Planning over MDPs through Probabilistic HTNs

Yuqing Tang
Graduate Center
City University of New York
New York, USA
ytang@cs.gc.cuny.edu

Felipe Meneguzzi, Katia Sycara
Robotics Institute
Carnegie Mellon University
Pittsburgh, USA
meneguzz@cs.cmu.edu
katia@cs.cmu.edu

Simon Parsons
Brooklyn College
City University of New York
New York, USA
parsons@sci.brooklyn.cuny.edu

Abstract

In this paper, we propose a new approach to using probabilistic hierarchical task networks (HTNs) as an effective method for agents to plan under conditions in which their problem-solving knowledge is uncertain, and the environment is non-deterministic. In such situations it is natural to model the environment as a Markov Decision Process (MDP). We show that using Earley graphs, it is possible to bridge the gap between HTNs and MDPs. We prove that the size of the Earley graph created for any given HTN is bounded by the total number of tasks in the HTNs and show that from the Earley graph we can construct a plan for a given task that has the maximum expected utility when it is executed in an MDP environment.

1 Introduction

The ability to plan is crucial for autonomous agents. Designing planning-capable agents requires a domain expert to formally define the actions available to the agent so that when new plans are needed, a planning algorithm can generate new plans to achieve the agent's goals. To that effect, Hierarchical Task Networks (HTNs) have been widely used as the formalism of choice to design planning agents in *deterministic* domains. Its popularity is due not only to the way domains are defined, providing a convenient way to specify domain knowledge as hierarchically structured *recipes* (Nau, Ghallab, and Traverso 2004) closer to how humans structure their own knowledge, but also in the fact that solving an HTN is significantly faster than other deterministic planning methods. HTNs, however, are used mainly where there is no uncertainty over action outcomes, and autonomous agents that function in realistic environments must be able to create plans that take into consideration the possibility that actions could either fail or have unintended outcomes. Markov Decision Process (MDPs) have been widely studied as an elegant mathematical formalism to model *stochastic* domains (Bellman 2003).The solution to an MDP planning problem, given a stochastic state transition system and a reward function, is a *policy* that defines the optimal (or maximum expected utility) action for every state in the domain.

In this paper, we present work towards bridging the gap between HTNs and MDPs by performing maximum expected utility (MEU) planning on an HTN domain specified in terms of tasks that are hierarchically structured from high-level tasks down to executable actions through a library of *methods*. In order to accomplish this, we look at the HTN methods as if they were the rules of a context-free grammar and apply our own modified version of an Earley parser (Earley 1970) to generate a data structure known as *Earley state chart* (Stolcke 1995). Probabilistic Earley parsing is a dynamic programming technique widely used in the processing of natural language that has been adapted to parse sentences probabilistically in order to cope with the ambiguity inherent to human languages. The Earley state chart not only supports an efficient implementation of the Earley parsing algorithm but also provides a semantic understanding of the grammar parsing procedure on a given input string. These parsing algorithms can also be employed for posterior revisions of the probabilistic grammars, given a set of annotated parsing strings (Stolcke 1995). This class of algorithm performs a parallel top-down search over all possible grammar parses for a given input sentence, and its complexity is bounded by $O(N^3)$ where N is the number of words in the input (Earley 1970).

Our adaptation of Earley parsing for probabilistic HTN planning was inspired by earlier efforts relating task decomposition to grammar parsing (Barrett and Weld 1994), where the solution concept is taken to be the intersection of two languages: 1) the language of sequences of actions resulting from the task decompositions modeled as grammar parsing; 2) the language of sequences of actions of all the valid chaining of the actions (primitive tasks) defined by the action templates (or operators). In this paper, by constructing our modified Earley graph, we take into consideration the preconditions of tasks and the effects of actions to make sure that the generated plans follow the constraints imposed by the HTN domain specification. While earlier work relates planning and parsing only for deterministic domains, we extend this concept into probabilistic domains by annotating probabilities in the HTN methods, allowing us to calculate the probabilities of generating plans in the domain. Furthermore, we allow a user to specify rewards for specific states in the HTN specification in the same way as goal states are specified in classical planning, allowing us to use the Earley graph to calculate the expected utilities of these plans and to perform MEU planning using the HTN domain.

While we share the same grammar passing idea for task decompositions as in (Barrett and Weld 1994), we adopt a different task decomposition approach using Earley graph (Stolcke 1995). The advantage of using Earley graphs to represent task decompositions is of twofold: 1) a graphical semantics of the decomposition procedure; 2) an intuitive framework to accommodate probabilistic information. In turn, this helps us nicely link HTNs with MDPs in this paper. It also opens up scope for further investigation on how the Earley graph itself can become a generalized solution concept to MDP planning problems with HTN knowledge and how to automatically revise the domain knowledge (in HTN form) from past experiences.

2 Basic Definitions

In this section we introduce and provide a formal model of the main concepts required throughout the paper.

The language of states and actions

Our extensions to Earley parsing are underpinned by an adaptation of the formalization of the state-space model commonly used in in non-deterministic planning (Cimatti et al. 2003). We define a non-deterministic state transition domain (NSTD) to be a tuple $\mathcal{M} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ where:

- $\mathcal{P} = \mathcal{P}_S \cup \mathcal{P}_A$ is a finite set of propositions resulting of the union of, respectively, state and action propositions;
- $\mathcal{S} \subseteq 2^{\mathcal{P}_S}$ is the finite set of all possible states;
- $\mathcal{A} \subseteq 2^{\mathcal{P}_A}$ is the finite set of actions; and
- $Tr \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the state-transition relation.

We can describe and manipulate the logical structure of the states, actions and state transitions using a language $\mathcal{L}$ of quantified boolean formulae (QBFs) over the propositions in $\mathcal{P}$, which can be efficiently implemented with Binary Decision Diagrams (BDDs) (Bryant 1992). The language of QBFs is an extension to the propositional language by allowing standard connectives $\wedge, \vee, \rightarrow, \neg$ and quantifiers $\exists, \forall$. We define a *variable renaming operation* on $\mathcal{L}$ as follows. Let $\vec{x}$ and $\vec{x}'$ be two vectors of propositional variables for a $\xi \in \mathcal{L}$, then a variable renaming operation is defined as $\xi[\vec{x}/\vec{x}']$ where all the occurrences of variables $\vec{x}$ are substituted by $\vec{x}'$.

States and actions are specified by the QBF formulas respectively as $\xi(s)$ on $\mathcal{P}_S$, denoted by $s \models \xi(s)$; and $\xi(a)$ on $\mathcal{P}_A$, denoted by $a \models \xi(a)$. Extending $\mathcal{P}$ with an additional set of set propositions $\mathcal{P}_{S'}$ which is a copy of $\mathcal{P}_S$, a state transition $\langle s, a, s' \rangle$ can be specified by a formula $\xi(s) \wedge \xi(a) \wedge \xi(s')$ on $\mathcal{P}_S$, $\mathcal{P}_A$ and $\mathcal{P}_{S'}$, denoted by $\langle s, a, s' \rangle \models \xi(s) \wedge \xi(a) \wedge \xi(s')$. Furthermore ,a set of states S can be represented by $\xi(S) = \bigvee_{s_i \in S} \xi(s_i)$, a set of actions A can be represented by $\xi(A) = \bigvee_{a_i \in A} \xi(a_i)$, and a set of state transitions R can be represented by $\xi(Tr) = \bigvee_{tr_i \in Tr} \xi(tr_i)$.

In what follows, we give small examples to illustrate the basic idea of representing domain knowledge efficiently in QBFs, while more detailed study can be found in (Edelkamp and Helmert 1999). Consider an agent that can be in 8 locations $\{L_0, L_1, \ldots, L_7\}$ and owns one of 4 types $\{V_0, V_1, V_2, V_3\}$ of vehicles at any one time, then in $\mathcal{P}_S$ we need 3 (binary) variables $\vec{x}_l = \langle x_{l_0}, x_{l_1}, x_{l_2} \rangle$ to encode the locations, and 2 variables $\vec{x}_v = \langle x_{v_0}, x_{v_1} \rangle$ to encode the vehicles being used. Thus, the agent being at location L_0 is represented by $at(L_0) \stackrel{\text{def}}{=} \neg x_{l_0} \wedge \neg x_{l_1} \wedge \neg x_{l_2}$, while the agent not being at location L_0 is represented by $\neg at(L_0)$. Similarly, if the agent has 3 actions available, we represent the actions with 2 action variables $\vec{a} = \langle y_{a_0}, y_{a_1} \rangle$. If there are 16 possible values that can be assigned to the parameters to these actions, we need another four parameter variables $\vec{pa} = \langle y_{pa_0}, y_{pa_1}, y_{pa_2}, y_{pa_3} \rangle$. If the agent can either own or not own each type of 4 vehicles, we need 4 variables instead of 2 to denote the $2^4 = 16$ vehicle usage possibilities.

Hierarchical Task Networks

An HTN consists of a *network* of *tasks* connected by ordering *constraints* (Ghallab, Nau, and Traverso 2004). Tasks can be at various levels of abstraction, from very high-level (*e.g.* go to conference C) to very specific (*e.g.* flip a switch). An HTN domain consists of a set of concrete *actions* (or operators) directly executable in the environment, and a set of methods that refine higher-level tasks into actions.

Definition 1 *An HTN planning domain $\mathcal{D}$ is a tuple $(\mathcal{T}, \mathcal{M})$ where: (i) $\mathcal{T}$ is a finite set of task symbols which are composed by two disjoint sets: non-primitive tasks $N\mathcal{T}$ and primitive tasks $\mathcal{A}$; and (ii) $\mathcal{M}$ is a finite set of methods to refine the tasks in $\mathcal{T}$.*

The set $\mathcal{A}$ of primitive tasks corresponds to the actions in the NSTD defined in Section 2, whereas the set $N\mathcal{T}$ of non-primitive tasks must be refined into primitive tasks by applying methods from Definition 3. Methods are used to refine a non-primitive task by replacing it with a task network.

Definition 2 *A task network $\mathcal{H}$ is a pair (T, C) where T is a finite set of tasks to be accomplished and $C = \{t_i \prec t_j\}$ is a set of partial ordering constraints on tasks in T.*

In a task network, a *precedes* constraint $\prec$, $t_i \prec t_j$ denotes that task t_i must be executed *before* t_j. We denote the set of tasks which do not have preceding tasks in $\mathcal{H}$ as $first(\mathcal{H})$, and the set of tasks which do not have succeeding tasks in $\mathcal{H}$ as $last(\mathcal{H})$. Formally, $first(\mathcal{H}) = T - \{t' | t \prec t' \in C\}$ and $last(\mathcal{H}) = T - \{t | t \prec t' \in C\}$.

Definition 3 *An HTN method $m \in \mathcal{M}$ is represented as a tuple $m = \langle t, \mathcal{H} \rangle$, where t is a task to be decomposed, and $\mathcal{H} = \langle T, C \rangle$ is a task network that specifies how t can be achieved. Each method m has a precondition $precond(m)$ specified by a formula in $\mathcal{L}_S$ which is interpreted into a set of states $S = \{s | s \models precond(m)\}$.*

To ease presentation, we refer to the components of an HTN method $m = (t, \mathcal{H})$ respectively as $task(m)$ and $network(m)$. To simplify the planning algorithms, we assume that every task network $network(m)$ is totally ordered, and leave partial order task networks for future research.

Definition 4 *A method m* decomposes *a task network $\mathcal{H} = \langle T, C \rangle$ into a* resulting task network $\mathcal{H}' = \langle T', C' \rangle$*, denoted by $\mathcal{H}' = decompose(\mathcal{H}, m)$, iff $t = task(m) \in T$; $\mathcal{H}_m =$*

network(m); $T' = (T - \{t\}) \cup T_m$; *and* $C' = (C - C_t) \cup (C'_t \cup C_m)$ *where*

$$\begin{aligned} C_t &= \{t \prec t_i | t \prec t_i \in C\} \cup \{t_i \prec t | t_i \prec t \in C\} \\ C'_t &= \{t_i \prec t_j | t_i \prec t \in C \text{ and } t_j \in \mathit{first}(\mathcal{H}_m)\} \\ &\quad \cup \{t_i \prec t_j | t \prec t_j \in C \text{ and } t_j \in \mathit{last}(\mathcal{H}_m)\} \end{aligned}$$

Here, $C_t \subseteq C$ are constraints referring to t in $\mathcal{H}_m$, and C'_t are the constraints linking the tasks preceding t in $\mathcal{H}$ to the all the un-preceded tasks, and the constraints linking the unsucceeded tasks in $\mathcal{H}_m$ to the immediate successors of t.

Definition 5 *A sequence* $m_1, \ldots, m_k$ *of methods* decomposes *a task network* $\mathcal{H} = \langle T, C \rangle$ *into a resulting task network* $\mathcal{H}'$ *if there is a sequence* $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_k$ *of task networks such that: (i)* $\mathcal{H}_0 = \mathcal{H}$*, (ii)* $\mathit{task}(m_i) \in \mathcal{H}_{i-1}$*, (iii)* $\mathcal{H}_i = \mathit{decompose}(\mathcal{H}_{i-1}, m_i)$*, and (iv)* $\mathcal{H}' = \mathcal{H}_k$*. The application of a sequence of methods* $m_1, \ldots, m_k$ *is said to have* fully decomposed *a task network* $\mathcal{H}$ *if the resulting task network* $\mathcal{H}'$ *contains only primitive tasks.*

An action a is expressed by its precondition $\mathit{precond}(a)$ and its effect $\mathit{effect}(s, a)$ in a state s. Here we use QBFs and the state transitions Tr defined in Section 2 to express $\mathit{precond}(a)$ and $\mathit{effect}(a)$ as follows:

$$\begin{aligned} \mathit{precond}(a) &= \exists_{\mathcal{P}_A, \mathcal{P}_{S'}} [Tr \wedge \xi(a)] \\ \mathit{effect}(s, a) &= \exists_{\mathcal{P}_S, \mathcal{P}_A} [Tr \wedge \xi(a)] \end{aligned}$$

Using this encoding, we define action execution.

Definition 6 *Let* s *and* s' *be two states in the environment and* a *an action. An action* a *is executable in state* s *if* $s \models \mathit{precond}(a)$*, and the execution of* a *in state* s *is a new state* s' *such that* $s' \models \mathit{effect}(s, a)$ *and* $s' \not\models \bot$*.*

Now, from Definition 6 it is easy to see that the execution of a sequence of actions $\{a_1, \ldots, a_n\}$ from an initial state s_0 induces a sequence of state transitions $\langle s_0, a_1, s_1, \ldots, s_{n-1}, a_{n-1}, s_n \rangle$ where each state s_i is the result of executing action a_i from state s_{i-1}. We call such as sequence of state transitions *trajectory of execution*.

Taking into account the task network constraints, the preconditions of methods, and the preconditions and effects of primitive tasks, we define valid task decompositions.

Definition 7 *Let* $\mathcal{H} = \langle T, C \rangle$ *be a task network composed of primitive tasks, i.e.* $T \subseteq \mathcal{A}$*,* $\mathcal{H}$ *is a* valid primitive task network *at a state* s_0 *iff for each* $t_0 \in \mathit{first}(\mathcal{H})$ *we have* $s_0 \models \mathit{precond}(t_0)$*; and there exists an execution trajectory* $s_0, t_0, \ldots, s_n, t_n, s_{n+1}$ *such that* $t_i \prec t_{i+1} \in C$*,* $s_i \models \mathit{precond}(t_i)$*,* $s_{i+1} \models \mathit{effect}(s_i, t_i)$ *and* $s_{i+1} \models \mathit{precond}(t_j)$*.*

Definition 8 *Let* $m_1, \ldots, m_k$ *be a sequence of methods that fully decomposes a task network* $\mathcal{H}$ *into* $\mathcal{H}'$*, then the decomposition is valid at state* s_0 *iff the resulting task network* $\mathcal{H}'$ *is a valid primitive task network at* s_0*.*

3 From HTN Methods to Earley Graphs

Our approach is based on the Earley Parsing algorithm for probabilistic context free grammar parsing (Stolcke 1995) adapted to accommodate the components of states (of preconditions and effects), and task decompositions. We modify the concept of Earley states, which we refer to as *Earley nodes* to avoid confusion with the planning state-space, to include the information of states and actions.

Earley Graphs for HTN Methods

To facilitate the construction of our Earley graph, we add two dummy tasks to every task network $\mathcal{H}$: $\mathit{start}(\mathcal{H})$ is a dummy starting task in $\mathcal{H}$ preceding the first tasks in $\mathcal{H}$, $\mathit{end}(\mathcal{H})$ is a a dummy ending task succeeding last tasks in $\mathcal{H}$; and $\mathit{next}(\mathcal{H}, t)$ denotes the set of tasks immediately succeeding t in $\mathcal{H}$. An Earley graph constructed from an HTN is defined by Earley nodes and links as follows.

Definition 9 *Let* $m = \langle t, \mathcal{H} \rangle$ *be an HTN method from which we generate* $|T|$ *Earley nodes. Each Earley Node EN is of the form* $EN_{m,t_i} = \langle m, t_i \rangle$ *where* $t_i \in \mathit{network}(m)$*. For notational convenience, we denote* m *by* $\mathit{method}(EN_{m,t_i})$*,* t_i *by* $\mathit{current}(EN_{m,t_i})$*, and* $\mathit{task}(m)$ *by* $\mathit{root}(EN_{m,t_i})$*.*

Definition 10 *An Earley graph for a method library* $\mathcal{M}$ *is a graph* $\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$ *where*

- $\mathcal{N} = \{EN\}$ *is the set of Earley nodes; and*
- $\mathcal{E}$ *is the set of Earley links of three types:*
  - *A* predicting link $\langle EN_{m,t_i}, EN_{m',t'_{start}} \rangle$ *where* $\mathit{task}(m') = t_i$ *and* $t'_{start} = \mathit{start}(m')$ *is the starting task of* m' *which precedes all the other tasks in* m'*.* $EN_{m',t'_{start}}$ *is then a* predicting node.
  - *A* scanning link $\langle EN_{m,a}, EN_{m,t_i} \rangle$ *where* a *is a primitive task in* m*, and* $t_i \in \mathit{next}(m, a)$ *is a task immediately succeeding* a *in* m*.* $EN_{m,a}$ *is then a* scanning node.
  - *A* completing link $\langle EN_{m',t'_{end}}, EN_{m,t_i} \rangle$ *where* $t'_{end} = \mathit{end}(m')$ *is the last task of* m'*, and* $t_i \in \mathit{next}(m, \mathit{task}(m'))$ *is a task immediately succeeding* $\mathit{task}(m')$ *in* m*.* $EN_{m',t'_{end}}$ *is then a* completing node.

A predicting link $\langle EN_{m,t_i}, EN_{m',t'_{start}} \rangle$ marks a possible decomposition of a task t_i; a completing link $\langle EN_{m',t'_{end}}, EN_{m,t_i} \rangle$ marks a possible completion of a task in m resulting in the investigation of the next task t_i in m; a scanning link marks an execution of a primitive task a resulting in the investigation of the next task t_i in m. A predicting node $EN_{m,t_{start}}$ and a completing node $EN_{m,t_{end}}$ are said to be *corresponding with each other*. An Earley graph is illustrated in Figure 1.

Given an HTN method library $\mathcal{M}$, an Earley graph can be constructed using Algorithm 1. The algorithm is directly derived from the Definition 10, and we can establish bounds on the Earley graph it constructs:

Proposition 1 *Let* $|\mathit{network}(m)|$ *be the number of tasks in the network of a method* m*;* $|\mathit{Methods}(t)|$ *the number of methods that can be used to decompose a non-primitive task* t*, and* $\mathit{Appearances}(t) = \{\langle m, t \rangle | m \in \mathcal{M}\}$ *the number of instances of* t *in all the methods. The number of nodes in the Earley graph constructed by Algorithm 1 is* $\Sigma_{m \in \mathcal{M}} |\mathit{network}(m)|$*, and the number of edges in the constructed Earley graph is* $O(\Sigma_{m \in \mathcal{M}} |\mathit{network}(m)|) + 2\Sigma_{t \in N\mathcal{T}} (|\mathit{Appearances}(t)| \cdot |\mathit{Methods}(t)|)$

Proof 1 *Each Earley node is identified by a pair* $\langle m, t \rangle$ *for every method* $m \in \mathcal{M}$ *and every task* $t \in \mathit{network}(m)$*. The number of such pairs is* $\Sigma_{m \in \mathcal{M}} |\mathit{network}(m)|$ *which is the number of nodes in the constructed Earley graph.*

For a non-primitive task t*, the number of predictive links leaving a node identified by* $\langle m, t \rangle$ *is* $|\mathit{Methods}(t)|$

linking and number of completing links entering a node $\langle m, next(m,t)\rangle$ *is also* $|Methods(t)|$. *Therefore the number of predictive links and completing links is* $2\Sigma_{t\in N\mathcal{T}}(|Appearances(t)|\cdot|Methods(t)|)$. *The number of scanning links for each method m is the number of primitive tasks in m. The total number of scanning links is* $\Sigma_{m\in\mathcal{M}}|\{a|a \in \mathcal{A} \wedge a \in network(m)\}|$ *which is bounded by* $O(\Sigma_{m\in\mathcal{M}}|network(m)|)$. *As a result, the total number of Earley edges is* $O(\Sigma_{m\in\mathcal{M}}|network(m)|) + 2\Sigma_{t\in N\mathcal{T}}(|Appearances(t)|\cdot|Methods(t)|)$.

Note that multiple appearances of the same task t in an $\mathcal{H}$ is counted multiple times in $|Apperances(t)|$. For example, in an HTN $\mathcal{H} = \{t_1 \prec t_1 \prec t_1\}$, $|apperances(t_1)| = 3$. Another subtle aspect of the Earley graph involves processing recursive method decompositions. For example, if $m_0 = (t_0, \{t_0 \succ t_1\}), m_1 = (t_0, \{a_1\})$, then the Earley graph will contain two loops: one is a predictive loop from $\langle m_0, t_0\rangle$ to $\langle m_0, start\rangle$, and $\langle m_0, start\rangle$ to $\langle m_0, t_0\rangle$; another one is a self-loop $\langle m_0, end\rangle$ to $\langle m_0, end\rangle$. These loops help to avoid infinite Earley node instantiations of recursive task decompositions.

Figure 1 illustrates the Earley graph generated from Example 3.1, which consists of planning to arrive in London departing from New Jersey.

Example 3.1 *Planning consists of selecting the appropriate means of transportation and route to maximize the utilities of reaching London. The domain* $\mathcal{D}$ *contains a set of five tasks* $\mathcal{T} = \{goTo(L), obtainVehicle, moveTo(L), moveTo(L,V), getVehicle(V)\}$ *of which two are actions* $\mathcal{A} = \{moveTo(L_1, L_2, V), getVehicle(V)\}$. *These tasks are decomposed using a set of six methods* $\mathcal{M} = \{ml, mo_1, mo_2, mtl_1, mtl_2, mtl_3\}$

The methods are defined as (in all method definitions we assume that the set of constraints C orders the tasks as they are written within their set definitions):

- $ml = \langle goTo(L), \langle\{getVehicle(car), obtainVehicle, moveTo(L)\}, C\rangle\rangle$, *with* $precond(ml) = \top$
- $mo_1 = \langle obtainVehicle\langle\{moveTo(airport, car), getVehicle(airplane)\}, C\rangle\rangle$, *with* $precond(mo_1) = at(airport)$
- $mo_2 = \langle obtainVehicle\langle\{moveTo(harbor, car), getVehicle(ship)\}, C\rangle\rangle$, *with* $precond(mo_2) = at(harbor)$
- $mtl_1 = \langle moveTo(L), \langle\{moveTo(nyc, airplane), moveTo(london, airplane)\}, C\rangle\rangle$, *with* $precond(mtl_1) = has(airplane)$
- $mtl_2 = \langle moveTo(L), \langle\{moveTo(soton, ship), moveTo(london, ship)\}, C\rangle\rangle$, *with* $precond(mtl_2) = has(ship)$
- $mtl_3 = \langle moveTo(L), \langle\{moveTo(lpool, ship), moveTo(london, ship)\}, C\rangle\rangle$, *with* $precond(mtl_3) = has(ship)$

The actions are specified as follows:

- $moveTo(L_1, L_2, V)$ *has preconditions* $has(V) \wedge at(L1)$ *and effects* $at(L2) \wedge \neg at(L1)$
- $getVehicle(V)$ *has preconditions* $\top$ *and effects* $has(V2) \wedge \neg has(V1)$

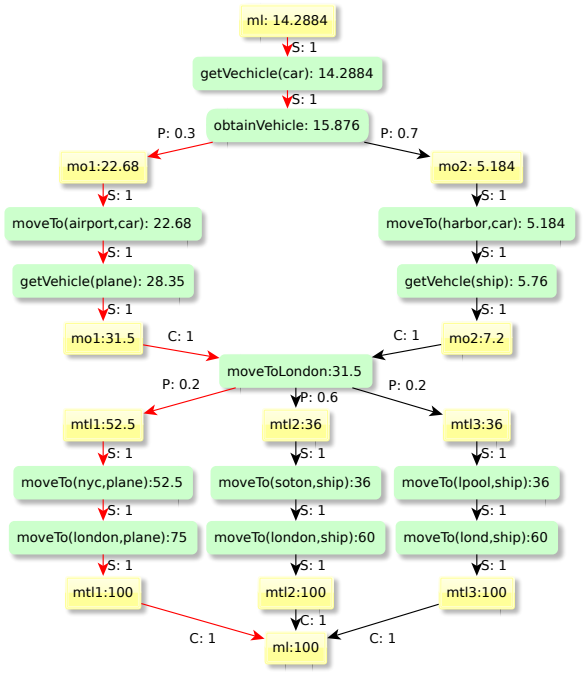

Figure 1: Earley Graph for "Go to London"

Valid Decompositions and Executions

In an Earley graph, a path from $EN_{m,t_{start}}$ to $EN_{m,t_{end}}$ corresponds to a decomposition of $task(m)$ and an execution trajectory of $task(m)$ according to the methods in the library $\mathcal{M}$ if the traversal of the paths is carefully managed to ensure that 1) the task decompositions corresponding to the path are valid, and 2) the preconditions of the methods and primitive tasks in the path are met. The first condition is to avoid linking a completing node to a parent method which does not lead to such a method. To ensure the first condition, we need to maintain a stack of applied methods; to ensure the second condition, we use the BDD representation to enforce the preconditions. In Algorithm 2, we provide a generic algorithm to traverse all valid decomposition and execution paths. The parameters of *Traverse* are:

- $\mathcal{G}$ is the input Earley graph;
- *EN* is an predictive node for an initial task *t*;
- *cache* is a table to store previous explorations of methods;
- *stack* is a stack to keep track of the decompositions;
- *precond* is the set of preconditions corresponding to the initial states;
- *Order* is an order function on Earley nodes (e.g. ascending in probability or utility);
- *Process* is an algorithm called whenever a valid decomposition and execution path is found; and
- *PostProcess* is post-processing algorithm called to perform additional book keeping on the current Earley node once all its children have been processed

Algorithm 1: Construct Earley Graph for $\mathcal{M}$

input : A method library $\mathcal{M}$
output: The Earley graph $\mathcal{G}$ for $\mathcal{M}$
$\mathcal{G} \leftarrow \emptyset$;
for *each* $m \in \mathcal{M}$ **do**
  for *each* $t_i \in network(m)$ **do**
    if t_i *is a non-primitive task* **then**
      for *each* $m' \in \{m' | task(m') = t_i\}$ **do**
        Add a predicting link $\langle EN_{m,t_i}, EN_{m',first(m')} \rangle$ into $\mathcal{G}$;
      end
    else if t_i *is a primitive task and* $t_i \neq end(m)$ **then**
      Add a scanning link $\langle EN_{m,t_i}, EN_{m,next(m,t_i)} \rangle$ into $\mathcal{G}$;
    else if $t_i = end(m)$ **then**
      for *each* $m' \in \{m' | task(m) \in network(m')\}$ **do**
        Add a completing link $\langle EN_{m,t_i}, EN_{m',next(m',task(m))} \rangle$ into $\mathcal{G}$;
      end
    end
  end
end
return $\mathcal{G}$

In the initial invocation of *Traverse*, the *precond* parameter is set to an initial state of interest S. If no special initial states are considered, we can apply *Traverse* with $precond = \top$. Since *Traverse* examines all the possible decomposition paths with the control of a stack, and the branching factor for a non-primitive task in any method is $|Methods(t)|$, the path space can grow exponentially in the worst case. In this case, we use a *cache* to store the exploration result of a method into a starting Earley node. Given a set of states, the set of valid decompositions using a method does not depend on decompositions preceding such a method, this enables us to cache the decomposition results and reuse them later. For non-recursive HTNs, the Earley graph enables us to employ dynamic programming with complexity bounded by the size of the Earley graph and the sizes of the BDDs which encode the sets of states in which the methods can be explored. BDDs[1] also enable us to look up the cache given a precondition using one BDD operation whose complexity does not depend on the number of states but the complexity of information in preconditions and the state information in the cache. For recursive HTNs, the *Traverse* procedure might produce an infinite-depth stack. Although we do not deal with this issue in this paper, approaches to parse recursive grammars exist (Stolcke 1995, Section 4.5) that could be leveraged to handle recursive task decompositions.

4 Integrating HTNs and MDPs

Markov Decision Processes

We start by formally laying out the elements of an MDP in Definition 11.

[1]By BDDs, we also refer to BDD variants such as MTBDDs when numerical values are encoded along with the boolean values.

Algorithm 2: Traverse($\mathcal{G}$, *EN*, *precond*, *cache*, *stack*, *path*, *Order*, *Process*, *PostProcess*)

if $method(top(stack)) \neq task(method(EN))$ **then**
  return *FAIL* ;
end
Append *EN* to the end of *path* ;
if *EN is a completing node* **then**
  $pop(stack)$;
  if *stack is empty* **then**
    Call *Process* with appropriate parameters, e.g. *EN*, *NN*, *path*, etc. ;
    return *SUCCESS*
  end
end
if *EN is a first*$(method(EN))$ **then**
  Look up *cache* for $\langle precond, method(EN), resultState \rangle$ for previous result;
  if *previous result available* **then**
    return previous result;
  end
end
Sort the children of *EN* with respect to *Order* ;
for *each child NN of EN in Order* **do**
  if $\langle EN, NN \rangle$ *is a predicting link and* $precond(method(NN)) \wedge precond \neq FALSE$ **then**
    push NN into *stack* ;
    $precond \leftarrow precond \wedge precond(method(NN))$;
  else if $\langle EN, NN \rangle$ *is a scanning link and* $precond(current(NN)) \wedge precond \neq FALSE$ **then**
    $precond \leftarrow effect(current(NN), precond)$;
  end
  if $precond \neq FALSE$ **then**
    $result_{NN} \leftarrow$ *Traverse*($t, \mathcal{G}$, *NN*, *precond*, *stack*, *path*, *Order*, *Process*, *PostProcess*) ;
  end
  Restore the context associated with $top(stack)$;
end
Call *PostProcess* **with appropriate parameters, e.g.** *EN*, *NN*, *path*, **etc.** ;
if $result_{NN}$ *is SUCCESS* **then**
  return *SUCCESS*
else
  return *FAIL*
end

Definition 11 *A Markov Decision Process (MDP)* Σ *is a 4-tuple* (S, A, Pr, R) *where S is a finite set of states, A is a finite set of actions, Pr is a state-transition system and R is a reward function.*

We equate the set of MDP states and actions S and A to the same set of states and actions of the non-deterministic state transitions defined in section 2. In this way, we can specify the MDP with QBFs.

The dynamics of the environment in an MDP is modeled by a probability distribution Pr on the state transitions.

Definition 12 *An MDP state-transition system defines a probability distribution for each state transition through a function* $Pr : S \times S \times A \rightarrow [0, 1]$. *Given* $s, s' \in S$ *and* $a \in A$, $Pr_a(s, s')$ *denotes the probability of transitioning from state s to state* s' *when executing action a. That is,* $Pr_a(s, s') = Pr(X_{\tau+1} = s' | X_\tau = s, Y_\tau = a)$ *where* X_τ *and*

Y_τ are random variables denoting the state and action at time τ respectively.

In order to calculate an *optimal* solution to the decision problem posed by an MDP, it is necessary to associate rewards to the states in an MDP through a *reward function*.

Definition 13 *An MDP reward function $R : S \to \Re$ associates a real number to each state of an MDP. Given $s \in S$, $R(s)$ denotes the reward associated with reaching state s.*

Within this work, we specify the conditional probability table and reward table respectively with QBFs

$$\begin{aligned} Pr &= \{\langle \xi(\langle s_i, a_i, s_i' \rangle), Pr_{a_i}(s_i, s_i') \rangle\} \\ R &= \{\langle \xi(s_i), R(s_i) \rangle\} \end{aligned}$$

The entries with the same probability or reward can be grouped into one QBF expression as in Example 4.1.

Example 4.1 *Extending Example 3.1 with MDP probabilities and rewards, with QBF representation we can set*

$$R = \{\langle at(London), 100\rangle, \langle \neg at(London, 0\rangle\}$$

$$\begin{aligned} Pr = \{&\langle getVehicle(car), 0.9\rangle, \langle moveTo(airport, car), 0.8\rangle, \\ &\langle getVehicle(plane), 0.9\rangle, \langle moveTo(harbor, car), 0.9\rangle, \\ &\langle getVehicle(ship), 0.8\rangle, \langle moveTo(nyc, plane), 0.7\rangle, \\ &\langle moveTo(london, plane), 0.75\rangle, \langle moveTo(soton, ship), 0.6\rangle, \\ &\langle moveTo(london, ship), 0.6\rangle, \langle moveTo(lpool, ship), 0.6\rangle\} \end{aligned}$$

The rewards on the states that are not satisfied the formulae in the reward table R are set to 0*. The other probabilities on the state transitions that can not satisfied the above formulae in the table Pr are inferred by the above with the sum of probabilities on each state-action pair to be* 1*.*

We note that in our current implementation it is the burden of the input to guarantee that the list of pairs of QBF expressions and numbers can be interpreted into consistent reward assignments to states and consistent probability assignments to state transitions.

In a classic MDP problem, the solution of an MDPs is a *policy*, which indicates the best action to take in each state. Thus, an MDP policy is a total function mapping states into actions, so a policy π is represented as a function $\pi : S \to A$. Information on the rewards of states makes it possible to compute the value of a a given state under a particular policy π – it is the expected value of carrying out the policy from that state, given some *discount factor* γ:

$$V^\pi(s) = [R(s) + \gamma \sum_{s' \in S} Pr_{a \in \pi(s)}(s, s') V^\pi(s')].$$

Semantics of an HTN Earley Graph

Given a task t, a method library $\mathcal{M}$ contains a set of methods $M(t)$ which can be used to decompose t. Assuming that the application of m is independent of choice of the methods used to decompose the other tasks, we have $Pr(m_i \mid t_i) = Pr(m_i \mid t_i, m_{i-1}, t_{i-1}, \ldots, m_0, t_0)$ where m_i is used to decompose t_i.

The probability of applying a method m on a task t can be assigned according to the application domains. Typically, we will assign human's subjective view about how likely a method m can be used to decompose a task t as $Pr(m|t)$.

For example, in Figure 1, we set

$$\begin{aligned} Pr(mo1|ObtainVehicle) &= 0.7, \\ Pr(mo2|ObtainVehicle) &= 0.3, \\ Pr(mtl1|moveToLondon) &= 0.6, \\ Pr(mtl2|moveToLondon) &= 0.2, \\ \text{and } Pr(mtl3|moveToLondon) &= 0.2. \end{aligned}$$

Thus, we can induce that the probability of a sequence of decompositions of a task t with a library $\mathcal{M}$ into a network of primitive tasks by the chain rule of probability and the independence assumption

$$\begin{aligned} &Pr(m_i, t_i, m_{i-1}, t_{i-1}, \ldots, m_0, t_0) \\ &\quad = Pr(m_i \mid t_i) \times Pr(m_{i-1} \mid t_{i-1}) \times \ldots \times Pr(m_0 \mid t_0) \end{aligned}$$

To embed the probability of decompositions into the Earley graph, we assign probability to the edges in an Earley graph induced from a method library:

- For a predicting link $\langle EN, NN\rangle$, set $Pr(EN|NN) = Pr(m \mid t)$ where $m = method(NN)$ and $t = root(NN)$.
- For a scanning link $\langle EN, NN\rangle$, set $Pr(EN|NN) = 1$
- For a completing link $\langle EN, NN\rangle$, set $Pr(EN|NN) = 1$

The probabilities assigned to the Earley links are about the uncertainty in decomposing tasks. The predicting link stores the subjective knowledge on how probable it is that a method can be used to successfully decompose a task, so it is assigned number $Pr(m|t)$. A scanning link is assigned probability 1 since, in terms of task decomposition, encountering a primitive task in the task network entails a move to the next task with probability 1. A completing link is assigned probability 1 because once a method has been fully "parsed", the next method in the parent task must be parsed next, as enforced by Algorithm 2

Thus, the probability of a path $\langle EN_0, EN_1, \ldots, EN_N\rangle$ extracted by our technique is

$$Pr(\langle EN_0, EN_1, \ldots, EN_N\rangle) = Pr(EN_0|EN_1) \times \ldots \times Pr(EN_{n-1}|EN_n)$$

This is the probability of a pure task network decomposition which models the uncertainty of how a computer program or a human expert uses a library of methods.

The probability of an Earley path with initial states can be similarly computed. Associated with a valid Earley graph path $path = \langle EN_0, EN_1, \ldots, EN_n\rangle$, we will have a set of execution trajectories each of which are of the form $\langle s_0, EN_{a_0}, s_1, EN_{a_1}, \ldots, EN_{a_m}, s_{m+1}\rangle$ such that

- $EN_{a_0}, \ldots, EN_{a_m}$ is a sub-sequence of scanning nodes of $\langle EN_0, EN_1, \ldots, EN_n\rangle$;
- $s_0 \in precond(method(EN_0) \wedge \ldots \wedge precond(EN_j) \ldots \wedge precond(EN_{a_0})$ where EN_0 is an initial node with $task(EN_0) = t$, each EN_j $(0 < j < a_0)$ is a predictive node, and there are no scanning nodes between EN_0 and EN_{a_0} in the sequence; and

- $s_i \in effect(current(EN_{a_{i-1}})) \wedge \ldots \wedge precond(EN_j) \wedge \ldots \wedge precond(EN_{a_{i+1}})$ where each EN_j $(a_{i-1} < j < a_i)$ is a predictive node or a completing node and there are no scanning nodes between EN_0 and EN_{a_0} in the sequence.

Such a set of execution trajectories of an Earley path is denoted by $E(path) = \{\langle s_0, EN_{a_0}, s_1, EN_{a_1}, \ldots, EN_{a_m}, s_{m+1}\rangle\}$. Incorporating decompositions, we have a decomposition-execution path $DE(path) = \{\langle EN_0, \ldots, EN_j, \ldots, s_{a_0}, EN_{a_0}, \ldots, s_{a_1}, EN_{a_1}, \ldots, s_{a_i}, EN_{a_i}, \ldots, EN_m, \ldots, s_{m+1}\rangle\}$. For convenience we denote the set of decomposition-execution paths of *path* starting from state s by $DE(path, s)$. Assuming independence between the MDP environment and the choices of method decompositions, we can then compute the probability of a decomposition-execution path *de* using

$$Pr(de) = \prod_{EN_i \in de} Pr(EN_{i+1}|EN_i) \cdot \prod_{s_j, a_j, s_{j+1} \in de} Pr(s_{j+1}|s_j, a_j)$$

Utility of Earley Paths

Given a decomposition-execution path $de \in DE(path)$, the value of this path is the summation of all the rewards encountered, so that $V(de) = \sum_{a_j \in de} R(s_j)$. As a decomposition path corresponds to a collection of decomposition-execution paths, the expected value of a decomposition path can be computed by summing up the values of all these decomposition-execution paths weighted by their probabilities:

$$V(path) = \sum_{de \in DE(path)} (V(de) \cdot Pr(de))$$

Similar to the MDP value computation, the expected value of a path can be computed iteratively with the Earley graph. Let $sub^{path}(s, EN)$ be the sub-path of *path* starting from $\langle s, EN\rangle$, we define

$$V^{path}(s|EN) = V(sub^{path}(s, EN))$$

and

$$V^{path}(EN) = \sum_s V^{path}(s|EN).$$

Related to a decomposition $path = \langle EN_0, \ldots, EN_n\rangle$, we define the value of the complete Earley node EN_n to be $V^{path}(s|EN_n) = R(s)$; if EN_{i+1} is a predicting or completing node, then

$$V^{path}(s|EN_i) = Pr(EN_{i+1}|EN_i) \cdot V(s|EN_i)$$

and if EN_{i+1} is a scanning node, then

$$V^{path}(s|EN_i) = Pr(EN_{i+1}|EN_i) \cdot \left(R(s) + \sum_{s'} Pr(s'|s, a)\ V^{path}(s'|EN_{i+1})\right)$$

Proposition 2 *The expected value of the starting Earley node EN_0 of a decomposition path is the expected value of the path: $V^{path}(EN_0) = V(path)$.*

The expected value of the starting Earley node EN_0 of a decomposition path situated at a state s_0 is is the expected value of all the decomposition-execution paths of path: $V^{path}(s_0|EN_0) = \Sigma_{de \in DE(path, s_0)} V(de)$.

Function MaxValProcess(*EN*, *precond*)

```
V(·|EN) ← ∅;
if EN is a fully completing node then
    for each s ⊨ precond do
        V(s|EN) ← R(s) if V(s|EN) is not in the table ;
    end
end
return V(·|EN);
```

Function MaxValPostProcess(*EN*,*SucEdges*,*precond*)

```
if En is a predicting node then
    for each ⟨EN, NN⟩ ∈ SucEdges do
        V ← ∅;
        for each s ⊨ precond do
            V(s) ← Pr(NN|EN) V(s|NN) if
            s ∈ precond(NN) ;
        end
    end
    Select ⟨EN, NN⟩ as the decomposition edge with
    maximum Σ_s V(s) and set V(·|EN) ← V(·);
end
if EN is an internal completing node then
    Let ⟨EN, NN⟩ ∈ ValidEdges ;
    for each s ⊨ precond do
        V(s|EN) ← Pr(NN|EN) V(s|NN) if
        s ∈ precond(NN) ;
    end
end
if EN is a scanning node then
    for each s ⊨ precond and each action a in EN do
        V(s|EN) ← R(s) + Σ_{s'∈S} Pr(s'|s, a) · V(s') ;
    end
end
return V(·|EN);
```

Proof 2 *By construction.*

The solution for a probabilistic hierarchical planning problem is then a search for a decomposition $path^*$ with the maximum expected $V^*(path^*)$ with a given set of initial states $S \models precond$.

MEU HTN Planning

In order to search for a maximum value decomposition path, we traverse the Earley graph using Algorithm 2 with the *Process* and *PostProcess* functions set to the `MaxValProcess` and `MaxValPostProcess`. The input to `MaxValPostProcess` is the set of successful edges. As we traverse the graph, we store entries of the form $\langle s, m, s', V_{max}(s, m, s'), Choices\rangle$ in the *cache*. Each entry contains a method m which, when applied in state s results in state s' with $V_{max}(s, m, s')$ as the maximum possible reward that can be accumulated by executing m on state s. Moreover, the choices of methods to decompose non-primitive tasks in m are stored in *Choices*. Once a method is explored in a given state, an agent can lookup in the cache providing a sequence of states and actions encountered, from which the agent can compute the probabilities of the Earley Nodes it is in, and in turn, compute the choices of actions that can take

it to the maximum expected values.

We have implemented a preliminary version of these algorithms in C++. Invoking Algorithm 2 with `MaxValProcess` and `MaxValPostProcess`, we obtain the Earley graph with node value filled in Figure 1, for which the optimal HTN solution is: $\langle$ *ml* : *START*, *getVechicle*(*car*), *obtainVehicle*, *mo*1 : *START*, *moveTo*(*airport*, *car*), *getVehicle*(*plane*), *mo*1 : *END*, *moveToLondon*, *mtl*1 : *START*, *moveTo*(*NYC*, *plane*), *moveTo*(*London*), *mtl*1 : *END*, *ml* : *END* $\rangle$.

The corresponding states experienced by the system are: $\langle at(NJ), at(NJ) \wedge has(car), at(airport) \wedge has(car), at(airport) \wedge has(plane), at(NYC) \wedge has(plane), at(London) \wedge has(plane) \rangle$. The expected utilities are shown in Figure 1, and the optimal HTN solution is marked in bold red.

5 Discussion

The Earley graph provides an intuitive probabilistic graphical model by which we can understand the interplay between the knowledge in the HTNs and the stochastic environment.

In this paper, the solution for a probabilistic hierarchical planning problem is simplified to be a decomposition $path^*$ with the maximum expected $V^*(path^*)$ given a set of initial states. In a more realistic setting, we should seek to produce a task decomposition table and a policy table for each node in the Earley graph as a robust solution concept to handle more effectively the failure of the action execution and the uncertainty in task decomposition knowledge. Furthermore, with the Earley graph we will be able to calculate the probability of being in an Earley node given a sequence of previously observed states and a sequence of actions taken. This can be done in a manner similar to probabilistic grammar parsing (Stolcke 1995). With this information, it is possible to revise the decomposition and action decisions conditional on the previous observations and actions.

Regarding the decomposition, if no information about the predictive probability is given, we can assume that the application of a method $m \in M(t)$ is equally chosen:$Pr(m \mid t) = \frac{1}{|M(t)|}$. Starting with this maximum entropy assumption, or some other assumption about the conditional priors $Pr(m|t)$, we can learn these probabilities for a set D of the sequences of state-transitions and their parsing as HTN decompositions, and update $Pr(m|t)$ to be the posterior $Pr(m|t, D)$. This can be done similarly to learning grammar probabilities given annotated samples as proposed in (Stolcke 1995).

6 Conclusions and Future Work

In this work, we developed a new approach to using probabilistic hierarchical task networks (HTNs) as an effective method for agents to plan when their problem-solving knowledge is uncertain, and the environment is nondeterministic. In such situations it is natural to model the environment as a Markov decision process (MDP) and we show that using Earley parsing techniques we can bridge the gap between HTNs and MDPs. We prove that the size of the Earley graph created for given HTNs is bounded by the total number of tasks in the HTNs and show that from the Earley graph we can then construct a plan for a given task that has the the maximum expected value when it is executed in an MDP environment.

Our ultimate goal is not only to perform probabilistic hierarchical planning for an uncertain environment, but also to extend the approach for multiagent system control. With these extensions, a system of cooperative agents can communicate to share the same set of task networks while working in the same environment (and the same uncertainty). As every agent can construct the same Earley graph structure from the task network library, they should be able to incrementally adapt to the environment and revise their task decomposition probabilities. The system of agents would then converge to a set of cooperative behaviors prescribed by the communicated set of task networks. In this way, we will have a system which allows us to specify its group behaviors in a way that is close to how humans think about the problem solving while accommodating both the uncertainty in the knowledge of problem solving and the uncertainty in the environment.

Acknowledgement: This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[Barrett and Weld 1994] Barrett, A., and Weld, D. S. 1994. Task-decomposition via plan parsing. In *Procs. of the 12th National Conference on Artificial Intelligence (vol. 2)*, 1117–1122. Menlo Park, CA, USA: American Association for Artificial Intelligence.

[Bellman 2003] Bellman, R. E. 2003. *Dynamic Programming*. Dover Publications, Incorporated.

[Bryant 1992] Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3):293–318.

[Cimatti et al. 2003] Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

[Earley 1970] Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of the ACM* 13(2):94–102.

[Edelkamp and Helmert 1999] Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *Procs. 5th European Conference on Planning*, volume 1809 of *LNCS*, 135–147. New York: Springer-Verlag.

[Ghallab, Nau, and Traverso 2004] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

[Nau, Ghallab, and Traverso 2004] Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[Stolcke 1995] Stolcke, A. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics* 21(2):165–201.