

A Replanning Algorithm for Decision Theoretic Hierarchical Planning: Principles and Empirical Evaluation

Guido Boella guido@di.unito.it

Rossana Damiano rossana@di.unito.it

Dipartimento di Informatica, Universita' di Torino

C.so Svizzera 185, 10149 Torino Italy

Abstract

In this paper, we present a replanning algorithm for a decision-theoretic hierarchical planner, illustrate the experimental methodology we designed to investigate its performance, and provide an evaluation of the algorithm. The methodology relies on an agent-based framework, in which plan failures can emerge from the interplay of the agent and the environment. Given this framework, the performance of the replanning algorithm is compared with the one of planning from scratch the solution to the planning problem by executing experiments in different domains. The empirical evaluation shows the superiority of replanning with respect from planning from scratch. However, the observation of significant differences in the data collected across planning domains confirm the importance of empirical evaluation in practical systems.

1 Introduction

Replanning is a central research issue for a variety of AI applications involving the execution of plans in realistic contexts, ranging from real-time architectures for automation to agent-based and multi-agent systems. Although Nebel and Koehler [Nebel and Koehler, 1993] have shown that some forms of replanning are harder than planning itself, the interest for replanning is motivated by both practical and theoretical considerations. From the practical point of view, in real time systems, the interleaving of deliberation and execution is an effective strategy to tackle non-determinism and incomplete knowledge, but does not eliminate completely the need to replan from time to time [Myers, 1999, Lemai and Ingrand, 2004]. From the theoretical point of view, the stability of commitment in intention remains a key property of agents [Cohen and Levesque, 1990], especially in social contexts. For example, in multi-agent systems, a certain amount of a priori coordination on individual plans cannot be set aside, together with the necessity for the agents to keep to their plans as much as possible during the joint activity. So, efficient and effective replanning techniques are required to guarantee the stability and predictability of the behaviour of agents.

Multiagent systems pose another requirement on planning: agents are often considered as utility maximisers, so rather than accepting a generic solution to a planning problem, they aim at an optimal one - compatibly with their bounded rationality constraints [Boella et al., 2005]. Thus, beside planning [Dearden and Boutilier, 1997, Haddawy and Hanks, 1998, Pryor and Collins, 1996], also replanning should be based on a decision-theoretic perspective.

In this paper, we address the following research questions:

- How to introduce replanning in a decision-theoretic planning algorithm?

- How to evaluate the preferability of a replanning algorithm – when facing plan failures and changes in the environment – with respect to the solution of planning from scratch?

To define a replanning algorithm we start from the planning algorithm of the DRIPS system [Haddawy and Hanks, 1998], that conjugates hierarchical planning with Decision Theory [Luce and Raiffa, 1957]. We propose an agent-based methodology and framework for its empirical evaluation. The evaluation methodology rests on an experimental framework that simulates the occurrence of plan failures as a result of the interaction between an autonomous agent and a non-deterministic environment. The replanning algorithm is compared with the alternative of planning from scratch based on time performance and quality of output plans.

The objective of the evaluation is to show not only that replanning from the current solution has advantages over planning a new solution, but also to understand in which situations – in terms of structure of the plan library, planning problem, dynamics of the environment and failure types – replanning gives the best advantage.

The paper is structured as follows: Section 2 describes the replanning algorithm; the evaluation framework and methodology are described in Section 3. Section 4 reports the experiments performed according to the methodology, whose results are discussed in Section 5. Conclusions end the paper.

2 Extending Decision-Theoretic Hierarchical Planning to Replanning

Hierarchical planning [Sacerdoti, 1977] lends itself to the design of resource-bounded agents, as the levels of abstraction encoded in hierarchical plans can

be directly mapped to the elaboration of partial plans. Partial plans are a key notion in the theory of practical reasoning [Bratman, 1990], since they allow avoiding premature commitment to overdetailed plans. In the context of re-planning, they provide a useful instrument to manipulate commitment at the lower levels of detail, leaving it unmodified at the higher levels.

Decision-theoretic planning allows an agent to make a distinction between the achievement of a goal and the utility of the states in which the goal is achieved. States are represented as sets of attribute-value pairs and the individual preferences of an agent are expressed as functions which map the value of single attributes to real numbers, representing the utility values. By releasing the “all or nothing” goal satisfaction paradigm of classical planning [Blythe, 1999], in decision-theoretic planning the achievement of a goal can be weighted against the costs of achieving it, and goals can be partially satisfied.

2.1 The Planning Algorithm

The representation of the world in DRIPS consists of attribute-value pairs. Attributes can have boolean values as well as continuous or discrete ones (e.g., the consumption of fuel is represented by a continuous quantity). In order to account for the role of uncertain knowledge about the world, each attribute is associated with a probability distribution on its values. Since the attributes are assumed to be independent, this representation is equivalent to a set of alternative worlds in which each attribute has a certain value with a certain probability. Formally, the representation of the current state of the world W_s is a set of pairs $\{A_i, PD_i\}$, where A_i is an attribute and PD_i is a probability distribution on its values:

$$W_s = \{\{A_1, PD_1\}, \dots, \{A_n, PD_n\}\}$$

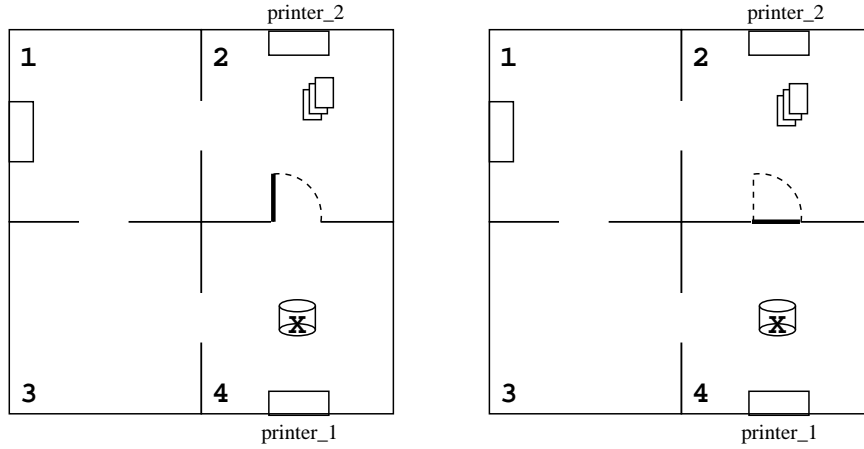


Figure 1: A graphical representation of the office micro-world.

For each attribute A_i , the probability distribution on its values is a set of pairs $\{V_i^j, P_i^j\}$, where V_i^j is one of the possible values, and P_i^j is the probability of that value:

$$PD_i = \{\{V_i^1, P_i^1\}, \dots, \{V_i^n, P_i^n\}\}$$

where $\sum(P_i^1, \dots, P_i^n) = 1$.

The effects of an action are non-deterministic and depend on a set mutually exclusive and exhaustive conditions.

For example, consider the toy domain illustrated in Figure 1. A robot, X , is situated in an office constituted by four rooms, and accomplishes simple tasks, like taking the mail from one room to another. Given this domain, the action go-4-2 defined in Figure 2 states that, when the action is executed while the agent is in room 4 and the door is open, there is one chance out of ten that the agent ends up being in the same room (see the :effs field in the first :cond-eff branch).¹ Notice that there is no proper goal in the action representation: the

¹The representation of effects in DRIPS allows representing the uncertainty associated with complex effects, defined on multiple attributes (see the :effs field in Figure 2), and elementary

```

(action
  :name          go-4-2
  :attributes    (at time fuel open-door)
  :effects       :cond-eff :cond ((and (at = 4)(door = open))
                                   :effs :eff ((at = 2) prob = 9/10
                                               (at = 4) prob = 1/10
                                               (time = time + 15) prob = 1
                                               (fuel = fuel - 0.5) prob = 1))
                                   :prob 1
                :cond-eff :cond ((or (not (at = 4))(door = closed))
                                   :effs :eff ((time = time + 15) 1
                                               (fuel = fuel - 0.5) 1))
                                   :prob 1
  :goal          (at = 2)
  :instantiation nil
  :decomposition nil
)

```

Figure 2: The definition of the primitive action, *go-4-2*, consisting of going from room 4 to room 2 in the office domain (see Section 4.3 for description of the domains).

DRIPS system provides an inter-subjective representation of actions, and relies on the agent’s utility function to represent individual goals. In order to account for the notion of action failure, we augmented the original representation with a field that specifies the action goal (see the `:goal` field).

In DRIPS, a plan library is an action hierarchy that includes two kinds of abstraction relations, as described by [Haddawy and Suwandi, 1994] and [Haddawy and Hanks, 1998]. The *sequential abstraction* relation is a task decomposition relation: an action type of this kind (*complex* action type) is a macro-operator that the planner can replace by a sequence of action types (`:decomposition` field). The *specification relation* describes how an *abstract* action type specializes into more specific action types (`:instantiation` field). A *primitive* action is a directly executable action.

The input to the planner is a plan that contains only the root of the action hierarchy; this plan is recursively refined by substituting complex actions with effects (defined on individual attributes, see the `:eff` field), by posing separate probability distributions on the two levels.

their decomposition and abstract actions with the more specific actions they subsume, until a set of primitive plans is obtained, i.e., a set of plans composed of primitive actions. At each cycle, the input to planning algorithm is the result of the refinement performed in the previous cycle, i.e. a set of more detailed plans, which subsume a smaller set of alternatives. Before execution, the utility of a plan is not known, but its *expected utility* can be computed on the set of states obtained by projecting the plan onto the current state of the world. The expected utility of a plan that has not been fully refined (a *partial plan*) is an interval that encompasses the expected utilities of all the alternatives plans it subsumes. Suboptimal plans are plans whose expected utility upper bound is lower than the lower bound of some other plan. In order to restrict the search space and to identify optimal plans, suboptimal plans are pruned at each refinement step.

2.2 The Replanning Algorithm

In classical planning, the validity of a plan is evaluated in terms of success or failure, intended as the truth or falsity of a logical expression. In decision-theoretic planning, the outcome (or expected outcome) of a course of action is mapped to an utility value, whose acceptability cannot be established a priori. The replanning algorithm presented here delegates to the agent model the task of defining the acceptable utility values, and assumes that a threshold is sufficient to identify the acceptable values. In this work, we consider as plan failure the situation in which the expected utility of plan drops below the acceptability threshold of the agent while the plan is being executed.

When a plan fails, it is possible that the current plan is ‘close’ to a similar feasible solution, where “closeness” is represented by the fact that the current plan and a new valid one are subsumed by the same partial plan at

```

function REPLAN (plan, world, threshold)
  FA := FIND-FOCUSED-ACTION (plan);
  candidate-sub-plan := FIND-CANDIDATE-SUB-PLAN (plan, FA);
  new-plan := PARTIALIZE (plan, FA, candidate-sub-plan, world, threshold);
  if IS-PLAN (new-plan)
    then return new-plan;
    else if (plan = plan-root)
      then return false;
    else
      begin
        partial-plan := COLLAPSE-PLAN (plan, candidate-sub-plan);
        REPLAN (partial-plan, world, threshold);
      end
    end if
  end if
end function

```

Figure 3: The main function of the replanning algorithm, REPLAN

some level of abstraction in the plan space. The key idea of the replanning algorithm is to retract a refinement choice operated during planning (*partialization*), and then to restart the refinement process on the partial plan thus obtained, in the attempt to verify if an alternative refinement is acceptable in the current context [Boella and Damiano, 2002a, Boella and Damiano, 2002b, Boella and Damiano, 2003]. After a plan has been partialized, the expected utility of the resulting plan is computed. Since this value encompasses the utility of all its refinements, it is possible to verify if it is a *promising* candidate, i.e., if it encompasses a refinement whose utility is above the acceptability threshold.

The abstraction and the decomposition hierarchies play complementary roles in the replanning algorithm: the abstraction hierarchy allows identifying alternatives to the steps of the failed plan, while the decomposition hierarchy focusses the partialization process on a sub-plan of the current plan, as described below. The partialization process is incremental: if a new valid refinement of the failed plan is not found after the first partialization step, the process is iterated, until either a new valid plan is found or the root of the action hierarchy is reached.

```

function PARTIALIZE (plan, FA, sub-plan, world, threshold)
  prioritized-steps := PRIORITIZE-STEPS (sub-plan);
  for each step in prioritized-steps do
    revision-node := FIND-REVISION-NODE (step);
    candidate-plan := RETRACT-STEP (plan, revision-node);
    if PROMISING (candidate-plan)
      new-plan := REFINE (candidate-plan, world);
      then return new-plan;
    end if
  end for each
  return false;
end function

```

Figure 4: The PARTIALIZE function

The functioning of the algorithm is the following:

1. The starting point of the replanning process (see the *replan* procedure in Figure 3) is the first non-executed step whose preconditions do not hold. This step becomes the initially *focused action* (FA).²
2. FIND-CANDIDATE-SUB-PLAN identifies the sub-plan of the input plan on which it will focus the search for alternatives. In order to do so, it consults the derivation tree of the plan to identify the lowest-level decomposition the FA appears into; then, it marks the steps which appear in this decomposition as *candidate sub-plan*. The rationale is to ensure that only the steps of the plan which are most directly related to the focused step (i.e. that are part of the same, lowest-level decomposition) are potentially affected by the revision process.

The subtree of the action hierarchy dominated by the root of the candidate sub-plan constitutes the *local context* of the FA; the local context is the portion of the action hierarchy that the algorithm will search for an alternative refinement of the input plan (see next step).

²Although the representation of actions does not explicitly report the preconditions of an action, they can be identified based on the action goal.

3. The partialization process (see the `PARTIALIZE` function in Figure 4) is the core of the algorithm: it tries to replace the steps of the candidate sub-plan with alternatives, in an incremental way, until no candidates are left or it succeeds. Each candidate step is replaced with an abstract action, so as to retract a refinement choice operated by the planning algorithm when the plan was created; then, after each retraction, the resulting plan is refined to verify if the retraction has made it possible to bring back the failed plan to a new valid plan. In detail:

- (a) The `PRIORITIZE-STEPS` function establishes the order in which the steps of the candidate sub-plan will be examined for partialization: the FA is examined first; then, the steps on its right (the ones that temporally follow it) are examined; finally, the steps on its left (the ones that precede it) are examined. This order reflects the assumption that it is preferable to modify actions that have not been executed yet.
- (b) The revision cycle is the core of the algorithm: for each step in the candidate sub-plan, the algorithm searches the current *local context* of the action hierarchy for an abstract action that subsumes that step (`FIND-REVISION-NODE`), and marks it as the *revision node*; if more than one abstract actions are found, the lowest level one is selected, in the attempt to limit the extent of the modification performed on the input plan. The step to be revised is then replaced by the revision node in the input plan (`RETRACT-STEP`), obtaining the *candidate plan*. If the plan candidate plan is promising, i.e., it subsumes a valid refinement, it is refined. The `PROMISING` predicate is evaluated by removing from the candidate plan the steps that correspond to executed actions: in this way, the utility evaluation becomes sensitive

to the execution context.

The revision process is incremental: the input to the subsequent revision cycle is the result of the previous revision.

4. If the partialization process has failed to generate a new valid plan, all the actions in the candidate sub-plan are removed from the plan and replaced by the complex action which subsumes them in the action hierarchy (COLLAPSE-PLAN). The plan thus obtained becomes the new input to the replanning algorithm, and the former root of the candidate sub-plan becomes the new FA (see Figure 3). In this way, as the input plan becomes more partial, the context within which alternatives are sought for becomes wider, until the plan eventually collapses onto the root of the action hierarchy and the refinement process is restarted from scratch.

In order to illustrate how the replanning algorithm works, we will resort to a portion of the office domain (see Section 4.3, Figure 1). Consider the situation in which the robot is in room 4 has the goal of getting the mail from room 2 to room 1, but wrongly believes that the door between 4 and 2 is open. In order to satisfy the goal to get the mail from room 2 to room 1, the robot has devised a plan composed of the steps: GO-4-2-door TAKE-MAIL GO-2-1 PUT-MAIL (see Figure 5, left box). After executing the step GO-4-2-door, the robot realizes that it is in room 4 and its plan is not valid anymore: executing the remaining plan steps from there will not bring about the desired state of affairs, i.e. that the mail is in room 1. So, the robot starts replanning:

- The replanning algorithm identifies the *focused action* (FA), the first non-executed action whose preconditions are not satisfied. In this example, TAKE-MAIL is marked as the FA, as it requires the agent to be in the same room as the mail (room 2).

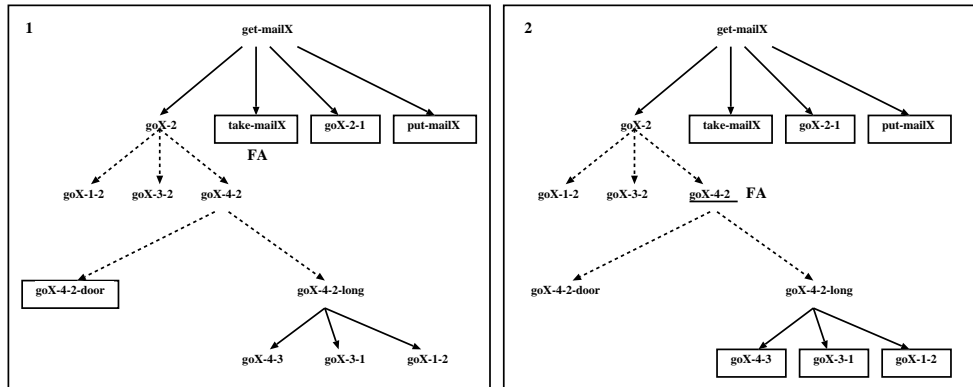


Figure 5: A representation of the steps performed by the replanning algorithm on the action hierarchy given the robot’s plan. The original plan (1); the new plan (2).

- The replanning algorithm identifies the complex action which directly subsumes *FA* in the action hierarchy, *GET-MAIL*, and identifies the candidate sub-plan, composed of the steps *GO-4-2-door*, *GO-2-1* and *PUT-MAIL*. The *local context* of the revision is the portion of the action hierarchy subsumed by *GET-MAIL* (in bold in the right box of Figure 5).
- The *partialization* function examines the right siblings of *GO-4-2-door*, *GO-2-1* and *PUT-MAIL*, in search of an abstract ancestor. None of the two is subsumed by any abstract actions in the local context. So, the only left-side sibling of the *FA* in the derivation tree hierarchy, *GO-4-2-door*, is examined: this action is subsumed in the action hierarchy by the abstract action *GO-4-2*, that is marked as the *revision node*.
- The step *GO-4-2-door* is replaced by the revision node (*GO-4-2*) in the plan, obtaining the candidate partial plan *GO-4-2 TAKE-MAIL GO-2-1 PUT-MAIL*. This plan is *promising*, as it subsumes an alternative refinement of the initial plan which is more appropriate than the failed plan (*GO-4-2-long* does not require the door to be open, as it consists of a

longer path through rooms 3 and 1).

- When the planner refines the new partial plan, it generates the following refinement: GO-4-3 GO-3-1 GO-1-2 TAKE-MAIL GO-2-1 PUT-MAIL.

2.2.1 Considerations and Related Work

The replanning algorithm rests on the assumption that a plan failure can normally be ascribed to a disrupted dependence relation between actions situated in the decomposition the same action, and that the latter is a low-level action in the plan library. If this assumption holds in empirical contexts, we expect that, in most cases, the revision of the failed plan will affect only a sub-part of it; more precisely, it will affect only the steps which are closer to the initially focused step, i.e. the first non executed step whose preconditions do not hold. Confirming this hypothesis is one of the goals of the empirical evaluation illustrated in this work (see Section 3.2).

As it has been remarked by Nebel and Koehler [Nebel and Koehler, 1993], reusing existing plans raises complexity issues. They show that modifying existing plans is advantageous only under some conditions, like replanning, in which it is crucial to retain as many steps as possible of the plan the agent is committed to. According to Liberatore [Liberatore, 1998], *conservative* plan modification leads replanning to be harder than planning itself, while non conservative plan modification - under some assumptions - is always less expensive than planning in computational terms. The replanning algorithm we propose is not conservative: it tends to limit the extent of the modifications to the failed plan, thus limiting the amount of computation required to repair the failed plan and promoting the reuse of plan steps, but it does not guarantee that the modifications to the failed plan are minimal. This choice of generating sub-optimal plans is in line with the notion of bounded optimality expressed by Russell

and Subramanian [Russell and Subramanian, 1995] and the critics expressed by Pollock [Pollock, 2006] to the notion of optimality in decision-theoretic planning (although Pollock’s observations address planning in real-world domains, while our algorithm has been experimented in toy domains of various sizes).

The replanning algorithm consults the decomposition relations encoded in the action hierarchy to focus the revision process on a subpart of the failed plan. By doing so, it exploits the structural information contained in the action hierarchy to reduce the complexity of the search, as recommended by Boutilier et al. [Boutilier et al., 1999]. The algorithm specifically addresses complexity issues in two main ways: the refinement process is restarted on a candidate partial plan only if the latter is promising, and the refinement process benefits from the same pruning heuristic applied by DRIPS during planning. In the worst case, the complexity of the replanning algorithm is the same as the planning algorithm, as the complexity of the latter will simply be increased by a factor of n , where n is the number of new refinements attempted. Most importantly, the replanning algorithm is complete: in the worst case, it reaches the root of the action hierarchy (and so will have explored all the plan space) before finding a solution.

Several approaches have been proposed to replanning and to the partially similar task of plan adaptation. Although all approaches can be ultimately reduced to a form of back-tracking of the search operated by planning algorithm in the plan space, the comparison is not always straightforward, due to the differences among the planning frameworks on which individual proposals are based. In particular, the bulk of work in replanning concerns partial order planning: to cite some, see [Wilkins et al., 1994, Haigh and Veloso, 1996, van der Krogt et al., 2000, van der Krogt and de Weerd, 2005, Lemai and Ingrand, 2003]. An exception is represented by the extension of heuristic search planning to

heuristic search replanning [Gerevini and Serina, 2000, Koenig et al., 2002, Likhachev et al., 2005]. Fox et al. [Fox et al., 2006] have recently proposed a plan repair algorithm for heuristic search planning that is inspired to the notion of plan stability. Our proposal is complementary to these approaches, as it relies on the paradigm of decision-theoretic hierarchical planning, in which dependence relations between actions are compiled out in abstract operators through the use of sequential abstraction.

A problem strictly related to replanning is suspension and resumption of plans. The resumption of a goal after a period of suspension is not necessarily automatic, due to the possible changes in the dynamic environment or in the goals of the agent. The approach of Thangarajah and Harland [Thangarajah and Harland, 2008] is to provide an operational semantics to label suspended tasks in a hierarchical plan and to resume them by possibly finding alternative if they are not executable anymore, similarly to our approach. The notion of plan suspension is accounted for also by the multi-agent architecture for planning and plan execution proposed by Hayashi [Hayashi, 2007]. In this architecture, the features of HTN planning are exploited to circumscribe the extent of replanning to the lower levels of the architecture.

The research on replanning is not necessarily concerned with plan failures: for example, in the seminal work by Kambhampati and Hendler [Kambhampati and Hendler, 1992] and Hanks and Weld [Hanks and Weld, 1995], plan modification is exploited as a planning strategy to adapt existing plans to new problems; in [van der Krogt et al., 2000], replanning is carried out to make the plan of different agents compatible in a multi-agent, distributed planning environment; in the work by Miksch and Seyfang [Miksch and Seyfang, 2000], or Haigh and Veloso [Haigh and Veloso, 1996] replanning is an instrument to account for new user goals, by incorporating them into the system's plans. In these situations, the tradeoff between costs

and benefits of replanning is different from the standard plan failure situation, making a proper comparison impossible.

3 An Agent-Based Evaluation Framework for Replanning

The framework for the empirical evaluation of the performance of the replanning algorithm is based on a utility-based agent architecture, presented in [Damiano, 2002, Boella and Damiano, 2002a, Boella and Damiano, 2002b]. The framework non-deterministically generates a planning problem, then simulates the execution of the plan devised by the agent. When a plan failure occurs, the alternatives of replanning and planning from scratch are attempted and evaluated based on their time performance and the quality of output plans. Since no other replanning algorithms are available for decision-theoretic, hierarchical planning, we resorted to planning from scratch as a benchmark to evaluate the performance of the replanning algorithm. Preliminary work is reported in [Boella and Damiano, 2003, Boella and Damiano, 2004].

The agent is situated in a dynamic environment, i.e. the world can change independently of the agent's actions, and actions can have non-deterministic effects. The framework does not assume full observability of the world, thus a perfect correspondence between the actual state of the environment and the agent's representation of it is not guaranteed. Plan failures arise either from the agent's incorrect knowledge of the world, or from execution failures; the non-determinism of the framework is also an independent source of failures. In this paper, we do not consider instead the problem of changes in the mental states of the replanning agents (see, e.g., [Thangarajah and Harland, 2008]).

While the methodology for comparing the performance of planners described

in [Long and Fox, 2003] involves elaborating a number of planning problems in several domains, the goal of the methodology we propose is somehow different: the focus is not on the formulation of the planning problems; rather, our proposal focusses on the definition of the experimental framework for the automated generation of the plan failures on which different replanning strategies are tested. Since it is agent-based, this framework provides a principled way to address the difficulties observed by Howe and Dahlman [Howe and Dahlman, 2002] in relation with the sampling of planning problems in planner comparison.

A similar evaluation scenario to compare planning from scratch and plan repair is described in the work by Fox et al. [Fox et al., 2006] for heuristic search planning; although this work considers also the modification of the agent's goal as a part of the variability of the execution context, it builds on previous work in the evaluation of planners and is not committed to a specific agent architecture.

3.1 The Agent Architecture

The agent is based on a BDI model [Rao and Georgeff, 1991, Georgeff et al., 2000]; its internal state is defined by its beliefs about the current world, its desires, and the intentions (or plans) it has formed in order to achieve a subset of its desires. Since the deliberation component of the agent is provided by the DRIPS planning system, it inherits the ontological commitments made by DRIPS in the representation of the world and the actions (the agent's beliefs), of the plans (the agent's intentions), and the utility function (the agent's desires, concretized into goals and preferences). We refer to [Boella and Damiano, 2002a, Boella and Damiano, 2002b] for a detailed description of the architecture, and to [Boella, 2002, Boella et al., 2004, Boella et al., 2005] for the integration of decision-theory in agent models.

The intentions of the agent are dynamic, and can be modified as a result of re-

deliberation. Following the literature on deliberation [Rao and Georgeff, 1995, Wooldridge and Parsons, 1999], the architecture is structured in two levels: the *object level* has the role of devising plans and executing them, performing sensing actions after each step to update the agent’s representation of the world; the *meta-deliberation level* controls the object level by selecting input goals to the deliberation process and monitoring the success of the plan being executed. Since the agent’s representation of the world is uncertain, and actions are non-deterministic, the expected utility of the initial plan may significantly vary as the execution of the plan proceeds. The meta-deliberation module relies on the agent’s initial expectations about the utility of the plan to detect plan failures: after the execution of each step, the agent computes the expected utility of the remaining steps; if the difference between the initial expected utility and the new one is above a certain threshold (acceptability threshold), this counts as a failure of the current plan, and replanning is performed. The acceptability threshold is obtained by applying a ratio to the higher bound of the initially expected utility interval. This ratio (acceptability ratio), together with the agent’s utility function, forms the *preference structure* of the agent.

3.2 Empirical Evaluation: Methodology

In order to generate replanning problems in an automated way, we arranged two different scenarios in which a plan may fail. Both scenarios simulate, though in a different way, the occurrence of a misalignment between the agent’s representation of the world and the representation maintained in the simulator. In particular, we distinguish misalignments that occur when the execution of one or more plan actions fail (*failure-determined misalignment*) from misalignments that occur in the absence of an execution failure (*execution-determined misalignment*). In the first case, the mismatch between the agent’s expectations

about the plan effects and the actual effects is caused by an execution failure, while, in the second case, the mismatch is determined by the fact that the world changes unexpectedly before or during the execution of a plan (similarly to the distinction between precondition failures and action failures in [Myers, 1999]).³

The relation between plan failure, replanning and planning from scratch is independent of the type of failure. Every time a plan fails, the replanning algorithm is invoked on the failed plan. If the replanning algorithm outputs a new plan, the planner is launched on the failed plan by simulating the world state holding at the time of failure. In this way, the performance of the two strategies - replanning and planning from scratch - are compared *as if* they were carried on in parallel by the same agent in the same situation. Since the replanning algorithm is complete, there is no point in attempting to plan from scratch if no plan has been found by replanning, so planning from scratch is attempted only if the replanning algorithm has succeeded. On the contrary, planning from scratch when replanning has succeeded is not pointless: since the replanning algorithm does not generate optimal plans, planning from scratch may generate a better plan, or employ less time to generate a plan of the same quality.

3.2.1 Experiment Setting

The execution of a set of replanning experiments is performed by the *Experiment Manager*. The input to the Experiment Manager is the specification of an experiment setting; the Experiment Manager automatically generates a planning problem and calls the agent loop on it, then invokes the *Scenario Manager* to generate plan failures according to the scenario specified in the input. When the agent's meta-deliberation component detects a failure, the Experiment Manager

³From the agent's point of view, these two situations are indistinguishable: the agent simply realizes, at a certain point of the plan execution, that executing the remaining plan steps will yield a significantly lower utility than initially expected.

invokes the replanning algorithm to repair the plan. If the replanning algorithm succeeds, the planner is assigned the task to find a new plan given the execution context at the failure time. During the entire process, the Experiment Manager records all the relevant data about the experiment: the initial representation of the world in the agent's beliefs and in the simulator, the execution trace, and the data about replanning and planning from scratch - if the latter has occurred. The specification of an **experiment setting** includes the following elements:

- The **agent definition** (see Section 3.1)
- The **environment definition** (see Section 3.1).
- A **planning problem specification**:
 - A **domain specification**: a plan library (see Section 2.1), and a specification of the world (a set of attributes, each with a range of possible values);
 - An agent **preference structure** (see Section 3.1);
- The **execution scenario specification** according to which the execution of the agent's plan will be carried out (see Section 3.2.2, below).

3.2.2 Defining Experiment Scenarios

Given an experiment setting definition, the Experiment Manager generates the initial representations of the world for the agent and the simulator in a non-deterministic way. At this stage, the two representations are consistent, i.e., the world represented in the simulator is simply one of the possible worlds listed in the agent's probabilistic representation of the world (see Section 2.1). Then, the Experiment Manager invokes the agent loop on the agent's initial representation of the world; the agent devises a plan to achieve the input goal and starts to execute it in the simulated world.

The Scenario Manager forces the occurrence of misalignments between the agent’s representation of the world and the world represented in the simulator by altering the execution mechanism and the agent’s representation of the world. Notice that the occurrence of a misalignment does not necessarily result in a plan failure: a plan failure occurs only if the misalignment affects the plan in a significant way, making its expected utility drop significantly with respect to the agent’s acceptability threshold.

The first scenario, **Execution Failure**, depicts the situation in which the execution of a plan step fails, i.e. it does not result in the intended effects. According to the terminology introduced in Section 3.2, this situation can be classified as a *failure-determined misalignment*. The second scenario, **Incorrect Representation**, reproduces the situation in which the actual world is incorrectly represented in the agent’s beliefs, i.e., it simulates a *representation-determined misalignment*.

In the first scenario, **Execution Failure**, the initial subjective and objective representations of the world coincide, but the effects of the plan steps are non-deterministically altered when they are executed in the simulator. Implementing this scenario involves modifying the action execution mechanism in the simulator so that the effects of actions can be non deterministically altered, i.e., the value of the attributes affected by the execution of the action differs from the value encoded in the action definition. The new value is constrained to the range of the values included in the interval between the expected value of the attribute (encoded in the action definition) and its value in the world state that precedes the action execution. In this way, the failure simulation process mimics the way the execution of an actions fails in the real world: an action may completely fail (the value of the affected attributes remain unchanged, or, in our implementation, it is brought back to its previous value), or it may yield a

partial achievement (the affected attributes take an intermediate value between the initial value and the expected one).

1. The algorithm computes the effects of the action in the current world state, obtaining a probability distribution on a set of outcomes; then, it selects an outcome in a way that accounts for this probability distribution.
2. If V_i^m and V_i^n are the values of attribute A_i before and after the execution of a step, the failure generation algorithm non-deterministically generates a new value V_x , where $n \leq x \leq m$.
3. In the absence of a statistical model of action failure, the number of steps to be altered is determined by the scenario specification, while the choice of the steps to be altered, and the number of attributes affected, are non deterministic.

The second scenario, **Incorrect Beliefs**, depicts the situation in which the agent has incorrect initial beliefs about the world. Executing experiments according to this scenario requires generating a subjective representation of the initial world that differs from the representation of the initial world in the simulator.

1. First, a random number of attributes are non-deterministically selected for alteration.
2. For each attribute, its value is replaced by a new value, drawn from the range of values specified for that attribute in the agent's specification of the world (see Section 3.2.1).

4 Empirical Evaluation

Given the experimental framework described in the previous section, we consider *successful replanning instances* the experiment runs in which the agent’s plan fails during execution and the agent’s attempt to repair it by replanning succeeds. For each experiment setting, we applied the experimental procedure for a number of runs sufficient to obtain a statistically appropriate set of successful runs. Consequently, the overall number of runs required to gather the same number of successful runs may not be same for all experiment settings.

The software developed for this evaluation is written in Allegro Common Lisp and run on a Pentium(R) 4-M (1,8 GHz) under Linux Redhat.

4.1 Comparing Replanning and Planning from Scratch

Before introducing the data on which the comparison is performed, further clarification is needed about the differences between the strategies of replanning and planning from scratch. Although, in the worst case, the replanning algorithm ends up searching the entire plan space from which the original plan has been derived, this is not true in general. At each partialization level, the replanning algorithm tries to modify only a subpart of the failed plan, the candidate subplan, so the search operated by the replanning algorithm is performed is simpler than the standard search operated by the planning algorithm. Moreover, this search is performed in a way that accounts for the execution context, so it tends to re-use executed steps. Reusing executed steps is more advantageous than starting a brand new plan from scratch, and the advantage is reflected by the expected utility.

In order to compare the efficiency and the effectiveness of the strategies of replanning and planning from scratch, the following data are recorded:

- First of all, we record the **replanning time** (RT) and the **planning from**

scratch time (ST), i.e. the time employed by the replanning algorithm and by the planning algorithm to generate the result.

In order to assess the extent to which one strategy is more efficient than the other, for each set of experiments, we compute the mean of the computation times employed by the two strategies and we compare these values by obtaining the **replanning time ratio**. This value indicates how much time is saved (or wasted) by replanning - instead of planning from scratch. Finally, we run tests to assess the statistical significance of data; we assumed that the distribution is normal for all the populations of data, based on the motivation that the replanning problems were generated non-deterministically by the experimental methodology given the initial specifications.

As the replanning algorithm that we propose tends to limit the extent of the plan modification, we expected to find that the replanning algorithm is more efficient than planning from scratch. This expectation is based on the assumption that, in most cases, there is a new valid plan that is similar to the failed one (see Section 2.2), and finding it does not require exploring the entire plan space.

- In order to evaluate which strategy is more *effective*, we record and compare the **expected utility** of the plans generated by replanning and by planning from scratch as a measure of plan quality.⁴

In order to assess to what extent one strategy is better than the other in terms of the quality of output plans, we compute the mean values of the expected utility of the two sets of plans and we compare them, by obtaining the **expected utility ratio**. This value indicates to what extent

⁴As the notion of plan failure incorporated in the experiment framework refers to the higher expected utility of the current plan, we record the higher expected utility of the generated plans.

the quality of the plans generated by replanning is better (or worse) than the plans generated by planning from scratch. Again, we run statistical tests on the data to assess if there is a statistically significant difference between the two.

Concerning plan quality, our expectation was that replanning would generate better plans than planning from the start, since it incorporates a preference for the modifying the steps that have not been executed yet.

4.2 Evaluating the Replanning Algorithm

Beside collecting data for comparing replanning and planning from scratch, we also record a set of specific data about the behavior of the replanning algorithm, in order to verify if the assumptions on which the algorithm is based hold in empirical contexts. The replanning algorithm consults the decomposition relations encoded in the action hierarchy to focus the revision process on a subpart of the failed plan, and gradually increases the scope of the revision, until it reaches the top of the action hierarchy (see Section 2.2). Clearly, the time performance of the replanning algorithm is negatively affected by the number of times it repeats the partialization process, and by the overall number of retractions - and tentative refinements - it performs.

- For each successful replanning instance, we recorded the overall **number of refinements** (NR) it attempted during the partialization process, and the number of **times it collapsed the focussed sub-plan** (NC), i.e., it enlarged the scope of the partialization process by collapsing the focussed sub-plan onto the complex action which constitutes its root in the decomposition hierarchy . We expected that a local revision of the failed plan would be sufficient to restore it in most cases, i.e., that the focus of the revision would be enlarged only a few times and that few refinements

would be necessary to find a new valid plan.

For each set of experiments, we collected some data which characterize the role of the setting in the occurrence of plan failure. Although a statistical assessment of the relation between the experiment setting and the occurrence of plan failures is out of the scope of this work, we think that these data raise some general issues for the management of replanning in real domains.

- For each experiment setting, we measured the **failure rate**, the number of plan failures encountered by the agent in that setting, given the overall set of runs. This value allows making an approximate evaluation of the frequency with which a certain agent would be likely to resort to replanning in that setting. Since plan libraries and initial world specifications differ by the degree of non-determinism, the rate of plan failures was expected to be affected by the domain specification.
- In order to verify the effectiveness of the replanning algorithm in each setting we measured the **relative successful replanning rate**, i.e., the number of times the replanning algorithm outputs a valid plan given the number of plan failures, and the **absolute successful replanning rate**, i.e. the number of successful replanning instances given the number of experiments in the set. In practice, given a planning problem specification and a scenario, the former value tells us how frequently replanning was able to overcome the failures it encountered, while the latter provides a means for evaluating the relevance of successful replanning in an experiment setting.

We expected to find no correlation between these two values: an experimental setting may generate more plan failures than another, but they may be harder to repair than those generated by setting that generated less failures.

4.3 Domains and Settings

In order to investigate the impact of plan failures and the trade-off between planning and replanning in a domain-independent way, the scenarios described above have been applied to three planning problem specifications (see Section 3.2.1), consisting of a domain specification (a plan library and a world specification) and of an agent preference structure. Domains differ along two main dimensions: the complexity of the plan library (measured in terms of maximal and minimal plan length and action hierarchy height), and the complexity of the representation of the world (the number of attributes involved in world definition). For sake of brevity, in the following, we will refer to an experiment setting as a domain-scenario pair.

In order to limit the factors involved in the experimentation, we made some simplifications. Concerning the agent preference structure specification, the acceptability ratio has been set to 0.1 for all settings. Moreover, the maximum number of steps affected by an execution failure in the Execution Failure scenario has been set to the arbitrary number of 2.

The first two domains (domain *A* and domain *B*) concern industrial brewing planning [Haddawy and Doan, 1994] and are characterized by non-deterministic actions. Although these two domains include similar action definitions, or reuse the same action definitions, they differ as they concern the brewing of different beer types: so, the plans they generate differ for the type of actions they include. In practice, given the same action hierarchy, different portions of it are relevant depending on the utility function employed to trade off the resource consumption against the beer quality. For these reasons, we decided to consider them as two different domains. They both contain 34 action types organized along in a hierarchy which includes three decomposition levels and four specification levels, for a total of seven abstraction levels. The initial world is defined by nine

attributes for both domains, and the output plans contain from 4 to 6 steps. The overall number of primitive plans is 1728.

The third domain (domain C) has been designed for this study. It represents an office toy world (a type of domain adopted also by [Dastani et al., 2003]) and concerns the planning of a complex mail delivery task. The plan library of domain C includes 32 operators and is characterized by a lower height than the other two domains (three decomposition levels and three specification levels, yielding six abstraction levels); differently from the other two domains, it contains non deterministic actions. The agent utility function accounts for time and resource consumption in the achievement of the goal. In domain C , the initial world representation is characterized by a lower number of defining attributes with respect to domains A and B (6 attributes); the length of output plans varies between 9 and 12 steps. The overall number of primitive plans is 288.

In order to compare the performance of our replanning algorithm with the strategy of planning from scratch, we applied to each domain the two scenarios presented above (see Section 3.2.2), the Execution Failure scenario and the Incorrect Belief Scenario. For each setting consisting of a domain-scenario pair, we performed as many experiments as necessary to collect a set of 30 instances of successful replanning.⁵

5 Discussion of the Results

In this section, we describe the results of the experiment we performed by applying the methodology presented above, and we discuss the results obtained.

⁵The number of 30 was chosen for it is conventionally considered to be the boundary between “small” and “large” samples [Cohen, 1995].

5.1 Comparing Replanning and Planning from Scratch: the Results

Concerning the *time performance* of the two repair strategies, the replanning algorithm is by and large more efficient than the planning from scratch strategy in all settings (see Figure 6, second column): replanning always employed less time than planning from scratch. The ratio between the mean time employed by the replanning algorithm, and the mean time required to plan from scratch, ranges from 26,43% (domain *C* with Execution Failure scenario) to 0,57% (domain *A* with the Incorrect Beliefs scenario).

In particular, the difference between the two strategies is bigger in more complex plan libraries, like domains *A* and *B*. This can be explained by considering the functioning of the replanning algorithm: since it embodies a preference for local repair, the most complex is the library (i.e., the deepest is the specialization library), the better is the performance of the algorithm, in comparison with the top-down search operated by the planning algorithm on the entire plan library. In fact, while the partial revision performed by the former tends to remain confined to one or two levels of the specialization hierarchy (as discussed below), the latter necessarily deals with all the levels of the action hierarchy.

In order to test the significance of these data from a statistical point of view, for each set of data, we applied the Z-test to the computation time of the two strategies. Since the Z-value is, for all sets, higher than the critical value of 1,644853⁶ with a $P < 0,01$, it is possible to affirm, with an acceptable probability (99%) that the difference between the computation times of the two strategies is not due to chance.

Regarding the *quality of the plans* generated by the two strategies, the data show the superiority of the replanning algorithm, although in a less clear-cut

⁶The test is one tailed since the difference between the replanning time and scratch planning time is always positive.

Experiment setting	Replanning time ratio	Executions where $RT < ST$	Z-test on RT and ST where hypothesized difference = 0 (two-tailed, $p < ,001$, critical $z = 1,959961$)
A Execution Failure	2,86	30	$z=353,44 P(Z \leq z)=0$
A Incorrect Beliefs	0,57	30	$z=244,75 P(Z \leq z)=0$
B Execution Failure	1,55	30	$z=236,14 P(Z \leq z)=0$
B Incorrect Beliefs	0,48	30	$z=163,85 P(Z \leq z)=0$
C Execution Failure	26,43	30	$z=29,33 P(Z \leq z)=0$
C Incorrect Beliefs	18,64	30	$z=34,00 P(Z \leq z)=0$

Figure 6: Computation time. Second column: the ratio between the time required for replanning (RT) and planning from scratch (ST); third column: the number of executions in which planning from scratch takes longer than replanning; fourth column: statistical significance of the difference between the two series of data.

Experiment setting	Expected utility ratio	Executions where $REU > SEU$	Z-test on REU and SEU where hypothesized difference = 0 (two-tailed, $p < ,001$, critical $z = 1,959961$)
A Execution Failure	152,06	28	$z=-4,78 P(Z \leq z)=1,78$
A Incorrect Beliefs	276,45	30	$z=14,38 P(Z \leq z)=0$
B Execution Failure	149,32	25	$z=3 P(Z \leq z)=6$
B Incorrect Beliefs	292	30	$z=14,84 P(Z \leq z)=0$
C Execution Failure	119,59	16	$z=1,48 P(Z \leq z)=0,14$
C Incorrect Beliefs	100,2	12	$z=-0,04 P(Z \leq z)=0,96$

Figure 7: Expected utility of plans. Second column: the ratio between the expected utility of plans obtained by replanning (REU) and planning from scratch (SEU); third column: the number of executions in which the expected utility (EU) of the plans obtained by replanning is higher than the expected utility of the plans obtained by planning from scratch; fourth column: statistical significance of the difference between the two series of data.

way. The expected utility ratio shows the superiority of the replanning algorithm in all settings (see Figure 7, second column), but only in two settings (domains *A* and *B* with the Incorrect Beliefs scenario) the quality of plans generated by replanning is always higher than the quality of the plans generated by planning from scratch (for all the 30 cases constituting the set). In the two settings obtained by associating the Execution Failure scenario with domains *A* and *B*, the quality of the plans generated by replanning is higher in most cases, but non in all cases. By contrast, in both settings with domain *C*, planning from scratch outperforms the replanning algorithm by the quality of plans.

Again, the scenario does not seem to influence the relative quality of the output plans, while the domain seems to play a central role. Action hierarchies that have been optimized to generate plans for batch execution in a well-defined initial states (like domains *A* and *B*) yield a poor performance on initial states in which one or more plan steps have already been executed. On the contrary, the action hierarchy contained in domain *C* has been purposely designed with an eye on replanning: the possibility for actions to have conditional effects has been exploited to account for a wider range of initial states. For this reasons, the expected utility values obtained by planning from scratch in domain *C* (with both scenarios) are comparable if not equal to the expected utility values obtained by the replanning algorithm.

As for the computation time, in order to test the significance of these data from a statistical point of view, we applied the Z-test to the expected utility of the plans generated by the two strategies. Not surprisingly, the Z-test indicates a significant difference only when the difference in expected utility between replanning and planning from scratch is sharp in 30 cases out of 30, (like in domains *A* and *B* with the Incorrect Belief scenario). In the remaining settings, there is no statistically significant difference between the two populations of

expected utility values obtained by replanning and by planning from scratch.

In conclusion, the data collected empirically substantially confirm the expectation that the replanning algorithm generates better plans (i.e. plans with a higher expected utility) than the strategy of replanning from start, although they do not show a complete dominance of replanning over planning from scratch for what concern the quality of output plans, and the difference is not statistically significant in all settings. These findings are in line with the results illustrated by [Fox et al., 2006], where planning from scratch and plan repair are compared in LPG planning. Although the replanning algorithm we tested does not explicitly accounts for the similarity to the failed plan, differently from [Fox et al., 2006], its functioning privileges the generation of plans that are slight variants of the original plan.

It is worth noting that preliminary experiments conducted on a simplified version of domain *C* [Boella and Damiano, 2003] yielded opposite results: the strategy of performing from scratch outperformed the replanning algorithm by time and plan quality, confirming the hypothesis that the structure of the plan library plays a crucial role in determining the advantage of replanning with respect to planning from scratch.

5.2 Evaluating the Replanning Algorithm: the Results

The data concerning the behavior of the replanning algorithm confirm the expectation that a solution is generally found at a local level. The number of times the replanning algorithm enlarged the scope of the revision by *collapsing* the focussed sub plan onto a complex action is very low in all settings (see Figure 8, third column, NC), ranging from 0,1 (domain *A* with Execution Failure Scenario) to 1,2 (domain *C* with Incorrect Beliefs scenario). In the large majority of cases, the scope of the plan revision was enlarged only 0 or 1 times, meaning

Experiment setting	Avg. NR	Avg. NC	Correlation NR / time (Pearson)	Correlation NC / time (Pearson)
A Execution Failure	2,1 (max 4 - min 1)	0,1 (max 1 - min 0)	0,749 (0,000)	-0,145 (0,444)
A Incorrect Beliefs	2,43 (max 3 - min 1)	0,67 (max 1 - min 0)	0,168 (0,374)	-0,072 (0,704)
B Execution Failure	1,93 (max 3 - min 1)	0,03 (max 1 - min 0)	0,537 (0,002)	-0,065 (0,732)
B Incorrect Beliefs	2,4 (max 3 - min 1)	0,67 (max 1 - min 0)	0,379 (0,039)	0,180 (0,342)
C Execution Failure	1,33 (max 4 - min 1)	1,2 (max 4 - min 0)	0,883 (0,000)	0,698 (0,000)
C Incorrect Beliefs	1,43 (max 4 - min 1)	0,6 (max 3 - min 0)	0,909 (0,000)	0,702 (0,000)

Figure 8: Correlation between the behavior of the replanning algorithm and its time performance. Second column: the average number of alternative refinements attempted (number of refinements, NR); third column, the average number of times the scope of the revision was enlarged by collapsing the focussed sub-plan onto a complex action (number of collapses, NC); fourth and fifth columns: Pearson’s correlation between NR and NC and the computation time.

that the replanning algorithm generally repairs the failed plan by operating at a very local level. While in the settings built by using domain *A* and *B* the value of NC is at most 1, a value of 4 has been observed episodically in domain *C* (in only one case), suggesting that the structure of the plan library is a crucial factor in determining the possibility of finding a local solution to plan failures.

The *number of refinements* executed by the replanning algorithm after the retraction of one or more steps is small and does not vary significantly across settings (see Figure 8, second column, NR), ranging from 1,33 (domain *C*, Execution Failure scenario) to 2,44 (domain *B*, Incorrect Beliefs scenario).

In order to test the of correlation between the number of refinements executed by the replanning algorithm and the computation time, we resorted to Pearson’s correlation coefficient (see Figure 8, fourth column, $p \leq 0,005$): a significant correlation was obtained only in three data sets out of six (domain

Experiment setting	Failure rate	Successful replanning	
		%(relative)	% (absolute)
A Execution Failure	57,64	36,14	20,83
A Incorrect Beliefs	38,68	16,57	6,41
B Execution Failure	43,49	20,41	8,88
B Incorrect Beliefs	34,82	22,57	7,85
C Execution Failure	39,58	52,63	20,83
C Incorrect Beliefs	42,86	100	42,86

Figure 9: The replanning rate in each domain-scenario pair and the successful replanning rate, relative and absolute.

A with Execution Failure Scenario and domain C with both scenarios). Concerning the correlation between the value of NC and the computation time (see Figure 8, fifth column), the experiments did not generate a sufficiently large set of data to perform a statistical evaluation. However, the fact itself that, in most runs, the algorithm was able to find a new valid plan by modifying only the initially focussed sub-plan, makes the assessment of this relation somewhat less relevant than we expected prior to the empirical evaluation.

There are considerable differences among experiment settings for what concern the *rate of plan failures* (see Figure 9), confirming the theoretical expectation that the probability of encountering a failure is not the same in all settings (some plan libraries are more tolerant to failures). The failure rate ranges from 34,82% (domain B with Incorrect Beliefs scenario) to 57,64% (domain A with Execution Failure scenario). Neither the scenario specification nor the planning problem specification determine, taken separately from each other, a higher or lower failure rate.

The *successful replanning* rate shows a larger variation across experiment settings, meaning that, in some settings, plan failures are more easily repaired than in other settings: the successful replanning rate ranges from 16,57% (domain A , Incorrect Beliefs scenario) to 100% (domain C , Incorrect Beliefs scenario).

This value does not depend on the failure rate: for instance, the setting where the failure rate is the highest, domain *B* with the Incorrect Beliefs scenario, is not the one in which the success rate of replanning is the highest. The variation of the successful replanning rate in turn affects the absolute successful replanning rate, which ranges from 6,41% (domain *A*, Incorrect Beliefs scenario) to 42,86% (domain *C*, Incorrect Beliefs scenario). The lower complexity of domain *C* seems to affect also the successful replanning rate, that is significantly higher in both settings that include this domain.

6 Conclusions and Future Work

In this paper, we presented an algorithm that extends decision-theoretic hierarchical planning to replanning and we proposed an empirical methodology for its evaluation. The replanning algorithm consults the decomposition relations between actions encoded in the action hierarchy to focus the revision process on a subpart of the failed plan; then, it retracts refinement choices within the focused sub-plan, in search for new, valid refinements. If a new valid plan cannot be found, the scope of the revision is enlarged, and the process is restarted [Boella and Damiano, 2002b, Boella and Damiano, 2002a].

The algorithm is based on the assumption that a solution to the replanning problem can be found in most cases by searching the portion of the space of possible plans which is “local” to the failed one in the plan space. Beside limiting the computational effort required to the agent, this approach is in line with the fact that intentions, according to theories of rational agents cite above, are characterized by stability. The algorithm is complete: eventually, the plan collapses onto the root of the action hierarchy, and planning is attempted from scratch. However, it is not conservative, according to definition provided by [Nebel and Koehler, 1993], i.e. it does not perform minimal modifications to

the failed plan.

The second research issue we pursued in this work is the design and implementation of an empirical methodology for evaluating the performance of the replanning algorithm within an agent-based framework. Starting from an agent specification and a planning problem specification, the methodology consists in defining execution scenarios in which the plan of an agent fails. When a plan fails, planning from scratch is attempted in parallel with replanning, and the performance of the two is compared in terms of computation time and quality of output plans. It is important to notice that this methodology is independent of the replanning strategies compared.

In order to limit the complexity of the experimental framework, in the experiments discussed here, we set some parameters to default values, leaving to future work the task of investigating how the experimental framework affects the performance of the replanning algorithm.

The replanning algorithm was largely faster than replanning from scratch in all settings and in all experiment runs. The quality of the plans obtained by replanning was on average equal or higher than the quality of the plans obtained by planning from scratch, but the superiority of the replanning algorithm was not as clear-cut as for the time performance. This fact, together with the observation that the rate of plan failures and the performance of the replanning algorithm vary across settings, leads to hypothesize a methodology according to which it is essential to assess the impact of plan failures and the effectiveness of different replanning strategies in a certain domain by performing extended simulations before committing to a redeliberation strategy.

7 Acknowledgments

The authors wish to thank Prof. Monica Bucciarelli (Dipartimento di Psicologia, Università di Torino) for her insights into the experimental setting and her advice on use of statistical techniques.

References

- [Blythe, 1999] Blythe, J. (1999). Decision-theoretic planning. *AI Magazine*, 20(2).
- [Boella, 2002] Boella, G. (2002). Decision theoretic planning and the bounded rationality of BDI agents. In *Procs. of GTDT 2002 workshop. Technical report WS-02-06*, pages 1–10, Menlo Park (CA). AAAI Press.
- [Boella and Damiano, 2002a] Boella, G. and Damiano, R. (2002a). An architecture for normative reactive agents. In Kawabara, K. and Lee, J., editors, *Intelligent Agents and Multi-Agent Systems, LNAI 2413*, pages 1–17. Springer.
- [Boella and Damiano, 2002b] Boella, G. and Damiano, R. (2002b). A replanning algorithm for a reactive agent architecture. In Scott, D., editor, *Artificial Intelligence: Methodology, Systems, and Applications, LNCS 2443*, pages 183–192. Springer.
- [Boella and Damiano, 2003] Boella, G. and Damiano, R. (2003). Empirical evaluation of a replanning algorithm. In *ICAPS Workshop on plan execution*, Trento.
- [Boella and Damiano, 2004] Boella, G. and Damiano, R. (2004). A decision-theoretic replanning algorithm. In Milani, A., editor, *Proceedings of the 3d Italian Workshop on Planning and Scheduling*, Perugia.

- [Boella et al., 2004] Boella, G., Hulstijn, J., and van der Torre, L. (2004). Decision-theoretic deliberation under bounded resources. In *Procs. of LOFT'04*, Leipzig.
- [Boella et al., 2005] Boella, G., Hulstijn, J., and van der Torre, L. (2005). Decision-theoretic deliberation in resource bounded self-aware agents. In *Procs. of Commonsense'05*.
- [Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.
- [Bratman, 1990] Bratman, M. (1990). What is intention? In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in communication*, pages 15–32. MIT Press.
- [Cohen, 1995] Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. The Mit Press, Cambridge, Massachusetts.
- [Cohen and Levesque, 1990] Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261.
- [Damiano, 2002] Damiano, R. (2002). *The Role of Norms in Intelligent Reactive Agents*. Ph.d. thesis, Uninversitá di Torino, Torino, Italy.
- [Dastani et al., 2003] Dastani, M., Dignum, V., and Dignum, F. (2003). Role-assignment in open agent societies. In *Procs. of AAMAS'03*, pages 489–496, Melbourne. ACM Press.
- [Dearden and Boutilier, 1997] Dearden, R. and Boutilier, C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1–2):219–283.

- [Fox et al., 2006] Fox, M., Gerevini, A., Long, D., and Serina, I. (2006). Plan stability: Replanning versus plan repair. *Proc. ICAPS*.
- [Georgeff et al., 2000] Georgeff, M., Pell, B., Pollack, M., Tambe, M., and Wooldridge, M. (2000). The belief-desire-intention model of agency. In Miller, J., Singh, M., and Rao, A., editors, *Intelligent Agents V. Proc. of Agent Theories, Architectures, and Languages: 5th International Workshop (ATAL98)*, LNAI, Paris. Springer-Verlag.
- [Gerevini and Serina, 2000] Gerevini, A. and Serina, I. (2000). Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*. AAAI Press.
- [Haddawy and Doan, 1994] Haddawy, P. and Doan, A. (1994). Abstracting probabilistic actions. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*.
- [Haddawy and Hanks, 1998] Haddawy, P. and Hanks, S. (1998). Utility models for goal-directed, decision-theoretic planners. *Computational Intelligence*, 14:392–429.
- [Haddawy and Suwandi, 1994] Haddawy, P. and Suwandi, M. (1994). Decision-theoretic refinement planning using inheritance abstraction. In *Proc. of 2nd Int. Conference on Artificial Intelligence Planning Systems*, pages 266–271, Menlo Park, CA.
- [Haigh and Veloso, 1996] Haigh, K. Z. and Veloso, M. (1996). Interleaving planning and robot execution for asynchronous user requests. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, pages 35–44. AAAI Press, Menlo Park, California.

- [Hanks and Weld, 1995] Hanks, S. and Weld, D. S. (1995). A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360.
- [Hayashi, 2007] Hayashi, H. (2007). Stratified multi-agent htn planning in dynamic environments. *Lecture Notes in Computer Science*, 4496:189–198.
- [Howe and Dahlman, 2002] Howe, A. E. and Dahlman, E. (2002). A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research*, 17:1–33.
- [Kambhampati and Hendler, 1992] Kambhampati, S. and Hendler, J. (1992). A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–238.
- [Koenig et al., 2002] Koenig, S., Furcy, D., and Bauer, C. (2002). Heuristic search-based replanning. In *Proc. of AIPS 02*.
- [Lemai and Ingrand, 2003] Lemai, S. and Ingrand, F. (2003). Interleaving temporal planning and execution: Ixtex-exec. In *Proc. of ICAPS 03 Workshop on Execution*, Trento.
- [Lemai and Ingrand, 2004] Lemai, S. and Ingrand, F. (2004). Interleaving temporal planning and execution in robotics domains. In *Proc. of AAAI 04*.
- [Liberatore, 1998] Liberatore, P. (1998). On non-conservative plan modification. In *Procs. of ECAI 98*. John Wiley and Son ltd.
- [Likhachev et al., 2005] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., and Thrun, S. (2005). Anytime Dynamic A*: An Anytime, Replanning Algorithm. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 262–271.

- [Long and Fox, 2003] Long, D. and Fox, M. (2003). The third international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59.
- [Luce and Raiffa, 1957] Luce, D. and Raiffa, H. (1957). *Games and Decisions*. John Wiley and Sons, Inc.
- [Miksch and Seyfang, 2000] Miksch, S. and Seyfang, A. (2000). Continual planning with time-oriented, skeletal plans. In *Proc. of ECAI 00*.
- [Myers, 1999] Myers, K. L. (1999). Cpef: Continuous planning and execution framework. *AI Magazine*, 20(4):63–69.
- [Nebel and Koehler, 1993] Nebel, B. and Koehler, J. (1993). Plan modification versus plan generation: A complexity-theoretic perspective. In *Proceedings of of the 13th International Joint Conference on Artificial Intelligence*, pages 1436–1441, Chambery, France.
- [Pollock, 2006] Pollock, J. (2006). *Thinking about Acting: Logical Foundations for Rational Decision Making*. Oxford University Press, USA.
- [Pryor and Collins, 1996] Pryor, L. and Collins, G. (1996). Planning for contingencies: a decision-based approach. *Journal of Artificial Intelligence Research*, 4:286–339.
- [Rao and Georgeff, 1995] Rao, A. and Georgeff, M. (1995). BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*.
- [Rao and Georgeff, 1991] Rao, A. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In *Proc. 2th Int. Conf. Principles of Knowledge Representation and Reasoning (KR:91)*, pages 473–484, Cambridge, MA.

- [Russell and Subramanian, 1995] Russell, S. J. and Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 2:575–609.
- [Sacerdoti, 1977] Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. American Elsevier, New York.
- [Thangarajah and Harland, 2008] Thangarajah, J. and Harland, J. (2008). Suspending and resuming tasks in bdi agents. In *Procs. of AAMAS'08*, pages 405–412. ACM Press.
- [van der Krogt et al., 2000] van der Krogt, R., Bos, A., de Weerdt, M., and Witteveen, C. (2000). An algorithm for replanning. In van den Bosch, A. and Weigand, H., editors, *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference BNAIC '00*, pages 21–28, Tilburg.
- [van der Krogt and de Weerdt, 2005] van der Krogt, R. and de Weerdt, M. (2005). Plan repair as an extension of planning. *Proc. of the Int. Conf. on Automated Planning and Scheduling*.
- [Wilkins et al., 1994] Wilkins, D., Myers, K., Lowrance, J., and Wesley, L. (1994). Planning and reacting in uncertain and dynamic environment. *Journal of Experimental and Theoretical AI*, 6:197–227.
- [Wooldridge and Parsons, 1999] Wooldridge, M. and Parsons, S. (1999). Intention reconsideration reconsidered. In Müller, J., Singh, M. P., and Rao, A. S., editors, *Proc. of ATAL-98*, volume 1555, pages 63–80. Springer-Verlag.