt.

Table 2. The standard deviation of the fits in Figure 2 and 3.

Δt	run-time fit sd.	reduction fit sd.
3	0.310	12.6
6	0.468	21.0
15	0.517	30.9

4.2 Improvement of the efficiency

Next, we are interested in the applicability of plan merging i.e., the cost reduction achieved. For each run we compared the total distance driven by the taxis, before and after the plan merging algorithm. In Figure 3, the difference between these values is plotted against the number of actions. As expected, more relaxed time constraints on the arrival time of the passengers lead to more improvement. Furthermore, the total improvement seems to be linear in the number of actions. The standard deviation of this fit is shown in the last column of Table 2. The *relative* improvement in drive distance (in percentage) is given in Table 3. The main disadvantage of reducing the distance driven by the taxis is that the

passengers need more time to get to their destinations. Table 3 also shows the *increase* in passenger travel time (in percentage).

Table 3. The decrease of the drive distance and the increase of the passenger travel time (relative, by giving a percentage (%) and the standard deviation (sd)).

Δt	reduction distance (%) sd.	increase time (%) sd.
3	3.32 2.06	2.60 1.85
6	8.41 4.12	7.05 3.94
15	18.0 7.62	16.4 7.26

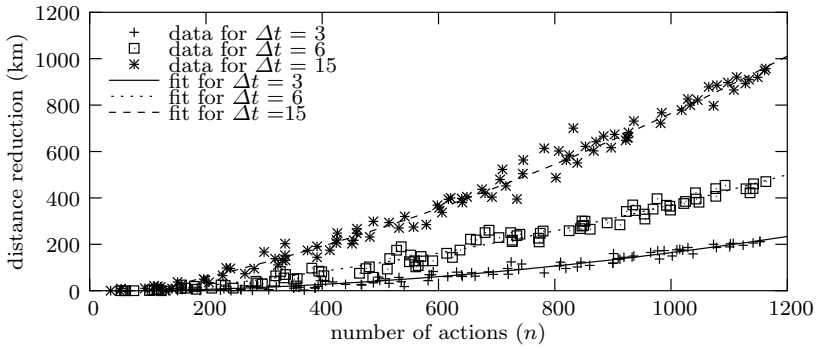

Figure 3. The improvement in drive distance versus the number of actions for three different values of Δt.

The strong relation between the plan size and both the reduction of the distance and the increase of the travel time can be explained by the fact that between larger plans more combinations are possible and thus more exchanges can be made.

To be able to compare the reduction of the drive distance of the taxis and the increase of the total travel time of the passengers, we calculated the relative values. The relation between the relative reduction of the drive distance (by sharing rides) and the relative increase of the total travel time of the passenger is shown in Figure 4. The correlation coefficient of the relation between the relative reduction of drive distance and the increase in passenger travel time is 0.947 (for $\Delta t \leq 15$).

Remark 3. For large plans and a pick-up and drop-off margin of at most 6 minutes, an increase of 11 percent in travel time was shown to result in a distance reduction of about 13 percent. Moreover, when passengers are allowed to arrive

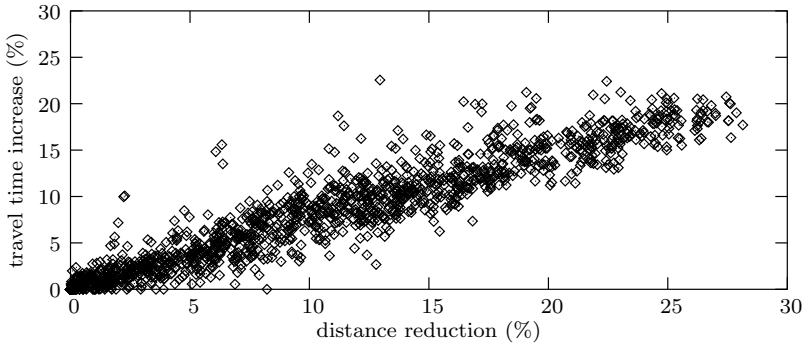

Figure 4. Relative increase of the total travel time versus the relative reduction of the total drive distance.

within an interval of 15 minutes, as is the current agreement for most Dutch low-budget elderly transportation services, the improvement can be up to 30 percent.

4.3 Greedy behavior

The greedy behavior of the algorithm is analyzed by the following experiment. For a specific run of the algorithm (in this case for two days (10 and 45) and for 34 agents), both the time and the total distance driven by the taxis are registered after each auction. The results of this experiment can be found in Figure 5. Clearly the additional improvement decreases over time. This greedy behavior makes the algorithm quite suitable to be used as an anytime algorithm.

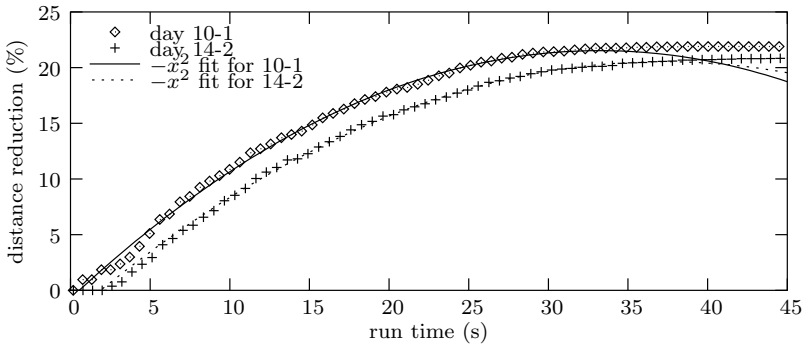

Figure 5. The relative reduction of the total drive distance versus the run time for two different days.

5 Discussion, Related and Future work

Formalism We discussed a formal resource based logic framework for multi-agent planning. The most popular planning specification systems proposed in the literature are based on first-order logic (cf. STRIPS [6] and PDDL [7]). STRIPS resembles our formalism with respect to the fact that for each operator explicitly is stated which atomic facts are to be deleted from the current state and which have to be added to create the new state. However, an action in the ARF by default deletes all its inputs. In the ARF it is easier to model cooperation between agents, because there we need to specify exchangeable objects. ARF provides such a specification by using resources: all relevant attributes of an exchangeable object are collected within one resource fact.

We remark that, like the planning logic by Masseron et al.[8], ARF can be conceived as based on a subset of linear logic [4]. Since resources can only be used once (no absorption), it seems quite natural to study the relationship between planning and linear logic. However, the fragment of linear logic used for the embedding is very restricted. It turns out that actions and resources have to be translated as a *linear theory*, using only one linear logical connective (multiplicative conjunction) in the deterministic case, and two (multiplicative conjunction and additive disjunction) in the nondeterministic one. In their paper, Masseron et al.[9] do not touch the problem of plan merging. In our opinion, our presentation of the planning formalism is better suited to study the operational behavior of planning.

Many planning systems have integrated the scheduling of resources by including some sort of Constraint Satisfaction Problem or Linear Programming problem solver [10,11,12] to deal with, e.g., consumable goods. In our work we may use such solvers to deal with the individual constraint satisfaction problems for the agents. Our contribution, however, lies in the use of planning algorithms in a *multi-agent* context. The importance of resources in a multi-agent system has led to making resource facts the main concept in modeling the *coordination* of the actions of agents.

Plan merging Using the ARF, we specified a polynomial any-time plan merging algorithm. This algorithm uses the idea that plan reduction can be obtained by removing actions from a plan and maintaining goal realizability by using (free) resources from other agents. We note that the plan merging algorithm discussed in this paper achieves a locally optimal merged plan. It is not difficult to show that searching for a globally optimal plan is NP-hard.

Application A schedule for 35 taxis was used to create a hypothetical experiment. In this experiment we divided the taxis randomly over a set of artificial taxi companies (i.e., the agents). The plans of these agents were then merged using the plan merging algorithm. These experiments showed that when the taxi company would allow passengers to share a taxi, the reduction of the costs of a plan were strongly related to the additional travel time of passengers. Furthermore,

evidence showed that the ground plan merging algorithm had a very good anytime behavior and a quadratic time complexity with a very low constant.

Summarizing, we conclude that both the formalism and the algorithm work quite well on realistic data and we believe that the proposed problem definition and the formalism are useful for further research on coordinated planning.

Extensions to Multi-agent Planning A disadvantage of plan merging in general is that it cannot be used in situations where an agent already needs to cooperate with others to *construct* a plan. To deal with these cases, the plan merging scheme has to be extended to a *multi-agent planning* method by (i) allowing agents to be able to *request* services from other agents and include the results in their plans, or (ii) by allowing agents to be able to *offer* services to other agents and, upon a request, add these to their plans. Exchanging services enables agents to not only offer resource facts that are already in its plan (and unused), but also to adapt its plan to produce resource facts that are desired by other agents. Such extensions should lead to a distributed algorithm where self-interested agents create plans including coordinated (efficient and conflict-free) actions.

Most solutions to multi-agent planning problems, however, *(i)* cooperatively create plans for all agents without dealing with the self-interestedness of agents, called cooperative distributed planning [13], such as PGP [14,15], *(ii)* focus on task allocation [16] and conflict resolution *before* planning [17,18], or *(iii)* conflict resolution *after* planning [19]. An extension of the plan merging algorithm should integrate coordination and conflict resolution in the planning phase, while maintaining the autonomous and self-interested aspects of agents.

We expect that such a coordinated planning algorithm yields even better results than the current plan merging algorithm, since opportunities to cooperate can be better utilized. Both the basic algorithm and the extensions use a resource fact oriented view on the world, and can be combined with most existing planning techniques. Such a general approach to coordinating plans of multiple agents might be used to solve many practical coordination problems, such as hospital scheduling, coordinating the transportation of goods or people, and managing the planning of joint forces on a mission of the UN.

To be able to use the proposed methods on integrating planning and coordination in these situations, still much work need to be done. Firstly, we need an adequate way to *reward* agents that offer services and share resource facts. Secondly, we need to know how to deal with agents that cannot or do not fulfill their contracts. Furthermore, we should test the developed algorithms in more realistic environments and improve them with (maybe even domain-dependent) heuristics. In addition, we need to look at a more dynamic (continual) version of the proposed algorithm where planning, replanning and execution are integrated. Finally, such approaches cannot be used in open multi-agent environments (e.g., the Internet) before a way is devised to deal with different *ontologies* (i.e., what are the resource facts in this domain), and a secure standard for agent communication and negotiation is chosen.

References

1. Kambhampati, S.: Refinement planning as a unifying framework for plan synthesis. AI Magazine **18** (1997) 67–97
2. de Weerdt, M.M., Bos, A., Tonino, J., Witteveen, C.: A resource logic for multi-agent plan merging. Annals of Mathematics and Artificial Intelligence, special issue on Computational Logic in Multi-Agent Systems **37** (2003) 93–130
3. Tonino, J., Bos, A., de Weerdt, M.M., Witteveen, C.: Plan coordination by revision in collective agent-based systems. Artificial Intelligence **142** (2002) 121–145
4. Girard, J.Y.: Linear logic. Theoretical Computer Science **50** (1987) 1–102
5. de Weerdt, M.M.: Plan Merging in Multi-Agent Systems. PhD thesis, Delft Technical University, Delft, The Netherlands (2003)
6. Fikes, R.E., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **5** (1971) 189–208
7. McDermott, D.: PDDL – the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control (1998)
8. Maturana, F., Norrie, D.: A multi-agent coordination architecture for distributed organizational systems. Technical report, Knowledge Science Institute and the Division of Manufacturing Engineering (1995)
9. Masseron, M.: Generating plans in linear logic. Theoretical Computer Science **113** (1993) 349–370
10. Ghallab, M., Laruelle, H.: Representation and control in ixtet, a temporal planner. In: Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94), Menlo Park, CA, (AAAI Press) 61–67
11. Koehler, J.: Planning under resource constraints. In: Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98), John Wiley & Sons (1998) 489–493
12. Wolfman, S.A., Weld, D.S.: The LPSAT engine and its applications to resource planning. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), San Mateo, CA, Morgan Kaufmann Publishers (1999)
13. DesJardins, M.E., Durfee, E.H., Ortiz, C.L., Wolverton, M.J.: A survey of research in distributed, continual planning. AI Magazine **4** (2000) 13–22
14. Decker, K.S., Lesser, V.R.: Generalizing the partial global planning algorithm. International Journal of Intelligent and Cooperative Information Systems **1** (1992) 319–346
15. Durfee, E.H.: Distributed problem solving and planning. In Weiß, G., ed.: A Modern Approach to Distributed Artificial Intelligence. The MIT Press, San Francisco, CA (1999)
16. Kraus, S.: Strategic Negotiation in Multi-Agent Environments. Intelligent Robots and Autonomous Agents. The MIT Press, San Francisco, CA (2001)
17. Foulser, D., Li, M., Yang, Q.: Theory and algorithms for plan merging. Artificial Intelligence Journal **57** (1992) 143–182
18. Yang, Q., Nau, D.S., Hendler, J.: Merging separately generated plans with restricted interactions. Computational Intelligence **8** (1992) 648–676
19. von Martial, F.: Coordinating Plans of Autonomous Agents. Volume 610 of Lecture Notes on Artificial Intelligence. Springer Verlag, Berlin (1992)