Learning in Depth-First Search

Blai Bonet

Universidad Simón Boívar

bonet@ldc.usb.ve

Héctor Geffner

ICREA / Universitat Pompeu Fabra

hector.geffner@upf.edu

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*, $\alpha\beta$-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 combine 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 = $\alpha\beta$-pruning with null windows + memory; Plaat et al. 1996); it also applies to other models like AND/OR, MDPs, games with chance nodes, etc

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*, $\alpha\beta$-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 combine 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 = $\alpha\beta$-pruning with null windows + memory; Plaat et al. 1996); it also applies to other models like AND/OR, MDPs, games with chance nodes, etc

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*, $\alpha\beta$-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 combine 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 = $\alpha\beta$-pruning with null windows + memory; Plaat et al. 1996); it also applies to other models like AND/OR, MDPs, games with chance nodes, etc

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*, $\alpha\beta$-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 combine 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 = $\alpha\beta$-pruning with null windows + memory; Plaat et al. 1996); it also applies to other models like AND/OR, MDPs, games with chance nodes, etc

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*, $\alpha\beta$-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 combine 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 = $\alpha\beta$-pruning with null windows + memory; Plaat et al. 1996); it also applies to other models like AND/OR, MDPs, games with chance nodes, etc

Deterministic Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into states $f(a, s) \in S$, and
- action costs $c(a, s)$ for non-terminal states s.

Deterministic Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into states $f(a, s) \in S$, and
- action costs $c(a, s)$ for non-terminal states s.

Solutions: sequences $a_0, \ldots, a_n$ of actions that "transform" s_0 into a terminal state

Deterministic Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into states $f(a,s) \in S$, and
- action costs $c(a,s)$ for non-terminal states s.

Solutions: sequences $a_0, \ldots, a_n$ of actions that "transform" s_0 into a terminal state

Optimal: if minimizes $\sum_{k=0}^{n} c(a_k, s_k)$

Deterministic Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into states $f(a,s) \in S$, and
- action costs $c(a,s)$ for non-terminal states s.

Solutions: sequences $a_0, \ldots, a_n$ of actions that "transform" s_0 into a terminal state

Optimal: if minimizes $\sum_{k=0}^{n} c(a_k, s_k)$

Algorithms: search algorithms for OR graphs; e.g. BFS, A*, IDA*, ...

Deterministic Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into states $f(a,s) \in S$, and
- action costs $c(a,s)$ for non-terminal states s.

Solutions: sequences $a_0, \ldots, a_n$ of actions that "transform" s_0 into a terminal state

Optimal: if minimizes $\sum_{k=0}^{n} c(a_k, s_k)$

Algorithms: search algorithms for OR graphs; e.g. BFS, A*, IDA*, ...

Preferred: IDA* is a linear space algorithm, memory can be exploited with a Transposition Table

Game Trees

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- moves $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and moves into sets of states $F(a, s) \subseteq S$, and
- terminal costs $c_T(s)$ for terminal states.

Game Trees

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- moves $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and moves into sets of states $F(a, s) \subseteq S$, and
- terminal costs $c_T(s)$ for terminal states.

Solutions: no longer a sequence of actions but a partial function π that maps states into actions such that it is closed with respect to s_0

Game Trees

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- moves $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and moves into sets of states $F(a, s) \subseteq S$, and
- terminal costs $c_T(s)$ for terminal states.

Solutions: no longer a sequence of actions but a partial function π that maps states into actions such that it is closed with respect to s_0

Optimal: if minimizes $V^{\pi}(s_0)$

Game Trees

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- moves $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and moves into sets of states $F(a,s) \subseteq S$, and
- terminal costs $c_T(s)$ for terminal states.

Solutions: no longer a sequence of actions but a partial function π that maps states into actions such that it is closed with respect to s_0

Optimal: if minimizes $V^{\pi}(s_0)$

Algorithms: Minimax, $\alpha\beta$-pruning, MTD, ...

AND/OR Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into sets of states $F(a, s) \subseteq S$,
- action costs $c(a, s)$ for non-terminal states s, and
- terminal costs $c_T(s)$ for terminal states.

AND/OR Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into sets of states $F(a, s) \subseteq S$,
- action costs $c(a, s)$ for non-terminal states s, and
- terminal costs $c_T(s)$ for terminal states.

Solutions: a partial function π that maps states into actions such that it is closed with respect to s_0

AND/OR Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into sets of states $F(a, s) \subseteq S$,
- action costs $c(a, s)$ for non-terminal states s, and
- terminal costs $c_T(s)$ for terminal states.

Solutions: a partial function π that maps states into actions such that it is closed with respect to s_0

Optimal: if minimizes $V^{\pi}(s_0)$

AND/OR Models

Understood in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into sets of states $F(a, s) \subseteq S$,
- action costs $c(a, s)$ for non-terminal states s, and
- terminal costs $c_T(s)$ for terminal states.

Solutions: a partial function π that maps states into actions such that it is closed with respect to s_0

Optimal: if minimizes $V^{\pi}(s_0)$

Algorithms: AO*, CFC/REV, ...

Unification via Dynamic Programming

DP deals with models given in terms of:

- a discrete and finite state space S,
- an initial state $s_0 \in S$,
- a non-empty set of terminal states $S_T \subseteq S$,
- actions $A(s) \subseteq A$ applicable in each non-terminal state,
- a function that maps states and actions into sets of states $F(a,s) \subseteq S$,
- action costs $c(a,s)$ for non-terminal states s, and
- terminal costs $c_T(s)$ for terminal states.

Solutions to DP

Understood in terms of solutions V^* for 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.

Solutions to DP

Understood in terms of solutions V^* for 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.

Solutions: a partial function π that maps states into actions such that it is closed with respect to s_0

Solutions to DP

Understood in terms of solutions V^* for 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.

Solutions: a partial function π that maps states into actions such that it is closed with respect to s_0

Optimal: if $V^\pi(s_0) = V^*(s_0)$

Solutions to DP

Understood in terms of solutions V^* for 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.

Solutions: a partial function π that maps states into actions such that it is closed with respect to s_0

Optimal: if $V^\pi(s_0) = V^*(s_0)$

Algorithms: Value Iteration, Policy Iteration, ...

Learning in Depth-First Search

```
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
```

Properties of LDFS

- LDFS computes π^* for **all models** the initial V is admissible (i.e. $V \leq V^*$)

Properties of LDFS

- LDFS computes π^* for **all models** the initial V is admissible (i.e. $V \leq V^*$)
- For DETERMINISTIC models and *monotone* V,

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

Bounded LDFS

```
B-LDFS-DRIVER(s_0)
begin
    repeat B-LDFS(s_0, V(s_0)) until V(s_0) ≥ U(s_0)
    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
```

Properties of Bounded LDFS

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

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

Properties of Bounded LDFS

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

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

- For ADDITIVE models,

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

Properties of Bounded LDFS

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

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

- For ADDITIVE models,

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

- For MAX models,

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

Properties of Bounded LDFS

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

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

- For ADDITIVE models,

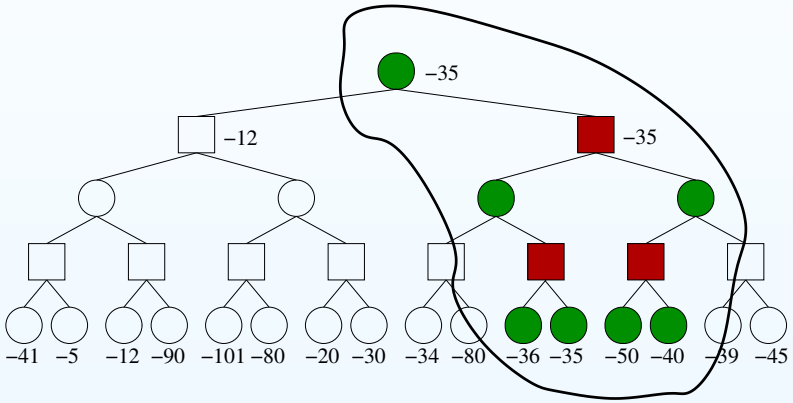
$$\text{LDFS} \quad = \quad \text{Bounded LDFS}$$

- For MAX models,

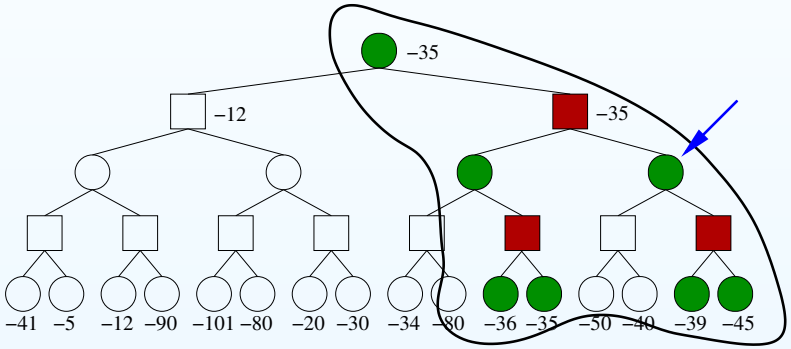
$$\text{LDFS} \quad \neq \quad \text{Bounded LDFS}$$

- 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 vs Local Optimality



Global Optimality



Local Optimality

Empirical Evaluation: Domains

- **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 $N = 10, 20, \ldots, 60$.
- **Diagnosis:** There are N binary tests for finding out the true state of a system among M different states. 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. 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. We consider $N = 15, 20, \ldots, 40$,

Empirical Evaluation: Domains

- **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 $N = 10, 20, \ldots, 60$.
- **Diagnosis:** There are N binary tests for finding out the true state of a system among M different states. 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. 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. We consider $N = 15, 20, \ldots, 40$,

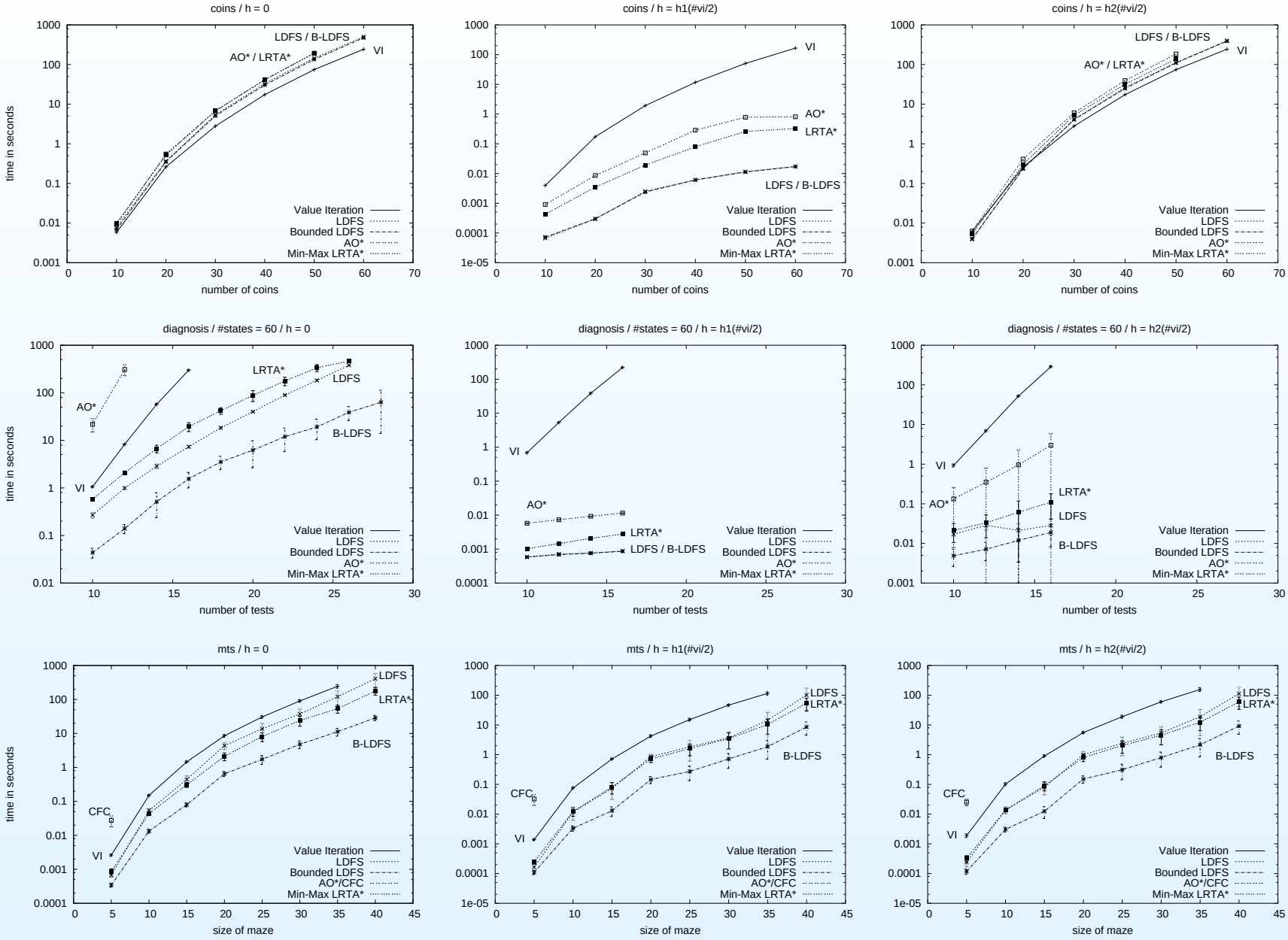
Empirical Evaluation: Domains

- **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 $N = 10, 20, \ldots, 60$.
- **Diagnosis:** There are N binary tests for finding out the true state of a system among M different states. 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. 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. We consider $N = 15, 20, \ldots, 40$,

Empirical Evaluation: Domains

- **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 $N = 10, 20, \ldots, 60$.
- **Diagnosis:** There are N binary tests for finding out the true state of a system among M different states. 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. 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. We consider $N = 15, 20, \ldots, 40$,

Empirical Evaluation: Results (3 Domains)



Conclusions

- DP formulations and heuristic search can be combined into a framework that reveals the similarities among different AI algorithms
- not only we were able to understand better previous algorithm, but to develop new more effective algorithms for Max models
- LDFS and Bounded LDFS can be thought as generalizations of IDA* for NON-DETERMINISTIC models, and, as the latter, it suffers some of its shortcomings
- We are currently working on methods to deal with them
- Papers (available at `http://www.ldc.usb.ve/~bonet`):

1. B. Bonet and H. Geffner. *Learning in Depth-First Search: A Unified Approach to Heuristic Search in Deterministic, Non-Deterministic, Probabilistic, and Game Tree Settings.* Tech Report. 2005.
2. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. of AAAI-05. Forthcoming.
3. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. IJCAI-05 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. Forthcoming.

Conclusions

- DP formulations and heuristic search can be combined into a framework that reveals the similarities among different AI algorithms
- not only we were able to understand better previous algorithm, but to develop new more effective algorithms for Max models
- LDFS and Bounded LDFS can be thought as generalizations of IDA* for NON-DETERMINISTIC models, and, as the latter, it suffers some of its shortcomings
- We are currently working on methods to deal with them
- Papers (available at `http://www.ldc.usb.ve/~bonet`):

1. B. Bonet and H. Geffner. *Learning in Depth-First Search: A Unified Approach to Heuristic Search in Deterministic, Non-Deterministic, Probabilistic, and Game Tree Settings.* Tech Report. 2005.
2. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. of AAAI-05. Forthcoming.
3. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. IJCAI-05 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. Forthcoming.

Conclusions

- DP formulations and heuristic search can be combined into a framework that reveals the similarities among different AI algorithms
- not only we were able to understand better previous algorithm, but to develop new more effective algorithms for Max models
- LDFS and Bounded LDFS can be thought as generalizations of IDA* for NON-DETERMINISTIC models, and, as the latter, it suffers some of its shortcomings
- We are currently working on methods to deal with them
- Papers (available at `http://www.ldc.usb.ve/~bonet`):

1. B. Bonet and H. Geffner. *Learning in Depth-First Search: A Unified Approach to Heuristic Search in Deterministic, Non-Deterministic, Probabilistic, and Game Tree Settings.* Tech Report. 2005.
2. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. of AAAI-05. Forthcoming.
3. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. IJCAI-05 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. Forthcoming.

Conclusions

- DP formulations and heuristic search can be combined into a framework that reveals the similarities among different AI algorithms
- not only we were able to understand better previous algorithm, but to develop new more effective algorithms for Max models
- LDFS and Bounded LDFS can be thought as generalizations of IDA* for NON-DETERMINISTIC models, and, as the latter, it suffers some of its shortcomings
- We are currently working on methods to deal with them
- Papers (available at `http://www.ldc.usb.ve/~bonet`):

1. B. Bonet and H. Geffner. *Learning in Depth-First Search: A Unified Approach to Heuristic Search in Deterministic, Non-Deterministic, Probabilistic, and Game Tree Settings.* Tech Report. 2005.
2. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. of AAAI-05. Forthcoming.
3. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. IJCAI-05 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. Forthcoming.

Conclusions

- DP formulations and heuristic search can be combined into a framework that reveals the similarities among different AI algorithms
- not only we were able to understand better previous algorithm, but to develop new more effective algorithms for Max models
- LDFS and Bounded LDFS can be thought as generalizations of IDA* for NON-DETERMINISTIC models, and, as the latter, it suffers some of its shortcomings
- We are currently working on methods to deal with them
- Papers (available at `http://www.ldc.usb.ve/~bonet`):

1. B. Bonet and H. Geffner. *Learning in Depth-First Search: A Unified Approach to Heuristic Search in Deterministic, Non-Deterministic, Probabilistic, and Game Tree Settings.* Tech Report. 2005.
2. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. of AAAI-05. Forthcoming.
3. B. Bonet and H. Geffner. *An Algorithm Better than AO*?* Proc. IJCAI-05 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. Forthcoming.