In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico. Morgan Kaufmann, 1233-1238, 2003.

TECHNICAL REPORT
R-317
August 2003

Faster Heuristic Search Algorithms for Planning with Uncertainty and Full Feedback

Blai Bonet
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024, USA
bonet@cs.ucla.edu

Héctor Geffner
Departamento de Tecnología
ICREA – Universitat Pompeu Fabra
Barcelona 08003, España
hector.geffner@tecn.upf.es

Abstract

Recent algorithms like RTDP and LAO* combine the strength of Heuristic Search (HS) and Dynamic Programming (DP) methods by exploiting knowledge of the initial state and an admissible heuristic function for producing optimal policies without evaluating the entire space. In this paper, we introduce and analyze three new HS/DP algorithms. A first general algorithm schema that is a simple loop in which 'inconsistent' reachable states (i.e., with residuals greater than a given ϵ) are found and updated until no such states are found, and serves to make explicit the basic idea underlying HS/DP algorithms, leaving other commitments aside. A second algorithm, that builds on the first and adds a labeling mechanism for detecting solved states based on Tarjan's strongly-connected components procedure, which is very competitive with existing approaches. And a third algorithm, that approximates the latter by enforcing the consistency of the value function over the 'likely' reachable states only, and leads to great time and memory savings, with no much apparent loss in quality, when transitions have probabilities that differ greatly in value.

1 Introduction

Heuristic search algorithms have been successfully used for computing optimal solutions to large deterministic problems (e.g., [Korf, 1997]). In the presence of non-determinism and feedback, however, solutions are not action sequences but policies, and while such policies can be characterized and computed by dynamic programming (DP) methods [Bellman, 1957; Howard, 1960], DP methods take all states into account and thus cannot scale up to large problems [Boutilier *et al.*, 1999]. Recent algorithms like RTDP [Barto *et al.*, 1995] and LAO* [Hansen and Zilberstein, 2001], combine the strength of Heuristic Search and Dynamic Programming methods by exploiting knowledge of the initial state s_0 and an admissible heuristic function (lower bound) h for computing optimal policies without having to evaluate the entire space. In this paper we aim to contribute to the theory and practice of Heuristic Search/Dynamic Programming methods by formulating and analyzing three new HS/DP algorithms. The first algorithm, called FIND-and-REVISE is a general schema that comprises a loop in which states reachable from s_0 and the greedy policy that have Bellman residuals greater than a given ϵ (we call them 'inconsistent' states), are found and updated until no one is left. FIND-and-REVISE makes explicit the basic idea underlying HS/DP algorithms, including RTDP and LAO*. We prove the convergence, complexity, and optimality of FIND-and-REVISE, and introduce the second algorithm, HDP, that builds on it, and adds a labeling mechanism for detecting *solved* states based on Tarjan's efficient strongly-connected components procedure [Tarjan, 1972]. A state is solved when it reaches only 'consistent' states, and solved states are skipped in all future searches. HDP terminates when the initial state is solved. HDP inherits the convergence and optimality properties of the general FIND-and-REVISE schema and is strongly competitive with existing approaches. The third algorithm, HDP(i), is like HDP, except that while HDP computes a value function by enforcing its consistency over all reachable states (i.e., reachable from s_0 and the greedy policy), HDP(i) enforces consistency over the 'likely' reachable states only. We show that this approximation, suitably formalized, can lead to great savings in time and memory, with no much apparent loss in quality, when transitions have probabilities that differ greatly in value.

Our motivation is twofold: to gain a better understanding of HS/DP methods for planning with uncertainty, and to develop more effective HS/DP algorithms for both optimal and approximate planning.

2 Preliminaries

2.1 Model

We model non-deterministic planning problems with full feedback with state models that differ from those used in the classical setting in two ways: first, state transitions become *probabilistic*; second, states are fully *observable*. The resulting models are known as Markov Decision Processes (MDPs) and more specifically as *Stochastic Shortest-Path Problems* [Bertsekas, 1995], and they are given by:[1]

M1. a discrete and finite state space S,
M2. an initial state $s_0 \in S$,
M3. a set $G \subseteq S$ of goal states,
M4. actions $A(s) \subseteq A$ applicable in each state $s \in S$,
M5. transition probabilities $P_a(s'|s)$ for $s \in S$, $a \in A(s)$,

[1] For discounted and other formulations, see [Puterman, 1994].

M6. positive action costs $c(a,s) > 0$, and
M7. fully observable states.

Due to the presence of full feedback (M7) and the standard Markovian assumptions, the solution of an MDP takes the form of a function π mapping states s into actions $a \in A(s)$. Such a function is called a *policy*. A policy π assigns a probability to every state trajectory $s_0, s_1, s_2, \ldots$ starting in state s_0 which is given by the product of the transition probabilities $P_{a_i}(s_{i+1}|s_i)$ where $a_i = \pi(s_i)$. If we further assume that actions in goal states have no costs and produce no changes (i.e., $c(a,s) = 0$ and $P_a(s|s) = 1$ if $s \in G$), the *expected cost* associated with a policy π starting in state s_0 is given by the weighted average of the probability of such trajectories times their cost $\sum_{i=0}^{\infty} c(\pi(s_i), s_i)$. An *optimal solution* is a policy π^* that has a minimum expected cost for all possible initial states. An optimal solution is guaranteed to exist provided the following assumption holds [Bertsekas, 1995]:

M8. the goal is reachable from every state with non-zero probability.

Since the initial state s_0 of the system is fixed, there is in principle no need to compute a complete policy but a *partial* policy prescribing the action to take in the states that can be reached following the policy from s_0. Traditional dynamic programming methods like value or policy iteration compute complete policies, while recent heuristic search DP methods like RTDP and LAO* compute partial policies. They achieve this by means of suitable heuristic functions $h(s)$ that provide admissible estimates (lower bounds) of the expected cost to reach the goal from any state s.

2.2 Dynamic Programming

Any heuristic or value function h defines a *greedy policy* π_h:

$$\pi_h(s) \stackrel{\text{def}}{=} \operatorname*{argmin}_{s \in A(s)} c(a,s) + \sum_{s' \in S} P_a(s'|s)\, h(s') \qquad (1)$$

where the expected cost from the resulting states s' is assumed to be given by $h(s')$. We call $\pi_h(s)$ the *greedy action* in s for the value function h. If we denote the optimal (expected) cost from a state s to the goal by $V^*(s)$, it is well known that the greedy policy π_h is *optimal* when h is the *optimal cost function*, i.e. $h = V^*$.

While due to the possible presence of ties in (1), the greedy policy is not unique, we will assume throughout the paper that these ties are broken systematically using an static ordering on actions. As a result, every value function V defines a *unique* greedy policy π_V, and the *optimal* cost function V^* defines a unique optimal policy π_{V^*}. We define the *relevant states* as the states that are reachable from s_0 using this optimal policy; they constitute a minimal set of states over which the optimal value function needs to be defined.[2]

Value iteration (VI) is a standard dynamic programming method for solving MDPs and is based on computing the optimal cost function V^* and plugging it into the greedy policy (1). This optimal cost function is the only solution to the fixed point equation:

$$V(s) = \min_{a \in A(s)} c(a,s) + \sum_{s' \in S} P_a(s'|s)\, V(s') \qquad (2)$$

also known as Bellman's equation. For stochastic shortest path problems like M1-M8 above, the border condition $V(s) = 0$ is also needed for goal states $s \in G$. Value iteration solves (2) by plugging an initial guess for V^* in the right-hand side of (2) and obtaining a new guess on the left-hand side. In the form of VI known as *asynchronous value iteration* [Bertsekas, 1995], this operation can be expressed as:

$$V(s) := \min_{a \in A(s)} c(a,s) + \sum_{s' \in S} P_a(s'|s)\, V(s') \qquad (3)$$

where V is a vector of size $|S|$ initialized arbitrarily (normally to 0) and where the equality in (2) is replaced by assignment. The use of expression (3) for updating a state value in V is called a state update or simply an *update*. In standard (synchronous) value iteration, all states are updated in parallel, while in asynchronous value iteration, only a selected subset of states is selected for update at a time. In both cases, it is known that if all states are updated infinitely often, the value function V converges eventually to the optimal value function. From a practical point of view, value iteration is stopped when the Bellman error or *residual* defined as the difference between left and right in (2):

$$R(s) \stackrel{\text{def}}{=} \left| V(s) - \left(\min_{a \in A(s)} c(a,s) + \sum_{s' \in S} P_a(s'|s)\, V(s') \right) \right|$$

over all states s is sufficiently small. In the discounted MDP formulation, a bound on the policy loss (the difference between the expected cost of the policy and the expected cost of the optimal policy) can be obtained as a simple expression of the discount factor and the maximum residual. In stochastic shortest path models, no similar closed-form bound is known, although such bound can be computed [Bertsekas, 1995]. Thus, one can execute value iteration until the maximum residual becomes smaller than a given ϵ, then if the bound on the policy loss is higher than desired, the same process can be repeated with a smaller ϵ (e.g., $\epsilon/2$) and so on (see [Hansen and Zilberstein, 2001] for a similar idea). For these reasons, we will take as our *basic task* below, the computation of a value function $V(s)$ with residuals no greater than a given parameter $\epsilon > 0$.

One last definition and a few known results before we proceed. We say that cost function V is *monotonic* iff

$$V(s) \leq \min_{a \in A(s)} c(a,s) + \sum_{s' \in S} P_a(s'|s)\, V(s') \qquad (4)$$

for every $s \in S$. Notice that a monotonic value function never decreases when updated, and moreover, must increase by more than ϵ when updated in a state s whose residual is greater than ϵ. As in the deterministic setting, a non-monotonic cost function can be made monotonic by simply taking the value $V(s)$ to be the max between $V(s)$ and the right-hand side of Bellman's equation. The following results are well known.

Theorem 1 *a) The optimal values $V^*(s)$ of a model M1-M8 are non-negative and finite; b) the* monotonicity *and* admissibility *of a value function are preserved through updates.*

[2] This definition of 'relevant states' is more restricted than the one in [Barto *et al.*, 1995] that includes the states reachable from s_0 by *any* optimal policy.

start with a lower bound function $V := h$
repeat
  FIND a state s in the greedy graph G_V with $R(s) > \epsilon$
  REVISE V at s
until *no such state is found*
return V

Algorithm 1: FIND-and-REVISE

3 Find-and-Revise

The FIND-and-REVISE schema is a general asynchronous VI algorithm that exploits knowledge of the initial state and an admissible heuristic for computing optimal or nearly optimal policies without having to evaluate the entire space. Let us say that a value function V is *ϵ-consistent* (inconsistent) over a state s when the residual over s is no greater (greater) than ϵ, and that V itself is ϵ-consistent when it is ϵ-consistent over all the states reachable from s_0 and the greedy policy π_V. Then FIND-and-REVISE computes an ϵ-consistent value function by simply searching for inconsistent states in the greedy graph and updating them until no such states are left; see Alg. 1.

The **greedy graph** G_V refers to the graph resulting from the execution of the greedy policy π_V starting in s_0; i.e., s_0 is the single root node in G_V,and for every non-goal state s in G_V, its children are the states that may result from executing the action $\pi(s)$ in s.

The procedures FIND and REVISE are the two parameters of the FIND-and-REVISE procedure. For the convergence, optimality, and complexity of FIND-and-REVISE, we assume that FIND searches the graph systematically, and REVISE of V at s updates V at s (and possibly at some other states), both operations taking $O(|S|)$ time.

Theorem 2 (Convergence) *For a planning model M1-M8 with an initial value function h that is admissible and monotonic,* FIND-*and*-REVISE *yields an ϵ-consistent value function in a number of loop iterations no greater than $\epsilon^{-1} \sum_{s \in S} V^*(s) - h(s)$, where each iteration has time complexity $O(|S|)$.*

Theorem 3 (Optimality) *For a planning model M1-M8 with an initial admissible and monotonic value function, the value function computed by* FIND-*and*-REVISE *approaches the optimal value function over all relevant states as ϵ goes to* 0.

4 Labeling

We consider next a particular instance of the general FIND-and-REVISE schema in which the FIND operation is carried out by a systematic *Depth-First Search* that keeps track of the states visited. In addition, we consider a *labeling scheme* on top of this search that detects, with almost no overhead, when a state is *solved*, and hence, when it can be skipped *in all future searches*. A state s is defined as *solved* when the value function V is ϵ-consistent over s and over all states reachable from s and the greedy policy π_V. Clearly, when this condition holds no further updates are needed in s or the states reachable from s. The resulting algorithm terminates when the initial state s_0 is solved and hence when an ϵ-consistent value function has been obtained.

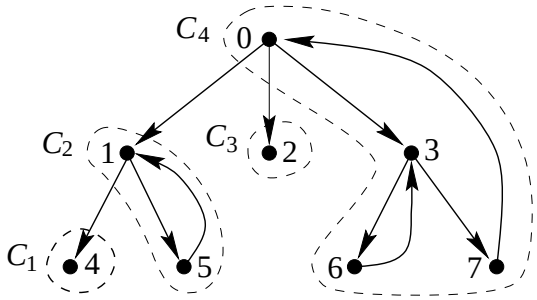

Figure 1: A graph and its strongly-connected components.

Due to the presence of cycles in the greedy graph, bottom-up algorithms common in AO* implementations cannot be used. Indeed, if s is reachable (in the greedy graph) from a descendant s' of s, then bottom-up approaches will be unable to label either state as solved. A labeling mechanism that works in the presence of cycles is presented in [Bonet and Geffner, 2003] for improving the convergence of RTDP. Basically, after each RTDP trial, an attempt is made to label the last unsolved state s in the trial by triggering a systematic search for inconsistent states from s. If one such state is found, it is updated, and a new trial is executed. Otherwise, the state s and all its unsolved descendants are labeled as solved, and a new cycle of RTDP trials and labeling checks is triggered. Here we take this idea and improve it by removing the need of an extra search for label checking. The label checking will be done as part of the FIND (DFS) search with almost no overhead, exploiting Tarjan's linear algorithm for detecting the *strongly-connected components* of a directed graph [Tarjan, 1972], and a correspondence between the strongly-connected components of the greedy graph and the minimal collections of states that can be labeled at the same time.

Consider the (directed) greedy graph G_V and the relation '$\sim$' between pairs of states s and s' that holds when $s = s'$ or when s is reachable from s' and s' is reachable from s in G_V. The strongly-connected components of G_V are the equivalence classes defined by this relation and form a partition of the set of states in G_V. For example, for the greedy graph in Fig. 1, where 2 and 4 are terminal (goal) states, the components are $C_1 = \{4\}$, $C_2 = \{1, 5\}$, $C_3 = \{2\}$, and $C_4 = \{0, 3, 6, 7\}$. Tarjan's algorithm detects the strongly-connected components of a directed graph in time $O(n + e)$ *while traversing the graph depth-first*, where n stands for the number of states ($n \leq |S|$ in G_V) and e for the number of edges.

The relationship between labeling and strongly-connected components in G_V is quite direct. Let us say first that a *component C is ϵ-consistent* when all states $s \in C$ are ϵ-consistent, and that a *component C is solved* when every state $s \in C$ is solved. Let's then define G_V^C as the graph whose nodes are the components of G_V and whose directed edges are $C \rightarrow C'$ when some state in C' is reachable from some state in C. Clearly, G_V^C is an *acyclic graph* as two components which are reachable from each other will be collapsed into the same equivalence class. In addition,

1. a state s is solved iff its component C is solved, and furthermore,

2. a component C is solved iff C is consistent and all components C', $C \rightarrow C'$, are solved.

The problem of *labeling states in the cyclic graph* G_V can thus be mapped into the problem of *labeling the components in the acyclic graph* G_V^C, which can be done in bottom up fashion.

From Fig. 1 is easy to visualize the component graph associated to the greedy graph. Thus, if 2 is the only inconsistent state, for example, we can label the components C_1 and C_2 as solved, while leaving C_3 and C_4 unsolved.

The code that simultaneously checks in depth-first fashion the consistency of the states and the possibility of labeling them is shown in Alg. 2. We call the resulting algorithm, HDP. HDP inherits its convergence and optimality properties from the FIND-and-REVISE schema and the correctness of the labeling mechanism.

We do not have space to explain HDP code in detail, yet it should be clear to those familiar with Tarjan's algorithm; in particular, the use of the state visit number, S.IDX, and the 'low-link' number, S.LOW, for detecting when a new component has been found. The flag $flag$ and the (normal) propagation of the visit numbers prevent a component from being labeled as solved when it is inconsistent or can reach an inconsistent component.

Theorem 4 (Correctness) *The value function computed by* HDP *for a planning model M1-M8, given an initial admissible and monotonic value function, is ϵ-consistent.*

5 Experimental Results

We now evaluate the performance of HDP in comparison with other recent Heuristic Search/DP algorithms such as the second code for LAO* in [Hansen and Zilberstein, 2001], that we call Improved LAO* (ILAO*), and Labeled RTDP (LRTDP), a recent improvement of RTDP that accelerates its convergence [Bonet and Geffner, 2003]. We use parallel Value Iteration as the baseline. We've implemented all these algorithms in C++ and the experiments have been run on a Sun Fire–280R with 750 MHz and 1Gb of RAM.

The domain that we use for the experiments is the racetrack from [Barto *et al.*, 1995]. The states are tuples (x, y, dx, dy) that represent the position and speed of the car in the x, y dimensions. The actions are pairs $a = (ax, ay)$ of instantaneous accelerations where $ax, ay \in \{-1, 0, 1\}$. Uncertainty in this domain comes from assuming that the road is 'slippery' and as a result, the car may fail to accelerate or desaccelerate. More precisely, an action $a = (ax, ay)$ has its intended effect with probability $1 - p$, while with probability p the action effects correspond to those of the action $a_0 = (0, 0)$. Also, when the car hits a wall, its velocity is set to zero and its position is left intact (this is different than in [Barto *et al.*, 1995] where for some reason the car is moved to the start position).

We consider the track `large-b` from [Barto *et al.*, 1995], `h-track` from [Hansen and Zilberstein, 2001],[3] and five other tracks (squares and rings of different size). Information about these instances can be found in the first three rows

[3]Taken from the source code of LAO*.

```
HDP(s : state)
begin
    while ¬ s.SOLVED do
        // perform DFS
        index := 0
        DFS(s)
        [ reset IDX to ∞ for visited states ]
        [ clean stack and visited ]
end

DFS(s : state)
begin
    // base case
    if s.SOLVED ∨ s.GOAL then
        s.SOLVED := true
        return false
    // check residual
    if s.RESIDUAL > ε then
        s.UPDATE()
        return true
    // mark state as active
    visited.PUSH(s)
    stack.PUSH(s)
    s.IDX := s.LOW := index
    index := index + 1
    // recursive call
    flag := false
    for s' ∈ s.SUCCESSORS do
        if s'.IDX = ∞ then
            flag := flag ∨ DFS(s')
            s.LOW := min{ s.LOW, s'.LOW }
        else if s' ∈ stack then
            s.LOW := min{ s.LOW, s'.IDX }
    // update if necessary
    if flag then
        s.UPDATE()
        return true
    // try to label
    else if s.IDX = s.LOW then
        while stack.TOP ≠ s do
            s' := stack.POP()
            s'.SOLVED := true
        stack.POP()
        s.SOLVED := true
    return flag
end
```

Algorithm 2: HDP.

of Table 1, including number of states, optimal cost, and percentage of states that are relevant.

As heuristic, we follow [Bonet and Geffner, 2003], and use the domain independent admissible and monotonic heuristic h_{min}, obtained by replacing the expected cost in Bellman equation by the best possible cost. The total time spent computing heuristic values is roughly the same for the different algorithms (except VI), and is shown separately in the fifth row in the table, along with its value for s_0. The experiments are carried with three heuristics: $h = h_{min}$, $h = h_{min}/2$, and $h = 0$.

The results are shown in Table 1. HDP dominates the other algorithms over all the instances for $h = h_{min}$, while LRTDP is best (with one or two exceptions) when the weaker heuristics $h_{min}/2$ and 0 are used. Thus, while HDP seems best for exploiting good heuristic information over these instances, LRTDP bootstraps more quickly (i.e., it quickly computes a good value function). We hope to understand the reasons for

algorithm	large-b	h-track	square-1	square-2	ring-1	ring-2	ring-3	ring-4
$\|S\|$	23880	53597	42071	383950	5895	33068	94369	353991
$V^*(s_0)$	18.73356	41.89504	8.07350	11.13041	11.04923	16.09833	22.04798	27.78572
% relevant	18.58	17.12	1.79	0.25	10.70	11.10	10.91	9.98
$h_{min}(s_0)$	16.0	36.0	7.0	10.0	10.0	14.0	19.0	24.0
time for h_{min}	3.157	13.422	4.509	72.853	0.555	4.558	16.745	85.766
VI(h_{min})	4.039	12.620	5.873	81.270	0.614	5.287	19.466	90.895
ILAO$*$(h_{min})	3.776	13.062	0.389	0.942	0.332	6.095	24.422	145.047
LRTDP(h_{min})	3.230	7.618	0.179	0.363	0.143	1.301	4.928	37.992
HDP(h_{min})	**1.888**	**7.547**	**0.148**	**0.275**	**0.121**	**1.101**	**4.145**	**30.511**
VI($h_{min}/2$)	**4.069**	14.275	6.147	89.248	0.636	5.704	24.385	100.130
ILAO$*$($h_{min}/2$)	6.346	26.057	2.106	68.028	0.765	9.801	32.415	174.089
LRTDP($h_{min}/2$)	4.744	**12.987**	**0.813**	**7.736**	**0.302**	**2.480**	**12.687**	152.200
HDP($h_{min}/2$)	6.402	26.663	1.255	20.952	0.331	3.536	15.982	**98.033**
VI($h=0$)	**5.785**	21.021	8.445	89.152	0.675	5.809	20.979	**101.616**
ILAO$*$($h=0$)	9.522	46.754	11.230	187.463	1.290	9.894	45.989	310.876
LRTDP($h=0$)	6.435	**15.276**	**3.286**	**45.514**	**0.431**	**3.682**	**19.424**	261.286
HDP($h=0$)	9.752	42.601	3.579	79.171	0.994	6.173	25.992	157.646

Table 1: Problem data and convergence time in seconds for the different algorithms with different heuristics. Results for $\epsilon = 10^{-3}$ and probability $p = 0.2$. Faster times are shown in bold.

these differences in the future.

6 Approximation

HDP, like FIND-and-REVISE, computes a value function V by enforcing its consistency over the states reachable from s_0 and the greedy policy π_V. The final variation we consider, that we call HDP(i), works in the same way, yet it enforces the consistency of the value function V *only* over the states that are reachable from s_0 and the greedy policy with some *minimum likelihood*.

For efficiency, we formalize this notion of likelihood, using a non-negative integer scale, where 0 refers to a normal outcome, 1 refers to a somewhat surprising outcome, 2 to a still more surprising outcome, and so on. We call these measures *plausibilities*, although it should be kept in mind, that 0 refers to the most plausible outcomes, thus 'plausibility greater than i', means 'a plausibility *measure* smaller than or equal to i.'

We obtain the transition plausibilities $\kappa_a(s'|s)$ from the corresponding transition probabilities by the following discretization:

$$\kappa_a(s'|s) \stackrel{\text{def}}{=} \lfloor -\log_2(P_a(s'|s)/\max_{s''} P_a(s''|s)) \rfloor \qquad (5)$$

with $\kappa_a(s'|s) = \infty$ when $P_a(s'|s) = 0$. Plausibilities are thus 'normalized': the most plausible next states have always plausibility 0. These transition plausibilities are then combined by the rules of the κ calculus [Spohn, 1988] which is a calculus isomorphic to the probability calculus (e.g. [Goldszmidt and Pearl, 1996]). The plausibility of a state trajectory given the initial state, is given by the sum of the transition plausibilities in the trajectory, and the plausibility of reaching a state, is given by the plausibility of the most plausible trajectory reaching the state.

The HDP(i) algorithm, for a non-negative integer i, computes a value function V by enforcing its (ϵ-)consistency over the states reachable from s_0 *with plausibility greater than or equal to* i. HDP(i) produces approximate policies fast by pruning certain paths in the search. The simplest case results from $i = 0$, as the code for HDP(0) corresponds exactly to the code for HDP, except that the possible successors of a state s in the greedy graph are replaced by the *plausible* successors.

HDP(i) computes lower bounds that tend to be quite tight over the states that can be reached with plausibility no smaller than i. At run time, however, executions may contain 'surprising' outcomes, taking the system 'out' of this envelope, into states where the quality of the value function and its corresponding policy, are poor. To deal with those situations, we define a version of HDP(i), called HDP(i, j), that *interleaves planning and execution* as follows. HDP(i, j) plans from $s = s_0$ by means of the HDP(i) algorithm, then executes this policy until a state trajectory with plausibility measure greater than or equal to j, and leading to a (non-goal) state s' is obtained. At that point, the algorithm replans from s' with HDP(i), and the same execution and replanning cycle is followed until reaching the goal. Clearly, for sufficiently large j, HDP(i, j) reduces to HDP(i), and for large i, HDP(i) reduces to HDP.

Table 2 shows the average cost for HDP(i, j) for $i = 0$ (i.e., most plausible transitions considered only), and several values for j (replanning thresholds). Each entry in the table correspond to an average over 100 independent executions. We also include the average cost for the greedy policy with respect to h_{min} as a bottom-line reference for the figures. Memory in the table refers to the number of evaluated states. As these results show, there is a smooth tradeoff between quality (average cost to the goal) and time (spent in initial planning and posterior replannings) as the parameter j vary. We also see that in this class of problems the h_{min} heuristic delivers a very good greedy policy. Thus, further research is necessary to assess the goodness of HDP(i, j) and the h_{min} heuristic.

7 Related Work

We have built on [Barto *et al.*, 1995] and [Bertsekas, 1995], and more recently on [Hansen and Zilberstein, 2001] and [Bonet and Geffner, 2003]. The crucial difference between

	h-track			square-2			ring-3			ring-4		
algorithm	time	quality	memory	time	quality	memory	time	quality	memory	time	quality	memory
HDP(h_{min})	7.547	41.894	35835	0.275	11.130	7245	4.145	22.047	43358	30.511	27.785	152750
HDP$(0, 2)$	0.853	42.950	7132	0.021	11.250	819	0.311	22.000	7145	1.758	28.500	24195
HDP$(0, 4)$	0.826	44.000	7034	0.022	11.500	650	0.327	23.300	6882	1.821	28.400	23857
HDP$(0, 16)$	0.701	46.800	6100	0.017	11.500	556	0.284	23.600	6331	1.580	30.800	21823
HDP$(0, 64)$	0.698	46.800	5899	0.014	11.610	564	0.264	25.500	6322	1.518	32.400	21823
greedy(h_{min})	N/A	47.150	356	N/A	12.450	104	N/A	25.390	192	N/A	31.600	241

Table 2: Results of HDP$(0, j)$ for $j = 2, 4, 16, 64$ and greedy policy with respect to h_{min} for $\epsilon = 10^{-3}$ and $p = 0.2$. Each value is the average over 100 executions. N/A in time for the greedy policy means "Not Applicable" since there is no planning.

FIND-and-REVISE and general asynchronous value iteration is the focus of the former on the states that are reachable from the initial state s_0 and the greedy policy. In RTDP, the FIND procedure is not systematic and is carried out by a stochastic simulation that may take time greater than $O(|S|)$ when the inconsistent states are reachable with low probability (this explains why RTDP final convergence is slow; see [Bonet and Geffner, 2003]). LAO*, on the other hand, keeps track of a subset of states, which initially contains s_0 only, and over which it incrementally maintains an optimal policy through a somewhat expensive REVISE (full DP) procedure. This is then relaxed in the second algorithm in [Hansen and Zilberstein, 2001], called Improved LAO* here. The use of an explicit envelope that is gradually expanded is present also in [Dean *et al.*, 1993] and [Tash and Russell, 1994]. Interestingly, these envelopes are expanded by including the most 'likely' reachable states not yet in the envelope. The algorithm HDP(i) exploits a similar idea but formulates it in a different form and has a crisp termination condition.

8 Summary

We have introduced and analyzed three HS/DP algorithms that exploit knowledge of the initial state and an admissible heuristic function for solving planning problems with uncertainty and feedback: FIND-and-REVISE, HDP, and HDP(i). FIND-and-REVISE makes explicit the basic idea underlying HS/DP algorithms: inconsistent states are found and updated, until no one is left. We have proved its convergence, complexity, and optimality. HDP adds a labeling mechanism based on Tarjan's SCC algorithm and is strongly competitive with current algorithms. Finally, HDP(i) and HDP(i, j) offer great time and memory savings, with no much apparent loss in quality, in problems where transitions have probabilities that differ greatly in value, by focusing the updates on the states that are more *likely* to be reached.

Acknowledgements: We thank Eric Hansen and Shlomo Zilberstein for making the code for LAO* available to us. Blai Bonet is supported by grants from NSF, ONR, AFOSR, DoD MURI program, and by a USB/CONICIT fellowship from Venezuela. Héctor Geffner is supported by grant TIC2002-04470-C03-02 from MCyT, Spain.

References

[Barto *et al.*, 1995] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[Bellman, 1957] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Bertsekas, 1995] D. Bertsekas. *Dynamic Programming and Optimal Control, (2 Vols)*. Athena Scientific, 1995.

[Bonet and Geffner, 2003] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. ICAPS-03*. To appear, 2003.

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Dean *et al.*, 1993] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In R. Fikes and W. Lehnert, editors, *Proc. 11th National Conf. on Artificial Intelligence*, pages 574–579, Washington, DC, 1993. AAAI Press / MIT Press.

[Goldszmidt and Pearl, 1996] M. Goldszmidt and J. Pearl. Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence*, 84:57–112, 1996.

[Hansen and Zilberstein, 2001] E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[Howard, 1960] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[Korf, 1997] R. Korf. Finding optimal solutions to rubik's cube using patterns databases. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conf. on Artificial Intelligence*, pages 700–705, Providence, RI, 1997. AAAI Press / MIT Press.

[Puterman, 1994] M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., 1994.

[Spohn, 1988] W. Spohn. A general non-probabilistic theory of inductive reasoning. In *Proc. 4th Conf. on Uncertainty in Artificial Intelligence*, pages 149–158, New York, NY, 1988. Elsevier Science Publishing Company, Inc.

[Tarjan, 1972] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[Tash and Russell, 1994] J. Tash and S. Russell. Control strategies for a stochastic planner. In B. Hayes-Roth and R. Korf, editors, *Proc. 12th National Conf. on Artificial Intelligence*, pages 1079–1085, Seattle, WA, 1994. AAAI Press / MIT Press.