Angelic Hierarchical Planning: Optimal and Online Algorithms

Bhaskara Marthi
MIT/Willow Garage Inc.
bhaskara@csail.mit.edu

Stuart Russell
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
russell@cs.berkeley.edu

Jason Wolfe[*]
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
jawolfe@cs.berkeley.edu

Abstract

High-level actions (HLAs) are essential tools for coping with the large search spaces and long decision horizons encountered in real-world decision making. In a recent paper, we proposed an "angelic" semantics for HLAs that supports proofs that a high-level plan will (or will not) achieve a goal, without first reducing the plan to primitive action sequences. This paper extends the angelic semantics with cost information to support proofs that a high-level plan is (or is not) *optimal*. We describe the Angelic Hierarchical A* algorithm, which generates provably optimal plans, and show its advantages over alternative algorithms. We also present the Angelic Hierarchical Learning Real-Time A* algorithm for situated agents, one of the first algorithms to do *hierarchical lookahead* in an online setting. Since high-level plans are much shorter, this algorithm can look much farther ahead than previous algorithms (and thus choose much better actions) for a given amount of computational effort.

Introduction

Humans somehow manage to choose quite intelligently the twenty trillion primitive motor commands that constitute a life, despite the large state space. It has long been thought that hierarchical structure in behavior is essential in managing this complexity. Structure exists at many levels, ranging from small (hundred-step?) motor programs for typing characters and saying phonemes up to large (billion-step?) actions such as writing an ICAPS paper, getting a good faculty position, and so on. The key to reducing complexity is that one can choose (correctly) to write an ICAPS paper without first considering all the character sequences one might type.

Hierarchical planning attempts to capture this source of power. It has a rich history of contributions (to which we cannot do justice here) going back to the seminal work of Tate (1977). The basic idea is to supply a planner with a set of high-level actions (HLAs) in addition to the primitive actions. Each HLA admits one or more *refinements* into sequences of (possibly high-level) actions that implement it. Hierarchical planners such as SHOP2 (Nau *et al.* 2003) usually consider only plans that are refinements of some top-level HLAs for achieving the goal, and derive power from constraints placed on the search space by the refinement hierarchy.

One might hope for more; consider, for example, the *downward refinement property*: every plan that claims to achieve some condition does in fact have a primitive refinement that achieves it. This property would enable the derivation of provably correct *abstract plans* without refining all the way to primitive actions, providing potentially exponential speedups. This requires, however, that HLAs have clear precondition–effect semantics, which have until recently been unavailable (McDermott 2000). In a recent paper (Marthi, Russell, & Wolfe 2007) — henceforth (MRW '07) — we defined an "angelic semantics" for HLAs, specifying for each HLA the set of states reachable by *some* refinement into a primitive action sequence. The angelic approach captures the fact that the *agent* will choose a refinement and can thereby choose which element of an HLA's reachable set is actually reached. This semantics guarantees the downward refinement property and yields a sound and complete hierarchical planning algorithm that derives significant speedups from its ability to generate and commit to provably correct abstract plans.

Our previous paper ignored action costs and hence our planning algorithm used no heuristic information, a mainstay of modern planners. The first objective of this paper is to rectify this omission. The angelic approach suggests the obvious extension: the exact cost of executing a high-level action to get from state s to state s' is the *least* cost among all primitive refinements that reach s'. In practice, however, representing the exact cost of an HLA from each state s to each reachable state s' is infeasible, and we develop concise lower and upper bound representations. From this starting point, we derive the first algorithm capable of generating *provably optimal* abstract plans. Conceptually, this algorithm is an elaboration of A*, applied in hierarchical plan space and modified to handle the special properties of refinement operators and use both upper and lower bounds. We also provide a satisficing algorithm that sacrifices optimality for computational efficiency and may be more useful in practice. Preliminary experimental results show that these algorithms outperform both "flat" and our previous hierarchical approach.

The paper also examines HLAs in the *online* setting, wherein an agent performs a limited lookahead prior to se-

[*]The authors appear in alphabetical order.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

lecting each action. The value of lookahead has been amply demonstrated in domains such as chess. We believe that *hierarchical* lookahead with HLAs can be far more effective because it brings back to the present value information from far into the future. Put simply, it's better to evaluate the possible outcomes of writing an ICAPS paper than the possible outcomes of choosing "A" as its first character. We derive an angelic hierarchical generalization of Korf's LRTA* (1990), which shares LRTA*'s guarantees of eventual goal achievement on each trial and eventually optimal behavior after repeated trials. Experiments show that this algorithm substantially outperforms its nonhierarchical ancestor.

Background

Planning Problems

Deterministic, fully observable planning problems can be described in a representation-independent manner by a tuple $(S, s_0, t, \mathcal{L}, T, g)$, where S is a set of states, s_0 is the initial state, t is the goal state,[1] $\mathcal{L}$ is a set of primitive actions, and $T : S \times \mathcal{L} \rightarrow S$ and $g : S \times \mathcal{L} \rightarrow \mathbb{R}$ are transition and cost functions such that doing action a in state s leads to state $T(s, a)$ with cost $g(s, a)$. These functions are overloaded to operate on sequences of actions in the obvious way: if $\mathbf{a} = (a_1, \ldots, a_m)$, then $T(s, \mathbf{a}) = T(\ldots T(s, a_1) \ldots, a_m)$ and $g(s, \mathbf{a})$ is the total cost of this sequence. The objective is to find a *solution* $\mathbf{a} \in \mathcal{L}^*$ for which $T(s_0, \mathbf{a}) = t$.

Definition 1. A solution $\mathbf{a}^*$ is *optimal* iff it reaches the goal with minimal cost: $\mathbf{a}^* = \operatorname{argmin}_{\mathbf{a} \in \mathcal{L}^* : T(s_0, \mathbf{a}) = t}\, g(s_0, \mathbf{a})$.

To ensure that an optimal solution exists, we require that every cycle in the state space has positive cost.

In this paper, we represent S as the set of truth assignments to some set of ground propositions, and T using the STRIPS language (Fikes & Nilsson 1971).

As a running example, we introduce a simple "nav-switch" domain. This is a grid-world navigation domain with locations represented by propositions $\mathrm{X}(x)$ and $\mathrm{Y}(y)$ for $x \in \{0, ..., x_{max}\}$ and $y \in \{0, ..., y_{max}\}$, and actions $\mathsf{U}, \mathsf{D}, \mathsf{L}$, and R that move between them. There is a single global "switch" that can face horizontally (H) or vertically ($\neg$H); move actions cost 2 if they go in the current direction of the switch and 4 otherwise. The switch can be toggled by action F with cost 1, but only from a subset of designated squares. The goal is always to reach a particular square with minimum cost. Since these goals correspond to 2 distinct states (H, $\neg$H), we add a dummy action Z with cost 0 that moves from these (pseudo-)goal states to the single terminal state t. For example, in a 2x2 problem ($x_{max} = y_{max} = 1$) where the switch can only be toggled from the top-left square $(0, 0)$, if the initial state s_0 is $\mathrm{X}(1) \wedge \mathrm{Y}(0) \wedge \mathrm{H}$, the optimal plan to reach the bottom-left square $(0, 1)$ is $(\mathsf{L}, \mathsf{F}, \mathsf{D}, \mathsf{Z})$ with cost 5.

High-Level Actions

In addition to a planning problem, our algorithms will be given a set $\mathcal{A}$ of *high-level actions*, along with a set $I(a)$ of allowed *immediate refinements* for each HLA $a \in \mathcal{A}$. Each immediate refinement consists of a finite sequence $\mathbf{a} \in \tilde{\mathcal{A}}^*$, where we define $\tilde{\mathcal{A}} = \mathcal{A} \cup \mathcal{L}$ as the set of all actions. Each HLA and refinement may have an associated precondition, which specifies conditions under which its use is appropriate.[2] To make a high-level sequence more concrete we may *refine* it, by replacing one of its HLAs by one of its immediate refinements, and we call one plan a *refinement* of another if it is reachable by any sequence of such steps. A *primitive refinement* consists only of primitive actions, and we define $I^*(\mathbf{a}, s)$ as the set of all primitive refinements of $\mathbf{a}$ that obey all HLA and refinement preconditions when applied from state s. Finally, we assume a special *top-level* action $\mathsf{Act} \in \mathcal{A}$, and restrict our attention to plans in $I^*(\mathsf{Act}, s_0)$.

Definition 2. (Parr & Russell 1998) A plan $\mathbf{a}^{h*}$ is *hierarchically optimal* iff $\mathbf{a}^{h*} = \operatorname{argmin}_{\mathbf{a} \in I^*(\mathsf{Act}, s_0) : T(s_0, \mathbf{a}) = t}\, g(s_0, \mathbf{a})$.

Remark. Because the hierarchy may constrain the set of allowed sequences, $g(s_0, \mathbf{a}^{h*}) \geq g(s_0, \mathbf{a}^*)$.

When equality holds from all possible initial states, the hierarchy is called *optimality-preserving*.

The hierarchy for our running example has three HLAs: $\mathcal{A} = \{\mathsf{Nav}, \mathsf{Go}, \mathsf{Act}\}$. $\mathsf{Nav}(x, y)$ navigates directly to location (x, y); it can refine to the empty sequence iff the agent is already at (x, y), and otherwise to any primitive move action followed by a recursive $\mathsf{Nav}(x, y)$. $\mathsf{Go}(x, y)$ is like Nav, except that it may flip the switch on the way; it either refines to $(\mathsf{Nav}(x, y))$, or to $(\mathsf{Nav}(x', y'), \mathsf{F}, \mathsf{Go}(x, y))$ where (x', y') can access the switch. Finally, Act is the top-level action, which refines to $(\mathsf{Go}(x_g, y_g), \mathsf{Z})$, where (x_g, y_g) is the goal location. This hierarchy is optimality-preserving for any instance of the nav-switch domain.

Optimistic and Pessimistic Descriptions for HLAs

As mentioned in the introduction, our angelic semantics (MRW '07) describes the outcome of a high-level plan by its *reachable set* of states (by some refinement). However, these reachable sets say nothing about *costs* incurred along the way. This section describes a novel extension of the angelic approach that includes cost information. This will allow us to find *good* plans *quickly* by focusing on better-seeming plans first, and pruning provably suboptimal high-level plans without refining them further. Due to lack of space, proofs are omitted.[3]

We begin with the notion of an *exact description* E_a of an HLA a, which specifies, for each pair of states (s, s'), the *minimum cost* of any primitive refinement of a that leads from s to s' (this generalizes our original definition).

Definition 3. The *exact description* of HLA a is a function $E_a(s)(s') = \inf_{\mathbf{b} \in I^*(a, s) : T(s, \mathbf{b}) = s'}\, g(s, \mathbf{b})$.

Remark. Definition 3 implies that if s' is not reachable from s by any refinement of a, $E_a(s)(s') = \infty$.

[1] A problem with multiple goal states can easily be translated into an equivalent problem with a single goal state.

[2] We treat these preconditions as *advisory*, so for our purposes a planning algorithm is complete even if it takes them into account, and sound even if it ignores them.

[3] An expanded paper with proofs is available at `http://www.cs.berkeley.edu/~jawolfe/angelic/`

We can think of descriptions as functions from states to *valuations* (themselves functions $S \rightarrow \mathbb{R} \cup \{\infty\}$) that specify a reachable set plus a finite cost for each reachable state (see Figure 1(b)). Then, descriptions can be extended to functions from valuations to valuations, by defining $\bar{E}_a(v)(s') = \min_{s \in S} v(s) + E_a(s)(s')$. Finally, these extended descriptions can be composed to produce descriptions for high-level *sequences*: the *exact* description of a high-level sequence $\mathbf{a} = (a_1, \ldots, a_N)$ is simply $\bar{E}_{a_N} \circ \ldots \circ \bar{E}_{a_1}$.

Definition 4. The *initial valuation* v_0 has $v_0(s_0) = 0$ and $v_0(s) = \infty$ for all $s \neq s_0$.

Theorem 1. *For any integer N, final state s_N, and action sequence $\mathbf{a} \in \tilde{\mathcal{A}}^N$, the minimum over all state sequences $(s_1, ..., s_{N-1})$ of total cost $\sum_{i=1}^{N} E_{a_i}(s_{i-1})(s_i)$ equals $\bar{E}_{a_N} \circ \ldots \circ \bar{E}_{a_1}(v_0)(s_N)$. Moreover, for any such minimizing state sequence, concatenating the primitive refinements of each HLA a_i that achieve the minimum cost $E_{a_i}(s_{i-1}, s_i)$ for each step yields a primitive refinement of $\mathbf{a}$ that reaches s_N from s_0 with minimal cost.*

Thus, an efficient, compact representation for E_a would (under mild conditions) lead to an efficient optimal planning algorithm. Unfortunately, since deciding even simple plan existence is PSPACE-hard (Bylander 1994), we cannot hope for this in general. Thus, we instead consider principled *compact approximations* to E_a that still allow for precise inferences about the effects and costs of high-level plans.

Definition 5. A valuation v_1 (weakly) *dominates* another valuation v_2, written $v_1 \preceq v_2$, iff $(\forall s \in S)\ v_1(s) \leq v_2(s)$.

Definition 6. An *optimistic description* O_a of HLA a satisfies $(\forall s)\ O_a(s) \preceq E_a(s)$.

For example, our optimistic description of Go (see Figure 1(a/c)) specifies that the cost for getting to the target location (possibly flipping the switch on the way) is at least twice its Manhattan distance from the current location; moreover, all other states are unreachable by Go.

Definition 7. A *pessimistic description* P_a of HLA a satisfies $(\forall s)\ E_a(s) \preceq P_a(s)$.

For example, our pessimistic description of Go specifies that the cost to reach the destination is at most four times its Manhattan distance from the current location.

(Optimistic and pessimistic descriptions generalize our previous complete and sound descriptions (MRW '07).)

Remark. For primitive actions $a \in \mathcal{L}$, $O_a(s)(s') = P_a(s)(s') = g(s, a)$ iff $s' = T(s, a), \infty$ otherwise.

In this paper, we will assume that the descriptions are given along with the hierarchy. However, we note that it is theoretically possible to derive them automatically from the structure of the hierarchy.

As with exact descriptions, we can extend optimistic and pessimistic descriptions and then compose them to produce bounds on the outcomes of high-level sequences, which we call *optimistic* and *pessimistic valuations* (see Figure 1(c/d)).

Theorem 2. *Given any sequence $\mathbf{a} \in \tilde{\mathcal{A}}^N$ and state s, the cost $c = \inf_{\mathbf{b} \in I^*(\mathbf{a}, s_0) | T(s_0, \mathbf{b}) = s}\ g(s_0, \mathbf{b})$ of the best primitive refinement of $\mathbf{a}$ that reaches s from s_0 satisfies $\bar{O}_{a_N} \circ \ldots \circ \bar{O}_{a_1}(v_0)(s) \leq c \leq \bar{P}_{a_N} \circ \ldots \circ \bar{P}_{a_1}(v_0)(s)$.*

(a) Properties of HLA Go(x_t, y_t) (precondition $\mathrm{X}(x_s) \wedge \mathrm{Y}(y_s)$)

refs	(Nav(x_t, y_t)) (Nav(x, y), F, Go(x_t, y_t)) $(\forall x, y)$ s.t. a switch at (x, y)
optimistic	$-\mathrm{X}(x_s)$, $-\mathrm{Y}(y_s)$, $+\mathrm{X}(x_t)$, $+\mathrm{Y}(y_t)$, $\tilde{\pm}\mathrm{H}$ *cost* $\geq 2 * (\lvert x_t - x_s \rvert + \lvert y_t - y_s \rvert)$
pessimistic	$-\mathrm{X}(x_s)$, $-\mathrm{Y}(y_s)$, $+\mathrm{X}(x_t)$, $+\mathrm{Y}(y_t)$ *cost* $\leq 4 * (\lvert x_t - x_s \rvert + \lvert y_t - y_s \rvert)$

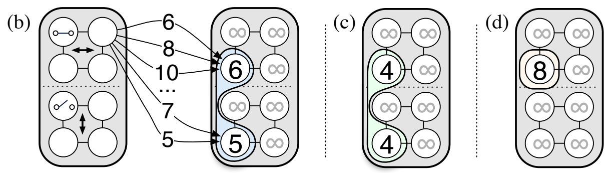


Figure 1: Some examples taken from our example nav-switch problem. (a) Refinements and NCSTRIPS descriptions of the Go HLA. (b) Exact valuation from s_0 for Go$(0, 1)$. Gray rounded rectangles represent the state space; in the top four states (circles) the switch is horizontal, and in the bottom four it is vertical. Each arrow represents a primitive refinement of Go$(0, 1)$; the cost assigned to each state is the min cost of any refinement that reaches it. The exact reachable set corresponding to this HLA is also outlined. (c) Optimistic simple valuation $\mathrm{X}(0) \wedge \neg\mathrm{X}(1) \wedge \neg\mathrm{Y}(0) \wedge \mathrm{Y}(1) : 4$ for the example in (b), as would be produced by the description in (a). (d) Pessimistic simple valuation $\mathrm{X}(0) \wedge \neg\mathrm{X}(1) \wedge \neg\mathrm{Y}(0) \wedge \mathrm{Y}(1) \wedge \mathrm{H} : 8$.

Moreover, following Theorem 1, these are the tightest bounds derivable from a set of HLA descriptions.

Representing and Reasoning with Descriptions

Whereas the results presented thus far are representation-independent, to utilize them effectively we require compact representations for valuations and descriptions as well as efficient algorithms for operating on these representations.

In particular, we consider *simple valuations* of the form $\sigma : c$ where $\sigma \subseteq S$ and $c \in \mathbb{R}$, which specify a *reachable set* of states along with a single numeric bound on the cost to reach states in this set (all other states are assigned cost ∞). As exemplified in Figure 1(c/d), an optimistic simple valuation asserts that states in σ *may* be reachable with cost at least c, and other states are *unreachable*; likewise, a pessimistic simple valuation asserts that each state in σ *is* reachable with cost at most c, and other states *may* be reachable as well.[4]

Simple valuations are convenient, since we can reuse our previous machinery (MRW '07) for reasoning with reachable sets represented as DNF (disjunctive normal form) logical formulae and HLA descriptions specified in a language called NCSTRIPS (Nondeterministic Conditional STRIPS). NCSTRIPS is an extension of ordinary STRIPS that can express a set of possible effects with mutually exclusive conditions. Each effect consists of four lists of propositions: add $(+)$, delete $(-)$, possibly-add $(\tilde{+})$, and possibly-delete $(\tilde{-})$. Added propositions are always made true in the resulting state, whereas possibly-added propositions may or may not be made true; in a pessimistic description, the agent can force either outcome, whereas in an optimistic one the outcome may not be controllable. By extending

[4]More interesting tractable classes of valuations are possible; for instance, rather than using a single numeric bound, we could allow linear combinations of indicator functions on state variables.

NCSTRIPS with cost bounds (which can be computed by arbitrary code), we produce descriptions suitable for the approach taken here. Figure 1(a) shows possible descriptions for Go in this extended language (as is typically the case, these descriptions could be made more accurate at the expense of conciseness by conditioning on features of the initial state).

With these representational choices, we require an algorithm for progressing a simple valuation represented as a DNF reachable set plus numeric cost bound through an extended NCSTRIPS description. If we perform this progression exactly, the output may not be a simple valuation (since different states in the reachable set may produce different cost bounds). Thus, we will instead consider an approximate progression algorithm that projects results back into the space of simple valuations. Applying this algorithm repeatedly will allow us to compute optimistic and pessimistic simple valuations for entire high-level sequences.

The algorithm is a simple extension of that given in (MRW '07), which progresses each (conjunctive clause, conditional effect) pair separately and then disjoins the results. This progression proceeds by (1) conjoining effect conditions onto the clause (and skipping this clause if a contradiction is created), (2) making all added (resp. deleted) literals true (resp. false), and finally (3) removing literals from the clause if false (resp. true) and possibly-added (resp. possibly-deleted). With our extended NCSTRIPS descriptions, each (clause, effect) pair also produces a cost bound. When progressing optimistic (resp. pessimistic) valuations, we simply take the min (resp. max) of all these bounds plus the initial bound to get the cost bound for the final valuation.[5]

Our above definitions need some minor modifications to allow for such approximate progression algorithms. For simplicity, we will absorb any additional approximation into our notation for the descriptions themselves:

Definition 8. An approximate progression algorithm corresponds to, for each extended optimistic and pessimistic description $\bar{O}_a$ and $\bar{P}_a$, (further) approximated descriptions $\tilde{O}_a$ and $\tilde{P}_a$. Call the algorithm *correct* if, for all actions a and valuations v, $\tilde{O}_a(v) \preceq \bar{O}_a(v)$ and $\bar{P}_a(v) \preceq \tilde{P}_a(v)$.

Intuitively, a progression algorithm is correct as long as the errors it introduces only further weaken the descriptions.

Theorem 3. *Theorem 2 still holds if we use any correct approximate progression algorithm, replacing each $\bar{O}_a$ and $\bar{P}_a$ with its further approximated counterpart $\tilde{O}_a$ and $\tilde{P}_a$.*

Offline Search Algorithms

This section describes algorithms for the *offline* planning setting, in which the objective is to quickly find a low-cost sequence of actions leading all the way from s_0 to t.

[5]A more accurate algorithm for pessimistic progression sorts the clauses by increasing pessimistic cost, computes the minimal prefix of this list whose disjunction covers all of the remaining clauses, and then restricts the max over cost bounds to clauses in this prefix. We did not implement this version, since it requires many potentially expensive subsumption checks.

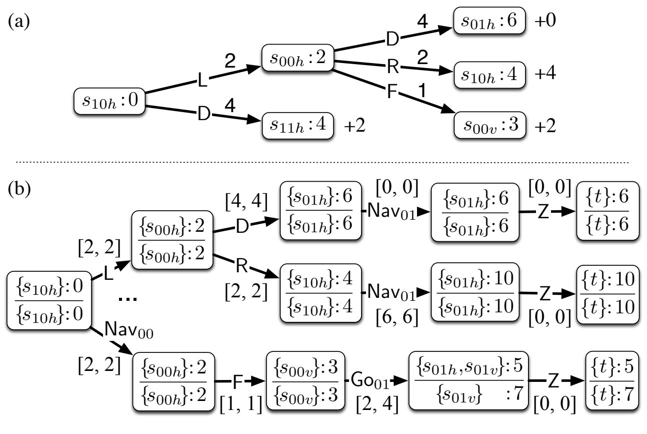


Figure 2: (a) A standard lookahead tree for our example. Nodes are labeled with states (written $s_{xy(h/v)}$) and costs-so-far, edges are labeled with actions and associated costs, and leaves have a heuristic estimate of the remaining distance-to-goal. (b) An *abstract* lookahead tree (ALT) for our example. Nodes are labeled with optimistic and pessimistic simple valuations and edges are labeled with (possibly high-level) actions and associated optimistic and pessimistic costs.

Because we have *models* for our HLAs, our planning algorithms will resemble existing algorithms that search over primitive action sequences. Such algorithms typically operate by building a *lookahead tree* (see Figure 2(a)). The initial tree consists of a single node labeled with the initial state and cost 0, and computations consist of leaf node *expansions*: for each primitive action a, we add an outgoing edge labeled with that action and its cost $g(s, a)$, whose child is labeled with the state $s' = T(s, a)$ and total cost to s'. We also include at leaf nodes a heuristic estimate $h(s')$ of the remaining cost to the goal. Paths from the root to a leaf are potential plans; for each such plan $\mathbf{a}$, we estimate the total cost of its best continuation by $f(s_0, \mathbf{a}) = g(s_0, \mathbf{a}) + h(T(s_0, \mathbf{a}))$, the sum of its cost and heuristic value. If the heuristic h never overestimates, we call it *admissible*, and this f-cost will also never overestimate. If h also obeys the triangle inequality $h(s) \leq g(s, a) + h(T(s, a))$, we call it *consistent*, and expanding a node will always produce extensions with greater or equal f-cost. These properties are required for A* and its graph version (respectively) to efficiently find optimal plans.

In hierarchical planning we will consider algorithms that build *abstract lookahead trees* (ALTs). In an ALT, edges are labeled with (possibly high-level) actions and nodes are labeled with optimistic and pessimistic valuations for corresponding partial plans. For example, in the ALT in Figure 2(b), by doing $(\mathsf{Nav}(0, 0), \mathsf{F}, \mathsf{Go}(0, 1))$, state s_{01v} is definitely reachable with cost in $[5, 7]$, s_{01h} may be reachable with cost at least 5, and no other states are possibly reachable. Since our planning algorithms will try to find low-cost solutions, we will be most concerned with finding optimistic (and pessimistic) bounds on the cost of the best primitive refinement of each high-level plan that reaches t. These bounds can be extracted directly from the final ALT node of each plan; for instance, the optimistic and pessimistic costs to t of plan $(\mathsf{Nav}(0, 0), \mathsf{F}, \mathsf{Go}(0, 1), \mathsf{Z})$ are $[5, 7]$.

We first present our optimal planning algorithm, AHA*, simultaneously introducing some of the issues that arise in our hierarchical planning framework. Then, we take a detour to describe our ALT data structures and how they address some of these issues in novel ways. Finally, we briefly describe an alternative "satisficing" algorithm, AHSS.

Angelic Hierarchical A*

Our first offline algorithm is *Angelic Hierarchical A** (AHA*), a hierarchically optimal planning algorithm that takes advantage of the semantic guarantees provided by optimistic and pessimistic descriptions. AHA* (see Algorithm 1) is essentially A* in *refinement space*, where the initial node is the plan (Act), possible "actions" are *refinements* of a plan at some HLA, and the goal set consists of the primitive plans that reach t from s_0. The algorithm repeatedly expands a node with smallest optimistic cost bound, until a goal node is chosen for expansion, which is returned as an optimal solution.

More concretely, at each step AHA* selects a high-level plan $\mathbf{a}$ with minimal optimistic cost to t (e.g., the bottom plan in Figure 2(b)). Then it *refines* $\mathbf{a}$, selecting some HLA a and adding to the ALT all plans obtained from $\mathbf{a}$ by replacing a with one of its immediate refinements.

While AHA* might seem like an obvious application of A* to the hierarchical setting, we believe that it is an important contribution for several reasons. First, its effectiveness hinges on our ability to generate nontrivial cost bounds for high-level sequences, which did not exist previously. Second, it derives additional power from our ALT data structures, which provide caching, pruning, and other novel improvements specific to the hierarchical setting.

The only free parameter in AHA* is the choice of which HLA to refine in a given plan; our implementation chooses an HLA with maximal gap between its optimistic and pessimistic costs (defined below), breaking ties towards higher-level actions.

Theorem 4. *AHA* is hierarchically optimal.*

AHA* always returns hierarchically optimal plans because optimistic costs are admissible, and always terminates as long as only finitely many plans have optimistic costs $\leq$ the optimal cost. This holds automatically if the problem is solvable, finite, and and has no nonpositive-optimistic-cost cycles.

In a generalization of the ordinary notion of consistency, we will sometimes desire *consistent* HLA descriptions, under which we never lose information by refining.[6] As in the flat case, when descriptions are consistent, the optimistic cost to t (i.e., f-cost) of a plan will never decrease with further refinement. Similarly, its best pessimistic cost will never increase. When consistency holds, as soon as AHA* finds an optimal high-level plan with equal optimistic and

Algorithm 1 : Angelic Hierarchical A*

```
function FINDOPTIMALPLAN($s_0, t$)
    $root \leftarrow$ MAKEINITIALALT($s_0, \{(\mathsf{Act})\}$)
    while $\exists$ an unrefined plan do
        $\mathbf{a} \leftarrow$ plan with min opt. cost to $t$ (tiebreak by pess. cost)
        if $\mathbf{a}$ is primitive then return $\mathbf{a}$
        REFINEPLANEDGE($root$, $\mathbf{a}$, index of any HLA in $\mathbf{a}$)
    return failure
```

pessimistic costs, it will find an optimal primitive refinement very efficiently. Consistency ensures that after each subsequent refinement, at least one of the resulting plans will also be optimal with equal optimistic and pessimistic costs; moreover, all but the first such plan will be skipped by the pruning described in the next section. Further refinement of this first plan will continue until an optimal primitive refinement is found *without backtracking*.

Abstract Lookahead Trees

Our ALT data structures support our search algorithms by efficiently managing a set of candidate high-level plans and associated valuations. The issues involved differ from the primitive setting because nodes store valuations rather than single states and exact costs, and because (unlike node expansion) plan refinement is "top-down" and may not correspond to simple extensions of existing plans.

Algorithm 2 shows pseudocode for some basic ALT operations. Our search algorithms work by first creating an ALT containing some initial set of plans using MAKEINITIALALT, and then repeatedly refining candidate plans using REFINEPLANEDGE, which only considers refinements whose preconditions are met by at least one state in the corresponding optimistic reachable set. Both operations internally call ADDPLAN, which adds a plan to the ALT by starting at the existing node corresponding to the longest prefix shared with any existing plan, and creating nodes for the remaining plan suffix by progressing its valuations through the corresponding action descriptions. In the process, partial plans that are provably dominated and plans that cannot possibly reach the goal are recognized and skipped over.

Theorem 5. *If a node n with optimistic valuation $O(n)$ is created while adding plan* $\mathbf{a}$, *and another node n' exists with pessimistic valuation $P(n')$ s.t. $P(n') \preceq O(n)$ and the remaining plan suffix of* $\mathbf{a}$ *is a legal hierarchical continuation from n', then* $\mathbf{a}$ *is safely prunable.*

(The continuation condition is needed since the hierarchy might allow better continuations from node n than n'.)

For example, the plan $(\mathsf{L}, \mathsf{R}, \mathsf{Nav}(0,1), \mathsf{Z})$ in Figure 2(b) is prunable since its optimistic valuation is dominated by the pessimistic valuation above it, and the empty continuation is allowed from that node. Since detecting all pruned nodes can be very expensive, our implementation only considers pruning for nodes with singleton reachable sets.

One might wonder why REFINEPLANEDGE refines a *single* plan at a given HLA edge, rather than simultaneously refining all plans that pass through it. The reason is that after each refinement of the HLA, it would have to continue progression for each such plan's suffix. This could be need-

[6] Specifically, a set of optimistic descriptions (plus approximate progression algorithm, if applicable) is consistent iff, when we refine any high-level plan, its optimistic valuation dominates the optimistic valuations of its refinements. A set of pessimistic descriptions (plus progression algorithm) is consistent iff the state-wise minimum of a set of refinements' pessimistic valuations always dominates the pessimistic valuation of the parent plan.

Algorithm 2 : Abstract lookahead tree (ALT) operations

```
function ADDPLAN($n$, $(a_1, ..., a_k)$)
    for $i$ from 1 to k do
        if node $n[a_i]$ does not exist then
            create $n[a_i]$ from $n$ and the descriptions of $a_i$
            if $n[a_i]$ is prunable via Theorem 5 then return
        $n \leftarrow n[a_i]$
    if $O(n)(t) < \infty$ then mark $n$ as a valid refinable plan

function MAKEINITIALALT($s_0$, $plans$)
    $root \leftarrow$ a new node with $O(root) = P(root) = v_0$
    for each $plan \in plans$ do ADDPLAN($root, plan$)
    return $root$

function REFINEPLANEDGE($root$, $(a_1, ..., a_k)$, $i$)
    mark node $root[a_1]...[a_k]$ as refined
    for $(b_1...b_j) \in I(a_i)$ w/ prec. met by $O(root[a_1]...[a_{i-1}])$ do
        ADDPLAN($root$, $(a_1, ..., a_{i-1}, b_1, ..., b_j, a_{i+1}, ..., a_k)$)
    $(o, p) \leftarrow (\min, \max)$ of the (opt., pess.) costs of $a_i$'s refs
    $a_i$'s opt. cost $\leftarrow \max$(current value, $o$)      /*  upward   */
    $a_i$'s pess. cost $\leftarrow \min$(current value, $p$)     /* propagation */
```

lessly expensive, especially if some such plans are already thought to be bad.

In any case, when valuations are simple, we can use a novel improvement called *upward propagation* (implemented in REFINEPLANEDGE) to propagate new information about the cost of a refined HLA edge to other plans that pass through it, without having to explicitly refine them or do any additional progression. This improvement hinges on the fact that with simple valuations, the optimistic and pessimistic costs for a plan can be broken down into optimistic and pessimistic costs for each *action* in that plan (see Figure 2(b)).

Theorem 6. *The* min *optimistic cost of any refinement of HLA a is a valid optimistic cost for a's current optimistic reachable set, and when pessimistic descriptions are consistent, the* max *such pessimistic cost is similarly valid.*

Thus, upon refining an HLA edge, we can tighten its cost interval to reflect the cost intervals of its immediate refinements, *without modifying its reachable sets*. This results in better cost bounds for all other plans that pass through this HLA edge, without needing to do any additional progression computations for (the suffixes of) such plans. [7]

Angelic Hierarchical Satisficing Search

This section presents an alternative algorithm, Angelic Hierarchical Satisficing Search (AHSS), which attempts to find a plan that reaches the goal with at most some pre-specified cost α. AHSS can be much more efficient than AHA*, since it can commit to a plan without first proving its optimality.

At each step, AHSS (see Algorithm 3) begins by checking if any primitive plans succeed with cost $\leq \alpha$. If so, the best such plan is returned. Next, if any (high-level) plans succeed with pessimistic cost $\leq \alpha$, the best such plan is committed to by discarding other potential plans. Finally, a plan with maximum *priority* is refined at one of its HLAs. Priorities can be assigned arbitrarily; our implementation uses the negative average of optimistic and pessimistic costs, to encourage a more depth-first search and favor plans with smaller pessimistic cost.

[7]Note that changing the costs renders the valuations stored at child nodes of the refined edge out-of-date. The plan selection step of AHA* can nevertheless be done correctly, by storing "Q-values" of each edge in the tree, and backing up Q-values up to the root whenever upward propagation is done.

Algorithm 3 : Angelic Hierarchical Satisficing Search

```
function FINDSATISFICINGPLAN($s_0, t, \alpha$)
    $root \leftarrow$ MAKEINITIALALT($s_0$, {Act})
    while $\exists$ an unrefined plan with optimistic cost $\leq \alpha$ to $t$ do
        if any plan has pessimistic cost $\leq \alpha$ to $t$ then
            if any such plans are primitive then return a best one
            else delete all plans other than one with min pess. cost
        $\mathbf{a} \leftarrow$ a plan with optimistic cost $\leq \alpha$ to $t$ with max priority
        REFINEPLANEDGE($root$, $\mathbf{a}$, index of any HLA in $\mathbf{a}$)
    return failure
```

Theorem 7. *AHSS is sound and complete.*

If any hierarchical plans reach the goal with cost $\leq \alpha$, AHSS will return one of them; otherwise, it will return failure. (Termination is guaranteed as long as only a finite number of high-level plans have optimistic costs $\leq \alpha$.) Like AHA*, when HLA descriptions are consistent, once AHSS finds a satisficing high-level plan it will find a satisficing primitive refinement in a backtrack-free search.

Online Search Algorithms

In the *online* setting, an agent must begin executing actions without first searching all the way to the goal. The agent begins in the initial state s_0, performs a fixed amount of computation, then selects an action a.[8] It then does this action in the environment, moving to state $T(s_0, a)$ and paying cost $g(s_0, a)$. This continues until the goal state t is reached. Performance is measured by the total cost of the actions executed. We assume that the state space is *safely explorable*, so that the goal is reachable from any state (with finite cost), and also assume positive action costs and consistent heuristics/descriptions from this point forward.

This section presents our next contribution, one of the first *hierarchical lookahead* algorithms. Since it will build upon a variant of Korf's (1990) Learning Real-Time A* (LRTA*) algorithm, we begin by briefly reviewing LRTA*.[9]

At each environment step, LRTA* uses its computation time to build a lookahead tree consisting of all plans $\mathbf{a}$ whose cost $g(s_0, \mathbf{a})$ just exceeds a given threshold. Then, it selects one such plan $\mathbf{a}_{min}$ with minimal f-cost and does its first action in the world. Intuitively, looking farther ahead should increase the likelihood that $\mathbf{a}_{min}$ is actually good, by decreasing reliance on the (error-prone) heuristic. The choice of candidate plans is designed to compensate for the fact that the heuristic h is typically biased (i.e., admissible) whereas g is exact, and thus the f-cost of a plan with higher h and

[8]More interesting ways to balance real-world and computational cost are possible, but this suffices for now.

[9]To be precise, Korf focused on the case of unit action costs; we present the natural generalization to positive real-valued costs.

lower g may not be directly comparable to one with higher g and lower h.

This core algorithm is then improved by a learning rule. Whenever a partial plan $\mathbf{a}$ leading to a previously-visited state s is encountered during search, further extensions of $\mathbf{a}$ are not considered; instead, the remaining cost-to-goal from s is taken to be the value computed by the most recent search at s. This augmented algorithm has several nice properties:

Theorem 8. *(Korf 1990) If g-costs are positive, h-costs are finite, and the state space is finite and safely explorable, then LRTA* will eventually reach the goal.*

Theorem 9. *(Korf 1990) If, in addition, h is admissible and ties are broken randomly, then given enough runs, LRTA* will eventually learn the true cost of every state on an optimal path, and act optimally thereafter.*

However, as described thus far, LRTA* has several drawbacks. First, it wastes time considering obviously bad plans. (Korf prevented this with "alpha pruning"). Second, a cost threshold must be set in advance, and picking this threshold so that the algorithm uses a desired amount of computation time may be difficult. Both drawbacks can solved using the following *adaptive LRTA** algorithm, a relative of Korf's "time-limited A*": (1) Start with the empty plan. (2) At each step, select an unexpanded plan with lowest f-cost. If this plan has greater g-cost than any previously expanded plan, "lock it in" as the current return value. Expand this plan. (3) When computation time runs out, return the current "locked-in" plan.

Theorem 10. *At any point during the operation of this algorithm, let $\mathbf{a}$ be the current locked-in plan, c_2 be its corresponding "record-setting" g-cost, and c_1 be the previous record g-cost ($c_1 < c_2$). Given any threshold in $[c_1, c_2)$, LRTA* would choose $\mathbf{a}$ for execution (up to tiebreaking).*

Thus, this modified algorithm can be used as an efficient, anytime version of LRTA*. Since its behaviour reduces to the original version for a particular (adaptive) choice of cost thresholds, all of the properties of LRTA* hold for it as well.

Angelic Hierarchical Learning Real-Time A*

This section describes Angelic Hierarchical Learning Real-Time A* (AHLRTA*, see Algorithm 4), which bears (roughly) the same relation to adaptive LRTA* as AHA* does to A*. Because a single HLA can correspond to many primitive actions, for a given amount of computation time we hope that AHLRTA* will have a greater effective lookahead depth than LRTA*, and thus make better action choices. However, a number of issues arise in the generalization to the hierarchical setting that must be addressed to make this basic idea work in both theory and practice.

First, while AHLRTA* searches over the space of high-level plans, when computation time runs out it must choose a *primitive* action to execute. Thus, if the algorithm initializes its ALT with the single plan (Act), it will have to consider its refinements carefully to ensure that in its final ALT, at least one of the (hopefully better) high-level plans begins with an executable primitive. To avoid this issue (and to ensure convergence of costs, as described below), we instead choose to initialize the ALT with the set of all plans consisting of

Algorithm 4 : Angelic Hierarchical Learning Real-Time A*

```
function HIERARCHICALLOOKAHEADAGENT($s_0, t$)
  $memory \leftarrow$ an empty hash table
  while $s_0 \neq t$ do
    $root \leftarrow$ MAKEINITIALALT($s_0, \{(a, \mathsf{Act}) \mid a \in \mathcal{L}\}$)
    $(g, a, f) \leftarrow (-1, nil, 0)$
    while $\exists$ unrefined plans from $root \wedge$ time remains do
      $\mathbf{a} \leftarrow$ a plan w/ min $f$-cost
      if the $g$-cost of $\mathbf{a} > g$ then
        $(g,a,f) \leftarrow$ ($g$-cost of $\mathbf{a}$, $a_1$, $f$-cost of $\mathbf{a}$)
      REFINEPLANEDGE($root$, $\mathbf{a}$, some index, $memory$)
    do $a$ in the world
    $memory[s_0] \leftarrow f$
    $s_0 \leftarrow T(s_0, a)$
```

a primitive action followed by Act.[10] With this set of plans, the choice of which HLA to refine in a plan is open; our implementation uses the policy described above for AHA*.

Second, as we saw earlier, an analogue of f-cost can be extracted from our optimistic valuations. However, there is no obvious breakdown of f into g and h components, since a high-level plan can consist of actions at various levels, each of whose descriptions may make different types and degrees of characteristic errors. For now, we assume that a set of higher-level HLAs (e.g., Act and Go) has been identified, let h be the sum of the optimistic costs of these actions, and let $g = f - h$ be the cost of the primitives and remaining HLAs.

Finally, whereas the outcome of a primitive plan is a particular concrete state whose stored cost can be simply looked up in a hash table, the optimistic valuations of a high-level plan instead provide a *sequence* of *reachable sets* of states. In general, for each such set we could look up and combine the stored costs of its elements; instead, however, for efficiency our implementation only checks for stored costs of singleton optimistic sets (e.g., those corresponding to a primitive prefix of a given high-level plan). If the state in a constructed singleton set has a stored cost, progression is stopped and this value is used as the cost of the remainder of the plan. This functionality is added by modifying REFINE-PLANEDGE and ADDPLAN accordingly (not shown).

Given all of these choices, we have the following:

Theorem 11. *AHLRTA* reduces to adaptive LRTA*, given a "flat" hierarchy in which Act refines to any primitive action followed by Act (or the empty sequence).*

(In fact, this is how we have implemented LRTA* for our experiments.) Moreover, the desirable properties of LRTA* also hold for AHLRTA* in general hierarchies. This follows because AHLRTA* behaves identically to LRTA* in neighborhoods in which every state has been visited at least once.

Theorem 12. *If primitive g-costs are positive, f-costs are finite, and the state space is finite and safely explorable, then AHLRTA* will eventually reach the goal.*

Theorem 13. *If, in addition, f-costs are admissible, ties*

[10]Note that with this choice, the plans considered by the agent may not be valid hierarchical plans (i.e., refinements of Act). However, since the agent can change its mind on each world step, the actual sequence of actions executed in the world is not in general consistent with the hierarchy anyway.

are broken randomly, and the hierarchy is optimality-preserving, then over repeated trials AHLRTA will eventually learn the true cost of every state on an optimal path and act optimally thereafter.*

If f-costs are inadmissible or the hierarchy is not optimality-preserving, the theorem still holds if s_0 is sampled from a distribution with support on S in each trial.

Our implementation of AHLRTA* includes two minor changes from the version described above, which we have found to increase its effectiveness. First, it sometimes throws away some of its allowed computation time, so that the number of refinements taken per allowed initial primitive action is constant; this tends to improve the interaction of the lookahead strategy with the learning rule. Second, when deciding when to "lock in" a plan it requires additionally that the plan is more refined than the previous locked in plan; this helps counteract the implicit bias towards higher-level plans caused by aggregation of costs from primitives and various HLAs into g-cost. Since both changes effectively only change the stopping time of the algorithm, its desirable properties are preserved.

Experiments

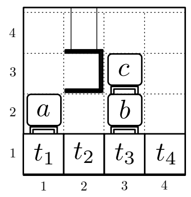
This section describes results for the above algorithms on two domains: our "nav-switch" running example, and the *warehouse world* (MRW '07).[11]

The warehouse world is an elaboration of the well-known blocks world, with discrete spatial constraints added. In this domain, a forklift-like gripper hanging from the ceiling can move around and manipulate blocks stacked on a table. Both gripper and blocks occupy single squares in a 2-d grid of allowed positions. The gripper can move to free squares in the four cardinal directions, turn (to face the other way) when in the top row, and pick up and put down blocks from either side. Each primitive action has unit cost. Because of the limited maneuvering space, warehouse world problems can be rather difficult. For instance, Figure 3 shows a problem that cannot be solved in fewer than 50 primitive steps. The figure also shows our HLAs for the domain, which we use unchanged from (MRW '07) along with the NCSTRIPS descriptions therein (to which we add simple cost bounds). We consider six instances of varying difficulty.

For the nav-switch domain, we consider square grids of varying size with 3 randomly placed switches, where the goal is always to navigate from one corner to the other. We use the hierarchy and descriptions described above.

We first present results for our offline algorithms on these domains (see Table 1). On the warehouse world instances, nonhierarchical (flat) A* does reasonably well on small problems, but quickly becomes impractical as the optimal plan length increases. AHA* is able to plan optimally in larger problems, but for the largest instances, it too runs out of time. The reason is that it must not only find the optimal plan, but also prove that all other high-level plans have higher cost. In contrast, AHSS with a threshold of ∞ is able to solve all the problems fairly quickly.

[11]Our code is available at
`http://www.cs.berkeley.edu/~jawolfe/angelic/`



HLA	Goal
Act	Achieve goal by seq. of Moves
Move(b, c)	Stack block b on c by NavT to one side of b, pick up, NavT to side of c, put down.
NavT(x, y)	Go to (x, y), possibly turning
Nav(x, y)	Go directly to (x, y)

Figure 3: Left: A 4x4 warehouse world problem with goal $\text{ON}(c, t_2) \wedge \text{ON}(a, c)$. Right: HLAs for warehouse world domain.

nav-switch

sz	A*	AHA*	AHSS
5	0	0	0
10	22	1	1
20	176	3	3
40	–	40	40

warehouse world

#	A*	AHA*	AHSS	HFS
1	1	0	0	1
2	9	4	2	12
3	–	63	9	135
4	–	526	27	–
5	–	–	60	–
6	–	–	48	–

Table 1: Run-times of offline algorithms, rounded to the nearest second, on some nav-switch and warehouse world problem instances. The algorithms are (flat) graph A*, AHA*, AHSS with threshold $\alpha{=}\infty$, and HFS from (MRW '07). Algorithms were terminated if they failed to return within 10^4 seconds (shown as "–").

We also included, for comparison, results for the Hierarchical Forward Search (HFS) algorithm (MRW '07), which does not consider plan cost. When passed a threshold of ∞, AHSS has the same objective as HFS: to find any plan from s_0 to t with as little computation as possible. However, AHSS has several important advantages over HFS. First, its priority function serves as a heuristic, and usually results in higher-quality plans being found. Second, AHSS is actually much simpler. In particular, whereas HFS required iterative deepening, cycle checking, and a special plan decomposition mechanism to ensure completeness and efficiency, the use of cost information allows AHSS to naturally reap the same benefits without needing any such explicit mechanisms. Finally, the abstract lookahead tree data structure provides caching and decreases the number of NCSTRIPS progressions required. Due to these improvements, HFS is slightly slower than the *optimal* planner AHA*, and a few orders of magnitude slower than AHSS.

On the nav-switch instances, results are qualitatively similar. Again, flat A* quickly becomes impractical as the problem size grows. However, in this domain, AHA* actually performs very well, almost matching the performance of AHSS. The reason is that in this domain, the descriptions for Nav are exact, and thus AHA* can very quickly find a provably optimal high-level plan and refine it down to the primitive level without backtracking, as described earlier.

The obvious next step would be to compare AHA* with other optimal hierarchical planners, such as SHOP2 on its "optimal" setting. However, this is far from straightforward, for several reasons. First, useful hierarchies are often not optimality-preserving, and it is not at all obvious how we should compare different "optimal" planners that use different standards for optimality. Second, as described in the related work section below, the type and amount of problem-specific information provided to our algorithms can be very different than for HTN planners such as SHOP2. We have yet to find a way to perform meaningful experimental com-

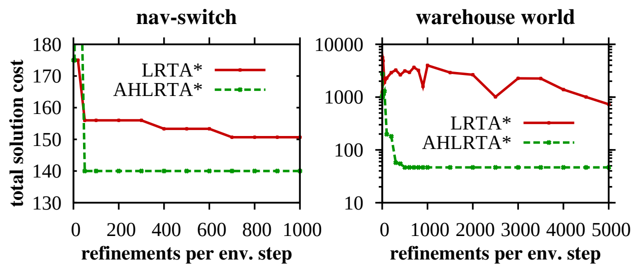


Figure 4: Total cost-to-goal for online algorithms as a function of the number of allowed refinements per environment step, averaged over three instances each of the nav-switch domain (left) and warehouse world (right). (Warehouse world costs shown in log-scale.)

parisons under these circumstances.

For the online setting, we compared (flat) LRTA* and AHLRTA*. The performance of an online algorithm on a given instance depends on the number of allowed refinements per step. Our graphs therefore plot total cost against refinements per step for LRTA* and AHLRTA*. AHLRTA* took about five times longer per refinement than LRTA* on average, though this factor could probably be decreased by optimizing the DNF operations. [12]

The left graph of Figure 4 is averaged across three instances of the nav-switch world. This domain is relatively easy as an online lookahead problem, because the Manhattan-distance heuristic for Act always points in roughly the right direction. In all cases, the hierarchical agent behaved optimally given about 50 refinements per step. With this number of refinements, the flat agent usually followed a reasonable, though suboptimal plan. But it did not display optimal behaviour, even when the number of refinements per step was increased to 1000.

The right graph in Figure 4 shows results averaged across three instances of the warehouse world. This domain is more challenging for online lookahead, as the combinatorial structure of the problem makes the Act heuristic less reliable. AHLRTA* started to behave optimally given a few hundred refinements per step. In contrast, flat lookahead was very suboptimal (note that the y-axis is on a log scale), even given five thousand refinements.

Here are some qualitative phenomena we observed on the experiments (data will be provided in full paper). First, as the number of refinements increased, AHLRTA* reached a point where it found a provably optimal primitive plan on each environment step. But it also had reasonable behavior when the number of refinements did not suffice to find a provably optimal plan (the left portion of the righthand graph), in that the "intended" plan at each step typically consisted of a few primitive actions followed by increasingly high-level actions, and this intended plan was usually reasonable at the high level. Second, when very few refinements (< 50) were allowed per step, AHLRTA* actually did worse than LRTA* on (a single instance of) the nav-switch world. While we do not completely understand the cause, what seems to be happening is that in the regime of very little deliberation time per step, lookahead pathologies and the LRTA* learning rule interact in complex ways, often causing the agent to spend long periods of time "filling out" local minima of the heuristic function in the state space. [13] This phenomenon is further complicated in the hierarchical case by the fact that the cost bounds for different HLAs tend to be systematically biased in different ways (for example, the optimistic bound for Nav is nearly exact, while the bound for Move tends to underestimate by a factor of two). Improved online lookahead algorithms that degrade gracefully in such situations, even given very little deliberation time, are an interesting topic for future work.

[12]It cannot be completely avoided because refinements for the hierarchical algorithms require multiple progressions.

Related Work

We briefly describe work related to our specific contributions, deferring to (MRW '07) for discussion of relationships between this general line of work and previous approaches.

Most previous work in hierarchical planning (Tate 1977; Yang 1990; Russell & Norvig 2003) has viewed HLA descriptions (when used at all) as constraints on the planning process (e.g., "only consider refinements that achieve p"), rather than as making true assertions about the effects of HLAs. Such HTN planning systems, e.g., SHOP2 (Nau *et al.* 2003), have achieved impressive results in previous planning competitions and real-world domains—despite the fact that they cannot assure the correctness or bound the cost of abstract plans. Instead, they encode a good deal of domain-specific advice on which refinements to try in which circumstances, often expressed as arbitrary program code. For fairly simple domains described in tens of lines of PDDL, SHOP2 hierarchies can include hundreds or thousands of lines of Lisp code. In contrast, our algorithms only require a (typically simple) hierarchical structure, along with descriptions that logically follow from (and are potentially automatically derivable from) this structure.

The closest work to ours is by Doan and Haddawy (1995). Their DRIPS planning system uses action abstraction along with an analogue of our optimistic descriptions to find optimal plans in the probabilistic setting. However, without pessimistic descriptions, they can only prove that a given high-level plan satisfies some property when the property holds for *all of its refinements*, which severely limits the amount of pruning possible compared to our approach. Helwig and Haddawy (1996) extended DRIPS to the online setting. Their algorithm did not cache backed-up values, and hence cannot guarantee eventual goal achievement, but it was probably the first principled online hierarchical lookahead agent.

Several other works have pursued similar goals to ours, but using *state abstraction* rather than HLAs. Holte *et al.* (1996) developed Hierarchical A*, which uses an automatically constructed hierarchy of state abstractions in which the results of optimal search at each level define an admissible heuristic for search at the next-lower level. Similarly, Bulitko *et al.* (2007) proposed the PR LRTS algorithm, a

[13]This is also why the LRTA* curve in the warehouse world is nonmonotonic.

real-time algorithm in which a plan discovered at each level constrains the planning process at the next-lower level.

Finally, other works have considered adding pessimistic bounds to the A* (Berliner 1979) and LRTA* (Ishida & Shimbo 1996) algorithms, to help guide search and exploration as well as monitor convergence. These techniques may also be useful for our corresponding hierarchical algorithms.

Discussion

We have presented several new algorithms for hierarchical planning with promising theoretical and empirical properties. There are many interesting directions for future work, such as developing better representations for descriptions and valuations, automatically synthesizing descriptions from the hierarchy, and generalizing domain-independent techniques for automatic derivation of planning heuristics to the hierarchical setting. One might also consider extensions to partially ordered, probabilistic, and partially observable settings, and better online algorithms that, e.g., maintain more state across environment steps.

Acknowledgements

Bhaskara Marthi thanks Leslie Kaelbling and Tomas Lozano-Perez for useful discussions. This research was also supported by DARPA IPTO, contracts FA8750-05-2-0249 and FA8750-07-D-0185 (subcontract 03-000219).

References

Berliner, H. 1979. The B* Tree Search Algorithm: A Best-First Proof Procedure. *Artif. Intell.* 12:23–40.

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph Abstraction in Real-time Heuristic Search. *JAIR* 30:51–100.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artif. Intell.* 69:165–204.

Doan, A., and Haddawy, P. 1995. Decision-theoretic refinement planning: Principles and application. Technical Report TR-95-01-01, Univ. of Wisconsin-Milwaukee.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.* 2:189–208.

Helwig, J., and Haddawy, P. 1996. An Abstraction-Based Approach to Interleaving Planning and Execution in Partially-Observable Domains. In *AAAI Fall Symposium*.

Holte, R.; Perez, M.; Zimmer, R.; and MacDonald, A. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI*.

Ishida, T., and Shimbo, M. 1996. Improving the learning efficiencies of realtime search. In *AAAI*.

Korf, R. E. 1990. Real-Time Heuristic Search. *Artif. Intell.* 42:189–211.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2007. Angelic Semantics for High-Level Actions. In *ICAPS*.

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Nau, D.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, W. J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *JAIR* 20:379–404.

Parr, R., and Russell, S. 1998. Reinforcement Learning with Hierarchies of Machines. In *NIPS*.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition.

Tate, A. 1977. Generating project networks. In *IJCAI*.

Yang, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Comput. Intell.* 6(1):12–24.