How to keep plan execution healthy ⋆

Femke de Jonge, Nico Roos, and Jaap van den Herik

Universiteit Maastricht, IKAT,
P.O. Box 616, NL-6200, Maastricht
{f.dejonge, roos, herik}@cs.unimaas.nl

Abstract. Developing a conflict-free plan for a multi-agent system in a complex and dynamic environment is a difficult task. Moreover, it is impossible to take into account all possible events that might occur during the execution of the plan. Unexpected events may cause a plan execution to lead to conflicts: we then say that the plan execution is unhealthy. This paper presents a new model that enables agents (1) to control plan-execution health and (2) to regain health when necessary. The agents can utilize the model to predict consequences of occurring disruptions and thus detect unhealthy situations. With the help of the model's predictions, agents can correct the execution of tasks within the plan to regain health. We emphasize that, in the case of bad health, the approach of correcting the plan execution should be applied before relying on the more drastic approach of replanning. The applicability of the presented model is demonstrated by introducing two multi-agent protocols to keep the plan execution healthy. Finally, we investigate the solving capabilities and the efficiency of our method in experiments using randomly generated plans. Our conclusion is that many unhealthy situations can be solved adequately by corrections in the plan execution instead of performing a replanning procedure.

1 Introduction

Plan development and plan execution in complex, dynamic environments are difficult tasks. This explains the tendency to apply intelligent computer programs to support these tasks. Currently, the (initial) plan development in fields such as Air Traffic Control (ATC) is to a large extent performed by planning software. For plan execution, however, such software is not widely available, even though the execution of plans in complex and dynamic environments requires continuous control and adaptation. Our research focusses on employing a multi-agent system for plan-execution control and adaptation. Multi-agent systems seem an obvious means to this end since the plans in environments such as ATC are mainly distributed.

An adequate plan normally satisfies all constraints imposed by its environment and by other plans. Hence, such a plan is conflict free. This is a property that should be kept consequently and persistently during the execution of the plan. We denote a plan execution as healthy, when during the execution of the plan no constraints are violated. A conflict-free plan can have an unhealthy plan execution when unexpected changes in the environment occur. The process of keeping a plan execution healthy can be viewed as a continuous cycle of detecting unhealthy situations and regaining health. Plan-execution

⋆ This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs (the Netherlands). Project DIT5780: Distributed Model Based Diagnosis and Repair

health can be regained by either correcting the execution or changing the plan (i.e., replanning).

In our opinion, corrections within the execution of a plan have three advantages when compared to replanning, viz. (1) they are often easier to accomplish, (2) they are less influential for the environment and the rest of the plan, and (3) especially within domains such as ATC, plan changes are more costly than changes in execution. For instance, gate changes require a large amount of organization as the passengers need to be informed, the engaged ground handling needs to be relocated, and so on. Not surprisingly, within the ATC practice, the first attempt to regain health is always to try and find solutions within the execution of the current plan. Therefore, we emphasize that before applying replanning, agents should try to regain health by correcting the execution of the plan without changing the plan itself.

In summary, the contribution of this paper is that it enables agents to keep the plan execution healthy by applying small corrections within the plan execution. For this purpose, we developed a model that agents can apply (1) to control the health of the plan execution and (2) to find corrections to regain health when necessary.

The outline of the paper is as follows. Section 2 discusses the background of our approach. In section 3, we present our model for plan-execution health control and repair in a multi-agent system. Section 4 provides formal definitions of when a plan execution is healthy, and how plan-execution health can by regained by applying small corrections to the execution. In section 5, we present two protocols that implement our model and in section 6, we applied these protocols in experiments to evaluate the applicability of our model. Section 7 concludes the paper.

2 Background

Planning notions As stated in the introduction, we address the execution of a plan after it is created. So, we assume that a plan is already developed. We view a plan as a partial ordered set of steps. These steps are actions carried out at specific points in the plan, while the actions are instantiations of general operations [1]. The execution of the steps usually has a certain duration and may require resources that have to be shared with other steps of the same plan or of other plans. We assume that a set of constraints describes requirements with respect to shared resources. Within ATC, for instance, we can think of safety constraints and of environmental constraints on noise pollution. Since we consider a multi-agent context, we assume that the plan is distributed over the agents. For example, in the ATC case, we can think of a multi-agent system containing one agent for each aircraft (controlling its plan).

Plan descriptions generally see the steps as atomic parts that make up the plan. Here, we view them as tasks that require several, often reactive, activities of the executing agents. These activities cannot be planned because they depend on the status of the environment (cf. when driving a car from A to B, not every overtaking manoeuvre can be planned in advance). Therefore, the way the plan should be executed is not specified exactly and we may state that the tasks have some boundaries or margins within which the execution may vary. In particular in air traffic, it is common to specify margins for the duration of tasks. For instance, it is the primary responsibility of a pilot or aircraft agent, flying from one waypoint to the next one, to keep the aircraft in the assigned

flight path within an assigned time interval. The activities of adjusting speed, height, and directions are not specified in the plan, but are assumed to be applied within the boundaries. However, the activities contribute to the attempt to follow the plan, i.e., to keep the plan execution within the specified margins such as the flight path and the time interval. So, the unplannable activities within plan execution influence whether the constraints are satisfied or violated. Even when a plan is executed within its margins, it still may happen that constraint violations occur (e.g., due to overtaking manoeuvres when driving a car from A to B).

Related research The main contribution of this paper is the model for plan-execution health control and repair. A fundamental property of such a model is, in our opinion, the ability to represent the current and future states of the plan and its environment. Models that are at the basis of such a property are Discrete Event Systems (DESs) and Markov Decision Models (see, for an overview [2]). A DES models (1) the states that a task (or object) can reach by nodes, and (2) the changes of states by events. Markov Decision Models are a specific type of DES, in which changes of states are probabilistically determined. Our model is partially inspired by these two models.

The TÆMS modelling framework used by [3] is also related to our model, since their plan representation is rather similar. In TÆMS, a plan is represented by task descriptions that express the uncertainty in plan execution. Raja et al. use TÆMS for plan development. In contrast, our research focusses on predicting the states that the tasks will reach, and how to influence this to regain plan execution health.

Our goal to keep plan execution healthy somewhat overlaps the goal of so-called continual planning (for an overview of distributed continual planning, see [4]). In continual planning, the processes of planning and execution are interleaved so as to deal with uncertainties in a dynamic environment. desJardins et al. [4] state that the most preferred planning technique for continual planning is hierarchical plan refinement.In our opinion that plan refinement cannot resolve all possible unhealthy situations, since there is a level within each plan for which its (sub)activities are unplannable (as discussed in section 2).

Running example The following example will be used as a running example throughout the text. Consider a small airport with only one runway used for both arrival and departure. It is a small but busy airport, so plans are tight. Assume that two aircraft agents, agent A and agent B, each have their own (sub)plan, connected through a constraint. A's plan is (1) to taxi from the gate to the runway, and (2) to take off from the runway. B's plan is (1) to arrive at the airport (at the runway), and then (2) to taxi from the runway to its gate. The obvious constraint that connects the two plans is that the runway cannot be used by more than one aircraft at the same time. Therefore, the agents have agreed on a mutual plan in which B lands before A takes off. (It is remarked that the aircraft can pass each other on the taxiway.) Although the plan satisfies the constraint, still, small changes in the execution (mostly caused by external influences) can cause a violation of constraints imposed on the plan execution. For instance, assume that A is a bit early as the aircraft speeded up while taxiing, and B is a bit delayed because of heavy head wind during its flight. Then, they still may not use the (same part of the) runway at the same time, but the two aircraft might pass one another at a close distance. However, a close distance could cause a violation of the safety constraints on the distance that should be kept between the two aircraft.

3 Model description

The model assigns a health state to each task in a plan. This health state may change during the execution of a task caused by unforeseen environmental influences or by activities of the agent executing the task. The external influences of the environment will be modelled as disruption events and the activities of agents, assuming that agents do not deliberately disrupt the execution of tasks, as repair events.

The assignment of health states to tasks will enable us to evaluate the effects of disruption events that have occurred during the execution of tasks. Our first (implicit) assumption of the model is that disruption events are observable. This assumption will not hold in general, especially in environments where not all possible disruption events can be known. However, the model is also useable, with minor adaptations, if agents are able to determine the actual health states of tasks, for instance through plan diagnosis (see, e.g., [5]). A second assumption is that the plans of the individual agents are linear. This assumption is mainly made for the clarity of the presentation of the model. Moreover, it is a common practice in ATC. The model is, however, also applicable if agents have partial ordered plans.

Formally, we model a multi-agent plan as a quadruple consisting of a four sets: $MAP = (A, PD, R, Cst)$. The sets are: (1) a set of agents A, (2) a set of plan descriptions PD, containing one plan description for each agent: $PD = \bigcup_{i=1}^{|A|} PD_i$, (3) a set of common rules R specifying the execution of the plan in general, and (4) a set of constraints Cst between the agents' plans. In the remainder of this section, these four sets will successively be explained in more detail.

We assume that each *agent* in the set A has its own individual plan. All plans are gathered within MAP. There are no other plans outside the model that the agents should consider.

A *plan description* $PD_i = (P_i, \mathcal{S}_i, \mathcal{E}_i, \tau_i, \sigma_i)$ describes how the plan of agent i will be executed. The base of the plan description is the sequence of tasks $P_i = \langle t_{i,0}, t_{i,1}, ..., t_{i,n} \rangle$ which the agent wants to execute in this specific order. We use $\overline{P_i}$ to denote the corresponding set of all tasks in sequence P_i. To describe the health of a task, we have the sets $\mathcal{S}_i$ and $\mathcal{E}_i$ containing for each task a set of states and a set of events respectively. The functions τ_i and σ_i formalize, in combination with the common rules R, the execution of tasks within a plan (we will specify this further on).

During the execution of an agent's plan, a task $t_{i,j}$ is in a certain *state*. Each task has its own set of possible states: $S_{i,j} \in \mathcal{S}_i$. We distinguish three types of state: pending, active, and finish. For each task $t_{i,j}$ holds: $S_{i,j} = S_{i,j}^{pending} \cup S_{i,j}^{active} \cup S_{i,j}^{finish}$. There is only one pending state for each task, this is the state in which the task is awaiting before it is being executed. Thus, $S_{i,j}^{pending} = \{s_{i,j}^{pending}\}$. When the current task (task1) finishes, the next task (task2) will become active by changing from the pending state to an active state (which state that is, depends on the execution of the previous task). Finally, when task2 is completed, task2 changes from an active state to a finish state and consequently, the then subsequent task (task3) is triggered.

Each plan has one start task: $t_{i,0}$, with $S_{i,0} = \{s_{i,0}^{pending}, s_{i,0}^{finish}\}$. The start task has only one pending and one finish state. When the start conditions are fulfilled, this start task will change from the pending to the finish state, which will cause the next task to begin execution (viz. go from the pending state to an active state).

State changes are caused by *events*. Each task $t_{i,j}$ has its own set of events: $E_{i,j} \in \mathcal{E}_i$, with $E_{i,j} = E_{i,j}^{finish} \cup E_{i,j}^{disrupt} \cup E_{i,j}^{repair}$. Finish events are triggered when predefined conditions are fulfilled. Moreover, finish events change tasks from an active to a finish state. Disruption events are externally caused and represent unexpected changes in the execution of a task that might effect the plan-execution health. Finally, the repair events are executed by the agent to regain the plan-execution health when necessary. They represent the corrections in the plan execution. A task's state is the result of the sequence of events during the plan execution, and will be represented by predicate $ts(t_{i,j}, s, E)$, where $t_{i,j} \in \overline{P_i}$ is the task for which event sequence $E = \langle e_1, ..., e_k \rangle$ leads to state $s \in S_{i,j}$. We use the predicate $ats(t, s)$ to denote that task t will achieve state s during the actual plan execution, i.e., the past, current, and expected events lead to s.

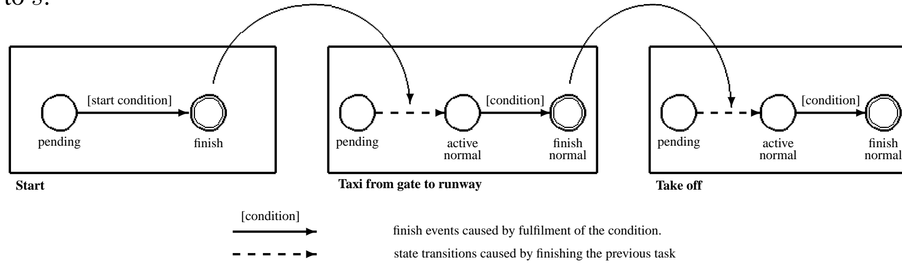

Fig. 1. Normal plan execution of agent A.

Figure 1 illustrates the normal execution of a plan of the departing agent A in our running example. The plan consists of three tasks: $P_1 = \langle t_{1,\text{Start}}, t_{1,\text{Taxi}}, t_{1,\text{Take_off}} \rangle$, and has event sequence $\langle e_{\text{finish_Start}}, e_{\text{finish_Taxi}}, e_{\text{finish_Take_off}} \rangle$. Note that, for reasons of clarity, the figure does not present the whole model, but shows only the occurring states.

As stated above, we formalize the execution of tasks within an agent's plan by the partial functions τ_i and σ_i, and by the set of common rules R from MAP. The partial function τ_i maps a task, its state, and an event to a new state: $\tau_i : \overline{P_i} \times \bigcup_j S_{i,j} \times \bigcup_j E_{i,j} \nrightarrow \bigcup_j S_{i,j}$. (with $\nrightarrow$ denoting a partial mapping). τ_i is defined such that only events in $E_{i,j}$ can change the state of a task $t_{i,j}$ into a new state in $S_{i,j}$. We assume that there is exactly one finish event for each task. A task can, by the definition of τ_i, reach different finish states depending on de previous state the task is in. The partial function σ_i returns the new state in the next task based on the previous task and its finish state: $\sigma_i : \overline{P_i} \times \bigcup_j S_{i,j}^{finish} \nrightarrow \bigcup_j S_{i,j}$.

The *set of common rules* R in MAP consists of three rules. The first rule in R describes how a state transition of a task is caused by an event e_k:

$$(ts(t_{i,j}, s, \langle e_1, ..., e_{k-1} \rangle) \wedge \tau_i(t_{i,j}, s, e_k) = s') \rightarrow ts(t_{i,j}, s', \langle e_1, ..., e_k \rangle) \qquad (1)$$

The second rule in R describes the immediate activation of the next task when the previous task is finished:

$$(ts(t_{i,j}, \text{pending}, \langle e_1, ..., e_{k-1} \rangle) \wedge ts(t_{i,j-1}, s, \langle e_1, ..., e_k \rangle) \wedge \sigma_i(t_{i,j-1}, s) = s') \rightarrow ts(t_{i,j}, s', \langle e_1, ..., e_k \rangle) \quad (2)$$

The third rule in R defines which states will or will not be reached during the plan execution. We use the predicate $Events(\{E_1, ..., E_m\})$ to denote that these sequences of events will occur (a sequence E_i for each P_i).

$$\exists e_1, ..., e_k(Events(\{\langle e_1, ..., e_k, ..., e_n\rangle_i, ...\}) \wedge ts(t_{i,j}, s, \langle e_1, ..., e_k\rangle)) \leftrightarrow ats(t_{i,j}, s) \tag{3}$$

We denote RPD as the set of all instantiations of the rules in R for all plan descriptions PD_i.

The set Cst in MAP is the set of *constraints*, with each constraint composed of predicates $ats(,)$ and logic symbols $\{\vee, \wedge, \neg\}$. Moreover, constraints are only defined on finish states, as they can be viewed as a summary of the execution of a task. An example of a constraint is $cst = \neg(ats(t, s) \wedge ats(t', s')) \vee ats(t'', s'')$, in which s, s', s'' are finish states. The constraints are 'demands' on the plan execution that should be fulfilled. A constraint violation or conflict occurs when the expected execution is inconsistent with a certain constraint. We will assume that when plans are executed normally (only finish events occur), all constraints will hold and the plan execution is in good health. Consequently, the constraint violations are caused by disruption events, and might be solved by repair events to regain the plan-execution health.

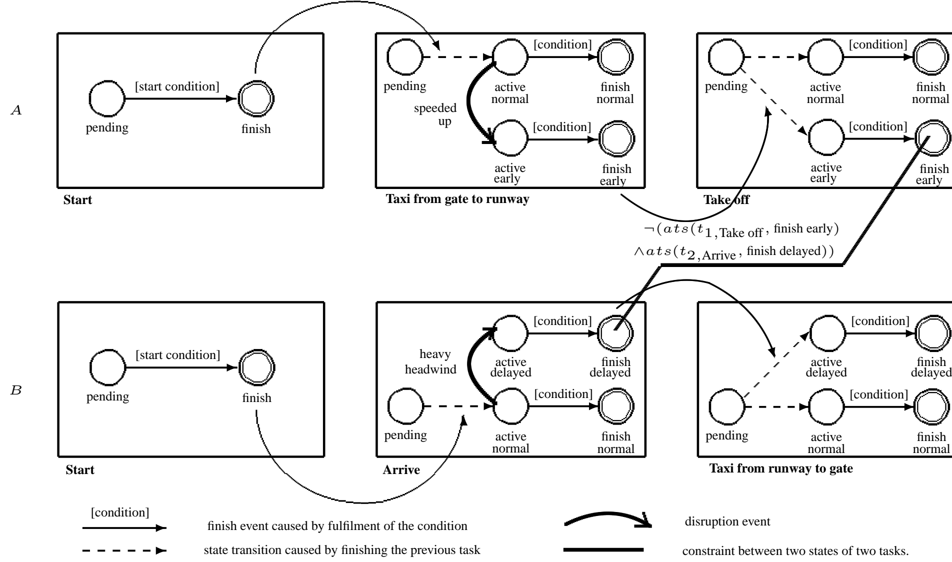

Fig. 2. Disturbed plan executions of agents A and B.

Figure 2 illustrates a disrupted execution of the plans of the departing agent A and arriving agent B in our running example. Both plans consist of three tasks: $P_1 = \langle t_{1,\text{Start}}, t_{1,\text{Taxi}}, t_{1,\text{Take_off}}\rangle$, and $P_2 = \langle t_{2,\text{Start}}, t_{2,\text{Arrive}}, t_{2,\text{Taxi}}\rangle$. The event sequences of the plan execution are $\langle e_{\text{finish_Start}}, e_{\text{Speeded_up}}, e_{\text{finish_Taxi}}, e_{\text{finish_Take_off}}\rangle_1$ and $\langle e_{\text{finish_Start}}, e_{\text{Heavy_head_wind}}, e_{\text{finish_Arrive}}, e_{\text{finish_Taxi}}\rangle_2$. In this setting, the constraint $\neg(ats(t_{1,\text{Take_off}}, \text{finish_early}) \wedge ats(t_{2,\text{Arrive}}, \text{finish_delayed}))$ between the two plans is violated.

In general, we assume that each agent has knowledge (i) of its individual plan description PD_i, (ii) of the common rules R, (iii) of the constraints $Cst_i \subseteq Cst$ that are relevant for its plan, and (iv) of the other agents to whose plans the constraints Cst_i apply.

4 Health and health repair

We assume that an agent notices when disruption events occur during the execution of a plan (for instance through its sensors). Based on the detected disruption events, an agent can construct the sequence of past events (up to and including the current or latest events) in the so-called current event history CEH_i (with $CEH = \bigcup_i CEH_i$). We assume that in the future, from current task $t_{i,j}$ on, no disruption or repair events will occur. Hence, for each task in the remaining plan, one finish event will occur. The resulting sequence of events $FE_i = \langle e_j, e_{j+1}, ..., e_n \rangle$, with $e_h \in E_{finish}$, will be called the future event sequence. The current event history can be combined with the future events sequence into the future event history: $FEH_i = CEH_i \circ FE_i$ (with $\circ$ denoting a concatenation of the two sequences, and $FEH = \bigcup_i FEH_i$). Based on the set of future event histories, FEH, we can define plan-execution health as follows.

Definition 1. *A plan execution is healthy iff* $Events(FEH) \cup RPD \vdash Cst$*.*

When an unhealthy plan execution has been detected, the agents should correct the execution of the plans such that no constraint violations will occur in the future and the plan-execution health is restored. To achieve this, each agent can insert repair events in the future event history in order to create new state paths in its plan execution.

A plan-execution health repair FER is a set of event sequences containing all future event sequences with some repair events inserted, in such a way that by applying FER, all constraints hold again.

Definition 2. *A plan-execution health repair FER is a set of sequences* $FER = FE \Cup RE$ *where* RE *is a minimal subset of* E_{repair} *s.t.* $Events(CEH \circ FER) \cup RPD \vdash Cst$*.*

We use $FER = FE \Cup RE$ to denote that the events in RE are placed at specified places within the sequences collected in FE. Note that for the same FE and RE different sets $FER = FE \Cup RE$ are possible, depending on the placement of the repair events in the sequences in FE. With a minimal RE we limit the subsets of RE to those which have no subset that will construct a plan-execution health repair as well.

Note that computing a plan-execution health repair corresponds to applying abduction. Without proof, we state that definition 2 is equivalent to a FER such that $Events(CEH \circ FER) \cup RPD \cup Cst \not\vdash \bot$ holds. This corresponds to applying consistency checks. Since both abduction and consistency-check problems are known to be NP-hard, in general, plan-execution health repair is NP-hard as well.

In our example, A can apply an event e_{Wait} during the taxi task, which changes the state of task $t_{1,\text{Taxi}}$ from 'active_early' to 'active_normal', and subsequently the state of task $t_{1,\text{Taxi}}$ to 'active_normal'. This correction of plan execution resolves the constraint violation. Therefore, an example of a plan execution health repair is $FER = \{\langle e_{\text{fin_Start}}, e_{\text{Speeded_up}}, e_{\text{fin_Taxi}}, e_{\text{Wait}}, e_{\text{fin_Take_off}} \rangle_1, \langle e_{\text{fin_Start}}, e_{\text{Head_wind}}, e_{\text{fin_Arrive}}, e_{\text{fin_Taxi}} \rangle_2\}$.

5 Two protocols

Health control During the execution of a plan, agents control its development to detect unhealthy states (conflicts) as follows. Based on the detected disruption events and the expected future events, the agents construct a future event history. Using the future event history, agents are able to predict which states will be reached in the future.

If these expected states are part of a possible constraint violation, the agents communicate the new values to other agents that participate in this constraint. This way, the agents individually have sufficient information to determine whether a constraint will be violated and an unhealthy plan execution is reached. The corresponding protocol for health control is presented below. When one or more conflicts are detected, i.e., when the plan-execution health is disturbed, the protocol for finding repair events to restore plan-execution health is activated.

```
Health control protocol of agent i
    while executing plan
        if disruption event occurs then
            determine expected future states;
            send message STATE_CHANGE to related agents;
        if message STATE_CHANGE received then update view on other agent's states;
        check for conflicts;
        if conflict detected then agent 0 start health repair protocol;
```

Health repair The protocol for health repair is based on a mapping from the problem of finding a plan-execution health repair to a constraint satisfaction problem. Simplified, through assignment of an events path, agents choose a state for each task such that all constraints hold. In this article, we sketch the protocol in broad outlines. For a more detailed description of the underlying algorithm, we refer to [6].

```
Health repair protocol of agent 0
    all agents start consistency subprotocol;
    if consistency succeeded then agent 0 start path assignment subprotocol;
    else health repair failed, no solution possible;
Consistency subprotocol of agent i
    repeat until no domain changes occur anymore
        apply domain reduction, check consistency;
        if domains changed then send message DOMAIN_CHANGE to related agents;
        receive all DOMAIN_CHANGE messages,
        update internal representation;
Path assignment subprotocol of agent i
    while!path assigned && !failed :
        assign a new event path (including repair events);
        if succeeded then
            all agents: start consistency subprotocol;
            if consistency succeeded then
                path assigned;
                if agent i+1 exists then agent i+1 start path assignment subprotocol;
                else all agents apply repairs
        else if i != 0 then failed, agent i-1 start path assignment subprotocol;
        else failed, no solution possible;
```

The principal part of the protocol is the *path assignment subprotocol*, in which agents one by one assign an event path they want to follow during their plan execution (to this end, the existing event path is extended by inserting repair events in the future part of the path). An event path is chosen only if it does not violate constraints given the already chosen paths. When an agent is not able to assign a conflict-free event path, the process backtracks to the previous agent, that should assign a new event path.

The *consistency subprotocol* is applied in between path assignments to increase efficiency. In this subprotocol, two steps are repeated. (1) Agents (with no path assignment yet) propagate all possible event paths to verify which states are still reachable given the current assignments. By removing the unreachable states, the agents achieve domain reduction (and thus search space reduction). (2) Changed domains are communicated to through constraints related agents, which, based on this new knowledge, apply domain reduction (step 1). The consistency subprotocol finishes when no domains change anymore and subsequently a state of consistency with maximal domain reduction is achieved. When during the consistency subprotocol a domain becomes empty, no solution is possible given the current path assignments. Consequently, the assignment subprotocol should backtrack, or when backtracking is not possible, the protocol fails.

6 Experiments

As stated in the introduction, the goal of the experiments is to gain insight into which unhealthy situations are suitable for our approach of correcting plan execution. Moreover, we would like to test the efficiency of the proposed protocols with respect to the communication overhead. For these two purposes, the protocols presented in the previous section have been implemented and tested with randomly generated plans. During the experiments, the complexity of the problem of finding repair events has been varied by altering two constraint parameters: (i) the percentage of constraints on the variables (or tasks), p1, and (ii) the percentage of value combinations that are allowed within a constraint between the variables, p2. The performance of the protocols is measured by the number of messages on state or domain changes.

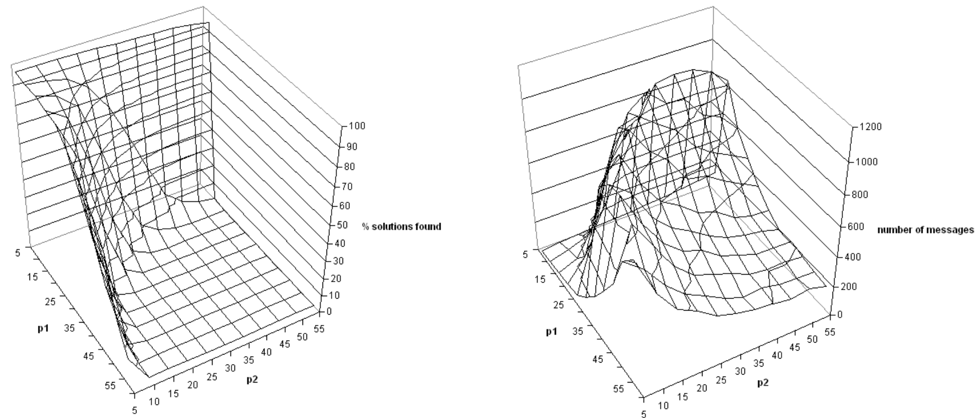

Fig. 3. Results of experiment.

In each experiment, a random plan is generated. Subsequently, an initial value assignment is made based on the expectations of a normal plan development. Then, a number of randomly generated disruption events are executed, which causes state changes. Based on the current and expected future states, the agents detect constraint violations. When an unhealthy plan execution is detected, the agents start the repair protocol to regain plan-execution health.

Figure 3 illustrates typical results of our experiments. The figure shows for a series of settings of constraint parameters ($10 < p1 < 100$ and $10 < p2 < 100$), the average

percentage of problems solved and the average number of messages on domain changes that were sent during the plan-execution health repair protocol. The other parameter settings for these specific experiments are: # agents = 5; # tasks per agent = 5; # states per task = 5; # tasks per constraint = 2; # possible repair events per state = 2; # executed disruption events = 10; # runs per constraint-parameter setting = 1000.

The results show that problems with high constraint density are unsolvable with health repair, as was to be expected since increasing the constraint density causes a decrease in the solution space. Given the settings of the experiments described, the phase transition from solvable to unsolvable problems lies roughly around the boundary $p1 + p2 = 100$. The ridge in the bottom figure shows that problems situated at the phase transition need the largest amount of messages.

7 Conclusion and future research

In this paper, we presented a model that enables agents to maintain plan-execution health. With help of the predicting capabilities of the model, agents can control the plan-execution health and regain health by correcting the plan execution. The protocols for health control and health repair together with their implementations demonstrate the applicability of the model in a multi-agent system. From the experiments we may conclude that a substantial proportion of unhealthy situations are solvable by small corrections in plan execution with a reasonable amount of communicative costs. In view of the observations presented in section 6, we may conclude that health repair is best applicable in problems with constraint density considerably lower than the transition area. Our overall conclusion is that many unhealthy situations can be solved adequately by a well-thought correction in plan execution instead of performing a replanning procedure. So, the benefits of the new model are large.

There are three topics we wish to examine in the near future. First, the efficiency of the protocols can be increased to reduce communication overhead. Second, the balance between health repair and replanning can be examined into more detail to gain a better insight into which unhealthy situations should be solved by plan-execution corrections, and which by replanning. Third, the model can be extended to a probabilistic model in which the probabilities that a disruption event will occur in the future are taken into account. This will improve the controlling power of the agents, in which they can anticipate on unhealthy situations in a much earlier stage.

References

1. Ghallab, M., Nau, D., Traverso, P.: Automated planning. Theory and practice. Morgan Kaufmann Publishers (2004)
2. Cassandras, C.G.: Discrete event systems: modeling and performance analysis. Aksen associates series in electrical and computer engineering. Homewood: Irwin (1993)
3. Raja, A., Lesser, V., Wagner, T.: Towards robust agent control in open environments. In: Proceedings of 5th International Conference of Autonomous Agents. (2000)
4. desJardins, M., Durfee, E., C.L. Ortiz, J., Wolverton, M.: A survey of research in distributed, continual planning. AI Magazine **4** (2000) 13–22
5. Witteveen, C., Roos, N., de Weerdt, M., van der Krogt, R.: Diagnosis of single and multi-agent plans. In: AAMAS 2005 (to appear). (2005)
6. de Jonge, F., Roos, N.: Plan-execution health repair in a multi-agent system. In: Proceedings of the 23rd annual workshop of PlanSIG. (2004)