AN ALGORITHM BETTER THAN AO*?

Blai Bonet bonet@ldc.usb.ve Universidad Simón Bolívar

Héctor Geffner hector.geffner@upf.edu ICREA / Universitat Pompeu Fabra

Introduction

- Dynamic Programming provides a convenient and unified framework for studying many state models used in AI, but no algorithms for handling large state spaces
- Heuristic search methods can handle large state spaces but have no common foundation; e.g. IDA*, AO*, αβ-pruning, ...
- In this work, we combine DP and heuristic search into an algorithmic framework, called **Learning in Depth-First Search**, that is both general and effective
- LDFS combines iterative depth-first searches with learning in the sense of Korf's Learning RTA* (LRTA*)
- On DETERMINISTIC problems, LDFS reduces to IDA* with Transposition Tables, on GAME TREE problems to MTD (MTD = αβ-pruning with null windows + memory; Plaat et al. 1996); it also applies to other models like AND/OR, MDPs, games with chance nodes, etc
- Here, we compare with AO* (and others) over AND/OR problems

Models

All models can be understood in terms of:

1. a discrete and finite states 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 that maps states and actions 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.

Accomodates various classes of AI models that correspond to:

- DETERMINISTIC: $|F(a,s)| = 1$,
- NON-DETERMINISTIC: $|F(a,s)| \geq 1$,
- MDP: with probabilities $P_a(s'|s)$ for $s' \in F(s,a)$
- ...

Solutions and Optimal Solutions

Can be expressed in terms of value function V satisfying Bellman equations:

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

where $Q_V(a,s)$ stands for the cost-to-go value defined as:

$c(a,s) + V(s')$, $s' \in F(a,s)$ for DETERMINISTIC,
$c(a,s) + \max_{s' \in F(a,s)} V(s')$ for Max AND/OR (NON-DET-MAX),
$c(a,s) + \sum_{s' \in F(a,s)} V(s')$ for Add AND/OR (NON-DET-ADD),
$c(a,s) + \sum_{s' \in F(a,s)} P_a(s'|s) V(s')$ for MDP,
$\max_{s' \in F(a,s)} V(s')$ for GAME TREE.

A policy π is a partial function from states to actions **closed** around the initial state s_0, and it is **optimal** if it minimizes $V^\pi(s_0)$: the (expected) cost from s_0. Optimal π denoted as π^*.

LDFS

```
LDFS-DRIVER(s0)
begin
    repeat solved := LDFS(s0) 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
```

Properties of LDFS and Bounded LDFS

1. LDFS and Bounded LDFS compute π^* for **all models** provided $V \leq V^*$
2. For DETERMINISTIC models and *monotone* V,

$$\text{LDFS} = \text{IDA* + Transposition Tables}$$

3. For GAME TREE models and $V = -\infty$,

$$\text{Bounded LDFS} = \text{MTD}(-\infty)$$

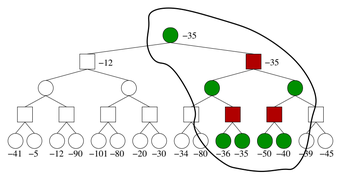
4. For ADDITIVE models,

$$\text{LDFS} = \text{Bounded LDFS}$$

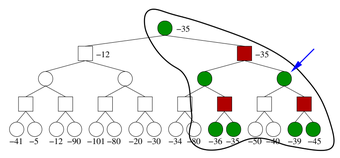
5. For MAX models,

$$\text{LDFS} \neq \text{Bounded LDFS}$$

6. LDFS (like VI, AO*, min-max LRTA*, etc) computes optimal solutions graphs where each node is an optimal solution subgraph (global optimality); over MAX models, however, this isn't needed, and is wasteful.



Global Optimality



Local Optimality

Bounded LDFS

```
B-LDFS-DRIVER(s0)
begin
    repeat B-LDFS(s0, V(s0)) until V(s0) ≥ U(s0)
    return (V, π)
end

B-LDFS(s, bound)
begin
    if s is terminal or V(s) ≥ bound then
        if s is terminal then V(s) := U(s) := c_T(s)
        return true
    flag := false
    foreach a ∈ A(s) do
        if Q_V(a,s) > bound then continue
        flag := true
        foreach s' ∈ F(a,s) do
            nb := bound − c(a,s)
            flag := B-LDFS(s', nb) & [Q_V(a,s) ≤ bound]
            if ¬flag then break
        if flag then break
    if flag then
        π(s) := a
        U(s) := bound
    else
        V(s) := min_{a∈A(s)} Q_V(a,s)
    return flag
end
```

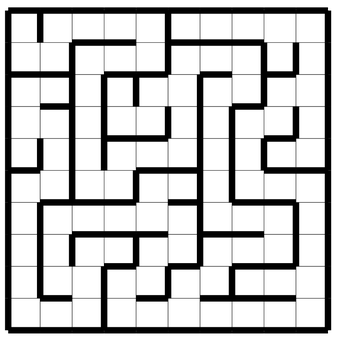
Empirical Evaluation: AO*, VI, LRTA*, LDFS, and Bounded LDFS

- **Coins:** There are N coins including a counterfeit coin that is either lighter or heavier than the others, and a 2-pan balance. A strategy is needed for identifying the counterfeit coin, and whether it is heavier or lighter than the others. We experiment with $N = 10, 20, \ldots, 60$. We use the representation proposed in Fuxi, Ming and Yanxiang (2003).

- **Diagnosis:** There are N binary tests for finding out the true state of a system among M different states as defined in Pattipati and Alexandridis (1990). An instance is described by a binary matrix T of size $M \times N$ such that $T_{ij} = 1$ iff test j is positive when the state is i. The goal is to obtain a strategy for identifying the true state. The search space consists of all non-empty subsets of states, and the actions are the tests. We performed two classes of experiments: a first class with N fixed to 10 and M varying in $\{10, 20, \ldots, 60\}$, and a second class with M fixed to 60 and N varying in $\{10, 12, \ldots, 28\}$.

- **Rules:** We consider the derivation of atoms in acyclic rule systems with N atoms, and at most R rules per atom, and M atoms per rule body. In the experiments $R = M = 50$ and N is in $\{5000, 10000, \ldots, 20000\}$.

- **Mts:** A predator must catch a prey that moves non-deterministically to a non-blocked adjacent cell in a given random maze of size $N \times N$. At each time, the predator and prey move one position. Initially, the predator is in the upper left position and the prey in the bottom right position. The task is to obtain an optimal strategy for catching the prey. In Ishida and Korf (1991), a similar problem is considered in a real-time setting where the predator moves 'faster' than the prey, and no optimality requirements are made. Solvable instances are generated by ensuring that the undirected graph underlying the maze is connected and loop free. We consider $N = 15, 20, \ldots, 40$,



problem	$\|S\|$	V^*	N_{VI}	$\|A\|$	$\|F\|$	$\|\pi^*\|$
coins-10	43	3	2	172	3	9
coins-60	1,018	5	2	315K	3	12
mts-5	625	17	14	4	4	156
mts-35	1,5M	573	322	4	4	220K
mts-40	2,5M	684	–	4	4	304K
diag-60-10	29,738	6	8	10	2	119
diag-60-28	> 15M	6	–	28	2	119
rules-5000	5,000	156	158	50	50	4,917
rules-20000	20,000	592	594	50	50	19,889