Continual Planning and Acting in Dynamic Multiagent Environments

Michael Brenner and Bernhard Nebel

Albert-Ludwigs-Universität
Freiburg, Germany
{brenner, nebel}@informatik.uni-freiburg.de

Abstract. In highly dynamic environments, e.g. multiagent systems, finding optimal action plans is practically impossible since individual agents lack important knowledge at planning time or this knowledge has become obsolete when a plan is executed. It is often more practical in such environments to enable agents to actively extend their knowledge as part of their plans and then revise their decisions in light of these update. In this paper, we describe a new principled approach to Continual Planning, i.e. the integration of Planning, Execution and Monitoring. The algorithm deliberately postpones parts of the planning process to later stages in an agent's plan-act-monitor cycle and automatically determines when to switch back to refining or revising a partly executed plan.
To evaluate our (and others') Continual Planning techniques we have developed a simulation environment where formal MA Planning domains are not only used by planning agents but also as the basis of the simulation model such that agents can not only plan, but execute actions and perceive their environment. Our experiments show that, using continual planning techniques, deliberate action planning can be used efficiently even in complex multiagent environments.

1 Introduction

Agents acting in the real world usually face a highly dynamic and only partially observable environment. Consequently, their beliefs about the current state of the world are limited, uncertain or simply incorrect. Traditionally, AI Planning research has addressed planning under such circumstances in the subfields of *conditional planning* [21, 3] and *probabilistic planning* [5, 18]. Conditional and probabilistic planning algorithms search for plans that succeed not only for a given initial state but under all potential circumstances, a fact that makes the problem computationally hard [18, 21]. Considering the large number of unobservable features and of possible contingencies in dynamic multi-agent environments it is clear that even small-sized problems will be hard to solve in practice by such planners. More importantly, realistic environments usually do not provide agents with a model of possible contingencies, let alone their probabilies, at all.

In this paper, we therefore advocate an alternative planning approach for dynamic environments, namely the integration of Planning, Execution and Monitoring, i.e. Continual Planning. The term is often used to describe *interleaving* of these three aspects of deliberative behavior. We argue, however, that only a more principled, more tightly *integrated* approach to Continual Planning can answer questions like the following: How

can a planner decide which parts of the problems solving process it should postpone? Can agents *plan* their later replanning? How are early plans that include knowledge gathering actions related to their later revisions that achieve the actual goal? How does a planner realize which kind of knowledge gathering is necessary in the first place?

For agents with limited perceptions and knowledge, even continual planning does not help in determining how to get going in the first place when crucial information for finding a plan is unknown. It is therefore necessary to explicitly model the agents' sensing capabilities as part of the planning domain [12, 17, 13, 20]. Agents can then actively seek to extend their knowledge. In contrast to similar approaches in conditional planning, however, we want to let a planner *postpone* the decision of what to do with that knowledge to later planning phases when the perceptions have actually been made. In other words, we want to "hide" a conditional subplan until the agent has enough information to resolve the contingency. For this purpose, we introduce the concept of an *assertion*. During planning, the agent uses assertions like normal actions in order to find a plan for his goals. However, when during execution assertions would actually become executable, they are instead *expanded*, i.e. a new planning phase is triggered in which the information gained during the last execution phase is used to replace the assertion with concrete, executable actions. The algorithm deciding about the inclusion of assertions into plans and about the switching between acting and (re)planning is described in detail in Section 3.

Dynamic systems (in particular: multiagent environments) can be stunningly complex, both as a whole and from the individual agents' perspective. In such systems, ignorance can indeed often become the proverbial bliss: using "lightweight" representations of their domain of action that are extended only when necessary and possible, agents can act efficiently without being overloaded with information and possibilities for action. We will illustrate how our approach to planning with sensing and assertions realizes this concept by allowing agents to extend their planning domain *during* the continual planning cycle. More philosophically, one could say that the approach enables agents to recognize new *affordances* while acting, i.e. to realize new possibilities for action in light of their current goals. A more practical side effect of this work was the development of generic simulation environment for evaluating multiagent planning approaches. It allows to describe "realistic" domain models, their abstractions to planning domains as well as agents that continually plan and act in this environment. We have used this testbed to evaluate some basic forms of collaborative action in MA systems. Furthermore, we hope that others may find it useful for modeling their own scenarios, thereby providing a set of benchmarks for MA planning techniques.

In the remainder of the paper, we first define the planning formalism used, then describe the basic algorithm for planning with assertions. Afterwards we present our environment for the empirical study of continual planning with different agent designs. Finally, we discuss current and future applications of our approach.

2 Planning Framework

In most multiagent systems, agents have limited knowledge of the environment and limited perceptive capabilities. Even if at some point an agent has perfect knowledge of

the world, it cannot be certain of its beliefs at later time points, since actions by other agents may change the world without those changes being perceived by the agent. It is therefore indispensable for planning and acting in MA environments to be able to represent *ignorance* about facts in addition to propositional truth and falsity. To enable this, we move from STRIPS-like propositional planning to the SAS$^+$ formalism [2] that allows non-boolean state variables instead of propositions[1].

For the sake of clarity, we will first define classical sequential planning in the SAS$^+$ formalism. A *planning domain* $\mathcal{D} = (\mathcal{V}, \mathcal{C}, \mathcal{O})$ consists of state variables $\mathcal{V}$, constants $\mathcal{C}$, and operators $\mathcal{O}$[2]. Each variable v from the set of *state variables* $\mathcal{V}$ is associated with a finite *domain* $dom_v \subseteq \mathcal{C}$. The set of constants will usually include the unique values *true* and *false*; a *proposition* is a variable v with $dom_v = \{\mathit{true}, \mathit{false}\}$.

A *partial variable assignment (PVA)* over $\mathcal{V}$ is a function s on some subset of $\mathcal{V}$ such that $s(v) \in dom_v$ wherever $s(v)$ is defined. def_s ($undef_s$) is the set of defined (undefined) variables in s. If a PVA $s(v)$ is defined for all $v \in \mathcal{V}$ then s is called a (complete) *state*. If $s(v)$ is defined (with value x), then the pair (v, x) is called an *assignment* (also written $v \doteq x$). Two PVAs s and s' are called *consistent* if the following holds: if both $s(v)$ and $s'(v)$ are defined, then $s(v) = s'(v)$. In the following, we will use set notation (as known from STRIPS-like planning) in a straightforward way to refer to assignments. For example, we we will denote the completely undefined PVA by $\emptyset$ and also write $(v \doteq x) \in s$ instead of $s(v) = x$. We define the *union* of two *consistent* PVAs s_1 and s_2 as the PVA $s = s_1 \cup s_2$ in which if $s_1(v) = x$ or $s_2(v) = x$, then also $s(v) = x$.

A planning task is a triple (A, I, G) consisting of an action set A, a completely defined initial state I and a (possibly incomplete) goal state G. Actions a are pairs $(pre(a), \mathit{eff}(a)$ where $pre(a)$ and $\mathit{eff}(a)$ are PVAs. An action a is *applicable* in a state s if, whenever $(v \doteq x) \in \mathit{pre}_a$, then also $(v \doteq x) \in s$. Applying an applicable action a in a state s results in state $app(s, a)$ where $(v \doteq x) \in app(s, a)$ iff $(v \doteq x) \in \mathit{eff}_a$ or $[(v \doteq x) \in s$ and $v \in undef_{\mathit{eff}_a}]$. The execution of a sequence of actions can be defined inductively in the usual recursive manner: $res(s, \langle\rangle) := s$, and $res(s, \langle a_1, ..., a_n\rangle) := app(res(s, \langle a_1, ..., a_{n-1}\rangle), a_n)$ if a_n is applicable in $res(s, \langle a_1, ..., a_{n-1}\rangle)$; otherwise $res(s, \langle a_1, ..., a_n\rangle)$ is undefined.

Definition 1. *An action sequence* $P = \langle a_1, ..., a_n\rangle$ *is a* sequential plan *for planning task* (A, I, G) *if* $res(I, P) \supseteq G$.

We will now extend the basic planning formalism in order to represent limited knowledge of the planning agent, sensor models, and replanning conditions (the latter will allow the planning algorithm to switch between planning and execution).

[1] Compiling SAS$^+$ to the STRIPS subset of PDDL is easy which currently enables us to use state-of-the-art PDDL planners for MA planning. However, as shown by Helmert, classical planning can also benefit greatly from using SAS$^+$ [15]. It might thus be more reasonable to use a SAS$^+$ planner in the first place, thereby saving the costs for translating to PDDL and back again. We will start experimenting with Helmert's FDD planner, the winner of the 2004 Planning Competition, as soon as an official release is available.

[2] Throughout this paper, we will assume ground actions, i.e. fully instantiated operators. Our implementation, though, allows to specifiy operator schemas exactly as in PDDL.

To describe limited knowledge of a planning agent, we relax the requirement of the initial state I being completely defined. In particular, this means that a planner can no longer assume that all facts not explicitly stated are false; instead unspecified values of state variables are regarded as *unknown*. For every state variable $v \in \mathcal{V}$ we introduce a new boolean *knowledge variable* k_v. If v is defined (undefined) in I, then $k_v := true$ ($k_v := false$) is added to I. Note that, in this paper, all definitions in a planning task implicitly refer to the knowledge and goals of the planning agent. In another paper, we will show how this model can be further extended to allow the planning agent to reason about the (common) knowledge of multiple agents.

Sensor models are actions $a = (pre(a), \mathit{eff}(a)$ where $\mathit{eff}(a) = \{(k_v \doteq true)\}$. The set of all sensor models is denoted by $A^s \subseteq A$. Intuitively, sensor models describe the circumstances ($pre(a)$) under which the planning agent will perceive the value of a specific state variable v. At planning time, what *exactly* will be perceived during execution is likely to be unknown; however, knowing the conditions for gathering the missing knowledge is the key to proactive information seeking (see below). Again, more complicated sensor models can be defined when reasoning about (joint) perceptions of multiple agents. The model presented here implicitly assumes that all sensors modelled will actually provide data to the planning agent during execution; therefore, the sensor model also assumes a direct effect on the knowledge of planning agent.

To enable an agent to autonomously determine when to switch between planning and execution, we now introduce the concept of *assertions*. Conceptually, assertions describe conditions under which the domain designer guarantees that an agent may achieve certain effects. While in this respect assertions resemble *actions*, they are also similar to planning *goals* because they do not specify how to achieve this effect, but instead describe which information the agent must gather before it can find a way to achieve the effect on its own by replanning. So assertions provide an agent with the means to postpone parts of its planning process, but also force it to change the world in such a way that replanning of the postponed parts becomes possible.

Formally, an assertion is an action a with a distinguished, non-empty set of preconditions $repl(a) \subseteq pre(a)$, called the *replanning conditions*. Preconditions $p \notin repl(a)$ are called the *ordinary preconditions* of a. If $repl(a) = \emptyset$, then a is an *ordinary action*. We denote the set of assertions among the actions A with $A^r \subseteq A$. Assertions become interesting to continual planning through a change in the semantics of plans:

Definition 2. *An action sequence $P = \langle a_1, ..., a_n \rangle$ is an* assertional plan *for the task (A, I, G) if (1) $res(I, P) \supseteq G$, and (2) $repl(a) \not\subseteq I$ for all assertions $a \in P$.*

This definition needs some explanation: Assertions are meant to be used to describe subproblems that cannot be be solved with the information currenlty at hand and therefore must be postponed. If, on the other hand, the missing information becomes available, the assertion shall be *expanded*, i.e. be planned for. Def 2 ensures both the postponement and the expansion of assertions: because of condition (2) the first executed action of a plan can never be an assertion; as soon as the agent has executed the prefix of the plan that achieves the replanning condition, condition (2) is violated and the plan as whole becomes invalid. Failure, however, is interpreted as the trigger for replanning in the continual planning algorithm (cf. Alg. 1). When replanning the agent

is not allowed to use the assertion again because of condition (2), thus it is forced to use the additional information gathered since the original plan was made for coming up with a more precise plan suffix or a new solution.

The multiagent plan representation we use here is simplest one we could imagine because the focus of this paper is on continual planning with assertions in general. The framework will be extended significantly in extended versions of this work. The version presented here can be regarded as a simplification of the model of Boutilier and Brafman [4]. However, in future work we will depart from their approach towards including more complex intentional modalities [14].

Definition 3. *An action a* interferes *with an action b if a affects a state variable assignment that b relies on or affects, too. Formally a interferes with b if there are v, o, o' such that $(v, o) \in \mathit{eff}(a) \wedge (v, o') \in \mathit{pre}(e_2) \cup \mathit{eff}(e_2)$.*

Actions a and b are mutex *if a interferes with b or b interferes with a.*

Definition 4. *An* asynchronous plan *is a pair $P = (A, C)$ where A is a set of actions and $C \subseteq A \times A$ is a set of ordering constraints among the actions. If two unequal action a and b are unordered in P, i.e. $(a, b) \notin C$ and $(b, a) \notin C$, then a and b must not be mutex.*

Note how the definition of mutual exclusitivity resembles its origins, the specification of Distributed Systems: actions that are mutex have a read-write conflict over a shared resource, so they must be prevented to occur concurrently. In this view a MA plan is a very basic form of a distributed algorithm automatically synthesized by one or several agents.

Corollary 1. *Let P_1, P_2 be sequential plans corresponding to different total orderings of an asynchronous plan P. Then $res(I, P_1) = res(I, P_2)$ for all possible states I.*

As a result of Corollary 1, we can define:

Definition 5. *An asynchronous plan P is a solution for a planning task $T = (A, I, G)$ if any total order P_{TO} of P is a solution of T. We define $res(I, P) = res(I, P_{TO}$ for some total order P_{TO} of P.*

Based on these results, we can use sequential planners to find asynchronous multiagent plans, as described in the next section.

3 Continual Planning with Assertions

Algorithm 1 shows the basic algorithm for continuous planning with assertions. It uses calls to a generic sequential planner PLANNER as a subroutine. In order to allow this modularization and still be compliant with the semantics of plans with assertions, the algorithm removes all expandable assertions from the set of possible actions before calling the PLANNER subroutine. However, it may be more convenient to modify the planner (as we have done in our implementation) so that it assures itself that expandable assertions are not included in a plan.

Algorithm 1 Continual planning agent using assertions

```
function CONTINUAL-PLANNING-AGENT(S, G)
  while S ⋡ G do
    if res(S, P) ⋡ G or
                                   repl(a) ⊆ S for any assertion a ∈ P then
      A = 𝒜 \ {a ∈ A^r | repl(a) ⊆ S}
      REMOVEOBSOLETESUFFIXFROM(P)
      P' = PLANNER(A, res(S, P), G)
      P = MAKEASYNCPLAN(CONCAT(TO(P), P'))
    if P = ∅ then
      return "cannot achieve goal G"
    a = REMOVEFIRSTLEVELACTION(P)
    S' = app(S, a)
    EXECUTE(a)
    exp = EXPECTEDPERCEPTIONS(S', A^s)
    perc = GETSENSORDATA()
    S = app(S', perc)
    if perc ⋡ exp then
      S = STATEESTIMATION(S', exp, perc)
  return "goal reached"
```

The first part of Algorithm 1 is the core of continual planning with assertions, whereas the second half shows the execution and how plan monitoring can be used to provide "high-level" input for sensor interpretation and state estimation. We will concentrate on the planning part here. After the agent has determined that its current plan is no longer valid (either it becomes clear that the plan cannot reach the goals or replanning is triggered by an assertion in the plan), it first determines the prefix of the asynchronous plan that is still executable. Of course, this step can be omitted if *plan stability* is not an issue. It is also possible to evaluate both full replanning and repairing the old plan. However, especially in collaborative settings where agents need to reach agreement over their plans, breaking plan stability is costly since it may lead to *asynchronous backtracking*, i.e. plan revision recursively concerning other agents[23]. Since in this paper we deliberately set aside the issue of communication and, consequently, also the sharing of plans, we will not discuss plan stability further. Cf. Section 6 for further discussion of collaboration.

We assume that the generic planner produces sequential plans like most state-of-the-art planners. In order to convert such a plan into an asynchronous one, and in order to combine it with the existing plan stump, the algorithm uses MAKEASYNCPLAN, specified in Algorithm 2. For an analysis of Alg. 2 and its relation to the techniques discussed by Backstrom [1] we must, for lack of space, again refer the reader to the extended paper.

Definition 6. *A state variable v is* written *by a if $(v, o) \in \mathit{eff}(a)$ for some o. v is* read-only *in a if $(v, o) \in pre(a)$ for some o, but v is not written by a. By $written(a)$ [$readOnly(a)$] we denote the set of variables v that are written by [read-only in] a.*

Algorithm 2 Converting TO plans to asynchronous plans

```
function MAKEASYNCPLAN(P_TO)
  initialize P with dummy action root
  readers(v) = ∅, provider(v) = root for all variables v
  for a ∈ P_TO do
    for v ∈ readOnly(a) do
      add a to P as successor of provider(v)
      add a to readers(v)
    for v ∈ written(a) do
      for prev ∈ readers(v) do
        add a to P as successor of prev
      add a to P as successor of provider(v)
      provider(v) = a
      readers(v) = ∅
  return P
```

Theorem 1. *Let P_{TO} be a sequential plan. Then $P = \text{MAKEASYNCPLAN}(P_{TO})$ is an asynchronous plan of which P_{TO} is a total ordering. Thus, $res(P_{TO}) = res(P)$.*

As already mentioned, AP lends itself to particular way of describing and extending domains "on demand". Consider the example of a household robot who, for the first time, is told to clean the dishes. The robot knows that the dishwasher in the kitchen may be used to achieve that goal, but has never operated it yet and does not know how to do so. In other words, the robot knows of some affordance of the dishwasher, but not of concrete actions to realize these. This knowledge can be expressed using the following assertion. Its syntax should be intuitive for readers familiar with PDDL [19]. In particular, since boolean state variables can be expressed propositionally as seen in the example, the STRIPS subset of PDDL is also a subset of the MA planning language.

```
(:action operate_dishwasher
 :parameters ?robot
 :precondition (pos ?robot : kitchen)
 :replan (canOperate ?robot dishwasher)
 :effect (dishesClean))
```

Using this assertion, the robot can come up with an initial plan to achieve its goal (sensor rules and assertions are set in italics):

```
1 move livingroom kitchen
2 senseBrandOf dishwasher
3 downloadManual dishwasher
4 operate dishwasher
```

As demanded by the semantics of assertional plans, the first action in this plan is exectuable. Its purpose is to bring the robot in a position where it can gather more information, namely find out the brand of the dishwasher. In our example, *senseBrandOf* is not only a sensor rule, but also an assertion: it hides possible conditional branches

for finding out the brandname, depending on the information the robot will have once it is in the kitchen. For example, the robot might automatically be provided with the brand information through RFID information sent by the dishwasher or it may have to actively query the dishwasher (which of course is another intelligent household agent). Let us assume, that the former is the case: as soon as the robot starts executing its plan and enters the kitchen, it receives the RFID information that the dishwasher is of some brand X. As shown below, this knowledge is the replanning condition for assertion *downloadManual*.

```
(:action downloadManual ?tool
 :parameters ?robot
 :precondition ()
 :replan (K ?robot (brandOf ?tool))
 :effect (canOperate ?robot ?tool))
```

The assertion will be replaced a number of concrete actions (e.g. connecting to the web site of company X and querying some web service there). What's most interesting here is that, as a result, the robot will not only have a changed belief state, but also extended capabilities, i.e. the planning domain will be extended with actions to operate the dishwasher. In other words, this process can be described as recognition (or even active determination) of the *affordances* of the dishwasher that are pertinent to the robot's task. To conclude the example: since `(canOperate dishwasher)` has been made true by the download action, `operate dishwasher` must now be expanded. Using its newly learned actions, the robot is, hopefully, able to achieve its goal now.

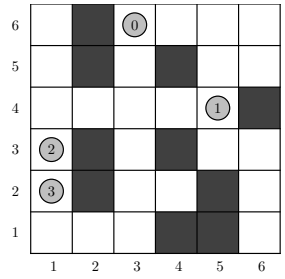
4 Empirical Evaluation

We have developed a simulation environment for Continual Multiagent Planning approaches, called MAPSIM, that allows to easily describe the relation between a formal Planning domain and the "realistic" model that the simulation itself uses to execute actions and to provide perceptions to agents. Thus, planning agents can directly use actions from their plans and "execute" them in the simulation; they make "perceptions" described using the ontology defined by the planning domain, so that they can used as input for plan monitoring or replanning immediately.

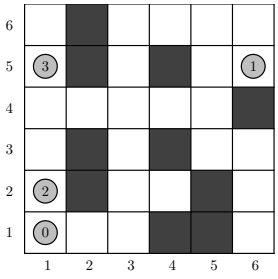
For evaluating our approach, we have devised a simple gridworld domain (similar to the *grid* domain in the Internal Planning Competitions) that allows automatic generation of arbitrary complex tasks for arbitrary numbers of agents[3]. We have created a test suite of 50 random problems varying in the number of agents (2 to 10), the grid size and the number of blocked cells. Fig. 1 shows one such problem from an omniscient perspective, i.e. the initial and goal position for all agents are shown in the same grid. Each agent, implemented as an individual MAPSIM process, knows only about its own goal state and its own perceptions. For a centralized planner with complete knowledge, each problem would be solvable, i.e. there is a sequential or asynchronous plan leading each agent into its goal position.

[3] The basic form of the domain discussed here was chosen for investigating problem scalability; more complex extensions include object transportation tasks in the gridworld as well as collaborative tasks where the set of actions includes *speech acts* between agents.

Fig. 1 A multiagent planning problem in the gridworld



Initial state



Goal state

The purpose of our experiments was to study the success rate of agents planning and acting distributedly under different configurations. We have run the simulation for each problem under 70 different configurations (explained below) for a total of 3500 test runs. In each run, each individual agent received perceptions from the simulator, checked whether its former plan was still valid, replanned if necessary, and executed one of the possible first-level actions. The planning domain contains the single assertion shown below. It describes the fact that before attempting to move to some position an agent needs to know whether it is occupied:

```
(:assertion move
 :parameters (?a - agent ?op ?np - position)
 :replan (K ?a (occupant ?np))
 :precondition (and (occupant ?op : ?a) (connected ?op ?np))
 :effect (and (occupant ?op : empty) (occupant ?np : ?a)))
```

The generic planner used by the agents in our experiments is a modified version of FF [16] that prevents application of assertions enabled in the initial state. The experiments were run on a 1.8 GHz Intel Pentium using 512 MB of its RAM. If after at most 10 minutes all agents had achieved their goals, the run was counted as succesful, otherwise as a failure. Since each run consisted of many planner calls by several agents, individual planner calls were timed out after 10s. (It is instructive to compare these time constraints to the 5 minute limit for a single planner call in the International Planning Competition to see the different constraints for Continuals MA planning!)

	$\mathbf{M}_0$	$\mathbf{M}_5$	$\mathbf{M}_{\text{perm}}$
$\mathbf{S}_1$	37	*62*	54
$\mathbf{S}_2$	84	*88*	83
$\mathbf{S}_{\text{half}}$	96	*99*	90
$\mathbf{S}_{\text{full}}$	100	-	-

Table 1. Memory span vs. sensor range. Entries show success rate in %.

For lack of space, in this first presentation of our approach we will describe only one basic experiment in this setting that confronts two important dimensions of agent design, namely sensing restrictions and memory decay/memory span of agents. When sensing is limited, agents have to rely on their memory for things they cannot currently see. However, when several agents act in the same environment old beliefs may have become obsolete by the time they are used by the planner. In such situations, agents may do better to rely on AP, i.e. treat the information as unknown and *plan to replan* based on information gathering actions. In our experiment, we investigated the relation between sensing limits and the use of AP by providing agents with different rates of *memory decay*. In Tab. 1, $\mathbf{M}_{\text{perm}}$ indicates that agents never "forget" perceptions and treat them as valid as long as they are not replaced by a newer one. M_0 indicates that only the current perceptions are used as initial state for the planner and forgotten immediately afterwards. M_5 indicates a "memory span" of 5 executed actions: all perceptions that were made within the last 5 execution steps are kept as valid, everything before is forgotten, i.e. treated as unreliable. In Tab. 1, $\mathbf{S}_i$ indicates that agents can see all grid-cells in a maximum distance of i cells. $\mathbf{S}_{\text{full}}$ is full observability of the whole grid, and $\mathbf{S}_{\text{half}}$ means that agents could see to a distance of k in a $2k \times 2k$ grid.

With full observability, remembering old perceptions is unnecessary; we have therefore restricted the test to $\mathbf{S}_{\text{full}}/\text{M}_0$ there. It comes as no surprise that this setting provides the best results. Since we are not interested in other configuration aspects in this experiment, we use the $\mathbf{S}_{\text{full}}/\text{M}_0$ as a normalized success rate of 100%. The main result of this experiment is that AP with sensing restricted to only a quarter of the total grid cells (half the grid both horizontally and vertically) and memory span limited to 5 cycles is almost as good. In fact, relying on memory instead of AP turns out to be a drawback in *all* settings with limited sensing. This is particularly interesting since *all* agents do replanning as soon as new perceptions violate their old plans; but the proactive information gathering strategy induced by AP seems to be more appropriate in a dynamic MA domains as this one. For all sensing limits, the best results are always obtained with M_5, i.e. a limited memory span that is able to compensate somehow for the limit sensing range, yet enforces information gathering instead of relying on dated beliefs.

5 Related Work

Relations of our approach to conditional and probabilistic have already been discussed in Sec. 1. The focus of this paper has been on single-agent planning in MA environments, with only indirect consideration of other agents; we therefore only briefly mention Distributed and MA Planning methods that will become relevant for studying cooperation in our framework. An survey of techniques for Distributed Planning that are relevant to this work is given by desJardins et. al. [8]. Most work within this field is based on hierarchical representations of multiagent plans [10, 9, 7]. Indeed, the expansion of assertions is similar to the decomposition of HTN schemata [22, 11]. In our approach, however, the abstraction hierarchy need not be explicitly given by the domain designer, but is resolved by the planner itself. Also the purpose of the abstraction is different from HTN planning: while HTN decompositions embody knowledge about how to solve subtasks, assertions essentially represent a way to postpone parts of the

planning process. Thus AP produces a *sequence* of *flat* plans, whereas HTN produces one abstraction hierarchy. While it has been proposed in textbooks that HTN planners may leave parts of the plan hierarchy unexpanded until a plan has been partially executed, we are not aware of work that describes how an HTN planner should make such decisions. A more in-depth comparison of AP and HTN planning will be the topic of a future paper. An interesting practical feature of AP is, however, that in its main loop it can include any general-purpose Planning algorithm. Thus, we can make use of performant state-of-the-art Planning systems, as shown in Sec. 4.

6 Current and Future Work

We have presented a new principled approach to Continual Planning in dynamic systems. For lack of space, we could not present some extensions we have already developed and implemented:

Richer Language: To enable agents to reason about their peers we extended the model to include beliefs about other agents and *mutual beliefs* among agents. Similarly, *copresence models* are generalizations of sensor models that describe when a common perception becomes commonly known. Based on these extensions we can model *speech acts* in our planning language.

Learning Assertions: Since assertions are syntactically and semantically very similar to normal actions, it should be possible to learn them automatically from past planning episodes or derive them from a planning domain. For some very basic cases we have already managed to do this.

Collaboration: As presented, our framework specifies only a limited form of MA plans. In particular, intentional modalities that are necessary for joint activity can not be represented [14]. However, even in its limited form our explicit modeling of sensing and communication complements BDI-inspired MAP models by describing how and why agents plan actions changing the beliefs of other or of their own. AP provides the additional perspective of describing how goals can be exchanged and adopted during a collaborative planning process. We have already developed agents for MAPSIM that use Joint Continual Planning to develop basic collaborative plans.

Most importantly, our continual planning approach is currently being integrated into a robot where it is used not only for action planning, but also, e.g., for disambiguation of natural language utterances, behavior prediction of other agents. First results of the use of the planner for interpretation and execution of action commands in natural language have been published[6]. We expect similarly interesting results when using the planner for continual human-robot interaction.

7 Acknowledgments

This work has been supported by the EC under contract number FP6-004250-CoSy.

References

1. Christer Bäckström. Computational aspects of reordering plans. *JAIR*, 9:99–137, 1998.

2. Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, November 1995.
3. Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. IJCAI-01*, pages 473–478, 2001.
4. Craig Boutilier and Ronen Brafman. Partial order planning with concurrent interacting actions. *JAIR*, 2001.
5. Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proc. AAAI-96*, pages 1168–1175. MIT Press, July 1996.
6. M. Brenner, N. Hawes, J. Kelleher, and J. Wyatt. Mediating between qualitative and quantitative representations for task-orientated human-robot interaction. In *Proc. of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, India, 2007.
7. Bradley Clement and Edmund Durfee. Top-down search for coordinating the hierarchical plans of multiple agents. New York, NY, USA, 1999. ACM Press.
8. M. DesJardins, E. Durfee, Jr. C. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 2000.
9. Marie DesJardins and Michael Wolverton. Coordinating a distributed planning system. *The AI Magazine*, 20(4), 1999.
10. E. Durfee and T. Montgomery. Coordination as distributed search in hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 1991.
11. Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence*, 18:69–93, 1996.
12. Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In *Proc. KR-92*, pages 115–125, 1992.
13. Keith Golden. Leap before you look: Information gathering in the puccini planner. In *Proc. AIPS-98*, pages 70–77, 1998.
14. Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.
15. Malte Helmert. A planning heuristic based on causal graph analysis. In *Proc. ICAPS 2004*, pages 161–170, 2004.
16. Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.
17. Hector J. Levesque. What is planning in the presence of sensing? In *Proc. AAAI-96*, pages 1139–1146. MIT Press, July 1996.
18. Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *JAIR*, 9:1–36, 1998.
19. D. McDermott. PDDL – the planning domain definition language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
20. Ronald Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In P. Traverso M. Ghallab, J. Hertzberg, editor, *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pages 212–221, Toulouse, France, 2002. AAAI Press, Menlo Park.
21. Jussi Rintanen. Constructing conditional plans by a theorem-prover. *JAIR*, 10:323–352, 1999.
22. Qiang Yang. *Intelligent Planning: A decomposition and abstraction based approach*. 1997.
23. Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: a review. 3(2), 2000.