Research
Report
TM-98-05

Transportation Scheduling and Simulation in a Railroad Scenario: A Multi-Agent Approach

Jürgen Lind and Klaus Fischer

**German Research Center for AI (DFKI)
Im Stadtwald, B36
66123 Saarbrücken, Germany**

{lind,kuf}@dfki.de

December 1998

Transportation Scheduling and Simulation in a Railroad Scenario: A Multi-Agent Approach

Jürgen Lind and Klaus Fischer

German Research Center for AI (DFKI)
Im Stadtwald, B36
66123 Saarbrücken, Germany
{lind,kuf}@dfki.de

DFKI-TM-98-05

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-).

ISSN 0946-008X

Transportation Scheduling and Simulation in a Railroad Scenario: A Multi-Agent Approach

Jürgen Lind and Klaus Fischer

German Research Center for AI (DFKI)
Im Stadtwald, B36
66123 Saarbrücken, Germany
{lind,kuf}@dfki.de

December 15, 1998

1 Introduction

Efficient transportation – be it of persons or goods – is a key issue in todays industrial world. Because of the immense amount of transportation tasks, it is necessary to use the available resources most effectively. Thus, computer aided – or entirely controlled – scheduling systems are key technologies in the forthcoming century.
Railroad freight haulage as it is performed today is depicted in Figure 1: a company that wants to ship something via railroad to its customers delivers the freight to the local freight center (usually a railroad station) where it is stored until enough freight from other companies has arrived to justify a train to the regional freight center. At the regional freight center, containers from other local freight centers that have the same direction are assembled and sent to the next interregional freight center where another re-assemblance process takes place. The decomposition of the trains is achieved in reverse order.

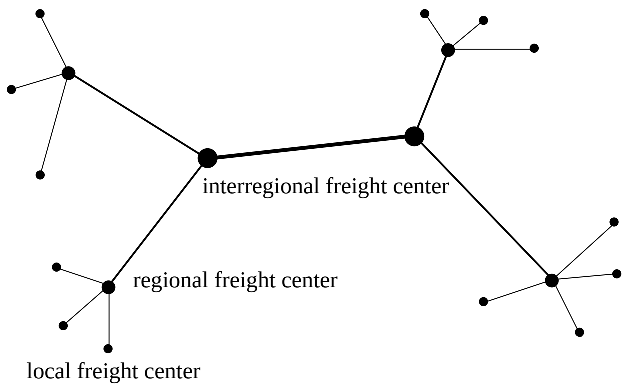

Figure 1: Hierarchical freight haulage

This approach, however, has some serious drawbacks. First of all, the containers of individual customers must wait at the local freight centers until enough freight is delivered to make a train to the next local freight center profitable. Secondly, the re-assemblance of trains in regional and interregional freight centers is a very time

consuming process that introduced additional delays in the producer-to-customer route.
An alternative approach [Krummheuer, 1997] to the classical freight haulage process uses small railroad *transportation modules* (or simply *modules*, e.g. [Windhoff AG, 1996]) instead of complete trains. Whereas a normal train is made up of one railcar and several freight cars, a transportation module is a unit of an engine and a storage area with approximately the size of an individual freight car. When a company wants to deliver some freight to a customer, it orders a transportation module at a local freight center and loads its goods on this module. The module itself is then responsible to find its way through the railroad network. The problem is now, that a location route in a railroad network cannot be used by two independent modules at the same time, i.e. either a route is blocked by a single module or two (or more) modules share a route by hooking together at the beginning of a location route and splitting up afterwards. In order to use the underlying railroad infrastructure most efficiently, the railroad modules should share as many location routes as possible.
The main advantage of this approach is that it avoids a central planning authority that schedules all transportation modules. Instead, each module is responsible to achieve its goal which is to deliver its freight to some destination node in the network. Additionally, each module performs some local optimization of the network throughput by sharing as many location routes with other modules as possible. The local optimization process of all modules eventually leads to a high, though usually not optimal, degree of resource efficiency. Besides this major advantage, a decentralized approach implies less coupling operations during the train composition process, a high degree of customer accessibility and lower costs because of the effective location route usage.
The coordination of the local optimization processes of many thousand modules in a practical scenario is a computationally demanding process and requires a sophisticated algorithmic framework. The major requirements towards a real-world system are listed in the following paragraphs.
First of all, a scheduling system should be capable of dealing with an incomplete problem specification. The classical operations research based approaches assume that the entire problem specification is given at the systems start-up time. Unfortunately, this assumption often does not hold in the real world! Transportation companies usually receive customer tasks over time and not only at the beginning of the planning process. Thus, the company cannot wait until all task specifications are available, then start the planning process in order to find an optimal plan and finally start to execute this plan. Instead, the company must overlap the planning process based on the available data and plan execution monitoring. Incoming tasks must then be integrated in the ongoing planning and execution process.
The second requirement is highly related to the first point and deals with the problem to establish a proper relation between the system and the real world it is supposed to model. A prominent example for the gap between research and reality are order dispatching systems for haulage companies: usually, the respective systems try to compute optimal routes and schedules for the companies trucks, but they fail to monitor the plan execution process and are thus not able to react to unforeseen situations such as mechanical failure of trucks or traffic jams making it impossible to maintain the original schedule. An exception from this shortcomings is the TELETRUCK system presented in [Bürckert et al., 1998] which uses a multi-agent approach for planning and monitoring of transportation tasks.
Finally, a system should not necessarily try to find an optimal solution for a given

problem. Although optimality is a desirable property of a solution, it is often the case that the computational effort to find an optimal solution is too high for realistic problems. Thus, a mechanism that is capable of finding a rather good solution quickly and then improving this solution if sufficient time is available, is an alternative to classical approaches. Algorithms of this type are usually referred to as *anytime-algorithms*. The name stems from the fact, that these algorithms can be aborted at any time and still yield a solution. The quality of the solution simply depends on the resources assigned to the algorithm.
Figure 2 illustrates the basic idea: a solution that is 80% as good as the optimal solution is found within a short computation time. Subsequently, this solution is improved as long as computational resources are available, finally leading to the optimal solution. Obviously, anytime algorithms are highly desirable if computational resources are a critical factor – which is often the case in real world applications.

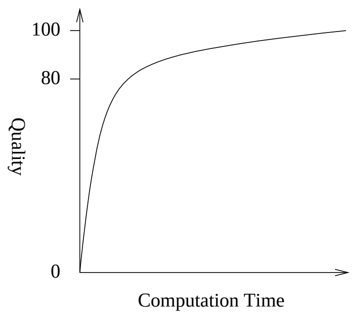

Figure 2: Timeline of an anytime algorithm

This paper is organized as follows: Section 2 describes the problem to be solved and the basic technical constraints to be maintained. We will then introduce some basic mechanisms of agent-based problem solving and explain how these mechanisms are used in our system. Afterwards, some remarks on the planning unit of our agents are presented before we briefly describe the functional unit that monitors the execution of the resulting plans. A conclusion summarizes the paper.

2 Problem Description

The overall goal of the system presented in this paper is to reduce the cost for a given set of transportation tasks in a railroad network. Each task is specified as a tuple consisting of the origin and the destination node, the earliest possible departure time (EDT), the latest allowed arrival time (LAT) and an additional time stamp indicating when the task is announced to the system. Thus, the set of tasks is not fully known to the system at start-up time, new tasks arrive during the planning process and may require a revision of the already assembled plan in order to reduce cost. A typical time profile for incoming tasks is depicted in Figure 3: at start-up time, 2000 tasks are known to the system; during the next 24 hours (=1440 minutes) additional tasks arrive, summing up to a total of 5000 tasks at the end of the day.
A transportation task is served by a transportation module mentioned in Section 1, we assume that each task can be served by a single module, i.e. there is no need to hook two or more modules together to serve a single task. Vice versa, we assume also that a module cannot serve more than one task at a time. All tasks occurring in the system are transportation requests in a railroad network; in the current version

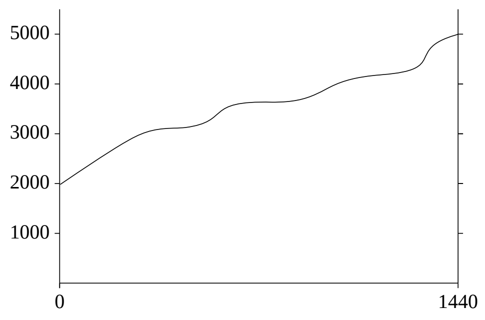

Figure 3: Task arrival time profile

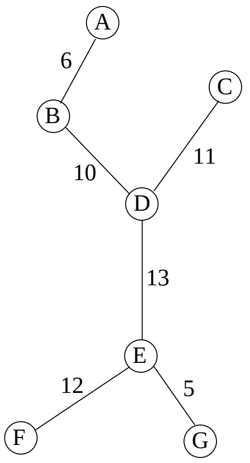

Figure 4: An example railroad network

of the system, we use an abstracted map of the German railroad network with approximately 250 nodes and 350 links. In this paper, however, we use the seven node network shown in Figure 4 to illustrate the basic ideas of our approach.
The net consists of several nodes connected via so-called *location routes*. The numbers on the routes in Figure 4 indicate the distance between two nodes connected via a location route. Whenever a module serves a transportation task, it computes the path from the origin to the destination node with a shortest path algorithm. The module then rents the intermediate location routes for a certain time window from the *net manager*. The time window for each location route is uniquely determined by the earliest departure time and the latest arrival time of the transportation task. When a location route is allocated by a certain module, the route is blocked for other modules during this time interval. In order to reduce route blocking, however, two or more modules can decide to share a particular location route.
Route sharing means, that two or more modules hook together at the beginning of a location route (or of a sequence of consecutive routes) and split up afterwards. Route sharing has two advantages: firstly, it enlarges the average utilization of location routes because it enables more than one module to use a location route at the same time. Secondly, the cost for renting a location route are reduced for an individual module by distributing the full cost among the participating modules.
To illustrate the idea of route sharing, consider the following example with two transportation task $T_1\langle B, F, 10, 60, 0\rangle$ and $T_2\langle C, G, 11, 150, 0\rangle$. Let the cost function for a location route be $c(w, n) = \frac{w}{n}$, i.e. the weight w of the route divided through number n of modules using it simultaneously. Now, if the two modules serving the respective tasks act independently, the transportation costs for the first module are 35 units and 29 units for the second. If, on the other hand, the two modules decide to cooperate and to share the common location route between node D and node E, the transportation cost reduces to 29.5 and 23.5, respectively. The computational problem in conjunction with location route sharing is to identify sets of tasks (i.e. their respective modules) that can share location routes. The limiting factors in this respect are the compatibility of transportation paths and time windows.
In the course of this paper, we will refer to sets of cooperating modules as “unions” where each union is determined by the participating modules. Thus, unions are meta level concepts; a union emerges when at least two modules decide to cooperate

by sharing location routes and it ceases to exists when all modules within the union have completed their respective tasks.
In this section, we have presented the requirements and goals of our system. In the next section, we will introduce some basic ideas from the field of multi-agent systems and demonstrate, how these ideas can be instantiated in this scenario.

3 Agent-Oriented Problem Solving

Agent oriented problem solving is a programming paradigm based on autonomous entities – the *agents*. Besides their autonomy, agents are supposed to react to changes in their environment and to be capable of planning their actions in order to achieve their goals. The field of multi-agent systems considers agent based systems with more than one agent. In these systems, an additional agent capability gains importance – the ability to communicate and cooperate with other agents in the system.
Multi-agent systems are particularly well-suited for the scenario described in the previous section because they allow a very natural mapping from the entities occurring in the scenario to agents. At first glance, it seems very natural to model the transportation modules introduced in the previous section as agents that pursue their local goals. Doing this, however, results in a conceptual break when it comes to modelling the coupling activities that are necessary for location route sharing. Coupling two or more modules together to form a union requires an election process in order to decide which module should represent the resulting union towards the other unions. Additionally, this approach implies a high degree of intra-union communication whenever the union leader wants to integrate a new module in the existing union. To avoid these problems, we have decided to model the unions as the agents in our system. Applying this scheme results in an important simplification of the system design and the resulting implementation. Merging several unions into a single union does no longer require an election a coordination process among the participating modules as they are straightforwardly integrated in another existing union. The roles of the participating unions (either *master* or *slave*) are determined by the negotiation protocol. Whenever a new task is announced to the system, a new union, consisting only of a single module, is created, we will sometimes refer to unions with only one module as *degenerated* unions. The advantage of applying this scheme is that we do not have to differentiate between modules and unions; every active entity in the system is a union and that's it!
Any agent cooperation within a multi-agent system is based on a negotiation process during which the agents try to figure out a deal that results in mutual benefits for them. The negotiation process amongst several agents is steered by so-called *cooperation protocols*. These protocols tell the individual agents what messages to expect from the peer agents or what messages to send to them, respectively. The two protocols used in our system are the *contract-net* [Smith, 1980] protocol described in Section 3.1 and the *simulated trading* [Bachem et al., 1992] protocol explained in Section 3.2.
We have combined these protocols in our scheduling approach to achieve the aforementioned properties (incrementality and anytime) in the following way: an initial solution for the module schedule is obtained by running the contract-net protocol whenever a new task is announced to the system. New tasks are incrementally integrated in the existing scheduling which guarantees, that always a solution for the problem (as far as it is known to the system) exists. However, this solution may be

(and usually is) not optimal. In order to improve the quality of the existing solution, the simulated trading protocol is run on the set of tasks (or the respective modules) currently known to the system. Unfortunately, executing the simulated trading protocol is a computationally expensive operation and so it is executed only periodically – either after a fixed number of new tasks has been added to the existing solution or explicitly triggered by a user request.
In the following sections, we present the contract-net and simulated trading protocols in detail.

3.1 Contract-Net Protocol

In the context of the contract-net protocol [Smith, 1980] depicted in Figure 5, there are two types of participants: one *manager* and a group of *bidders*. The protocol is initiated by the manager which sends a description of the task under consideration to the bidders. Note, that "task" is a not transportation task mentioned earlier but rather some abstract description of a problem to be solved. We will present the instantiation of the general protocol to our scenario later.
After the bidders have received the task description, each of them computes a bid that informs the manager about costs that will be charged if the task is assigned to that particular bidder. After all bidders have submitted their bids to the manager, the manager selects the bid that minimizes his cost and assigns the task to the respective bidder (+) and rejects the offers of the other bidders (-).

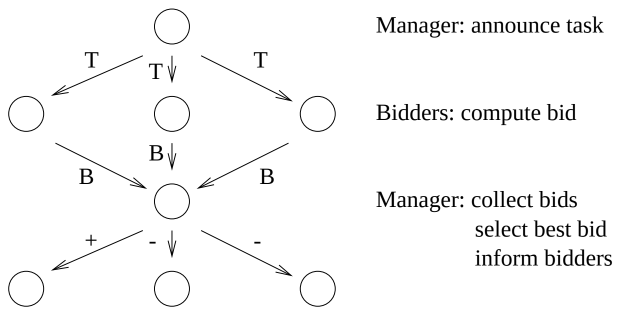

Figure 5: Contract Net Protocol

In our system, this protocol is adopted by creating a new (degenerated) union when a new task is announced to the system. The module in the union plans its path and time constraints for the task and then the parent union initiates the contract-net protocol as the manager and offers the modules plan to the other currently active unions. These unions check if they contain one or more modules that are a potential sharing peers and if this is the case, they offer a sharing commitment to the new union. The new union collects these offers and selects the one that has the largest cost saving potential. It then transfers the module to the winning union and ceases to exist because it does not contain other modules. If no union offers a sharing commitment, the new union remains active as degenerated union.

3.2 Simulated Trading

The simulated trading protocol [Bachem et al., 1992] is an algorithm designed to improve existing solutions, not to construct new solutions from scratch. In our case, the input and the output of the protocol are valid schedules where the cost of the output are always less or equal to the cost of the input. This is trivially true since the output can always be the input if no cheaper schedule exists. However, this property is nonetheless important because it guarantees, that the protocol can be aborted at any time and still yield a valid solution. Furthermore, if the protocol is given enough computation time, it is guaranteed to find the optimal solution. Now, how does this work?

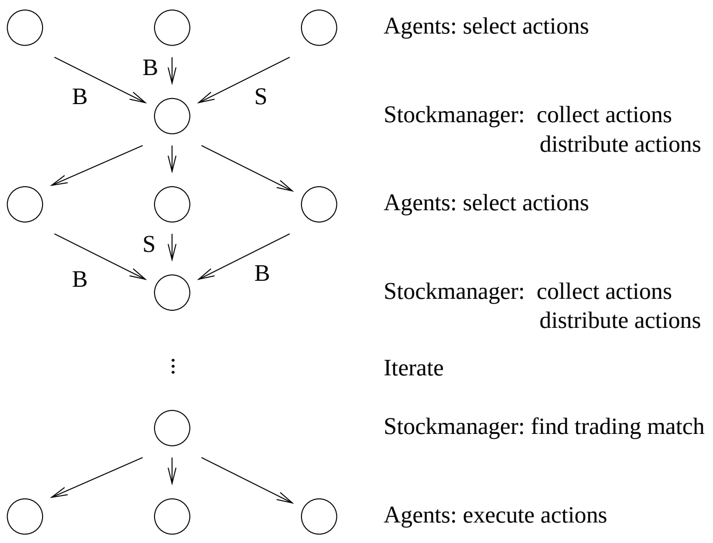

Figure 6: Simulated Trading

The protocol is initiated by a special agent, the *stock manager*. In the course of the protocol execution, the agents (here called *traders*) perform several rounds of hypothetical trading, i.e. the traders either choose to sell some of their goods or to buy something from others. In our context, a sell operation corresponds to removing a module from a union and a buy operation corresponds to integrating a module in a union. Thus, the unions try to optimize their cost by exchanging unprofitable modules with better ones. The decision which module to sell depends on a probability distribution induced by the potential cost reduction if the module was sold. Vice versa, the decision to buy a module offered by another union depends on the potential cost reduction if the module would be integrated in the union.
After the stock manager has collected the hypothetical sell and buy actions, it must find a valid *trading match* in the set of actions. There a several validity requirements for a trading match e. g. there must be no two buy actions on the same sell operation, etc. Finding a trading match is a nontrivial task [Bachem et al., 1992] and accounts for the computational complexity of the simulated trading protocol. If a trading match is found, the stock manager informs the traders which actions must be executed, i.e. which modules must be exchanged.
In this section, we have outlined some basic ideas of agent-oriented problem solving. In the next section, we will present the local planning algorithm of the unions. The plan integration operator developed in there enables a union to find a module schedule with a maximum number of location route sharings.

4 Local planning algorithm of unions

4.1 Notation and Datastructures

First of all, we introduce some basic notations and datastructures used in the subsequent detailed description of the algorithms implemented in our system.
Each of the m modules has a unique identification number $i \in \{1, \ldots, m\}$ and is denoted by M_i. Furthermore, each union has a unique identification number $j \in \{1, \ldots, k\}$ as well. A union of n modules $M_{i_0}, \ldots M_{i_n}$ is written as $U_j^{i_0, \ldots, i_n}$. A plan for an individual modules is a sequence of plan steps where each step consists of a node identifier, a time window and a list of actions to be performed during that plan step. Each of these actions is described by its type, the peer modules, its duration and the time window $\underline{t}, \overline{t}$ during which it must be executed. Time windows used in the module plans have a lower bound ($\underline{t}$) and an upper bound ($\overline{t}$), location routes between any two nodes in the network are written as sequence of node names, e. .g. $\overline{AB}, \overline{BDEF}$,
Each element of a plan be accessed via a dot notation scheme, list elements are accessed via their index number. For example let
[(NodeId:A $\underline{t}$:10 $\overline{t}$:16 actions:nil)
(NodeId:C $\underline{t}$:10 $\overline{t}$:16 actions:[(type:join peers:[M_2 M_3] duration:4 $\underline{t}$:11 $\overline{t}$:15)])
(NodeId:D $\underline{t}$:10 $\overline{t}$:16 actions:nil)]
be the plan of module M. Then we have $M.A.\overline{t} = 16$ or $M.C.actions.1.peers$ =[M_2 M_3].

4.2 Plan Integration operator

In this section, we will develop a plan integration operator for the plans of n distinct modules. Plan integration means, that the operator takes the n plans as input and modifies these plans by inserting *join* and *split* actions such that the resulting n plans imply a maximum degree of location route sharing. The integration operator is used by the unions to decide whether they can integrate a new module in their set of modules or not.
In the course of this section, we will use a rather simple example to illustrate the basic ideas and to give the reader rough feeling for what is going on during the plan integration process. In the example, we have two unions U_1^1 and U_2^2 with modules M_1 and M_2 serving tasks $T_1\langle A, F, 10, 60, 0\rangle$ and $T_2\langle C, G, 9, 50, 0\rangle$ respectively. Basically, plan integration is achieved in five steps:

Find location route matches The first step in the plan integration operation is to find an overlapping sequence of location routes in the plans. In the example, the path of module M_1 is $\overline{ABDEF}$ and the path of module M_2 is $\overline{CDEG}$. Thus, the two paths overlap in $\overline{DE}$. If no overlapping is found, the plan integration process is aborted. For n plans, the general overlap condition is $\forall i \exists j : overlap(M_i, M_j) \neq \emptyset$, i.e. for each module, there must exist at least one sharing peer in the union.

Generate joint actions If the overlap condition holds, the next step in the plan integration process is to generate *join* and *split* actions for the respective plans. These actions are referred to as *joint actions* because they require two modules to coordinate the individual actions. In the example, the two modules join in node D and split in node E, the actions to be inserted are therefore

M_1*.D.actions =[(type:join peers:[*M_2*] ...)]*
M_2*.D.actions =[(type:join peers:[*M_1*] ...)]*
M_1*.E.actions = [(type:split peers:[*M_2*] ...)]*
M_2*.E.actions = [(type:split peers:[*M_1*] ...)]*

Minimize number of joint actions The number of joint actions generated in the previous step is not optimal because the generation process only considers the local context of the action, i.e. only a single step in the plan. Due to prior actions of a module, however, some actions are obsolete and can be eliminated. To illustrate this situation, assume another Module M with task $T\langle B, G, 10, 60, 0\rangle$. Integrating the three modules M_1, M_2 and M, yields three overlappings

1. $(M_1, M_2) = \overline{DE}$
2. $(M_1, M) = \overline{BDE}$
3. $(M_2, M) = \overline{DEG}$

resulting in three bilateral actions pairs

1. (M_1, M_2): *join* at D, *split* at E
2. (M_1, M): *join* at B, *split* at E
3. (M_2, M): *join* at D, *split* at G

This results in a generation of two *join* actions for module M_2 at node D: one with M_1 and one with M. These two actions can be reduced to a single action M_2*.D.actions = [(type:join peers:[*$M_1 M$*] ...)]* because M_1 and M are already linked due to their prior *join* operation at node B. While this appears to be rather trivial in this context, this is not the case in more complex plans where previous *join* and *split* actions must be recursively traced for a large number of modules.

Specify joint action constraints In this step, the time windows of the newly generated actions are specified. The conditions that must hold in this respect are that actions must take place within the time windows of the plan steps of the respective modules and that the joint actions must take place simultaneously. In the example, the resulting constraints are

$$M_1.D.actions.1.\underline{t} = M_2.D.actions.1.\underline{t} = max(M_1.D.\underline{t}, M_2.D.\underline{t})$$
$$M_1.D.actions.1.\overline{t} = M_2.D.actions.1.\overline{t} = min(M_1.D.\overline{t}, M_2.D.\overline{t})$$

Find action schedule In the last step of the plan integration process, the operator must guarantee the existence of a schedule for all actions in each plan step of each module. This means, that all actions occurring in a plan step must be serialized in a way that the action executions do not overlap in time.
For example let M_i*.N.actions = [(type:join peers:[*M_j*] duration:4* $\underline{t}$*:10* $\overline{t}$*:30)*
(type:join peers:[M_k*] duration:4* $\underline{t}$*:14* $\overline{t}$*:34)]*
be the plan step of module M_i at node N. The time windows for the two actions are shown in Figure 7, the task of this step in the plan integration process is to arrange

these actions within their respective time windows such that they do not overlap and that they take place in the time interval given by cutting the action execution time intervals. A valid schedule for the two actions is also shown in Figure 7.

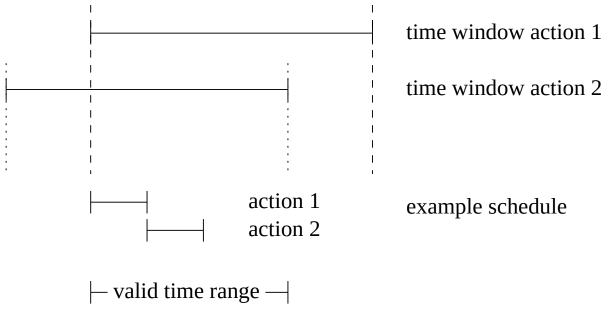

Figure 7: Action Scheduling

Note that, as one can see in Figure 7, usually multiple schedules for the actions exist. However, the time windows of the actions are not committed to a particular schedule because this implies an unnecessary restriction imposed at planning time. At planning time, it is sufficient to know that a schedule exists, concrete commitment to a particular schedule are delayed until execution time giving the participating modules a maximum degree of flexibility. A delay of commitment until it is necessary is called a *least commitment strategy*.
After these steps of the plan integration process have been successfully performed, the output of the plan integration operator are n module plans that satisfy the overlap condition mentioned before. The execution of the resulting plan must now be monitored in order to be able to react to plan deviations and to maintain the integrity constraints imposed by the plans.

5 Plan Execution Monitoring

Each union has a functional unit that monitors the plan execution process of the modules in the union. The plan execution monitor (PEM) is the link between the planning unit and the real world (or a simulation engine that simulates the real world). The PEM controls the usage of location routes and the coupling activities of the modules. To illustrate how plan execution monitoring works, recall the plan of module M_2 from the example in the previous section:
[(NodeId:C $\underline{t}$:9 $\overline{t}$:21 actions:nil)
(NodeId:D $\underline{t}$:20 $\overline{t}$:32 actions:[(type:join peers:[M_1] duration:4 $\underline{t}$:11 $\overline{t}$:15)])
(NodeId:E $\underline{t}$:33 $\overline{t}$:45 actions:[(type:split peers:[M_1] duration:3 $\underline{t}$:11 $\overline{t}$:15)])
(NodeId:G $\underline{t}$:38 $\overline{t}$:50 actions:nil)]
When the module starts to execute this plan at time $t = 9$, it asks the PEM whether the location route from C to D is available as it should be because it was allocated at planning time. However, the route may not be available at execution time due to external reasons, e.g. route blocking due to mechanical failure of another module while using that particular route. It is the task of the PEM to communicate with the net manager in order to check if the route is available or not. If the module is allowed to use the route, it departs from node C. When it arrives at node D, it issues a coupling request to the PEM, indicating that it is waiting for module M_1 to join with it. If module M_1 has already arrived at node D, the coupling action can start if M_1

is not engaged in an ongoing coupling activity with another module. If M_1 has not arrived yet or is currently engaged, the coupling action of M_2 is stalled. If M_1 does not become available within the time interval specified in the *join* action of M_2, M_2 departs from D without coupling with M_1 because otherwise M_2 will miss its scheduled latest arrival time at the goal node. If the coupling activity can be completed as it is scheduled, the two modules depart from D after the coupling is finished. When they arrive at node E, the two modules inform the PEM of their parent union that they want to split and after completing the *split* action they depart from node E for their respective goal nodes.
This example is, again, highly simplified. If more than two modules are involved in *join* or *split* actions additional plan integrity constraints must be satisfied. If, for example, three modules $M_1M_2M_3$ are coupled in this order and M_2 must split from M_1 and M_3, an additional *join* action between M_1 and M_3 must be generated because M_1 and M_3 are supposed to remain coupled. However, no additional *join* action is necessary if the original module order is $M_2M_1M_3$. The decision of whether to generate additional coupling actions or not must be taken by the PEM upon plan execution time, depending on the actual coupling order of the modules.

6 Conclusion

In this paper, we have presented a scheduling system that optimizes the usage of the infrastructure in a railroad scenario. We have used a multi-agent approach for planning and plan execution monitoring in this real-time scenario because of the natural link between the theoretical concepts and the scenario. An incremental planning approach that takes incomplete tasks specifications into account uses the contract net protocol to obtained an initial plan. Then, a post-optimization of the initial solution is achieved by means of the simulated trading protocol. The link between the agents planning unit and the external world is drawn by the plan execution monitoring unit of the agent that dynamically reacts to external events and that can initiate a revision of the actual plan of the agent.
The system presented here is not only applicable in the context of freight haulage, it can as well be used in a passenger transport scenario. Even scheduling of road trucks can be of interest when new technologies in road freight haulage become available [1].

References

[Bachem et al., 1992] Bachem, A., Hochstättler, W., and Malich, M. (1992). Simulated Trading: A New Approach For Solving Vehicle Routing Problems. Technical Report 92.125, Mathematisches Institut der Universität zu Köln.

[Bürckert et al., 1998] Bürckert, H.-J., Fischer, K., and Vierke, G. (1998). Transportation scheduling with Holonic MAS, the TeleTruck approach. In *Proc. PAAM98*.

[Krummheuer, 1997] Krummheuer, E. (1997). Rendezvous und schnelle Sprinter. *Verkehrsforum.* http://www.verkehrsforum.de/magazin/archiv/1_97/1_97_2.html.

[1] "Zaubermittel gegen das Verkehrschaos?", Axel Freyberg, FAZ vom 2.9.1998

[Smith, 1980] Smith, R. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. on Computers*.

[Windhoff AG, 1996] Windhoff AG (1996). CargoSprinter. http://www.windhoff.de http://www.fortunecity.de/anlagen/atlantik/23/sprinter.htm.

TM-98-05
Research Report

Transportation Scheduling and Simulation in a Railroad Scenario: A Multi-Agent Approach

Jürgen Lind and Klaus Fischer

German Research Center for AI (DFKI)
Im Stadtwald, B36
66123 Saarbrücken, Germany
{lind,kuf}@dfki.de