Integration of Online Learning into HTN Planning for Robotic Tasks

Stéphane Magnenat
Autonomous Systems Lab - ETH Zürich
stephane at magnenat dot net

Jean-Cédric Chappelier
LIA - EPFL
jean-cedric.chappelier@epfl.ch

Francesco Mondada
LSRO - EPFL
francesco.mondada@epfl.ch

Abstract

This paper extends hierarchical task network (HTN) planning with lightweight learning, considering that in robotics, actions have a non-zero probability of failing. Our work applies to A*-based HTN planners with lifting. We prove that the planner finds the plan of maximal expected utility, while retaining its lifting capability and efficient heuristic-based search. We show how to learn the probabilities online, which allows a robot to adapt by replanning on execution failures. The idea behind this work is to use the HTN domain to constrain the space of possibilities, and then to learn on the constrained space in a way requiring few training samples, rendering the method applicable to autonomous mobile robots.

Introduction

The hierarchical task network (HTN) planning method (Ghallab, Nau, and Traverso 2004, ch. 11) has proved to be a powerful block for building high-level robot behaviours in various contexts, such as navigation (Belker, Hammel, and Hertzberg 2003), activity planning (Beaudry, Kabanza, and Michaud 2005), human-robot interaction (Alami et al. 2006), or autonomous construction (Magnenat 2010). This planning method allows to express knowledge and to reason about the possible actions of the robot and their consequences on the world. Planners store knowledge in the form of task networks, that are recursive trees of alternative task decompositions, including preconditions and action effects. The set of all task networks used by a planner is called the *planning domain*. Task networks are expressive, but they must provide a complete and totally correct world knowledge. Yet this is hardly possible in the reality of robotics, where there might be several ways to perform an action, some better than others given the specificity of the particular environment in which the robot evolves. In robotic applications, HTN planning thus needs adaptation/learning.

Let us illustrate the limitations of basic HTN planning in the real world with an example. Imagine a robot that must fetch objects from a cupboard and put them on the ground as fast as possible. There are two ways to do so: gently putting the object down or dropping it. Dropping is faster, and therefore preferred. But while some objects such as balls can be dropped, others such as glasses must be handled with care: if dropped, they will probably break and the plan execution will fail. In this paper, we propose a way to learn from the results of action executions in order to provide the planning process with a moderate adaptability. In the former example, our method will make the planner gently put down glasses after a few failed drops. If the glasses are in fact made of plastic and can be dropped safely, the same planning domain will let the robot prefer this faster way. The behavioural difference solely results from the experimental interaction with the environment.

The contribution of this paper is thus the integration of a lightweight form of learning into HTN planning, suitable for real-time use by mobile robots. This learning process operates in the relatively small space of possible actions and therefore does not require a large experimental corpus. This is possible because the HTN planning domain already heavily constrains the space of possible action sequences. Note that action success rate might depend on the context (the previously executed actions), but this contextual relationship is known a priori and can be encoded by hand. The main difference with most of the works integrating learning into planning is that these employ Markov decision process (MDP) and link probabilities to states. In naive MDP formulations, the size of the state space is proportional to the Herbrand base of the planning problem, which grows exponentially with the number of objects considered (Schulz 2002). As discussed in the next section, this size remains significant despite the various ways to reduce it that have recently been discussed in the literature (see for instance Meneguzzi et al. 2011). On the contrary, the method we propose in this paper maintains probabilities on actions, not on states. Therefore, it can take advantage of high-level information on the action success rate, using only a small number of trials. This allows the developer of the planning domain to write all *possible* ways to perform a task and to let the robot explore at run time which actions are the most suited for its particular environment.

Related work

Our work is not the first attempt at integrating learning into HTN planning. In this section, we briefly review related work and discuss how it compares to ours.

Li, Kambhampati, and Yoon (2009) define a HTN scheme in which probabilities are associated with task decompositions. In contrast, we are more interested in adding probability of success to basic actions, because for robotic applications this is the point where the plan might fail. Note

that the work of Li et al. focuses on learning the planning domain from plan traces, while we concentrate on improving the adaptability of hand-crafted domains.

In the field of autonomous mental development (AMD), the work of Mugan and Kuipers (2011) aims at learning high-level states and actions in continuous environments. While this developmental system has many layers, at the planning level it associates success rates to actions, and chooses the plan with the best chance of success considering these rates. Therefore, albeit formulated in a different context and missing a formal proof of optimality, this work is the closest to ours regarding high-level hypotheses.

Morisset and Ghallab (2008) present a robot that learns from experience in a similar way to our method. However, these authors maintain separate *control states* linked to the HTN skills hierarchy through a MDP, that is the object of learning. In contrast, our method attaches probabilities of success to contextualized actions and considers these to be independent.

Meneguzzi et al. (2011) propose to construct a MDP from a hand-crafted HTN, and then solve the MDP. Similarly to our work, this allows to consider error rates in action executions. However, in that work the MDP operates in a grounded space, while our approach allows the planner to work in the lifted space (Russell et al., Sect. 9.2, p. 275), reducing dimensionality by considering classes of objects rather than instances. In the experiment presented in Meneguzzi et al., the number of MDP states seems to be proportional to the logarithm of the Herbrand base of the planning problem. But, the MDP still holds about 85 states for a Herbrand base of 100, which corresponds to a very small domain. Furthermore, the lack of details on the experimental scenario prevents a complete complexity analysis. The MDP approach also requires a transformation phase, which can be long. Therefore we believe that, when applicable, our approach proposes a better solution in terms of simplicity and efficiency.

Starting from considerations similar to our work, Parr and Russell (1997) propose the concept of hierarchical abstract machines to introduce a HTN-style hierarchy to MDP and reinforcement learning. These authors show huge performance improvements compared to flat MDP. But the method cumulates the disadvantages of operating on grounded states and being less expressive than the HTN formalism. In the same direction, Kuter and Nau (2005) have incorporated HTN-style search strategies to MDP planning. The idea of planning on a MDP is very common and representative of the field of planning under uncertainty, where hierarchy is exploited to reduce the search space. However, the MDP forces the use of grounded predicates and states (C. Boutilier and Hanks 1999).

When compared to related work, our method clearly has the advantage of being practically implementable on a physical robot and capable of showing adaptability after only a few tries. Besides, it does not add any complexity to the planning algorithm, nor does it require a transformation or preprocessing of the planning domain. The extra-work lies at the level of defining the contextualized actions, which encode the a priori knowledge about the dependency between the success rate of sequential actions in the real world.

HTN planning

We propose to extend the HTN planning framework with probabilities and to learn these probabilities online. A HTN planner decomposes a goal task into subtasks until it finds a sequence of actions the robots can perform. The planner knows the available methods and their possible decompositions. When given the goal task and the initial state of the world, the HTN planner seeks an admissible sequence of actions. The actions can affect the state of the world; and the planner records these alterations.

As implementation, we use Planner 9 (Magnenat, Voelkle, and Mondada 2009), which plans partially-ordered graphs of tasks using forward decomposition. It keeps track of each possible decomposition in a different search node. Planner 9 starts planning with a single node containing the goal task and the initial state of the world. When visiting a node, Planner 9 iterates through all tasks that have no predecessor. If the task is an action, it applies this action to the current state of the world and stores the action as part of the plan. Otherwise, Planner 9 instantiates the different possible decompositions of the task. This process goes on until there is no more node left or until Planner 9 has found a node with no more task to decompose. When Planner 9 decomposes a task, it performs lifting: it accumulates its preconditions for delayed check. Planner 9 assigns a value to a variable only when an action changes a relation this variable appears in.

Planner 9 chooses the node to visit by selecting the least expensive one using A* (Hart, Nilsson, and Raphael 1968). In terms of cost, it adds the total cost of the decomposition so far (path-cost in A*) and the number of remaining tasks to be decomposed (heuristic-cost in A*). One advantage of A* with respect to a depth-first search is to allow free recursions in the definition of the planning domain. This is useful in robotics because real-world problems are often expressed in a recursive way. The execution of the plan consists in sequentially executing the actions.

Planning under task-execution uncertainty

As the HTN algorithm decomposes a high-level task into a sequence of low-level actions, in general there are several admissible sequences of actions that can achieve a given task. Planner 9 finds the shortest plan when every task is decomposed into one or more action. This might not be the case in general, but it is always possible to transform a planning domain so that every task decomposes into one or more action(s). Indeed, if a task might result in no action, we can move this task one level up in the task hierarchy along its preconditions. We can apply this scheme recursively until no task could result in no action or a top-level task might result in no action. In that case, we can remove this task from the planning domain and check its precondition prior to calling the planning algorithm. Thus Planner 9 will find the plan with the shortest number of actions.

However, in the robotics context, the optimality of a plan does not depend only on the number of actions, but also on the actions themselves. In particular, each action has a different utility depending on its type and on its outcome. With a physical robot, actions might fail, and different actions have

different failure rates. Consequently, we are not only interested in finding an admissible sequence of actions in the HTN sense, but we also want to find a plan whose execution holds a high chance of success. To this end we take into account the expected probability of success of a plan. Real world considerations suggest to limit the maximum probability of success of any action to be strictly smaller than 100 %. For instance, the robot consumes energy, and from time to time needs recharging, which may prevent it from completing its plan. We also have to acknowledge that the robot is a physical device which wears down and will eventually break. Moreover, HTN planning assumes a stationary world, where nothing changes beside the robot: this is not true in reality, and the longer the execution lasts, the more probable it is that a change occurs. With success probabilities of actions less than 100 %, the usefulness of a plan implicitely depends on its length, and therefore, on the duration of its execution. Technically speaking, and because of the way A* does search, implementing these real-world considerations also prevents the HTN algorithm from expanding infinite loops of recursive tasks with actions having a 100 % success probability (the HTN planning domain allows recursions).

To model the usefulness of a plan, we associate to each action type a a corresponding utility $u(a; r)$, which is a random variable depending on the outcome r of the action type a. r is a binary variable corresponding either to success ($r = 1$) or failure ($r = 0$). For the utility function u, we only consider the *type* of the action, not the parameters of every instance, because the parameters represent real-world objects that change with the robot's goal. We want Planner 9 to find the plan π that has the maximum expected utility $\mathbb{E}[u(\pi)]$ (Schoemaker 1982). In the general case of HTN decomposition, a plan π is a directed acyclic graph (DAG) of partially ordered actions. However, let us first concentrate on the special case of a single robot and consider the plan as a sequence of k actions $\pi = (a_1, \cdots, a_k) = \boldsymbol{a}_1^k$. We define the utility of a plan π to be the product[1] of the utilities of its actions:

$$u(\pi; \boldsymbol{r}) = \prod_{i=1}^{k} u(a_i; r_i) \tag{1}$$

Thus, by the definition of the mathematical expectation:

$$\begin{aligned} \mathbb{E}[u(\pi)] &= \sum_{\boldsymbol{r} \in \{0,1\}^k} p(\boldsymbol{r}|\pi)\, u(\pi; \boldsymbol{r}) \\ &= \sum_{\boldsymbol{r}} p(\boldsymbol{r}|\pi) \prod_{i=1}^{k} u(a_i; r_i) \end{aligned} \tag{2}$$

where $p(\boldsymbol{r}|\pi)$ is the probability of the sequence of possible outcomes $\boldsymbol{r} = (r_1, \cdots, r_k)$ for plan π.

If an action fails, the robot stops the execution of the plan. We thus consider the utility of a failure $u(a; 0)$ to be 0. Therefore, the product of the utility in Equation 2 is non zero if and only if all action executions result in a success. Equation 2 then becomes:

$$\begin{aligned} \mathbb{E}[u(\pi)] &= p(\boldsymbol{r} = \mathbf{1}|\pi) \prod_{i=1}^{k} u(a_i; r_i = 1) \\ &= \prod_{i=1}^{k} \underbrace{p(r_i = 1 | \boldsymbol{r}_1^{i-1} = \mathbf{1}, \pi)}_{\theta(a_i;\pi)} \prod_{i=1}^{k} u(a_i; r_i = 1) \end{aligned} \tag{3}$$

where $\mathbf{1} = (1, \cdots, 1)$. From now on, we will denote $u(a_i; r_i = 1)$ by $u(a_i)$. Without loss of generality, we can enforce $0 < \max_a u(a) \leq 1$ by a simple re-normalisation.

Equation 3 shows that the expectation of the utility of a plan is the product of the probabilities of success of each successive action multiplied by the product of the utility of each action. In all generality, the probability $\theta(a; \pi)$ of successfully executing an action a in some plan π only depends on some of the previous actions of π performed by the robot[2]. We call these the dependency list $\Delta(a)$ of a; and define a *contextualized action* $\rho = (a, \Delta(a))$. Through their dependency list, actions fulfill a Markov property: $\theta(a_i; \pi) = p(r_i = 1 | \boldsymbol{r}_1^{i-1} = \mathbf{1}, \pi) = p(r_i = 1 | a_i, \Delta(a_i)) = p(r_i = 1 | \rho_i) = \theta(\rho_i)$. Therefore a unique parameter $\theta(\rho)$ is attached to every contextualized action ρ and represents its probability of success. The possible dependency lists are known a priori from domain knowledge, and can be used by the developer to build the Markov dependency graph, that is, to specify all possible ρ. In robotics, we expect the set of ρ to be relatively small, and clearly much smaller than the Herbrand base of the planning problem.

As explained in the introduction of this section, HTN domain can be transformed so that every task is decomposed into one or more actions. In this domain, we can use the expected utility of a partial plan, $\mathbb{E}\left[u(\boldsymbol{a}_1^i)\right]$, to guide the A* search (Hart, Nilsson, and Raphael 1968) of Planner 9 so that it always finds the plan that has the largest expected utility:

Theorem 1. *Using $-\log \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right]$ as the path cost for a partial plan containing i actions and $-n \log \widehat{\theta}$ as the heuristic cost, where n is the number of remaining tasks to be decomposed, Planner 9 finds the plan that has the largest expected utility.*

where $\widehat{\theta} = \max_\rho \theta(\rho)$, which shall be strictly less than 1 since never-ending success does not happen in reality, as previously pointed out.

Proof. A* is optimal if the heuristic function, in this case $-n \log \widehat{\theta}$, is admissible, that is, if this function never overestimates the distance to the goal. Let us consider a node with a partial plan $\boldsymbol{a}_1^i$ and n remaining tasks to be decomposed. If Planner 9 can decompose this node into a plan π, the latter will have at least $i + n$ actions, as every task is decomposed into one or more actions. Let us consider that this plan π has

[1] If an additive framework was preferred, we could use the log of utilities instead.

[2] In practice, it is even often independent from all other actions.

k actions, with $k \geq i+n$; its expected utility is:

$$\begin{aligned}\mathbb{E}\left[u(\pi)\right] &= \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right] \underbrace{\mathbb{E}\left[u(\boldsymbol{a}_{i+1}^{i+n})\right]}_{\leq \widehat{\theta}^n} \underbrace{\mathbb{E}\left[u(\boldsymbol{a}_{i+n+1}^{k})\right]}_{\leq 1} \\ &\leq \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right] \widehat{\theta}^n \end{aligned} \tag{4}$$

(with the convention that $\mathbb{E}\left[u(\boldsymbol{a}_{i+n+1}^k)\right] = 1$ when $k = i+n$). By plugging Inequality 4 into the path cost of the final plan, and noting that log is a monotonic function, we see that:

$$-\log \mathbb{E}\left[u(\pi)\right] \geq -\log \mathbb{E}\left[u(\boldsymbol{a}_1^i)\right] + (-n \log \widehat{\theta}) \tag{5}$$

This inequality shows that the path cost of the completed plan is always larger than the path cost of a node plus the heuristic cost. The heuristic function is thus admissible, and A* always finds the plan of minimum cost, that is, the plan of maximum expected utility. □

Adaptable HTN planning

In the previous section, we have introduced a way to take uncertainty into consideration within the HTN planning process. Our method depends on knowing the probability of successfully executing a contextualized action. If the world is static and the properties of the elements do not change with time, we can compute this probability based on statistics about previous executions. However, if the world is dynamic or if we want to estimate the probability of success online, we can update an estimation of this probability during the operations of the robot, which is the object of this section.

Let us suppose that the robot has executed N times a contextualized action ρ, at times $\boldsymbol{t} = (t_1, \cdots, t_N)$, and that these executions resulted in N outcomes $\boldsymbol{r} = (r_1, \cdots, r_N)$. We use an exponential forgetting model to estimate the probability $\theta(\rho)$ that the action ρ executed after time t_N results in a success by the following equation in which λ is a time constant, accounting for possible changes in the world model:

$$\theta(\rho) = \frac{\sum_{i=1}^{N} e^{-\lambda(t_N - t_i)} r_i}{(1+\varepsilon) \sum_{i=1}^{N} e^{-\lambda(t_N - t_i)}} \tag{6}$$

where ε is a small constant encoding that infinite success is not possible in reality, ensuring $\widehat{\theta} < 1$ (see discussion in the former section).

This equation suggests iteration over the whole history to compute the final probability. However, it can be implemented into an iterative version requiring only two parameters α and β such that $\theta(\rho) = \frac{\alpha}{\beta}$. Initially, $\alpha_1 = r_1$ and $\beta_1 = 1 + \varepsilon$. Then, knowing α_i and β_i at time step i, their values at time step $i+1$ become:

$$\begin{aligned} \alpha_{i+1} &= f \cdot \alpha_i + r_{i+1} \\ \beta_{i+1} &= f \cdot \beta_i + 1 + \varepsilon \\ \text{where } f &= e^{-\lambda(t_{i+1} - t_i)} \end{aligned} \tag{7}$$

The form of this learning algorithm is simple as long as we can model the success rate of an action by a single parameter.

Relations

```
unary relations = {isBall, isGlass}
```

Actions

```
takeBall(o): precond: isBall(o)

takeGlass(o): precond: isGlass(o)

dropObject(o): precond: ∅

putObjectDown(o): precond: ∅
```

Methods

```
takeObjectBall(o)
  task:     takeObject(o)
  precond:  isBall(o)
  subtasks: ⟨takeBall(o)⟩
takeObjectGlass(o)
  task:     takeObject(o)
  precond:  isGlass(o)
  subtasks: ⟨takeGlass(o)⟩
fetchObjectCarefully(o)
  task:     fetchObject(o)
  precond:  ∅
  subtasks: ⟨takeObject(o), putObjectDown(o)⟩
fetchObjectQuickly(o)
  task:     fetchObject(o)
  precond:  ∅
  subtasks: ⟨takeObject(o), dropObject(o)⟩
```

Figure 1: Planning domain of object-fetching example.

Example

We have implemented the aforementioned algorithms in Planner 9, our open-source HTN planner[3], and we also have validated the implementation with the scenario described in the introduction. Figure 1 shows the planning domain, which is available in Planner 9's distribution[4].

1. Fixed success rate

In order to validate the probabilistic planning and the contextualized actions, in a first experiment we have set the utilities and success rates as in Table 1. In this example, dropping an object is 5 times more useful than putting it down gently. However, doing so with a glass only succeeds 10 % of times. Starting from an initial state `isBall`(ball) ∧ `isGlass`(glass), setting `fetchObject`(o) as the goal returns `takeBall`(ball), `dropObject`(ball) with a cost of 1.82 for the ball and `takeGlass`(glass), `putObjectDown`(glass) with a cost of 3.43 for the glass. The alternative plan `takeGlass`(glass), `dropObject`(glass) has a cost of 4.02 and is therefore not the best. This shows that Planner 9 successfully takes the probability of success into account when planning.

[3] source code: `https://gitorious.org/planner9`, revision used for the results in this article: `b4409ed9`, 2011-12-07

[4] domain in: `problems/robot-proba2.hpp`, experiment 1 in `programs/simple-proba.cpp`: `testContextualizedAction`, experiment 2 in `programs/simple-proba.cpp`: `testLearning`

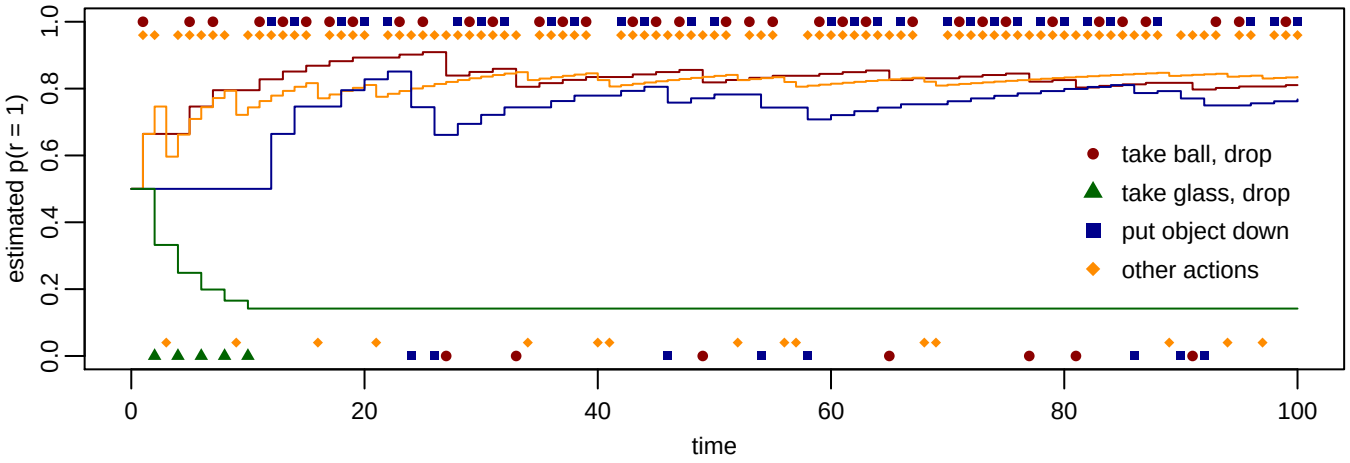

Figure 2: Simulation of learning the success rate for 100 plan-act loops. The utilities and success rates in the task-execution simulator are set as in Table 1, and $\varepsilon = 0.01$ and $\lambda = 1/10$. The top marks indicate a successful action, and the bottom marks indicate a failed action. We can see that the probability of success of dropping a glass quickly decays.

Utilities of actions	
`takeBall`	1
`takeGlass`	1
`dropObject`	5
`putObjectDown`	1

Success rates of context. actions	
`takeBall, dropObject`	0.9
`takeGlass, dropObject`	0.1
`putObjectDown`	0.8
default	0.9

Table 1: Utilities and success rates for experiment 1.

2. Learning the success rate

To validate the fact that the exponential forgetting model provides adaptation to HTN planning, we have built a plan-act simulation experiment. For 100 trials, Planner 9 must plan `fetchObject`(o) with o being alternatively "glass" and "ball". Initially, Planner 9 knows the contextualized actions but not their success rates. For every action, we initialize the forgetting model with $a = 1$ and $b = 2$ to set an uninformative prior of 50 % success. The utilities and contextualized actions are like in Table 1.

Figure 2 shows the action outcomes and estimated success rates resulting from the simulation. We see that initially, Planner 9 tries to drop objects rather than to put them down, because the former has a higher utility. However, in the case of glass, this results in failures. Therefore, Planner 9 switches to putting the glass down, which succeeds most of the times. This experiment shows that the plan-act loop converges to an estimation of the true rates which is sufficiently good to always select the plan of maximum expected utility. Because we select actions deterministically according to the mode of the probability distribution, the process does not recover the true success rates.

Discussion

The method we present in this paper is best suited when there are different ways to implement a given task and conditions specific to a particular environment affect what is the best way. It still has some limitations, which we discuss in this section, pointing out ways to alleviate them.

The proposed adaptation mechanism does not take the values of the variables into account. This means that we consider that the probability of success is independent from the objects involved in the action. It is of importance, since this allows the planner to reason in the lifted space, which is much smaller than the grounded space. However, it might be limiting when the values of the grounded variables have a strong influence on the action execution. For such cases, we advise to add properties through the perception stage reflecting this variability, and then to differentiate the actions following these properties. With such measures, our approach is suitable to handle these cases.

Until now, we have only considered the utility of plans that consist of a linear sequence of actions, which are typically the case for single robots. However, our reasoning still holds for plans that are DAG of partially ordered actions. Indeed, the utility function does not change, neither does the consideration that a failure of any action results in a 0 utility. However, the dependency lists might become complicated and hard to define. Yet this is a difficulty for the formulation of the a priori knowledge, not a limitation of the proposed algorithm.

We have defined the utility solely as a function of the action type. However, once an action is grounded, it is often possible to compute its utility more precisely. Equation 4 shows that as long as the utility of the grounded action is smaller than $\widehat{\theta}$, the heuristic function is admissible. Thus it is possible to employ the additional information from grounded actions, if utilities are properly defined. For example, if an action consists in going from one location to another, the utility of the grounded action can be the utility of the action type, multiplied by a factor starting from 1 and decreasing with the distance.

In general, the choice of which amount of knowledge to hand-code in a robot and what to leave to learning is not easy.

If too much is hand-coded, the robot lacks adaptivity; but if too much is left to learning, the robot learns too slowly or even not at all. In the extreme case when everything is probabilistic and subject to learning, the planning becomes a partially observable MDP (Sridharan, Wyatt, and Dearden 2008). In this case, even modest-size problems are often intractable with physical robots. We think that our approach of defining the HTN domain and the dependency lists by hand and letting the robot learn online the success rate of alternatives is a good compromise for current hardware. However, the difficulty to hand-code domain knowledge is a limiting factor. Recent results on learning from plan traces (Zhuo et al. 2010) or on developmental systems (Mugan and Kuipers 2011) might help with this respect, and are compatible with the method we propose. Moreover, future developments might shift this balance, probably in the direction of learning, especially if robots can share the acquired knowledge with their peers (Waibel et al. 2011). Indeed this might help to overcome the limited amount of available training experience, which we think is one of the factors currently limiting real-world deployment of complex high-level learning algorithms in robotics.

Conclusion

In this paper, we have proposed a method that adds adaptation capabilities to robots using HTN planning. We have proved that the planner finds the plan of maximal expected utility, while retaining its lifting capability and efficient heuristic-based search. Our method has been validated experimentally through its implementation in our open-source HTN planner and the simulation of a plan-act loop. Possible improvements such as taking into account information about the objects of action were also discussed.

In general, we think that the idea of using the HTN domain to constrain the space of possibles, and then learn on the constrained space, is interesting and might bring a new life to the crossroad of automated planning, artificial intelligence, machine learning, and robotics.

Acknowledgements

We thank the reviewers for their constructive comments. We thank Ève Lasserre for proof-reading the manuscript. This work was supported by the Swarmanoid (FP7-IST-FET 022888) and myCopter (FP7-AAT-2010-RTD-1) European projects.

References

Alami, R.; Clodic, A.; Montreuil, V.; Sisbot, E.; and Chatila, R. 2006. Toward human-aware robot task planning. In *AAAI Spring Symp'To boldly go where no human-robot team has gone before*, 39–46.

Beaudry, E.; Kabanza, F.; and Michaud, F. 2005. Planning for a mobile robot to attend a conference. *Advances in Artificial Intelligence* 199–213.

Belker, T.; Hammel, M.; and Hertzberg, J. 2003. Learning to optimize mobile robot navigation based on htn plans. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, 4136–4141. IEEE.

C. Boutilier, T. D., and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research (JAIR)* 11:1–94.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Morgan Kaufmann Publishers.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4:100–107.

Kuter, U., and Nau, D. 2005. Using domain-configurable search control for probabilistic planning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, 1169–1175.

Li, N.; Kambhampati, S.; and Yoon, S. 2009. Learning probabilistic hierarchical task networks to capture user preferences. In *Proceedings of the 21th International Joint Conference on Artificial Intelligence (IJCAI)*.

Magnenat, S.; Voelkle, M.; and Mondada, F. 2009. Planner9, a HTN planner distributed on groups of miniature mobile robots. In *Intelligent Robotics and Applications, Proceedings of the Second International Conference on Intelligent Robotics and Application*, volume 5928 of *Lecture Notes in Computer Science*, 1013–1022. Springer.

Magnenat, S. 2010. *Software integration in mobile robotics, a science to scale up machine intelligence*. Phd thesis, EPFL, Lausanne, Switzerland.

Meneguzzi, F.; Tang, Y.; Sycara, K.; and Parsons, S. 2011. An approach to generate MDPs using HTN representations. In *IJCAI Workshop on Decision Making in Partially Observable Uncertain Worlds: Exploring Insights from Multiple Communities*.

Morisset, B., and Ghallab, M. 2008. Learning how to combine sensory-motor functions into a robust behavior. *Artificial Intelligence* 172(4–5):392–412.

Mugan, J., and Kuipers, B. 2011. Autonomous learning of High-Level states and actions in continuous environments. *IEEE Transactions on Autonomous Mental Development* PP(99):1.

Parr, R., and Russell, S. 1997. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems* 10:1043–1049.

Russell, S.; Norvig, P.; Canny, J.; Malik, J.; and Edwards, D. 2003. *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, NJ. second edition.

Schoemaker, P. 1982. The expected utility model: Its variants, purposes, evidence and limitations. *Journal of Economic Literature* 529–563.

Schulz, S. 2002. A comparison of different techniques for grounding near-propositional cnf formulae. In *Proceedings of the 15th International FLAIRS Conference*, 72–76.

Sridharan, M.; Wyatt, J.; and Dearden, R. 2008. HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Haussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; Schiessle, B.; Tenorth, M.; Zweigle, O.; and van de Molengraft, R. 2011. Roboearth. *Robotics Automation Magazine, IEEE* 18(2):69–82.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174(18):1540–1569.