Learning in Depth-First Search: A Unified Approach to Heuristic Search in Deterministic, Non-Deterministic, Probabilistic, and Game Tree Settings

Blai Bonet and Héctor Geffner

Abstract

Dynamic Programming provides a convenient and unified framework for studying many state models used in AI but no algorithms for handling large spaces. Heuristic-search methods, on the other hand, can handle large spaces but lack a common foundation. In this work, we combine the benefits of a general dynamic programming formulation with the power of heuristic-search techniques for developing an algorithmic framework, that we call *Learning in Depth-First Search*, that aims to be both general and effective. The basic LDFS algorithm searches for solutions by combining iterative, bounded depth-first searches, with learning in the sense of Korf's LRTA* and Barto's *et al.* RTDP. In each iteration, if there is a solution with cost not exceeding a lower bound, then the solution is found, else the process restarts with the lower bound and the value function updated. LDFS reduces to IDA* with Transposition Tables over deterministic models, but solves also non-deterministic, probabilistic, and game tree models, over which a slight variation reduces to the state-of-the-art MTD algorithm. Over Max AND/OR graphs, on the other hand, LDFS is a new algorithm which appears to be quite competitive with AO*.

Introduction

Dynamic Programming provides a convenient and unified framework for studying many state models used in AI (Bellman 1957; Bertsekas 1995) but no algorithms for handling large spaces. Heuristic-search methods, on the other hand, can handle large spaces effectively, but lack a common foundation: algorithms like IDA* aim at deterministic models (Korf 1985), AO* at non-deterministic models (Martelli & Montanari 1973), Alpha-Beta at Game Trees (Newell, Shaw, & Simon 1963), and so on (see (Nilsson 1980; Pearl 1983)), and it is not always clear what these tasks and techniques have in common, nor how they can be generalized in a principled way to other models like non-deterministic models with cycles or Markov decision processes. In this work, we combine the benefits of a general dynamic programming formulation with the effectiveness of heuristic-search techniques for developing an algorithmic framework, that we call *Learning in Depth-First Search*, that aims to be both general and effective. The basic LDFS algorithm searches for solutions by combining iterative, bounded depth-first searches, with learning in the sense (Korf 1990) and (Barto, Bradtke, & Singh 1995). In each iteration, if there is a solution with cost not exceeding a lower bound, then the solution is found and reported, else the process restarts with the lower bound and the value function updated. LDFS reduces to IDA* with Transposition Tables (Reinefeld & Marsland 1994) over deterministic models, but solves also non-deterministic, probabilistic, and game tree models, over which a slight variation reduces to the state-of-the-art MTD algorithm (Plaat *et al.* 1996).

The LDFS framework makes explicit and generalizes two key ideas underlying a family of effective search algorithms across a variety of models: *learning* and *lower bounds*. We build on recent work that combines DP updates with the use of lower bounds and knowledge of the initial state for computing *partial* optimal policies for MDPs (Barto, Bradtke, & Singh 1995; Hansen & Zilberstein 2001; Bonet & Geffner 2003). However, rather than developing another algorithm for MDPs, we make use of these notions to lay out a general framework covering a wide range of models which we hope is transparent and useful. Preliminary experiments over Max AND/OR graphs, suggest indeed that LDFS is quite competitive with AO* (Bonet & Geffner 2005).[1]

Models

All the models can be defined in terms of the following common elements:

1. a discrete and finite state space S,
2. an initial state $s_0 \in S$,
3. a non-empty set of terminal states $S_T \subseteq S$,
4. actions $A(s) \subseteq A$ applicable in each non-terminal state,
5. a function mapping non-terminal states s and actions $a \in A(s)$ into *sets* of states $F(a, s) \subseteq S$,
6. action costs $c(a, s)$ for non-terminal states s, and
7. terminal costs $c_T(s)$ for terminal states.

We assume that both $A(s)$ and $F(a, s)$ are non-empty. The various models correspond to:

[1] **Note for reviewers:** (Bonet & Geffner 2005) and this paper feature the LDFS algorithm, and both papers are submitted to AAAI-05 with ID's 294 and 296 respectively. This one introduces the algorithm and analyzes its scope and properties, while 294 evaluates performance over AND/OR graphs.

- Deterministic Models (DET): $|F(a,s)| = 1$,
- Non-Deterministic Models (NON-DET): $|F(a,s)| \geq 1$,
- Markov Decision Processes (MDPs): with probabilities $P_a(s'|s)$ for $s' \in F(a,s)$ s.t. $\sum_{s' \in F(a,s)} P_a(s'|s) = 1$.

In addition, for DET, NON-DET, and MDPs

- action costs $c(a,s)$ are all positive, and
- terminal costs $c_T(s)$ are non-negative.

When terminal costs are all zero, terminal states are called *goals*. Finally,

- Game Trees (GT): are non-deterministic models (NON-DET) with zero action costs, arbitrary terminal costs, and a *tree-structure*.

A model has a *tree-structure* when two different paths cannot lead to the same state. A path $s_0, a_0, s_1, a_1, \ldots, a_{n-1}, s_n$ is a sequence of states and actions starting in the initial state s_0, such that each action a_i is applicable in s_i, $a_i \in A(s_i)$, and each state s_{i+1} is a possible successor of s_i given action a_i, $s_{i+1} \in F(a_i, s_i)$. We also define the *acyclic models* as those which do not accommodate *cyclic paths*, i.e., paths $s_0, a_0, s_1, a_1, \ldots, a_{n-1}, s_n$ with $s_i = s_j$ for $i \neq j$. We write aNON-DET and aMDPs to refer to the subclass of acyclic NON-DET and MDPs models. For example, the type of problems in the scope of the AO* algorithm, are defined by those in aNON-DET.

Solutions

The solutions to the various models can be expressed in terms of the so-called Bellman equation that characterizes the optimal cost function (Bellman 1957; Bertsekas 1995):

$$V(s) \stackrel{\text{def}}{=} \begin{cases} c_T(s) & \text{if } s \text{ terminal} \\ \min_{a \in A(s)} Q_V(a,s) & \text{otherwise} \end{cases} \quad (1)$$

where the $Q_V(a,s)$ values express the cost-to-go and are *short-hand* for:

$c(a,s) + V(s')$, $s' \in F(a,s)$ for DET,
$c(a,s) + \max_{s' \in F(a,s)} V(s')$ for NON-DET-Max,
$c(a,s) + \sum_{s' \in F(a,s)} V(s')$ for NON-DET-Add,
$c(a,s) + \sum_{s' \in F(a,s)} P_a(s'|s) V(s')$ for MDPs,
$\max_{s' \in F(a,s)} V(s')$ for Game Trees.

We make a distinction between worst-case (Max) and additive (Add) non-deterministic models, as both are considered in AI, and yet, they have slightly different properties. We will refer to the models (NON-DET-Max and GT) whose Q-values are defined with Max as Max models, and to the rest of the models, defined with Sums, as Additive models.

Under some conditions, there is a unique value function $V^*(s)$, the optimal cost function, that solves the Bellman equation, and the optimal solutions to all the models can be expressed in terms of the policies π that are *greedy* with respect to $V^*(s)$. A policy π is a function mapping states $s \in S$ into actions $a \in A(s)$, and a policy π_V is greedy with respect to a value function $V(s)$, or simply greedy in V, iff π_V is the best policy assuming that the cost-to-go is given by $V(s)$; i.e.

$$\pi_V(s) = \operatorname*{argmin}_{a \in A(s)} Q_V(a,s) \,. \quad (2)$$

Often, however, these conditions are not met, and the set of $|S|$ Bellman equations have no solution. These happens for example in the presence of *dead-ends*. Also, for Max models, as we will see, it is not the case that optimal solutions must be greedy with respect to V^*. For these reasons, we characterize optimal solutions in a slightly different way, taking into account the information about the initial state s_0 of the system which is assumed to be available and known.

We deal then with *partial policies* that map *some states* into actions only. We say that a partial policy π is *closed* (relative to s_0) if π prescribes the action to be done in all the (non-terminal) *states reachable from* s_0 *and* π; this set $S' \subseteq S$ is defined inductively as comprising s_0 and all the states $s' \in F(\pi(s), s)$ for $s \in S'$. In particular, closed policies for deterministic models correspond to action sequences, for game trees, to actual trees, and so on.

Any closed policy π relative to a state s has a cost $V^\pi(s)$ that expresses the cost of solving the problem starting from s. The costs $V^\pi(s)$ are given by the solution of (1) but with the operator $\min_{a \in A(s)}$ removed and the action a replaced by $\pi(s)$. These costs are thus well-defined when the resulting equations have a solution *over the subset of states reachable from* s_0 *and* π. Moreover, for all models above, except MDPs, it can be shown that (closed) policies π have a well-defined finite cost $V^\pi(s_0)$ when they are *acyclic*, and for MDPs, when they are *proper*. Otherwise $V^\pi(s_0) = \infty$. A closed policy π is *cyclic* if it gives rise to cyclic paths $s_0, a_0, s_1, a_1, \ldots, a_{n-1}, s_n$ where $a_i = \pi(s_i)$, and it is *proper* if a terminal state is reachable from every state s reachable from s_0 and π (Bertsekas 1995).

For all models except for MDPs, since solutions π are acyclic, the costs $V^\pi(s_0)$ can be defined also recursively, starting with the terminal states s' for which $V^\pi(s') = c_T(s')$, and up to the non-terminal states s reachable from s_0 and π for which $V^\pi(s) = Q_{V^\pi}(\pi(s), s)$. In all cases, we are interested in computing a solution π that minimizes $V^\pi(s_0)$. The resulting value is the optimal problem cost $V^*(s_0)$.

A General Algorithm

We assume throughout the paper that we have an initial value (or heuristic) function V that for all non-terminal states is a *lower bound*, $V(s) \leq V^*(s)$, and *monotonic*, $V(s) \leq \min_{a \in A(s)} Q_V(a,s)$. Also for simplicity, we assume $V(s) = c_T(s)$ for all *terminal* states. We summarize these conditions by simply saying that V is **admissible.** This value function is then modified by *learning* in the sense of (Korf 1990) and (Barto, Bradtke, & Singh 1995), where the values of selected states are made consistent with the values of successor states; an operation that takes the form of a Bellman *update*:

$$V(s) := \min_{a \in A(s)} Q_V(a,s) \,. \quad (3)$$

If the initial value function is admissible, it remains so after one or more updates. Methods like value iteration perform the iteration $V(s) := \min_{a \in A(s)} Q_V(a,s)$ until the difference between right and left does not exceed some $\epsilon \geq 0$. The difference $\min_{a \in A(s)} Q_V(a,s) - V(s)$, which is non-negative for monotonic value functions, is called the *residual* of V over s, denoted $Res_V(s)$. Clearly, a value function

V is a solution to Bellman equation and is thus equal to V^* if it has zero residuals over all states. Given a fixed initial state s_0, however, it is not actually necessary to eliminate all residuals for ensuring optimality:

Proposition 1 *Let V be an admissible value function and let π be a policy greedy in V. Then π minimizes $V^\pi(s_0)$ and hence is optimal if $Res_V(s) = 0$ over all the states s reachable from s_0 and π.*

This suggests a simple and general schema for solving all the models, that avoids the strong assumptions required by standard DP methods and yields partial optimal policies.

Since there may be many policies π greedy in V, we will assume an ordering on actions and let π_V refer to the particular greedy policy obtained by selecting in each state the action greedy in V that is minimal with respect to this ordering. Then, in order to obtain a policy and a value function satisfying Proposition 1, it is sufficient to search for a state s reachable from s_0 and π_V with residual $Res_V(s) > \epsilon$ and update the state, keeping this iteration until there are no such states left. If $\epsilon = 0$ and the initial value value function V is admissible, then the resulting (closed) greedy policy π_V is optimal. This FIND-and-REVISE schema, shown in Fig. 1 and introduced in (Bonet & Geffner 2003) for solving MDPs, can be used to solve all the models, without the strong assumptions made by DP algorithms and without having to compute complete policies:[2]

Proposition 2 *Starting with an admissible value function V, the* FIND-*and*-REVISE *schema for $\epsilon = 0$, solves all the models (DET, NON-DET, GT, MDPs) provided they have solution.*

For the non-probabilistic models with *integer* action and terminal costs, the number of iterations of FIND-and-REVISE with $\epsilon = 0$ is bounded by $\sum_{s \in S}[\min(V^*(s), MaxV) - V(s)]$, where $MaxV$ stands for any upper bound on the optimal costs $V^*(s)$ of the states with finite cost. This is because updates increase the value function by at least one in some states, decrease it in none, and preserve its admissibility. In addition, states with values above $MaxV$ are not reachable from s_0 and the greedy policy. Since the Find procedure can implemented by a simple DFS procedure that keeps track of visited states in time $O(|S|)$, it follows that the time complexity of FIND-and-REVISE over those models can be bounded by the same expression times $O(|S|)$. For MDPs, the convergence of FIND-and-REVISE with $\epsilon = 0$ is asymptotic and cannot be bounded in this way. However, for any $\epsilon > 0$, the convergence is bounded by the same expression divided by ϵ.

Learning DFS

We have seen that all the models admit a common formulation and a common algorithm. This algorithm, while not practical, will be useful for understanding and proving the correctness of other, more effective approaches. We will say that an iterative algorithm is instance of FIND-and-REVISE$[\epsilon]$, if each iteration of the algorithm terminates, either identifying and updating a state reachable from s_0 and π_V with residual $Res_V(s) > \epsilon$, or proving that no such state exists, and hence, that the model is solved. Such algorithms will inherit the correctness of FIND-and-REVISE, but by performing more updates per iteration will converge faster.

[2] It is assumed that the initial value function is represented intensionally and that the updated values are stored in a hash table.

starting with an admissible V
repeat
  FIND s reachable from s_0 and π_V with $Res_V(s) > \epsilon$
  Update $V(s)$ to $\min_{a \in A(s)} Q_V(a, s)$
until *no such state is found*
return V

Algorithm 1: The FIND-and-REVISE schema

We focus first on the models whose *solutions* are necessarily acyclic, excluding thus MDPs but not acyclic MDPs (aMDPs). We are not excluding *models* with cycles though; only models whose solutions may be cyclic. Hence the requirements are weaker than those of algorithms like AO*.

We will say that a state s is *consistent* relative to a value function V if the residual of V over s is no greater than ϵ. Unless mentioned otherwise, we take ϵ to be 0. The first practical instance of FIND-and-REVISE that we consider, LDFS, implements the Find operation as a DFS that considers all the greedy actions in a state, backtracks on inconsistent states, and updates not only the inconsistent states that are found, but upon backtracking, their ancestors too. The code for LDFS is shown in Fig. 2. The Depth-First Search is achieved by means of two loops: one over the (greedy) actions $a \in A(s)$ in s, the other, nested, over the possible successors $s' \in F(a, s)$. The tip nodes in this search are the the inconsistent states s, (where for all the actions $Q_V(a, s) > V(s)$), the terminal states, and the states that are labeled as solved. A state s is labeled as solved when the search beneath s did not find any inconsistent state. This is captured by the boolean $flag$. If s is consistent, and $flag$ is true after searching beneath the successors $s' \in F(a, s)$ of a greedy action a, then s is labeled as solved, $\pi(s)$ is set to a, and no more actions are tried at s. Otherwise, the next greedy action is tried, and if no one is left, s is updated. Similarly, if the search beneath a successor state $s' \in F(a, s)$ reports an inconsistency or a no longer satisfies the greedy condition $Q_V(a, s) \leq V(s)$, the rest of the successor states $s'' \in F(a, s)$ are skipped.

LDFS is called iteratively over s_0 from a driver routine that terminates when s_0 is solved, returning a value function V and a greedy policy π that satisfies Proposition 1, and hence is optimal. We show this by proving that LDFS is an instance of FIND-and-REVISE. First, since no model other than MDPs can accommodate a cycle of *consistent* states, we get that:

Proposition 3 *For DET, NON-DET, GT, and aMDPs, a call to* LDFS *cannot enter into a loop and thus terminates.*

Then, provided with the same ordering on actions as FIND-and-REVISE, it is simple to show that the first state s that is updated by LDFS is inconsistent and reachable from s_0 and π_V, and if there is not such state, LDFS terminates with π_V.

Proposition 4 *Provided an initial admissible value function,* LDFS *is an instance of* FIND-*and*-REVISE$[\epsilon = 0]$, *and*

```
LDFS-DRIVER(s_0)
begin
    repeat solved := LDFS(s_0) until solved
    return (V, π)
end
LDFS(s)
begin
    if s is SOLVED or terminal then
        if s is terminal then V(s) := c_T(s)
        Mark s as solved return true
    flag := false
    foreach a ∈ A(s) do
        if Q_V(a, s) > V(s) then continue
        flag := true
        foreach s' ∈ F(a, s) do
            flag := LDFS(s') & [Q_V(a, s) ≤ V(s)]
            if ¬flag then break
        if flag then break
    if flag then
        π(s) := a
        Mark s as SOLVED
    else
        V(s) := min_{a∈A(s)} Q_V(a, s)
    return flag
end
```

Algorithm 2: Learning DFS Algorithm (LDFS)

hence, it terminates with a closed partial policy π that is optimal for DET, NON-DET, GT, and aMDPs.

In addition, for the models that are *additive*, it can be shown that all the updates performed by LDFS are *effective*, in the sense that they are all done on states that are inconsistent, and which as a result, strictly increase their values:

Proposition 5 *Provided an initial admissible value function, all the updates in* LDFS *over the* additive models *DET, NON-DET-Add, and aMDPs, strictly increase the value function.*

An immediate consequence of this is that for DET and NON-DET-Add models with integer action and terminal costs, the bound on the number of iterations can be reduced to $V^*(s_0) - V(s_0)$, which corresponds to the maximum number of iterations in IDA* under the same conditions. Actually, provided that LDFS and IDA* (with transposition tables (Reinefeld & Marsland 1994)) consider the actions in the same order, it can be shown that they will both traverse the same paths, and maintain the same value (heuristic) function in memory:

Proposition 6 *Provided an admissible (and monotonic) value function V, and that actions are applied in the same order in every state,* LDFS *is equivalent to* IDA* *with Transposition Tables over the class of deterministic models (DET).*

Actually, for Additive Models, the workings of LDFS can be characterized as follows:

Proposition 7 *Over the Additive Models DET, Non-DET-Add, and aMDPs,* LDFS *tests whether there is solution π with cost $V^\pi(s_0) \leq V(s_0)$ for an initial admissible value function V. If a solution exists, one such solution is found and reported; else $V(s_0)$ is increased, and the test is run again, til a solution is found. Since V remains a lower bound, the solution found is optimal.*

This is indeed the idea underlying IDA* and the Memory-enhanced Test Driver algorithm or MTD($-\infty$) for Game Trees (Plaat *et al.* 1996). Interestingly, however, while LDFS solves Game Trees, it does not exhibit this pattern: the reason is that over Max models, like GT and NON-DET-Max, updates in LDFS are not always effective. We discuss this next.

Local and Global Optimality

An optimal solution is one that minimizes $V^\pi(s_0)$. If π also minimizes $V^\pi(s)$ for all the states s reachable from s_0 and π, we say that π is *globally optimal*. A characteristic of additive models is that the first condition implies the second. In Max models, however, this is not true. This distinction arises because while all arguments count in a sum, not all arguments count in a maximization. Game Tree algorithms make use of this difference for computing optimal policies that are not necessarily globally optimal. On the other hand, LDFS and FIND-and-REVISE, compute only globally optimal policies. This is because they keep updating V until *all inconsistencies* over the states reachable from s_0 and the greedy policy π_V are eliminated. Some of these inconsistencies, however, are harmless in Max models. Interestingly, the AO* algorithm has the same limitation and computes globally optimal solutions even in models like aNON-DET-Max where this is not required.

Bounded LDFS

The properties that LDFS exhibits over additive models, in particular the notion of effective updates, can be extended to Max models by adding an extra argument to LDFS: a $Bound$ parameter. For simplicity, we will restrict our attention to *Max models where no state can be reached through paths of different costs.* This includes of course Game Trees and NON-DET-Max Tree models. For the general case, see (Bonet & Geffner 2005).

The key observation is that while an increase in $V(s')$ for some $s' \in F(a, s)$ does not necessarily translate into an increase of $Q_V(a, s)$ in Max models, an increase of $V(s')$ above a certain bound will. Namely, $Q_V(a, s) \leq Bound$ iff $V(s') \leq Bound'$ for $Bound'$ equal to $Bound$ in Game Trees, to $Bound - c(a, s)$ in DET, to $Bound - c(a, s) - \sum_{s'' \in F(a,s) \setminus \{s'\}} V(s'')$ in NON-DET-Add, etc. The procedure Bounded LDFS shown in Fig. 3 takes advantage of this, replacing the tests $Q_V(a, s) \leq V(s)$ in LDFS with $Q_V(a, s) \leq Bound$ where $Bound$ is the extra parameter, which is initialized to $V(s_0)$ in the driver routine, and passes as the $Bound'$ above in the recursive calls. For Additive models, this change has no effect because the following invariant holds before the loop over the actions:

Proposition 8 *For the additive models, in all* LDFS-BOUND *calls, the invariant $V(s) = Bound$ holds before the A-loop.*

```
LDFS-BOUND-DRIVER(s_0)
begin
    repeat solved := LDFS-BOUND(s_0, V(s_0)) until solved
    return (V, π)
end
LDFS-BOUND(s,Bound)
begin
    ...
    foreach a ∈ A(s) do
        if Q_V(a,s) > Bound then continue
        flag := true
        foreach s' ∈ F(a,s) do
            Bound' := see text
            flag := LDFS-BOUND(s', Bound') &
                                    [Q_V(a,s) ≤ Bound]
            if ¬flag then break
        if flag then break
    ...
end
```

Algorithm 3: LDFS-BOUND: fragment that differs from LDFS

As a result, directly from the code, it can be established that

Proposition 9 LDFS *and* LDFS-BOUND *are equivalent over the Additive models.*

For Max models, however, a weaker invariant holds:

Proposition 10 *For Max models, in all* LDFS-BOUND *calls, the invariant* $V(s) \leq Bound$ *holds before the A-loop.*

The difference between LDFS and LDFS-BOUND over Max models is that while the former regards $\min_{a \in A(s)} Q_V(a,s) > V(s)$ as an inconsistency that needs to be removed, LDFS-BOUND removes this inconsistency only when $\min_{a \in A(s)} Q_V(a,s) > Bound$ holds, which is weaker due to the invariant $V(s) \leq Bound$. As a result, LDFS-BOUND, unlike LDFS, is not an instance of FIND-and-REVISE over Max models, and thus termination and correctness need to be proved in a different way. First, a LDFS-BOUND call terminates over Game Trees as there cannot be cyclic paths, and over NON-DET-Max models, as the invariant $0 \leq V(s) \leq Bound$ holds and the Bound decreases monotonically along any path:

Proposition 11 *A call to* LDFS-BOUND *over Max models cannot enter into a loop and thus it always terminates.*

Then with arguments similar to the ones used for FIND-and-REVISE, one can bound the number of iterations, so that upon termination LDFS-BOUND returns a policy π and a value function V such that $Q_V(\pi(s), s) \leq Bound$ holds in all the invocations LDFS-BOUND$(s, Bound)$ over the states s reachable from s_0 and π. Yet since π must be acyclic, inductively once can prove that $V^\pi(s_0) \leq V(s_0)$ and hence:

Proposition 12 *Provided an initial admissible value function,* LDFS-BOUND *terminates with a closed policy* π *that is optimal over the Max models GT and NON-DET-Max.*

In addition, like the updates in LDFS over the additive models, updates in LDFS-BOUND are *effective* over Max models:

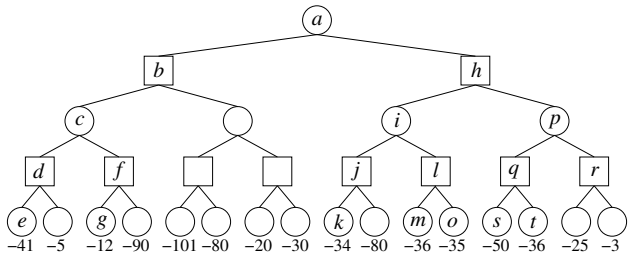

Figure 1: Game Tree with MIN player playing first

Proposition 13 *Provided an initial admissible value function, all updates in* LDFS-BOUND *over Max models, strictly increase the value function.*

As result LDFS-BOUND over Max models can be also described as an algorithm that finds a solution π with cost $V^\pi(s_0) \leq V(s_0)$ if there is one such solution, and else updates $V(s_0)$ and other values, and tries again. It is thus not entirely surprising that over Game Trees, LDFS-BOUND reduces to the MTD$(-\infty)$ algorithm (Plaat *et al.* 1996); i.e., given the same ordering on actions and successor states, both algorithms traverse the same paths, and maintain the same value function:[3]

Proposition 14 *With the initial value function* $V(s) = -\infty$, *for all* s, LDFS-BOUND *is equivalent to* MTD$(-\infty)$.

The proof involves code transformations and invariants. For example, the MT algorithm works with both lower and upper bounds, yet it can be shown that when the initial value function is $-\infty$, upper bounds (from the perspective of the MIN player) play the same role as labels in LDFS-BOUND. For the equivalence, it is necessary to assume that both the top-nodes and the terminal nodes correspond to MIN-player moves. While MT is symmetric in this sense, LDFS-BOUND is not. Finally, for keeping the presentation simple, we have assumed that $V(s) = c_T(s)$ for terminal states, yet this condition is not required.

The reader is encouraged to try the LDFS-BOUND algorithm on the Game Tree shown in Fig. 1, modified from (Plaat *et al.* 1996) so that it retains the same solution but with the MIN player moving first (thus payoffs have been made negative). The top node a is the initial state where there are two applicable actions, 'left' and 'right'. The action 'right' in a has non-deterministic effects i and p, and so on (note that MAX nodes like b and h do not correspond to states but to 'And' nodes). With an initial value function $V = -\infty$, LDFS-BOUND, like MTD$(-\infty)$, traverses the subtree formed by $a, b, c, d, e, f, g, h, i, j, k, l, m$ in the first iteration and sets $V(a) = -41$. This value is used as the bound of the second iteration, which also returns unsuccessfully with a new bound $V(a) = -36$, updated in the third iteration to $V(a) = -35$. The fourth iteration ends successfully proving this value optimal, with a solution π that chooses 'right' at a and i, and 'left' at p.

[3] As in MTD, $-\infty$ refers to a large negative number. Thus, $-\infty < -\infty + k$ for positive k.

MDPs

In order to handle MDPs, two features are needed: an $\epsilon > 0$ bound on the size of the residuals allowed for avoiding asymptotic convergence, and a bookkeeping mechanism for avoiding loops and recognizing states that are solved. To illustrate the subtleties involved, let us assume that there is a single action $\pi(s)$ applicable in each state. By performing a single depth-first pass over the descendants of s, keeping track of visited states for not visiting them twice, we want to know when a state s can be labeled as solved. The subtlety arises due to the presence of cycles. In particular, it is no longer correct to label a state s as solved when the variable $flag$ indicates that all descendants of s are consistent (i.e., have residuals no greater than ϵ); as there may be ancestors of s that are also reachable from s with unexplored descendents. Yet, even in such case, there must be states s in the DFS tree spanned by LDFS-MDP (recall that no states are visited twice) such that all the states that are reachable from s and π are beneath s, and for those states, it is *correct* to label them as solved when $flag$ is true. Moreover, the labeling scheme becomes *complete* if at that point not only s is labeled but also its descendants. The question of course is how to recognize such 'top' elements in the state graph during the depth-first search. The answer is given by Tarjan's strongly connected component algorithm (Tarjan 1972) that keeps track of two indices $s.low$ and $s.idx$ for each state s encountered. The top elements s are precisely those for which $s.low = s.idx$.

The resulting algorithm for MDPs, LDFS-MDP, is LDFS+ ϵ-RESIDUALS + TARJAN. We lack the space to show it here but can prove that:

Proposition 15 LDFS-MDP *is an instance of* FIND-*and*-REVISE$[\epsilon]$ *and hence for a sufficiently small* ϵ*, solves MDPs provided they have a solution with finite (expected) cost.*

LDFS-MDP is similar to the HDP algorithm (Bonet & Geffner 2003) that introduced Tarjan's algorithm for labeling states in MDPs, yet by trying all greedy actions in every state LDFS-MDP ensures that all updates remain effective.

Discussion

We have developed a computational framework, LDFS, which makes explicit and generalizes two key ideas underlying a family of effective search algorithms across a variety of models: *learning* and *lower bounds*.[4] The same LDFS algorithm handles deterministic and non-deterministic models, with or without cycles, and a simple variation handles MDPs where solutions can be cyclic. The framework uncovers also a key distinction between Additive and Max models that has apparently gone unnoticed: optimal solutions to Additive models are globally optimal, but optimal solutions to Max models need not be. Still algorithms like AO* compute globally optimal solutions, which is adequate for Additive AND/OR graphs but is not required for Max AND/OR graphs. We have actually implemented the LDFS and Bounded LDFS algorithms and compared them with AO* and Value Iteration over Max AND/OR Graphs. The results, reported in (Bonet & Geffner 2005), show that over a wide variety of instances and heuristic functions, LDFS and Bounded LDFS are almost never worse than either AO* or Value Iteration, and instead are often one or more orders of magnitude faster. We do not expect similar practical gains over DET and GTs where LDFS and Bounded LDFS reduce to well known algorithms. The value of the proposed framework, however, goes beyond the particular algorithms obtained for the various models. For example, since all LDFS algorithms can be understood as 'extensions' of IDA* some of their limitations can be understood in terms of the limitations of IDA* itself. For example, it is well known that IDA* doesn't do well when action costs are real numbers. In MDPs, this problem arises even when action costs are integers because of the probabilities. We are thus currently exploring variations on the basic LDFS-MDP algorithm that exploit this parallelism.

[4] For other works emphasizing the common ideas between single-agent and two-player search (DET and GTs in our terms); see (Marsland & Reinefeld 1993) and (Schaeffer, Plaat, & Junghanns 2001).

References

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

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

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Vols 1 and 2*. Athena Scientific.

Bonet, B., and Geffner, H. 2003. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. IJCAI-03*, 1233–1238.

Bonet, B., and Geffner, H. 2005. An algorithm better than AO*?

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

Korf, R. 1985. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Marsland, T. A., and Reinefeld, A. 1993. Heuristic search in one and two player games. Technical Report TR 93-02, University of Alberta.

Martelli, A., and Montanari, U. 1973. Additive AND/OR graphs. In *Proc. IJCAI-73*, 1–11.

Newell, A.; Shaw, J. C.; and Simon, H. 1963. Chess-playing programs and the problem of complexity. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought*. McGraw Hill. 109–133.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Tioga.

Pearl, J. 1983. *Heuristics*. Addison Wesley.

Plaat, A.; Schaeffer, J.; Pijls, W.; and Bruin, A. 1996. Best-first fixed-depth minimax algorithms. *Artificial Intelligence* 87(1-2):255–293.

Reinefeld, A., and Marsland, T. 1994. Enhanced iterative-deepening search. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 16(7):701–710.

Schaeffer, J.; Plaat, A.; and Junghanns, A. 2001. Unifying single-agent and two-player search. *Inf. Sci.* 135(3-4):151–175.

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