Plan recovery in reactive HTNs using symbolic planning

Lydia OULD OUALI
LIMSI-CNRS, UPR 3251, Orsay, France
Univ. Paris-Sud, Orsay, France
Email: ouldouali@limsi.fr

Charles RICH
Worcester Polytechnic Institute
Worcester, MA, USA
Email: rich@wpi.edu

Nicolas SABOURET
LIMSI-CNRS, UPR 3251, Orsay, France
Univ. Paris-Sud, Orsay, France
Email: Nicolas.Sabouret@limsi.fr

***Abstract*—Hierarchical reactive methods are very popular in the field of controlling complex artificial intelligent agents in dynamic environment. However, dynamic environments are incompletely known and can change in unpredictable way which make the planning systems fail. In this paper, we describe a hybrid planning system called *Discolog* which extends a reactive planning system with a linear symbolic planning system to propose a strategy to recover from breakdown. Our solution has been implemented on the reactive system *Disco* which been extended with the symbolic planning system STRIPS.**

I. Introduction

Automatic planning is an important field of controlling artificial agents in complex and dynamic environments where research built two different approaches. The first one is symbolic planning: this approach consists in constructing a complete symbolic and logical model of the environment that allows the agent to reason about this model and define a complete plan to carry out its goals. The most popular architecture used to describe the environment is HTN (i.e Hierarchical Taks Network) [Ero96], which allows a recursive decomposition of complex goals into sub-goals or primitive actions. The HTN architecture eases the design of the environment and gives more expressiveness. One limit of symbolic planning is to assume that the environment is fully defined. By consequence, the agent is able to predict all the possible situations to plan in advance. However, it becomes clear that authoring a complete representation of a dynamic and complex environment such as simulation of human behavior [CGS98] or the definition of dialog systems [AF02] requires significant knowledge-engineering effort [ZHH$^+$09], and even reveals to be impossible [Mae90]. However, with incomplete knowledge the agent cannot anticipate the future and the generated plan might be not executed as expected. Therefore, if at any point of the execution the plan breakdown (i.e action execution fails), the planner has to stop the execution and build another plan that achieves the agent's goals. Such operation might be costly in terms of time and resources.

Because of these limitations, another planning approach called reactive planning was proposed [Fir87]. Reactive planning avoids long-term prediction in order to make the execution faster. For this reason, it leaves all the planning during the execution phase: the agent plans only for the next step to be executed from the current state of the environment. Thus, it can adapt its next step according to the observed changes. The main advantage of reactive planning systems is they don't need a complete definition of the environment. Instead, they aim to define the policy of the agent in its environment by running through a pres-authored HTN structure with procedural knowledge. Procedural knowledge defines conditions in the HTN domain knowledge as black-box procedures (for example : JavaScript code) that contains no logical information (i.e no symbolic knowledge). This type of reactive HTN eases the design, reduce the complexity of planning and still can cope with complex dynamic environments [Bro05]. They are used in numerous application domains, such as dialog systems [BR03], games [Isl05] and simulating human behavior [Bro05].

Nevertheless, breakdowns can still appear in reactive planning. An action execution can fail and leads the HTH to a state where no possible action can be applicable to achieve the goal. In such situation, the agent has to stop and think of a strategy to reach its goal. However, reactive planning avoid long term prediction which make it unable to construct a plan recovery. In addition, without symbolic knowledge, the HTN has nothing to reason about.

Our claim is with a minimum definition of symbolic planning, the agent can build efficient local plans to recover from breakdowns. Since reactive planning avoid long term prediction, we propose in this paper to first extends the procedural knowledge to a symbolic one and integrate it to a symbolic planner to reason about and construct plan repair.

In order to deal with this limitation, we propose in this paper to extend reactive HTNs with a linear symbolic planner to build a plan allowing the HTN to recover from breakdowns. Moreover, after analyzing the HTN procedural knowledge, we find out that several procedural task's operators are boolean procedures that can be thought as symbolic predicates. We propose thus to the HTN designer to extend theses HTN procedures to symbolic knowledge allowing the symbolic planner to compute local recovery plans. We study the capacity of such hybrid model to recover from breakdowns in dynamic environment.

In section 2, we briefly present existing works in this domain. In section 3, we formalize the proposed solution *Discolog* and describe its implementation. Section 4, presents the preliminary evaluation to our model. We discuss the obtained solutions and the impact of both quantity and quality of the extracted symbolic knowledge in the recovery process.

II. Background and related works

Basic requirements to build planning systems is to have a formal representation of the environment (i.e domain knowledge) and planning engine to reason about this knowledge. In the following we present the main contributions in planning.

A. Symbolic planning

The first contribution in planning was symbolic linear planning systems (STRIPS)[FN72] which model the environment as a set of actions and transitions between those actions . Linear planning attempt to generate a plan (i.e. a sequence of actions) which once executed transform the initial state of the environment to a final state where the agent goal is achieved.

Due the complexity of planning problems environment, a new approach of planning called HTN (Hierarchical Task Network) [Ero96] was proposed. HTNs can be represented as AND/OR tree. AND nodes represents tasks. each task has preconditions to check the applicability of the task and postconditions parameter that checks the success of the task execution. There are two different types of tasks: primitive tasks (leaf nodes) are similar to linear planning actions which can directly be performed in the environment. Compound tasks have to be decomposed into subtasks using a corresponding recipe. Recipes of tasks are defined as OR nodes represent a method of decomposition of compound tasks.

HTN planners intent to plan for one or more goal task. Planning proceeds using task decomposition that decomposes goal task using a corresponding recipe into a sequence of simpler subtasks. This process is applied recursively until a conflict-free plan (sequence of primitive tasks or actions) that can make the goal task successful is found. HTN planners become popular this last decade where several systems were developed such as SHOP [NCLMA99], SIPE [Wil88] or NOAH [Sac75].

During the execution in a highly dynamic environment, an action execution might drift from what was expected because of the lack of information about all the changes in the environment. Such situation causes an execution failure or *breakdown* and the planner can no longer achieve its goal. Several researchers in symbolic planning have been attacking this problem by integrating a plan repair module [BD02], [VDKDW05], [HTHO06], [AKYG07], [WHLUMA07]. Thus, if a breakdown is detected, the plan repair module uses a causal graph that contains all the logical dependencies between the HTN task to calculate the task candidate to repair (the strategy for computing the task candidate differs). Next, the planning system is called again to propose a new decomposition to the task candidate. Theses propositions avoid replanning from the scratch and proposes a local repair with minimal costs. However, they remain dependent to the initial planning system. Thus, if this latter is unable to find a plan repair, then the HTN definitively fails to achieve the goal. Moreover, theses systems inherit the limitations of symbolic planning: with limited knowledge, the HTN can not always find other decompositions to tasks.

B. Reactive planning

The problem of planning in high dynamic environment with incomplete knowledge has been tackled in reactive planning systems [Sch87]. Reactive planning avoids long term prediction. Instead, they plan only for the next act to perform at every moment which allows reactive HTNs adapting their next act to the observed changes in the real environment. Therefore, action execution is not selected from a pre-constructed plan but it is computed directly in the execution process. Actions to perform are selected from a hand coded domain knowledge proposed by the planner designer. It models the policy of the agent in its environment (i.e all the tasks that it can perform). In addition, for some planners [Ric09] action's operators don't contain any logical information and are represented as simple code procedure (for example JavaScript, XML). Thus, the planner cannot reason about this knowledge. This type of domain knowledge is called *Procedural domain knowledge*.

Reactive planning becomes very popular in AI and for most of cases uses an HTN formalism to model their domain knowledge. For instance, in Robotics where [Fir87] defends a parallel reactive architecture with *Reactive Action packages* representing autonomous process used to achieve the different goals of the robot. [Bry01] and [Bro05] propose a reactive planning system for prototyping human-like behavior in a virtual environment to ensure a natural behavior with respect of time constraint and reactivity with the real world. Reactive planning is also used in Gaming with the name of behavior trees [Isl05]. Behaviors trees have the same hierarchical structure of HTNs and do real time decision which can be seen as reactive planning.

C. Previous Approaches

Reactive planning also rise this problematic among researchers dealing with investigations in reactive planning. The work presented in [Fir87] suggest that robots need strategic planning to detect problematic situations before they occur. Therefore, it proposes to add a strategic planner's job which put constraints on the planner behavior before its execution in order to prevent inefficiencies. Theses constraints can be an ordering tasks in the execution queue of the planner or choosing the most promising decomposition for compounded tasks. C.Brom [Bro05] observed some limitations in reactive planning for human agents. Among the limits discussed in this work, they discussed the necessity of planning to make the behavior of agents appear more intelligent and believable for example to be able to see the distance of its tasks especially to achieve goals with time constraint. We support this claim in our work by proposing a hybrid system that extends a reactive planning system with a symbolic planner to recover from breakdowns.

III. Motivation example

This section shows an example of HNT reactive planning for a robot arm responsible for charging object into trucks.

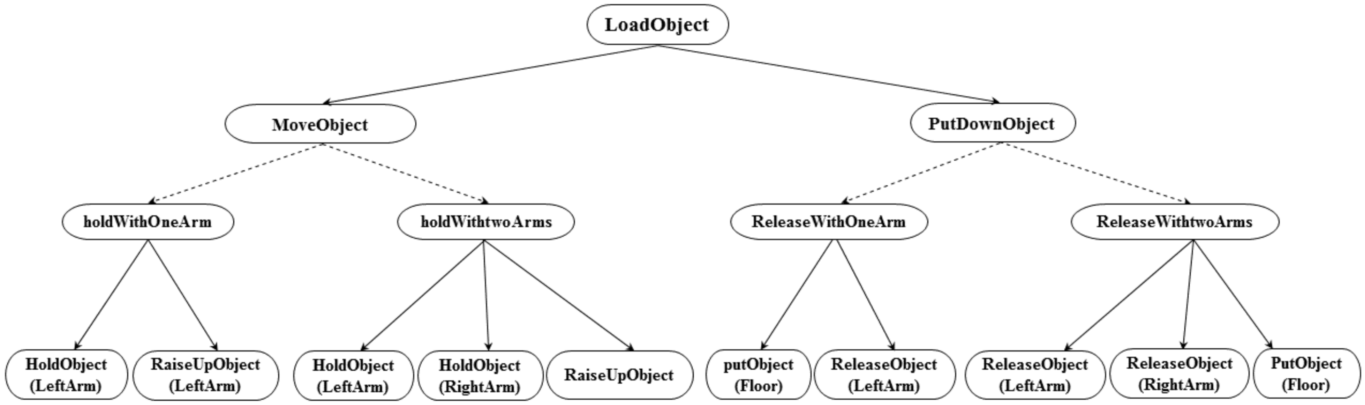


Fig. 1: Load Object task decomposition for robot arm motion

The goal of the robot is to load objects and dispose them into the truck. Figure **??** describes the goal task decomposition to achieve the load task. In order to load an object into the truck, the robot has to move the object from its initial place and put it into the truck. To move this object the robot starts by holding the object. Depending on the object's weight, the robot will choose the corresponding recipe as described in figure **??**: the first recipe proposes the robot to use one arm if the object weight is below than 5 kilograms. The second recipe proposes to use both of the robot arms if the object weight is higher than 5 kilos and below than 10 kilos. In this example, the weight of the object is 20 kilos. Thus, when the robot tries to hold the object with one arm, it finds that the object is too heavy which made the recipe inapplicable. therefore, it tries the second recipe and holds the object with both of its arms but it fails again because the object is still too heavy for the robot. At this moment, the robot has no other recipe to try and remains blocked with no possible task to execute. A breakdown is then reached by the robot. When a breakdown happens, the robot has to think of a strategy or a plan to get out from this breakdown.

In the next section we present our proposition to overcome the breakdown problem.

IV. Solution Approach

Reactive HTNs don't look ahead to predict the future. Instead, they plan only for the next act to perform at every moment which make the agent reactive with the changes in its environment. Nevertheless, as dynamic environment is unpredictable, primitive task execution might lead to a dead end where no possible task can be performed next and a breakdown is detected. In order to recover from such breakdown, the agent has to think of a strategy, but reactive planning avoid long term prediction and without symbolic knowledge the agent has no way to think.

To overcome this problem, we propose to extend the reactive HTN with a symbolic linear planner that construct a local strategy to recover from the faced breakdown. Symbolic linear planning systems (see section II-A) reason on primitive tasks (i.e actions) represented with a symbolic formalism. To construct this symbolic domain knowledge from the HTN procedural knowledge, we analyzed the procedural knowledge to find out that several procedural task's operators are boolean procedures that can be thought as symbolic predicates. Therefore, we propose to the HTN designer to extend all theses primitives tasks to a symbolic formalism in order to build the symbolic domain knowledge.

The hybrid planning system proceeds as described in the algorithm 1. It calls the reactive HTN to achieve the goal. The environment is monitored at each step of the execution to check the success of the execution. Nevertheless, a breakdown might occur and make the HTN execution fails.

When a breakdown is detected, the hybrid system invokes the recover procedure. First, The recover procedure traverses the hierarchy of the goal task to detect all the children tasks affected by the breakdown and attempts to propose a plan repair for each of them.

Linear planning attempts to reach a goal state rather than achieving a task. Therefore, the recover algorithm determines candidates for recovery by looking for the failed conditions in the tasks affected by the breakdown using for that their task status. Every task in the HTN can be either *Live* (Task preconditions are valid and the task can be performed), *Done* (The task has been executed successfully), *Failed* (Task execution failed) or *Blocked* (Task preconditions are not valid). Therefore, the algorithm takes a task preconditions as candidate if its status is *Blocked*, or its postconditions if its status is *Failed*. If the task is a compound task with its status to *Live* and none of its recipes are valid then the applicability condition of those recipes are considered as candidates. Note that a condition is considered as candidate only if it can be converted to a symbolic formalism. Otherwise, the recover procedure ignores this condition.

For instance, taking back the example described in the section III, a breakdown is detected because any recipe can be applied to decompose the move task. The system, then calls the recover procedure to detect all the affected tasks by this breakdown. The candidates list includes both applicability

Algorithm 1 Reactive planning and plan recovery algorithm

procedure HYBRID SYSTEM(DomainKnowlege,Goal)
$\pi \leftarrow$ *ReactiveHTN(DomainKnowlege,Goal)*
If ($\pi \leftarrow$ *Success*)
return *Success*
else $plan \leftarrow Recover(Goal)$
If (plan $=$ *null*)
return Failure
else
foreach action $a_i \in$ plan
$Discolog(HTN, a_i)$
end procedure

procedure RECOVER(Goal)
Candidates $\leftarrow$ *findCandidate*(G)
If *Candidates* = $\emptyset$
return *null*
else $\Pi \leftarrow \emptyset$
foreach *candidate* $\in$ *Candidates*
$\Pi+=$ LinearPlanner(candidate, CurrentState)
$Cost \leftarrow \{cost(\pi)|\pi \in \Pi\}$
return $\pi \in \Pi$ with minimum cost(π)
end procedure

procedure FINDCANDIDATES(Goal)
foreach *child* $\in$ *Goal*
If (precondition(child)! $=$ $\emptyset$ and *status* (child)$\notin \{Done, Live, Blocked\}$)
add precondition(child) to Candidates
else If (postcondition(child)! $= \emptyset$ and *status* (child)$\in \{Failed\}$
add postcondition(child) to Candidates
If (*status* $\in \{Live\}$ and nonPrimitive(child) and applicability(child)!= $\emptyset$)
add App-condition(child) to Candidates
findCandidates$(children(child))$
return Candidates
end procedure

conditions of the failed recipes, the postconditions of the move task and the load task.

Once the list of candidates has been determined, the linear planning system is called for each candidate. This linear planner returns a list of possible plans to recover from the breakdown and most promising plan is applied. We decided to keep the shortest plan to allow local repair and prevents other breakdowns due to the execution of pre-constructed plan. Thus, in the example, the planner proposes as plan to decompose the object into two littles objects and load them separately.

V. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The proposition presented in this paper has been implemented using a reactive HTN called Disco [Ric09] and a simple linear STRIPS planner. The produced system is called *Discolog*.

Disco uses the ANSI/CEA-2018 standard for the procedural definition of its domain knowledge and a Java-based reactive planning system. Tasks are modeled using XML format. Primitive tasks contains grounding script parameter defined as JavaScript program which represent the effect of primitive task execution in the environment. The STRIPS version used in *Discolog* is an existing Prolog implementation proposed in [PGM98]. Prolog is based on inference engine which allows STRIPS reasoning on the possible links between the predicates and deducts extra knowledge.

The integration of STRIPS in the Disco system is performed using *tuProlog* [1] framework. The use of *tuProlog* presents two mains advantages. First, *tuProlog* is a Java-based framework that exploits a Prolog engine directly from a Java program. Thus, STRIPS can locally raised without any call to an external system. Second, it has specific libraries for the prolog predicates that helps with the conversion of recovery candidates from the procedural knowledge to symbolic knowledge.

The aim of this experiments is first to validate the hybrid architecture of Discolog system. Second, we want to test the contribution of the symbolic knowledge in the performances of recovery. We make the assumption that the effectiveness of the recovery process is relied to the level of knowledge in the symbolic domain knowledge. In fact, we assume that the more information STRIPS planner gets from the procedural domain knowledge, the more effective recover plans it can generates. The result of this experiment should proves that plan recovery follows a monotonic evolution in function of the level of knowledge defined in STRIPS.

In order to validate Discolog, we had to test it on different HTNs domain knowledge and analyze its ability of recovery on every possible breakdown. Nevertheless, in the absence of accurate models including reactive and symbolic knowledge, we implemented our own evaluation data.

As theses primary tests purpose is to validate the hybrid architecture of Discolog, we actually don't need a domain knowledge with a semantic description of its tasks. Therefore, we defined a procedural domain knowledge based only on synthetic data structured in such way to ensure a believable execution. Each compound task (T) is defined with a set of recipes (R) and each recipe is constituted by a set of children tasks (C) to decompose the parent task (T). Preconditions of compound task are propagated to its first child in each recipe, and the postconditions are propagated to its last child. The rest of task children in each recipe are defined with chained conditions: The postconditions of the $task_i$ activate the preconditions of the $task_{i+1}$. Symbolic knowledge is extracted from the procedural one. The goal is to study the affect of the level of symbolic knowledge defined on recovery, we randomly extract primitive tasks from the HTN and vary the number of tasks extracted depending on a predefined level.

We tested Discolog on different HTNs: (depth, number of

[1] http://apice.unibo.it/xwiki/bin/view/Tuprolog/

recipes/task, number task children/recipe). For each HTN, we varied first, the level of symbolic knowledge to be extracted $\{25\%, 50\%, 75\%\}$. For each level of knowledge, we randomly extracted different symbolic domain knowledge. Second, the initial state is randomly defined. This later affects the decomposition that the HTN will choose and the plan recover procedure. Once theses parameters are set, for each primitive task, a breakdown is caused and the plan recovery for this task is analyzed. The experiments have been done on two different HTN configuration as presented in table I.

TABLE I: Configuration of the HTNs

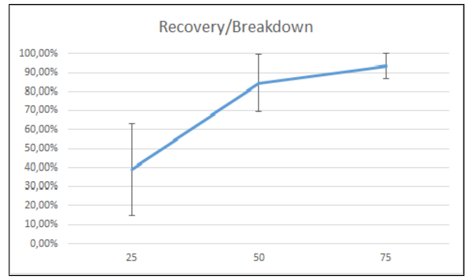
Configuration	Total Nb of nodes	Init state	Nb of symbolic DN
(3,3,3)	91	10	60
(5,1,4)	341	10	60

Figures 2a and 3a display the recover rate for HTNs. We notice that the hybrid system is able to propose plans recovery and switch between the procedural and the symbolic environment. Moreover, the results confirm our assumption: the graph follows a monotonic assumption in function of the symbolic knowledge. Thus, the more symbolic knowledge Discolog has the more it can recover from breakdowns.

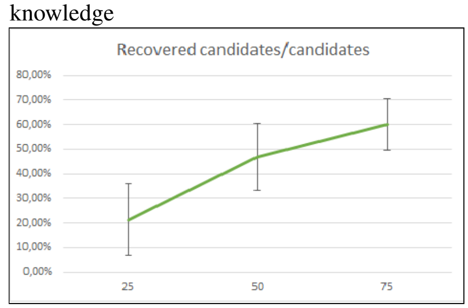
The error bars defined in the graphs represent the standard deviations fo each execution in function of the level of symbolic knowledge. We notice that the less symbolic knowledge Discolog has the bigger is the standard deviation. For the same HTN, Discolog gets different rates of performances. For example for HTNs defined with the 25 % of symbolic knowledge, Discolog performances varies from 0% to 65% of recover. This is noticed in the case of HTN with 25 % and 50 % of defined symbolic knowledge. Thus, HTNs defined with limited symbolic domain knowledge, are unable to cover all the possible breakdowns because of their lack of symbolic knowledge.

Figures 2b and 2a demonstrate the average of the number of candidates repaired during the execution. The results for repairing candidates also confirm our assumption. However, the error bars as demonstrated in graphs show that, Discolog performances varies for repairing all the possible candidates and this independently of the level of symbolic knowledge defined. Theses result raise a new question of the quality of the symbolic knowledge defined by the designer. Symbolic knowledge is limited, then it has to be expressive and very representative of the agent policy, or Discolog will not have the precise knowledge to recover from all the possible breakdowns.

These tests are very promising but remains far from being definite experimental analysis. An extensive tests on a set of realistic domain knowledge is the object of futures works to detail the problematic of the quality of knowledge and test Discolog on real planning problems.



(a) Recover rate for each level of knowledge



(b) Average number of candidates repaired for each level of knowledge

Fig. 2: Results for the (3, 3, 3) HTN

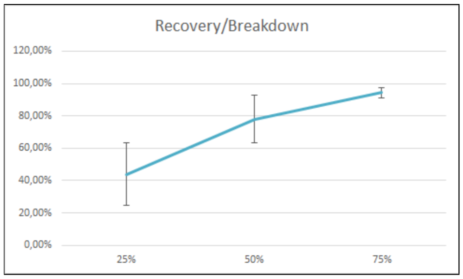
VI. Conclusion

In this paper we have presented *Discolog*, an algorithm to recover from breakdowns in reactive HTN planning systems. Despite the ability of reactive planning to deal with high dynamic world, breakdowns might occur because of the lack of knowledge on the different possible changes in the world and the ineptitude of reactive HTN to reason in long term to define a recover strategy.

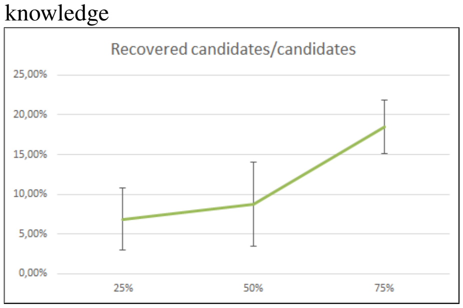
The proposed algorithm extends reactive HTNs with linear symbolic planner to produces a plan to recover from the breakdown. The symbolic knowledge is extracted from the procedural knowledge by the HTN designer. Thus if a breakdown is detected, the algorithm calculates the candidates (conditions which are not valid in the non-executed HTN tasks), then STRIPS is called to propose plans to repair these conditions. The most promising plan is then converted to procedural formalism and executed.

The solution have been implemented and validated. It combines a reactive HTN Disco with the symbolic linear planning system STRIPS in Prolog. The results of our preliminary experiments was promising and demonstrated, first, the ability of the hybrid planning system Discolog to propose viable plans recovery and for the different breakdowns. Second, the contribution of symbolic knowledge in the performances of the recovery. Finally, the experiments raise another problematic of the quality of knowledge in limited domain knowledge that we wish address. In addition, we intend to validate Discolog on real planning applications.

Future prospects of this research are, first, to construct a modeling tool for HTN model design. This tool will help the HTN designer in one hand to define the level of knowledge to integrate in the HTN. In the other hand, the system will use the history of breakdowns to propose adding knowledge



(a) Recover rate for each level of knowledge



(b) Average number of candidates repaired for each level of knowledge

Fig. 3: Results for the (4, 1, 5) HTN

in the HTN where breakdowns occurred. As second step, we propose to integrate Discolog in social dialog system between an agent and a human in order to support the dynamic nature of a social dialog.

References

[AF02] James Allen and George Ferguson. Human-machine collaborative planning. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, pages 27–29, 2002.

[AKYG07] N Fazil Ayan, Ugur Kuter, Fusun Yaman, and Robert P Goldman. Hotride: Hierarchical ordered task replanning in dynamic environments. In *Planning and Plan Execution for Real-World Systems–Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop. Providence, RI*, 2007.

[BD02] Guido Boella and Rossana Damiano. A replanning algorithm for a reactive agent architecture. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 183–192. Springer, 2002.

[BR03] Dan Bohus and Alexander I Rudnicky. Ravenclaw: Dialog management using hierarchical task decomposition and an expectation agenda. 2003.

[Bro05] Cyril Brom. Hierarchical reactive planning: Where is its limit. *Proceedings of MNAS: Modelling Natural Action Selection. Edinburgh, Scotland*, 2005.

[Bry01] Joanna Joy Bryson. *Intelligence by design: principles of modularity and coordination for engineering complex adaptive agents*. PhD thesis, Massachusetts Institute of Technology, 2001.

[Byl94] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994.

[CGS98] Rosaria Conte, Nigel Gilbert, and Jaime Simão Sichman. Mas and social simulation: A suitable commitment. In *Multi-agent systems and agent-based simulation*, pages 1–9. Springer, 1998.

[Ero96] Kutluhan Erol. Hierarchical task network planning: formalization, analysis, and implementation. 1996.

[Fir87] R James Firby. An investigation into reactive planning in complex domains. In *AAAI*, volume 87, pages 202–206, 1987.

[FN72] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[HTHO06] Hisashi Hayashi, Seiji Tokura, Tetsuo Hasegawa, and Fumio Ozaki. Dynagent: An incremental forward-chaining htn planning agent in dynamic domains. In *Declarative Agent Languages and Technologies III*, pages 171–187. Springer, 2006.

[Isl05] Damian Isla. Handling complexity in the halo 2 ai. In *Game Developers Conference*, volume 12, 2005.

[Mae90] Pattie Maes. *Designing autonomous agents: Theory and practice from biology to engineering and back*. MIT press, 1990.

[NCLMA99] Dana Nau, Yue Cao, Amnon Lotem, and Hector Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pages 968–973. Morgan Kaufmann Publishers Inc., 1999.

[PGM98] David I Poole, Randy G Goebel, and Alan K Mackworth. *Computational Intelligence*. Oxford University Press Oxford, 1998.

[Ric09] Charles Rich. Building task-based user interfaces with ansi/cea-2018. *Computer*, 42(8):20–27, 2009.

[Sac75] Earl D Sacerdoti. A structure for plans and behavior. Technical report, DTIC Document, 1975.

[Sch87] Marcel Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI*, volume 87, pages 1039–1046. Citeseer, 1987.

[VDKDW05] Roman Van Der Krogt and Mathijs De Weerdt. Plan repair as an extension of planning. In *ICAPS*, volume 5, pages 161–170, 2005.

[WHLUMA07] Ian Warfield, Chad Hogg, Stephen Lee-Urban, and Héctor Munoz-Avila. Adaptation of hierarchical task network plans. In *FLAIRS Conference*, pages 429–434, 2007.

[Wil88] David E Wilkins. *Practical planning: Extending the classical AI planning paradigm*. Morgan Kaufmann Publishers Inc., 1988.

[ZHH+09] Hankz Hankui Zhuo, Derek Hao Hu, Chad Hogg, Qiang Yang, and Hector Munoz-Avila. Learning htn method preconditions and action models from partial observations. In *IJCAI*, pages 1804–1810, 2009.