# Case-Based, Decision-Theoretic, HTN Planning

Luís Macedo[1,2], Amílcar Cardoso[2]

[1] Department of Informatics and Systems Engineering, Engineering Institute, Coimbra Polytechnic Institute,
3030-199 Coimbra, Portugal
lmacedo@isec.pt
http://www2.isec.pt/~lmacedo
[2] Centre for Informatics and Systems of the University of Coimbra, Department of Informatics, Polo II,
3030 Coimbra, Portugal
{lmacedo, amilcar}@dei.uc.pt

**Abstract.** This paper describes ProCHiP, a planner that combines CBR with the techniques of decision-theoretic planning and HTN planning in order to deal with uncertain, dynamic large-scale real-world domains. We explain how plans are represented, generated and executed. Unlike in regular HTN planning, ProCHiP can generate plans in domains where there is no complete domain theory by using cases instead of methods for task decomposition. ProCHiP generates a variant of a HTN - a kind of AND/OR tree of probabilistic conditional tasks - that expresses all the possible ways to decompose an initial task network. As in Decision-Theoretic planning, the expected utility of alternative plans is computed, although in ProCHiP this happens beforehand at the time of building the HTN. ProCHiP is used by agents inhabiting multi-agent environments. We present an experiment carried out to evaluate the role of the size of the case-base on the performance of the planner. We verified that the CPU time increases monotonically with the case-base size while effectiveness is improved only up to a certain case-base size.

## 1 Introduction

Hierarchical Task Network (HTN) planning [4] is a planning methology that is more expressive than STRIPS-style planning. Given a set of tasks that need to be performed (the planning problem), the planning process decomposes them into simpler subtasks until *primitive tasks* or actions that can be directly executed are reached. *Methods* provided by the domain theory indicate how tasks are decomposed into subtasks. However, for many real-world domains, sometimes it is hard to collect methods to completely model the generation of plans. For this reason an alternative approach that is based on cases of methods has been taken in combination with methods [14].

Real-world domains are usually dynamic and uncertain. In these domains actions may have several outcomes, some of which may be more valuable than others. Planning in these domains require special techniques for dealing with uncertainty. Actually, this has been one of the main concerns of planning research in recent years, and several decision-theoretic planning approaches have been proposed and used successfully, some based on the extension of classical planning and others on Markov-Decision Processes (see [3, 9] for a survey). In these decision-theoretic planning frameworks actions are usually probabilistic conditional actions, preferences over the outcomes of the actions is expressed in terms of an utility function, and plans are evaluated in terms of their Expected Utility (EU). The main goal is to find the plan or set of plans that maximizes an EU function [17], i.e., to find the optimal plan. However, this might be a computationally complex task.

In this paper we present ProCHiP (Probabilistic, Case-based, Hierarchical-task network Planning), a planner that combines CBR with the techniques of decision-theoretic planning and HTN planning in order to deal with uncertain, dynamic large-scale real-world domains. Unlike in regular HTN planning, we don't use methods for task decomposition, but instead cases of plans. ProCHiP generates a variant of a HTN - a kind of AND/OR tree of probabilistic conditional tasks - that expresses all the possible ways to decompose an initial task network. The EU of tasks and of the alternative plans is computed beforehand at the time of building the HTN. ProCHiP is implemented in artificial cognitive agents inhabiting multi-agent environments.

The next section describes the features of ProCHiP related with plan representation, generation and execution. Subsequently, we present an experiment in which we evaluate the influence of the case-base size on the performance of ProCHiP. Finally, we present related work, discuss our findings and present conclusions.

## 2 Case-Based, Decision-Theoretic, HTN Planning

### 2.1 Representation

Within our approach we may distinguish two main kinds of plans: concrete plans, i.e., cases of plans, and abstract plans (for more details about abstraction in CBR see for instance [2]). Concrete plans and abstract plans are interrelated since concrete plans are instances of abstract plans and these are built from concrete plans. Since the concept of abstract plan subsumes the concept of concrete plan, let us first describe the representation issues related with abstract plans and then present the main differences between concrete plans and abstract plans.

We represent abstract plans as a hierarchy of tasks (a variant of HTNs [4, 15]) (Fig. 1). Formally, an abstract plan is a tuple $AP = <T, L>$, where $T$ is the set of tasks and $L$ is the set of links. More precisely, we represent an abstract plan by a hierarchical graph-structured representation comprising tasks (represented by the nodes) and links (represented by the edges). We adopted the adjacency matrix approach to represent these graphs [11]. The links may be of hierarchical (abstraction or decomposition), temporal, utility-ranking or adaptation kind. This structure has the form of a planning tree [10], i.e., it is a kind of AND/OR tree that expresses all the possible ways to decompose an initial task network. Like in regular HTNs, this hierarchical structure of a plan comprises primitive tasks or actions (non-decomposable tasks) and non-primitive tasks (decomposable or compound tasks). Primitive tasks correspond to the leaves of the tree and are directly executed by the agent, while compound tasks denote desired changes that involve several subtasks to accomplish it (e.g., the leaf node *driveTruck* of Fig. 1 is a primitive task, while *inCityDel* is a compound task). The decomposition of a compound task into a sequence of subtasks is represented by linking the compound task to each subtask by a hierarchical link of type decomposition (denoted by *dcmp*). This corresponds to an AND structure. In addition, a hierarchical plan may also include special tasks in order to express situations when a decomposable task has at least two alternative decompositions. Thus, these special tasks are tasks whose subtasks are heads of those alternative decompositions. We called abstract tasks (e,g., the root task *transport* of Fig. 1) to those special decomposable tasks because they may be instantiated by one of their alternative subtasks. Thus, they are a kind of abstractions of their alternative instances. The subtasks of an abstract task may themselves be abstract tasks. The decomposition of abstract tasks into several alternative instances is expressed by linking the abstract task to each subtask by a hierarchical link of type abstract (denoted by *abst*). This corresponds to an OR structure.

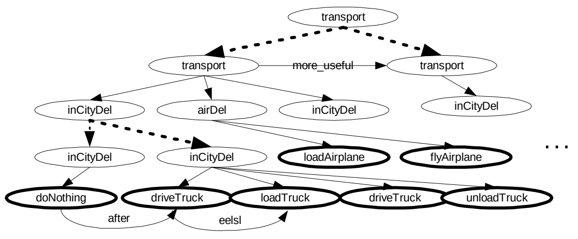

**Fig. 1.** Example of an abstract plan. Primitive tasks are represented by thick ellipses while non-primitive tasks are represented by thin ellipses. Dashed arrows represent *abst* links, while thin arrows represent *dcmp* links

As we said, in addition to hierarchical links that express AND or OR decomposition (*dcmp* and *abst*), there are also temporal, utility-ranking and adaptation links between tasks. Temporal links are just like in regular HTNs. We followed the temporal model introduced by [1]. Thus, links such as *after*, *before*, *during*, *overlap*, etc., may be found between tasks of an abstract plan. Utility-ranking links (denoted by *more_useful*) are used between subtasks of abstract tasks in order to express a relation of order with respect to their EU, i.e., the head tasks of the alternative decompositions of a given abstract task are ranked according to the EU of their decompositions. Adaptation links [8] are useful to generate an abstract plan from several cases of plans. They explain how tasks and their components are related in a plan and therefore they explain how to adapt portions of cases of plans when they are reused to construct an abstract plan. For instance, the link *eelsl* (*end location equal to start location*) means that the start location of the truck when *loadTruck* takes place is equal to the end location of the truck when *driveTruck* is executed.

A task $T$ is both conditional and probabilistic (e.g., [3, 6, 24]). This means each primitive task has a set of conditions C={$c_1$, $c_2$, ..., $c_m$} and for each one of these mutually exclusive and exhaustive conditions, $c_i$, there is a set of alternative effects $\varepsilon^i$={$< p^i_1, E^i_1 >$, $< p^i_2, E^i_2 >$, ..., $< p^i_{n_i}, E^i_{n_i} >$}, where $E^i_j$ is the $j$th effect triggered with probability $p^i_j \in [0,1]$ by condition $c_i$ (i.e., $P(E^i_j \mid c_i) = p^i_j$), and such that $\sum_{j=1}^{n_i} p^i_j = 1$. Fig. 2 presents the structure of a task. The probabilities of conditions are represented in that structure although we assume that conditions are independent of tasks. Thus, $P(c_i|T)=P(c_i)$. The main reason for this is to emphasize that the EU of a task, in addition to the probability of effects, depends on the probability of conditions too. In addition to conditions and effects, a task has other information components. Formally, a task (primitive or not) may be defined as follows.

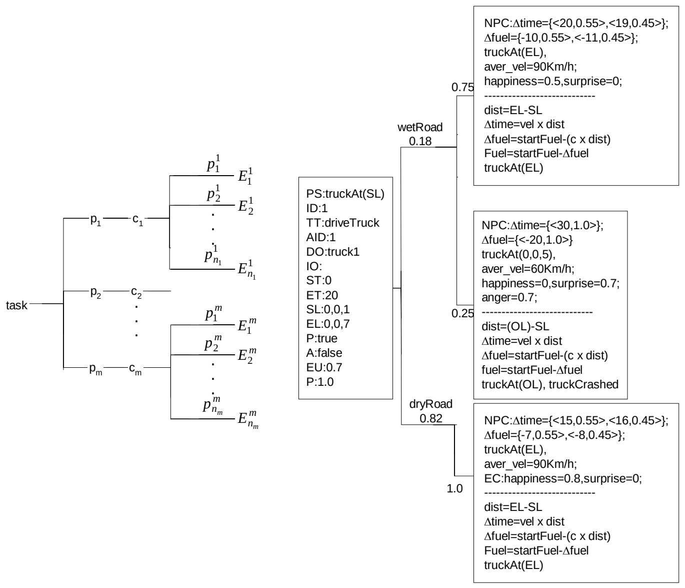

**Fig. 2.** Schematic representation of a task in an abstract plan: general form and example. Horizontal, dashed lines inside the boxes describing the effects separate non-procedural from procedural components

**Definition 1**. A *task* is a tuple <*PS*, *ID*, *TT*, *AID*, *DO*, *IO*, *ST*, *ET*, *SL*, *EL*, *PR*, *A*, *EP*, *EU*, *P*>, where: *PS* is the set of preconditions that should be satisfied so that the task can be executed; *ID* is the task's identifier, i.e., an integer that uniquely identifies the task in a plan; *TT* is the task category (e.g., *driveTruck*, *transport*); *AID* is the identifier of the agent that is responsible for the execution of the task[1]; *DO* is the direct object of the task, i.e., the identifier of the entity that was subjected to the task directly (e.g., for a task of type *driveTruck*, the direct object is the object - its identifier - to be driven; for a task of type *transport*, the direct object is the entity that is transported – for instance, a package); *IO* is the indirect object of the task, i.e., the answer to the question "To whom?" (e.g., for a task of type *give*, the indirect object is the entity that receives the entity (the direct object) that is given – for instance, the person who receives money); *ST* is the scheduled start time of the task; *ET* is the scheduled end time of the task, *SL* is the start location of the agent that is responsible for executing the task; *EL* is the end location of the agent that is responsible for the execution of the task; *PR* is a boolean value that is true when the task is primitive; *A* is a boolean value that is true when the task is abstract (for primitive tasks it is always false); *EP* is the set of alternative probabilistic conditional effects of the task, i.e., *EP* $= \{<c_i,\varepsilon^i>: 1 =< i <= m\}$; *EU* is the EU of the task; *P* is the probability of the task (this is always 1.0 for every task except the heads of alternative decompositions of an abstract task as we'll explain below).

Although non-primitive tasks are not directly executed by an agent, they are represented like primitive tasks. Some of the components are meaningful only for primitive tasks. However, others such as the set of alternative probabilistic conditional effects are essential for the ranking of the alternative decompositions of the abstract tasks in terms of the EU. That is why the set of conditional probabilistic effects and other meaningful properties are propagated upward through the hierarchy from the primitive tasks to the non-primitive tasks (this propagation will be explained below).

Each effect (see Fig. 2) is composed of a few components of several kinds such as temporal, emotional (notice that in our work, agents are of cognitive kind with a module of emotions and other motivations included in their architecture [12]), etc. These components may be of two kinds: non-procedural and procedural. The non-procedural component refers to the data collected from previous occurrences of the effect (contains the duration of the task, the emotions and respective intensities felt by the agent, the fuel consumed, etc., in previous executions of the task as stored in cases of plans). The procedural component refers to the process through which the temporal, emotional and other kinds of data may be computed (contains descriptions or rules of how to compute the components).

Formally, an effect may be defined as follows.

[1] The planner is used by agents inhabiting multi-agent environments.

**Definition 2**. An *effect* is a tuple <*ID*, *EC*, *EU*, *P*, *NPC*, *PC*>, where: *ID* is the identifier of the effect, i.e., an integer value that uniquely identifies the effect in the list of effects of the task; *EC* is the effect category to which it belongs (like tasks, effects are classified into categories); *EU* is the utility value (EU value for the case of tasks in abstract plans) of the effect; *P* is the probability value of the effect, i.e., the relative frequency of the effect (this gives us the number of times the effect occurred given that the task and the condition that triggers it occurred); *NPC* is the non-procedural component; *PC* is the procedural component.

Cases of plans share most of the features of abstract plans because they are also represented hierarchically. The major differences are: unlike abstract plans, cases of plans don't have OR structures and consequently don't have abstract tasks; the primitive tasks have a probability of 1.0 (otherwise they won't belong to the case) and can only have a conditional effect since the conditions are mutually exclusive and exhaustive. Notice that, although a non-primitive task of a case of a plan may exhibit an effect, this is not relevant, since in the real world only the primitive tasks are executed. However, the way a non-primitive task was decomposed is of primary importance for the generation of abstract plans, as we will explain in the following section. Fig. 3 shows an example of two cases of plans, which are instances of the abstract plan presented in Fig. 1.

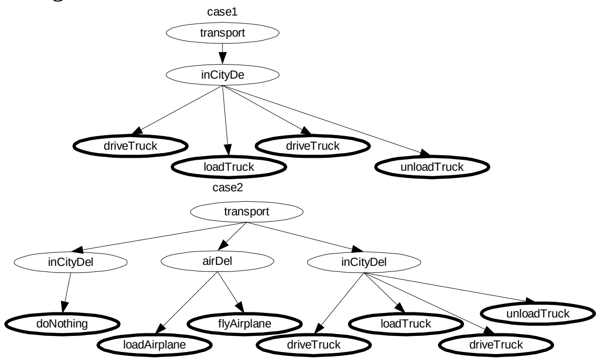

**Fig. 3.** Example of a case-base with two concrete plans (instances of the abstract plan of Fig. 1)

## 2.2 Plan Generation

Since the planner is used by an agent that is part of a multi-agent environment, in order to solve a planning problem, the agent should have in memory the information of the initial state of the environment. This comprises a three-dimensional metric map of the environment [21] in which inanimate and other animate agents are spatially represented. Fig. 4 presents an example of a metric map that represents an initial state of the world.

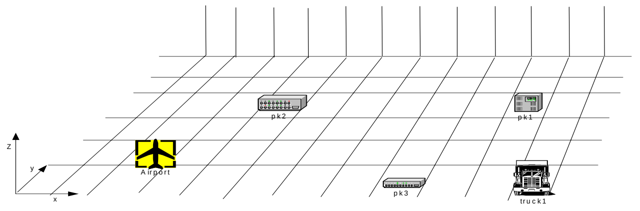

**Fig. 4.** Example of the metric map of an initial state of the environment in the logistics domain. It comprises: one truck (*truck1*) located at coordinates (11,0,0); three packages, *pk1*, *pk2* and *pk3*, located at, respectively, (10,3,0), (4,3,0), and (8,0,0); and one plane located at the airport with coordinates (2,1,0)

A problem is an initial and incomplete HTN, i.e., a set of goal tasks. Planning is a process by which that initial HTN is completed resulting in an abstract plan ready to be executed and incorporating alternative courses of action, i.e., it includes replanning procedures. Roughly speaking, this involves the following steps (the respective algorithms are presented in later figures): first, the structure of the abstract plan (HTN) is built based on cases of past plans (this is closely related to the regular HTN planning procedure); then the conditional effects, probabilities as well as the EU are computed for the primitive tasks of this abstract plan based on the primitive tasks of cases of past plans; finally, these properties (conditional effects and respective probabilities, and EU) are propagated upward in the HTN, from the primitive tasks to the main task of the HTN. Fig. 5 presents this algorithm.

```
Algorithm CONSTRUCT-ABSTRACT-PLAN(abstPlan)
    abstPlan ← BUILD-STRUCTURE(abstPlan)
    primTasks ← GET-PRIMTASKS(abstPlan)
    primTasksAllPlanCases← GET-PRIMTASKS-ALL-PLAN-CASES()
    COMPUT-PRIMTASKS-PROPS(primTasks,primTasksAllPlanCases)
    abstPlan←PROPAGAT-PROPS-UPWARD(primTasks,abstPlan)
    return abstPlan
end
```

**Fig. 5.** Algorithm for the construction of an abstract plan

Much like regular HTN planning, building the structure of the abstract plan (algorithm of Fig. 7) is a process by which the initial HTN is completed by recursively decomposing its compound tasks. Unlike regular HTN planning, within our approach the domain theory (methods and operators in regular HTN planning) is confined to a finite set of actions/operators. Thus there are no explicit methods to describe how to decompose a task into a set of subtasks. Actually, methods are implicitly present in cases of past plans (see [14] for a similar approach). This is particularly useful in domains where there is no theory available. Therefore, the process of decomposing a task into subtasks is case-based and is performed as follows. Given a task, the possible alternative decompositions (task and its subtasks, as well as the links between them) are retrieved from cases of past plans. Two situations might happen. If there are more than one alternative decomposition, the given task is set as abstract and the set of decompositions are added to the HTN, linking each head task to the abstract task through a hierarchical link of type *abst*. Thus, these head tasks are now the subtasks of the abstract task (see Fig. 6 for an illustration of this process). The result is a decomposition with an OR structure. On the other hand, if only one decomposition is retrieved, its subtasks are added as subtasks of the given task, linked by a hierarchical link of type *dcmp* (see Fig. 6 for an illustration of this process). This corresponds to an AND structure. Whether a single decomposition or multiple decompositions are retrieved, the addition of it/them comprises an adaptation process [8], i.e., the retrieved decomposition(s) is/are changed if necessary so that it/they is/are consistent with the rest of the HTN. Each adaptation link triggers a process. Thus, for instance, the adaptation link *ea* (*equal AID*) in Fig. 6 indicates that the tasks *transport* and *inCityDel* have the same component *AID* , i.e., they are executed by the same agent. This means that the *AID* component of those tasks retrieved from past plans are changed so that it refers to the agent whose identifier is referred to by the *AID* of *transport* belonging to the current abstract plan.

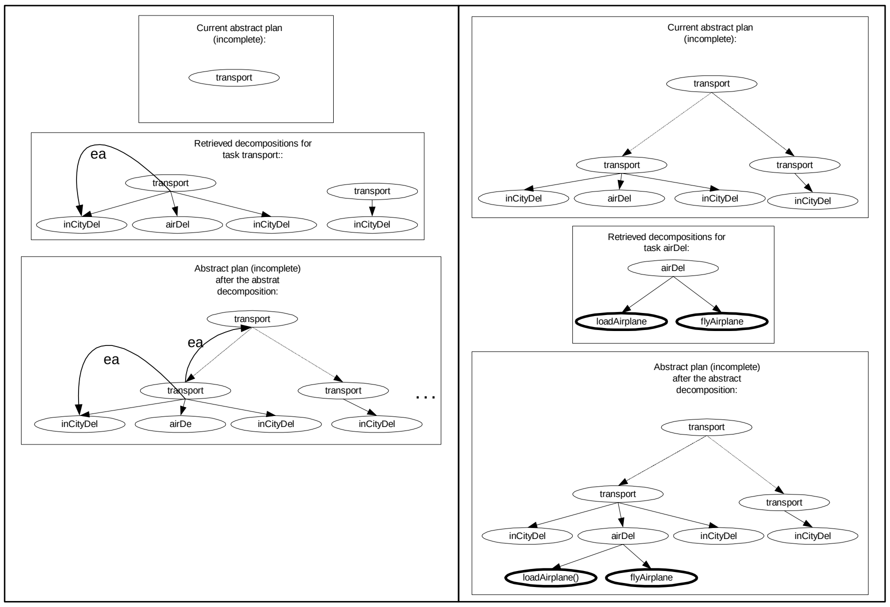

**Fig. 6.** Illustrative example of: an OR-decomposition of an abstract task (on the left); an AND-decomposition of a regular compound task (on the right)

The process of building the HTN ends when there are no more compound tasks to decompose, i.e., when the leaves of the tree are primitive tasks, or when there are no available decompositions in the case-base for at least one compound task.

```
Algorithm BUILD-STRUCTURE(abstPlan,CB)
    goalTasks ← getLeafTasks(AbstPlan)
    taskQueue ← goalTasks
    while taskQueue ≠ ∅
      task ← popFrontTask(taskQueue)
      listAlternDcmps ← getListAlternDcmps(task, CB)
      if size(listAlternDcmps) > 1
        task type ← "abstract"
        for each decomposition in listAlternDcmps do
          headTask ← getHeadTask(decomposition)
          $P(headTask/task) \leftarrow \frac{S_r(headTask \cap task)}{S_r(task)}$
          adapt(headTask, task, "abst")
          insert headTask in AbstPlan; link it to task by "abst" link
          subtasksDcmp ← getSubTasks(decomposition)
          for each subtask (with adaptationLinks from headTask) in subtasksDcmp do
              adapt(subtask, headTask, adaptionLinks)
              for each othertask with adaptationLinks to subtask do
                adapt(subtask, othertask, adaptionLinks)
              end for each
              if notPrimitive(subtask) then
                insertTask(subtask, taskQueue)
                $P(subtask/headTask) \leftarrow \frac{S_r(subtask \cap headTask)}{S_r(headTask)} = 1.0$
              insertTask(subtask, AbstPlanStructure)
          end for each
          copy all links from decomposition to AbstPlan
        end for each
      else
        subtasksDcmp ← getSubTasks(decomposition)
        for each subtask (with adaptationLinks from subTask) in subtasksDcmp do
          adapt(subtask, task, adaptionLinks)
          for each othertask with adaptationLinks to subtask do
              adapt(subtask, othertask, adaptionLinks)
          end for each
          if notPrimitive(subtask) then insertTask(subtask, taskQueue)
          $P(subtask/task) \leftarrow \frac{S_r(subtask \cap task)}{S_r(task)} = 1.0$
          insertTask(subtask, AbstPlan)
        end for each
        copy all links from decomposition to abstPlan
      endif
    endwhile
    return abstPlan
end
```

**Fig. 7.** Algorithm for constructing the structure of an HTN

Within our approach, a task belonging to an HTN has a probability value associated to it. This value expresses the probability of being executed given that its ancestor is executed. Thus, this probability is actually a conditional probability. Obviously, the probability of a task belonging to a case of a past plan is always 1.0 because it was executed (otherwise it won't belong to the case). The probability of the tasks belonging to an abstract plan is computed during the process of building the HTN as follows. Given the $i^{th}$ subtask, $ST_i$, of a task $T$ both belonging to an abstract plan, the probability of $ST_i$ be executed given that $T$ is executed is given by the conditional probability formula $P(ST_i / T) = \frac{P(ST_i \cap T)}{P(T)}$. According to frequency interpretation of probability, this is estimated by: $P(ST_i / T) = \frac{P(ST_i \cap T)}{P(T)} = \frac{S_r(ST_i \cap T)}{S_r(T)}$, which expresses the number of times $ST_i$ and $T$ occurred together in the total amount of times $T$ occurred, or in the context of HTN planning, this expresses the number of times $ST_i$ was subtask of $T$ in the total amount of times $T$ was the task decomposed in past HTN plans in $r$ decompositions. When $ST_i$ is not a head of an alternative decomposition in the new plan (i.e., when $T$ is not an abstract task), it means that $T$ was always decomposed in the same way in past plans, i.e., into the same subtasks, which means $ST_i$ occurred always when $T$ occurred, otherwise $ST_i$ won't be subtask of $T$. Thus, in this situation, the numerator and denominator of the above equation are equal and therefore $P(ST_i/T)$=1.0. However, when $ST_i$ is a head of an alternative decomposition, it means there were more than one way to decompose $T$ in past plans, the decomposition headed by $ST_i$ being one of them. Thus, counting the number of times the decomposition headed by $ST_i$ was taken to decompose $T$, i.e., the number of times $ST_i$ instantiated $T$, $S_r(ST_i \cap T)$, in all past plans and dividing this number by the number of times $T$ was decomposed, i.e., $S_r(T)$, yields the value for $P(ST_i/T)$ for this situation.

After the abstract HTN is built, the conditional effects (and respective probabilities) and the EU are computed for the primitive tasks based on the past occurrences of those primitive tasks (notice that the probability of the tasks has already been computed during the process of building the HTN as described above). Remember that tasks (either primitive or not) have a list of possible effects each one associated with a probability value (see Fig. 2). Thus, this is once more a case-based process that is carried out as described by the algorithm of Fig. 8.

After the primitive tasks have their properties computed based on cases of past plans, these properties are propagated bottom-up (from primitive to non-primitive tasks), from the subtasks to the task of a decomposition and from the subtasks (heads of alternative decompositions) to the abstract task of an abstract decomposition. The goal of this propagation is twofold: to complete the non-primitive tasks so that they can be ranked according to their EU when they are heads of alternative decompositions, and to know the overall EU of the abstract plan which is given by the EU of the main task of the plan. Fig. 9 presents the algorithm for the propagation of properties. Function PROPAGAT-PROPS-ABST and PROPAGAT-PROPS-DCMP relies heavily on the notions of inter-action abstraction described in [6].

```
Algorithm COMPUT-PRIMTASKS-PROPS(primTasks, primTasksAllPlanCases)
    for each primTask in primTasks do
        taskList ← {i: i ∈ primTasks and i is of the same type of primTask}
        condEffectList ← ∅
        for each task in taskList do
            condEffectListTask ← $\bigcup_{i=1}^{m} \langle c_i, E_i \rangle$, m is the number of conditional effects of task,
Ei={ $E^{i}_{a_{task}}$ }
            condEffectList ← condEffectList ∪ condEffectListTask
        end for each
        genCondEffectList ← GENERALIZE-COND-EFFECT-LIST(condEffectList )
        set the conditional effects of primTask with genCondEffectList
        EU(primTask)← $\sum_i P(\langle c_i, \varepsilon^i \rangle) \times EU(\langle c_i, \varepsilon^i \rangle) = \sum_i P(c_i) \times EU(\varepsilon^i)$
    end for each
    return primTasks
end
```

**Fig. 8.** Algorithm for computing the conditional effects (and respective probabilities) and the EU of primitive tasks

```
Algorithm PROPAGAT-PROPS-UPWARD(primTasks, mainTask, absPlan)
 if primitive(mainTask) nothing to do
 else
    subTasks ← getSubTasks(mainTask)
    for each subTask in subTasks do
        PROPAGAT-PROPS-UPWARD(primTasks, subTask, abstPlan)
    end for each
    if abstract(mainTask) then
        PROPAGAT-PROPS-ABST(subTasks, mainTask, mainTask1)
        replace mainTask by mainTask1 in abstPlan
    else
        PROPAGAT-PROPS-DCMP(subTasks, mainTask, mainTask1)
        replace mainTask by mainTask1 in abstPlan
    endif
 endif
end
```

**Fig. 9.** Recursive algorithm for propagating properties upward, from primitive tasks to all non-primitive tasks

### 2.3 Plan Execution and Replanning

Finding the optimal plan in ProCHiP consists simply of traversing the abstract plan, selecting the most EU subtask of an abstract task. Backtracking occurs when an alternative decomposition fails execution. In this case, the next alternative decomposition that follows the previous in the EU ranking is selected for execution.

### 2.4 Retaining Plans

As mentioned in section 2.3, executing a plan corresponds to an instantiation of an abstract plan. After a plan is executed, the instantiation that was actually executed is stored in memory for future reuse. In addition, the abstract plan is also stored in memory. This way, it might be useful in the future since it might avoid an unnecessary process of generating it again.

## 3 Experiment

We conducted an experiment in order to evaluate the role played by the case-base size on the performance of ProCHiP. Given a kind of goal task such as *transport*, we constructed 5 case-bases, ranging in size from 1 to 5 cases of plans, each case describing a different way of achieving the specified goal task. For each one of these case-bases, we ran ProCHiP with 10 different goal tasks of type *transport*. The CPU time taken by the planner to build an abstract plan for the specified goal task was measured. In addition, the number of those 10 goal tasks solved successfully was computed, as well as the number of tasks in those abstract plans. The results are plotted in Fig. 10.

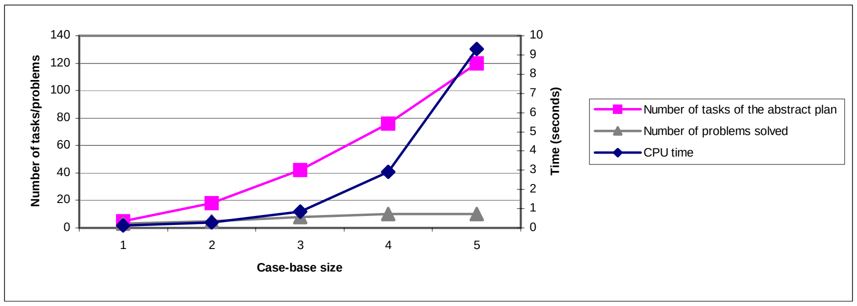

**Fig. 10.** Performance of ProCHiP with case-bases varying in size

The CPU time taken by the planner to build the abstract plan increases monotonically with the size of the case-base. The same happens with the number of tasks in the abstract plan. An interesting result is noticed with the number of problems successfully solved. With 3 cases, the planner is able to find a correct solution for 80% of the goal tasks, and with 4 or more cases the effectiveness is 100%, which means the addition to the case-base of more cases for solving problems of this kind (*transport*) seems to decrease the efficiency of the planner, because the effectiveness is not and could not be increased (100%). This issue is related to the utility problem (e.g., [5, 13, 19]) and case-base maintenance (e.g., [18, 20]).

## 4 Related Work

Our work is closely related to HTN planning. This methology has been extensively used in planning systems such as UMCP [4], SHOP and SHOP2 [16]. Unlike these planners, ProCHiP don't use methods as part of the domain theory for task decomposition, but instead methods that are implicitly included in cases that describe previous planning problem solving experiences. SiN [14] also uses a case-based HTN planning algorithm, in which cases are instances of methods.

Learning hierarchical plans or HTNs is still rarely addressed by the machine learning community, although there are a few exceptions. Garland, Ryall and Rich [Garland, 2001 #161 infer task models from annotated examples, i.e., through demonstration by a domain expert. [7]. van Lent and Laird [22] used a learning-by-observation technique which involves extracting knowledge from observations of an expert performing a task and generalizes this knowledge to a hierarchy of rules. Xu and Muñoz [23] use an algorithm that gather and generalize information on how domain experts solve HTN planning problems.

Among decision-theoretic planners, DRIPS [6] is probably the most closely related to ProCHiP. Actually, DRIPS shares a similar representation approach for abstract plans (an abstraction/decomposition hierarchy) and for actions. Besides, it also returns the optimal plan according to a given utility function. However, in contrast to DRIPS, in ProCHiP the variant of a HTN that represents abstract plans is automatically built from cases and not given as input for the planning problem. Besides, it includes temporal, utility ranking and adaptation links in addition to decomposition links. Another major difference is that, in ProCHiP, the EU of tasks and of alternative plans is computed when the abstract plan is built, while in DRIPS this occurs when the optimal plan is searched. In ProCHiP, there is the possibility of computing the EU of tasks based on the non-procedural component of their effects, which avoids some additional computations at the cost of being less accurate. Moreover, finding the optimal plan in ProCHiP consists simply of traversing the HTN with backtracking (or replanning) points located at the subtasks of an abstract task. In ProCHiP the propagation of properties upward in the hierarchy is closely related with the approach taken in DRIPS for abstracting actions [6]. A propagation of properties in the planning tree, bottom-up and left-to-right, is also used in GraphHTN [10] in order to improve the search algorithm.

## 5 Conclusions

We presented ProCHiP, a planner that combines CBR with the techniques of decision-theoretic planning and HTN planning in order to deal with uncertain, dynamic large-scale real-world domains. We conducted an experiment in order to evaluate the dependence of the time taken by ProCHip to build abstract plans on the size of a case-base containing cases representing implicit methods. We concluded that the CPU time increases monotonically with the case-base size. However, we also concluded that the case-base size improves the effectiveness of ProCHiP only up to a certain size. After that size the performance of ProCHiP corresponds to a low efficiency while the effectiveness is almost unaltered.

## Acknowledgments

The PhD of Luís Macedo is financially supported by PRODEP III.

## References

[1] J. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, pp. 832-- 843, 1983.
[2] R. Bergmann and W. Wilke, "On the Role of Abstraction in Case-Based Reasoning," in *Advances in Case-Based Reasoning - Proceedings of the Third European Workshop on Case-Based Reasoning*, vol. 1168, *Lecture Notes in Artificial Intelligence*, I. Smith and B. Faltings, Eds. Berlin: Springer Verlag, 1996, pp. 28-43.
[3] J. Blythe, "Decision-Theoretic Planning," *AI Magazine, Summer 1999*, 1999.
[4] K. Erol, J. Hendler, and D. Nau, "UMCP: A sound and complete procedure for hierarchical task-network planning," in *Proceedings of the International Conference on AI Planning Systems*, 1994, pp. 249-254.
[5] A. Francis and A. Ram, "The utility problem in case-based reasoning," in *Proceedings of the AAAI-93 Case-based Reasoning Workshop*, 1993.
[6] P. Haddawy and A. Doan, "Abstracting probabilistic actions," in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, 1994, pp. 270-277.
[7] O. Ilghami, D. Nau, H. Muñoz-Avila, and D. Aha, "CaMeL: Learning methods for HTN planning," in *AIPS-2002*, 2002.
[8] J. Kolodner, *Case-Based Reasoning*. San Mateo, CA: Morgan-Kaufmann, 1993.
[9] M. Littman and S. Majercik, "Large-Scale Planning Under Uncertainty: A Survey," in *Workshop on Planning and Scheduling for Space*, 1997, pp. 27:1--8.
[10] A. Lotem and D. Nau, "New advances in GraphHTN: Identifying independent subproblems in large HTN domains," in *Proceedings of the International Conference on AI Planning Systems*, 2000, pp. 206-215.
[11] L. Macedo and A. Cardoso, "Nested-Graph structured representations for cases," in *Advances in Case-Based Reasoning - Proceedings of the 4th European Workshop on Case-Based Reasoning*, vol. 1488, *Lecture Notes in Artificial Intelligence*, B. Smyth and P. Cunningham, Eds. Berlin: Springer-Verlag, 1998, pp. 1-12.

[12] L. Macedo and A. Cardoso, "SC-EUNE - Surprise/Curiosity-based Exploration of Uncertain and Unknown Environments," in *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing*. York, UK: University of York, 2001, pp. 73-81.

[13] S. Minton, "Qualitative results concerning the utility of explanation-based learning," *Artificial Intelligence*, vol. 42, pp. 363-391, 1990.

[14] H. Muñoz-Avila, D. Aha, D. Nau, L. Breslow, R. Weber, and F. Yamal, "SiN: Integrating Case-based Reasoning with Task Decomposition," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*. Seattle, WA: Morgan Kaufmann, 2001.

[15] D. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem, and S. Mitchell, "Total-order planning with partially ordered subtasks," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. Seattle, WA: Morgan Kaufmann, 2001.

[16] D. Nau, T. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379-404, 2003.

[17] S. Russel and P. Norvig, *Artificial Intelligence - A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[18] B. Smyth and M. Keane, "Remembering to forget: a competence preserving case deletion policy for CBR systems," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, 1995, pp. 377-383.

[19] B. Smyth and P. Cunningham, "The utility problem analysed," in *Advances in Case-Based Reasoning - Proceedings of the Third European Workshop on Case-Based Reasoning*, vol. 1168, *Lecture Notes in Artificial Intelligence*, I. Smith and B. Faltings, Eds. Berlin: Springer Verlag, 1996, pp. 392-399.

[20] B. Smyth and E. McKenna, "Building compact competent case bases," in *Proceedings of the Third International Conference on Case-Based Reasoning*. Berlin: Springer Verlag, 1999, pp. 329-342.

[21] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. San Mateo, CA: Morgan Kaufmann, 2002.

[22] M. van Lent and J. Laird, "Learning Hierarchical Performance Knowledge by Observation," in *Proceedings of the International Conference on Machine Learning*, 1999.

[23] K. Xu and H. Munõz-Avila, "CBM-Gen+: An algorithm for reducing case base inconsistencies in hierarchical and incomplete domains," in *Proceedings of the International Conference on Case-Based Reasoning*. Berlin: Springer, 2003.

[24] H. Younes, "Extending PDDL to model stochastic decision processes," in *Proceedings of the ICAPS-02 Workshop on PDDL*, 2003.