Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Iterative Depth-First Search for FOND Planning

Ramon Fraga Pereira[1], **André Grahl Pereira**[2], **Frederico Messa**[2], **Giuseppe De Giacomo**[1]

[1]Sapienza University of Rome, Rome, Italy
[2]Federal University of Rio Grande do Sul, Porto Alegre, Brazil
{pereira,degiacomo}@diag.uniroma1.it
{agpereira,frederico.messa}@inf.ufrgs.br

Abstract

Fully Observable Non-Deterministic (FOND) planning models uncertainty through actions with *non-deterministic* effects. Existing FOND planning algorithms are effective and employ a wide range of techniques. However, most of the existing algorithms are not robust for dealing with both non-determinism and task size. In this paper, we develop a novel *iterative depth-first search* algorithm that solves FOND planning tasks and produces *strong cyclic policies*. Our algorithm is explicitly designed for FOND planning, addressing more directly the non-deterministic aspect of FOND planning, and it also exploits the benefits of heuristic functions to make the algorithm more effective during the iterative searching process. We compare our proposed algorithm to well-known FOND planners, and show that it has robust performance over several distinct types of FOND domains considering different metrics.

Introduction

Fully Observable Non-Deterministic (FOND) planning is an important planning model that aims to handle the uncertainty of the effects of actions (Cimatti et al. 2003). In FOND planning, states are *fully observable* and actions may have *non-deterministic* effects (i.e., an action may generate a set of possible successor states). FOND planning is relevant for solving other related planning models, such as *stochastic shortest path* (SSP) planning (Bertsekas and Tsitsiklis 1991), *planning for temporally extended goals* (Patrizi, Lipovetzky, and Geffner 2013; Camacho et al. 2017; Camacho and McIlraith 2019; Camacho et al. 2018; De Giacomo and Rubin 2018; Brafman and De Giacomo 2019), and *generalized planning* (Hu and Giacomo 2011; Bonet et al. 2017, 2020). Solutions for FOND planning can be characterized as *strong policies* which guarantee to achieve the goal condition in a finite number of steps, and *strong cyclic policies* which guarantee to lead only to states from which a goal condition is satisfiable in a finite number of steps (Cimatti et al. 2003).

Existing FOND planning algorithms in the literature are based on a diverse set of techniques and effectively solve difficult tasks when the non-determinism of the actions must be addressed. Cimatti et al. (2003) and Kissmann and Edelkamp (2009) have introduced *model-checking* planners based on *binary decision diagrams*. Some of the most effective FOND planners rely on standard *Classical Planning* techniques by enumerating plans for a deterministic version of the task until producing a strong cyclic policy (Kuter et al. 2008; Fu et al. 2011; Muise, McIlraith, and Beck 2012; Muise, McIlraith, and Belle 2014; Muise, Belle, and McIlraith 2014). There are also planners that efficiently employ AND/OR heuristic search for solving FOND planning tasks, such as MYND (Mattmüller et al. 2010) and GRENDEL (Ramírez and Sardiña 2014). Recently, Geffner and Geffner (2018) have proposed a SAT encoding for FOND planning, and an iterative SAT-based planner that effectively handles the uncertainty of FOND planning. Nevertheless, these FOND planners present some limitations. Some of these planners address the non-determinism of the actions more indirectly, whereas others rely on algorithms with sophisticated and costly control procedures, and others do not take advantage of fundamental characteristics of planning models. As a result, such FOND planners are not robust for dealing with some of the non-determinism aspects of FOND planning and task size.

In this paper, we introduce a novel *iterative depth-first search* algorithm that solves FOND planning tasks and produces *strong cyclic policies*. Our algorithm is based on two main concepts: (1) it is explicitly designed for solving FOND planning tasks, so it addresses more directly the non-deterministic aspect of FOND planning during the searching process; and (2) it exploits the benefits of heuristic functions to make the iterative searching process more effective. We also introduce an efficient version of our algorithm that prunes unpromising states in each iteration. To better understand the behavior of our proposed *iterative depth-first search* algorithm, we characterize its behavior through fundamental properties of FOND planning policies.

We empirically evaluate our algorithm over two FOND benchmark sets: a set from IPC (Bryce and Buffet 2008) and (Muise, McIlraith, and Beck 2012); and a set containing new FOND planning domains, proposed by Geffner and Geffner (2018). We show that our algorithm outperforms some of the existing state-of-the-art FOND planners on planning time and coverage, especially for the new FOND domains. We also show that the pruning technique makes our algorithm competitive with existing FOND planners. Our contributions open new research directions in FOND planning, such as the design of more informed heuristic functions, and the development of more effective search algorithms.

Background

FOND Planning

A *Fully Observable Non-Deterministic* (FOND) planning task (Mattmüller et al. 2010) is a tuple $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A} \rangle$. $\mathcal{V}$ is a set of *state variables*, and each variable $v \in \mathcal{V}$ has a finite domain D_v. A *partial state* s maps variables $v \in \mathcal{V}$ to values in D_v, $s[v] \in \mathrm{D}_v$, or to a undefined value $s[v] = \bot$. $vars(s)$ is the set of variables in s with defined values. If every variable $\mathcal{V}$ in s is defined, then s is a *state*. s_0 is a state representing the *initial* state, whereas s_* is a partial state representing the *goal condition*. A state s is a *goal* state if and only if $s \vDash s_*$. $\mathcal{A}$ is a finite set of *non-deterministic actions*, in which every action $a \in \mathcal{A}$ consists of $a = \langle pre, \mathrm{EFFS} \rangle$, where $pre(a)$ is a partial state called *pre-conditions*, and $\mathrm{EFFS}(a)$ is a non-empty set of partial states that represent the possible *effects* of a. A *non-deterministic action* $a \in \mathcal{A}$ is applicable in a state s iff $s \vDash pre(a)$. The *application* of an effect $eff \in \mathrm{EFFS}(a)$ to a state s generates a state $s' = \mathrm{SUCC}(s, eff)$ with $s'[v] = eff[v]$ if $v \in vars(eff)$, and $s'[v] = s[v]$ if not. The application of $\mathrm{EFFS}(a)$ to a state s generates a set of successor states $\mathrm{SUCCS}(s, a) = \{\mathrm{SUCC}(s, eff) \mid eff \in \mathrm{EFFS}(a)\}$. We call $a \in \mathcal{A}$ simple *deterministic* if $|\mathrm{EFFS}(a)|$ has size one.

A solution to a FOND planning task Π is a *policy* π which is formally defined as a partial function $\pi : \mathcal{S} \mapsto \mathcal{A} \cup \{\bot\}$, which maps non-goal states of $\mathcal{S}$ into actions, such that an action $\pi(s)$ is applicable in the state s. A π-trajectory with length $k-1$ is a non-empty sequence of states $\langle s^1, s^2, \ldots s^k \rangle$, such that $s^{i+1} \in \mathrm{SUCCS}(s^i, \pi(s^i)), \forall i \in \{1, 2, \ldots, k-1\}$. A π-trajectory is called empty if it has a single state, and thus length zero. A policy π is *closed* if any π-trajectory starting from s_0 ends either in a goal state or in a state defined in the policy π. A policy π is a *strong policy* for Π if it is closed and no π-trajectory passes through a state more than once. A policy π is a *strong cyclic policy* for Π if it is closed and any π-trajectory starting from s_0 which does not end in a goal state, ends in a state s' such that exists another π-trajectory starting from s' ending in a goal state. Note that a *strong cyclic policy* may re-visit states infinite times, in a cyclic way, but the *fairness* assumption guarantees that it will *almost surely* reach a goal state at some point along the execution. The assumption of *fairness* defines that all action outcomes in a given state will occur infinitely often (Cimatti et al. 2003).

Determinization and Heuristics for FOND Planning

A *determinization* of a FOND planning task Π defines a new FOND planning task Π^{DET} where all actions are *deterministic*. Formally, $\Pi^{\mathrm{DET}} = \langle \mathcal{V}, s_0, s_*, \mathcal{A}^{\mathrm{DET}} \rangle$ is a task where $\mathcal{A}^{\mathrm{DET}}$ is a set of deterministic actions with one action a' for each outcome $eff \in \mathrm{EFFS}(a)$ of all actions in $a \in \mathcal{A}$. A s-plan for Π^{DET} is a sequence of actions that when applied to s reaches a goal state. A s-plan is optimal if it has minimum cost among all s-plans. A solution for Π^{DET} is a s_0-plan.

A *heuristic function* $h : \mathcal{S} \mapsto \mathbb{R} \cup \{\infty\}$ maps a state s to its h-value, an estimation of the cost of a s-plan. A *perfect heuristic* h^* maps a state s to its *optimal* cost plan or ∞, if no plan exists. A heuristic is *admissible* if $h(s) \leq h^*(s)$ for all $s \in \mathcal{S}$. *Delete-relaxation* heuristics (Bonet and Geffner 2001; Hoffmann and Nebel 2001) can be efficiently used in FOND planning by applying *determinization* (Mattmüller 2013). Other types of heuristics for FOND planning have been proposed in the literature, such as *pattern-database* heuristics (Mattmüller et al. 2010), and pruning techniques (Winterer, Wehrle, and Katz 2016; Winterer et al. 2017).

FOND Planners

One of the first FOND planners in the literature was developed by Cimatti et al. (2003), and it is called MBP (Model-Based Planner). MBP solves FOND planning tasks via *model-checking*, and it is built upon *binary decision diagrams* (BDDs). GAMER (Kissmann and Edelkamp 2009), the winner of the FOND track at IPC (2008), is also based on BDDs, but GAMER has shown to be much more efficient than MBP.

MYND (Mattmüller et al. 2010) is a FOND planner based on an adapted version of LAO* (Hansen and Zilberstein 2001), a heuristic search algorithm that has theoretical guarantees to extract strong cyclic solutions for Markov decision problems. NDP (Kuter et al. 2008) makes use of *Classical Planning* algorithms to solve FOND planning tasks. FIP (Fu et al. 2011) is similar to NDP, but the main difference is that FIP avoids exploring already explored/solved states, being more efficient than NDP. PRP (Muise, McIlraith, and Beck 2012) is one the most efficient FOND planners in the literature, and it is built upon some improvements over the state relevance techniques, such as avoiding dead-ends states. The main idea of these planners is selecting a reachable state s by the current policy that still is undefined in the current policy. Then, the planner finds a s-plan with Π^{DET} and incorporates the s-plan into the policy. The planner repeats this process until the policy is strong cyclic, or it finds out that it is not possible to produce a strong cyclic policy from the current policy, and then it backtracks. Since these planners find s-plan for Π^{DET} which do not consider the non-deterministic effects, they can take too much time to find that the current policy can not become a strong cyclic policy, or they can add actions to a policy that require too much search effort to become a strong cyclic policy.

GRENDEL (Ramírez and Sardiña 2014) is a FOND planner that combines regression with a symbolic fixed-point computation for extracting strong cyclic policies. Most recently, Geffner and Geffner (2018) developed FONDSAT, an iterative SAT-based FOND planner that is capable to produce strong and strong cyclic policies for FOND planning tasks.

Iterative Depth-First Search Algorithm for FOND Planning

In this section, we propose a novel *iterative depth-first search* algorithm called IDFS that produces strong cyclic policies for FOND planning tasks. IDFS performs a series of bounded depth-first searches that consider the non-determinism aspect of FOND planning during the iterative searching process. IDFS produces a strong cyclic policy in a bottom-up way and only adds an action to the policy if it determines that the resulting policy with the additional action has the potential to become a strong cyclic policy without exceeding the current search-depth bound.

Evaluation Function $\mathcal{F}$

A heuristic function $h(s)$ estimates the length of a trajectory from the state s to any goal state. It can assess whether a search procedure can reach a goal state without exceeding a search-depth bound. We define the f-value of a state s as $f(s) = g(s) + h(s)$, with $g(s)$ being the search depth from s_0 to s. In this paper, we assume that all actions have a uniform action cost equal to one[1]. During the iterative searching process, IDFS considers the application of an action $a \in \mathcal{A}$ to a state s by evaluating the set of generated successor states $\text{SUCCS}(s, a)$ using an *evaluation function* $\mathcal{F}_\xi$, which returns the estimate of the search depth required to reach a goal state through $\text{SUCCS}(s, a)$. The evaluation function $\mathcal{F}_\xi$ uses a parameter function ξ to aggregate the f-values of states in $\text{SUCCS}(s, a)$: $\mathcal{F}_{\min}(\text{SUCCS}(s, a))$ is $\min_{s' \in \text{SUCCS}(s,a)} f(s')$, and $\mathcal{F}_{\max}(\text{SUCCS}(s, a))$ is $\max_{s' \in \text{SUCCS}(s,a)} f(s')$. Note that the evaluation function $\mathcal{F}_\xi$ is "pessimistic" when $\xi = \max$, whereas it is "optmistic" when $\xi = \min$.

The IDFS Algorithm

We now present the IDFS, and Algorithm 1 formally shows its pseudo-code.

Main Iterative Loop (Lines 1-8) IDFS performs a series of bounded depth-first searches, called *iterations* to solve a FOND planning task Π. IDFS assumes that h is a heuristic function for the *deterministic* version of the task Π. Prior to the first iteration, IDFS initializes the *bound* with the estimated value of heuristic function h of the initial state s_0. At each iteration, IDFS aims to produce a solution by searching to a depth of at most *bound*. The main loop receives a flag indicating if the iteration produced a solution from state s_0. If the flag is SOLVED, then π is a strong cyclic policy for task Π, and IDFS returns it. If the flag is UNSOLVED, then IDFS_R could not produce a strong cyclic policy for task Π with the current *bound*. Thus, IDFS assigns to $bound$ the value of the global variable $nextBound$. The value of $nextBound$ is the minimum estimate ($\mathcal{F}_\xi$ or g-value+1) of a generated but not expanded set of successors. If no set of successors with a greater estimate than $bound$ is generated, the main loop returns UNSOLVABLE. This general strategy of depth-first search bounded by estimates is inspired by the Iterative Deepening A* algorithm by Korf (1985).

Recursion (Lines 9-36) IDFS iteratively tries to produce a strong cyclic policy for task Π in a bottom-up way, using a recursive procedure called IDFS_R. Definition 1 formally defines the concept of *partial strong cyclic policy*, which we use to explain the behavior of IDFS.

Definition 1. *A policy π is a **partial strong cyclic policy** from a state s of a* FOND *task Π for a set A of primary target states and a set B of secondary target states, iff A is **reachable** from s in π, and π is **sinking** to B. (We omit A and B, when the context is clear.)*

[1]All FOND planning domains in the available benchmarks have actions with unitary cost.

Algorithm 1: IDFS

```
   // Main Iterative Loop.
1  IDFS(s_0):
2      bound := h(s_0), nextBound := ∞
3      while bound ≤ |S| do
4          flag, π := IDFS_R(s_0, ∅, ∅, ∅)
5          if flag = SOLVED then
6              return π
7          bound := nextBound, nextBound := ∞
8      return UNSOLVABLE
   // Recursion.
9  IDFS_R(s, Z, Z_*, π):
       // Base Cases.
10     if s ⊨ s_* or π(s) ≠ ⊥ or s ∈ Z_* then
11         return SOLVED, π
12     if s ∈ (Z ∖ Z_*) then
13         return UNSOLVED, π
       // Evaluate Actions.
14     for a ∈ APPLICABLEACTIONS(s) do
15         if F_ξ(SUCCS(s, a)) > bound and Z_* = ∅ then
16             if F_ξ(SUCCS(s, a)) < nextBound then
17                 nextBound := F_ξ(SUCCS(s, a))
18             continue // Next action.
19         if g(s) + 1 > bound then
20             if g(s) + 1 < nextBound then
21                 nextBound := g(s) + 1
22             continue // Next action.
           // Fixed Point.
23         Z'_* := Z_*, π' := π, M := SUCCS(s, a), M_* := ∅
24         repeat
25             REACHEDFIXEDPOINT := TRUE
26             for s' ∈ (M ∖ M_*) do
27                 flag, π' := IDFS_R(s', Z ∪ {s}, Z'_*, π')
28                 if flag = SOLVED then
29                     M_* := M_* ∪ {s'}
30                     Z'_* := Z ∪ {s}
31                     REACHEDFIXEDPOINT := FALSE
32         until REACHEDFIXEDPOINT
33         if M_* = M then
34             π'(s) := a
35             return SOLVED, π'
36     return UNSOLVED, π
```

- *A is **reachable** from s in π iff $s \in A$ or there is a π-trajectory starting from s ending in a state of A that does not includes a state of $B \setminus A$.*
- *π is **sinking** to B iff any π-trajectory either goes through a state of B or ends in a state s', such that exists another π-trajectory starting from s' ending in a state of B.*

IDFS_R aims to produce a *partial strong cyclic policy* from state s of a FOND task Π by searching to a depth of at most the current $bound$. IDFS_R takes as input four arguments: the state s, the set $\mathcal{Z}$, the set $\mathcal{Z}_*$, and a policy π. These arguments are set to empty in each iteration of the main iterative loop. The set $\mathcal{Z}$ contains the ancestors of state s. The policy π is the policy that IDFS has built up to the moment of the current call of IDFS_R. The set $\mathcal{Z}_* \subseteq \mathcal{Z}$ contains all ancestors of state s

that: are not in π and are ancestors of states in π. Note that IDFS has found a trajectory to a goal state for all states in $\mathcal{Z}_*$. If IDFS$_R$ returns SOLVED, then the returned policy is a *partial strong cyclic policy* from state s. The sets A and B of the partial strong cyclic policy are $A = \mathcal{S}_* \cup \mathcal{S}_\pi \cup \mathcal{Z}_*$ (primary target states) and the set $B = A \cup \mathcal{Z}$ (secondary target states). $\mathcal{S}_\pi$ is the set $\{s \mid \pi(s) \neq \bot\}$, and $\mathcal{S}_*$ the set $\{s \mid s \vDash s_*\}$ of goal states. Since $A = B = \mathcal{S}_*$ in the call of IDFS$_R$ in the main loop, the returned partial strong cyclic policy from state s_0 is a *strong cyclic policy* for task Π.

Consider the FOND planning task example of Figure 1, in which, s_0 is the initial state, s_5 is the only goal state, and there are three non-deterministic actions, applied in states s_1, s_3 and s_{10}. In the example, the current call of IDFS$_R$ is evaluating the state s_{10} (with the current recursion path is in bold). In this call, the received policy π contains the states s_2, s_3, s_4, s_6 and s_8 (in purple), i.e., $\mathcal{S}_\pi = \{s_2, s_3, s_4, s_6, s_8\}$. The ancestors of state s_{10} are the states s_0, s_1 and s_9, i.e., $\mathcal{Z} = \{s_0, s_1, s_9\}$. The former two are in $\mathcal{Z}_* = \{s_0, s_1\}$ (in green). Thus, the set B of secondary target states includes states of A and the state s_9.

IDFS$_R$ Base Cases (Lines 10-13) IDFS$_R$ first checks whether the current state s of the recursion is either a primary target state ($s \vDash s_*$ or $s \in \mathcal{Z}_*$ or $\pi(s) \neq \bot$), or a state $s \in \mathcal{Z}$ which is not primary target state. If the first case occurs, IDFS$_R$ returns SOLVED. If the second case occurs, it returns UNSOLVED. Both cases return policy unmodified.

IDFS$_R$ Evaluate Actions (Lines 14-22) If the base cases do not address the state s, IDFS$_R$ proceeds to attempt to solve it (Line 14). To optimize the search, IDFS$_R$ evaluates first the *applicable actions* with least $\mathcal{F}_{\max}(\text{SUCCS}(s,a))$ and discards actions with $\mathcal{F}_{\max}(\text{SUCCS}(s,a)) = \infty$. If the estimated solution depth of the successor states $\text{SUCCS}(s,a)$ is greater than the current *bound* (i.e., $\mathcal{F}_\xi(\text{SUCCS}(s,a)) > bound$) and $\mathcal{Z}_* = \varnothing$ (Line 15), then the set of successor states is discarded, and $\mathcal{F}_\xi(\text{SUCCS}(s,a))$ is assigned to $nextBound$ (Line 17) if $nextBound$ was greater than it.

IDFS$_R$ verifies whether $\mathcal{Z}_* = \varnothing$ because it aims to find at least one trajectory from s to a primary target state in $A = \mathcal{S}_* \cup \mathcal{S}_\pi \cup \mathcal{Z}_*$. Note that $\mathcal{F}_\xi(\text{SUCCS}(s,a))$ aggregates f-values that only estimate the solution depth from s_0 through $\text{SUCCS}(s,a)$ to goal states. Thus, $\mathcal{F}_\xi(\text{SUCCS}(s,a))$ can only be used to estimate the solution depth to a primary target state when $\mathcal{Z}_* = \varnothing$, since it implies $A = \mathcal{S}_*$.

If $\mathcal{Z}_* \neq \varnothing$ the g-value the successor states $\text{SUCCS}(s,a)$ can be used to estimate the solution depth to a primary target state. In this case, if $g(n) + 1$ is greater than the current *bound*, the set of successor states is discarded, and $g(n) + 1$ is assigned to $nextBound$ if $nextBound$ was greater than it. If neither $\mathcal{F}_\xi(\text{SUCCS}(s,a))$ nor $g(s)$ prevent the search to proceed, IDFS$_R$ evaluates the successor states $\text{SUCCS}(s,a)$.

IDFS$_R$ Fixed Point (Lines 23-35) IDFS$_R$ recursively descends into the successor states $\text{SUCCS}(s,a)$ of s to determine whether it should or not add the mapping $s \mapsto a$ to the police π. Namely, it adds the mapping $s \mapsto a$ to π only if **all** the recursive calls on states of $\text{SUCCS}(s,a)$ returned SOLVED (Line 34). If not, it discards the possibility of using the action a on s, and proceeds to the next action.

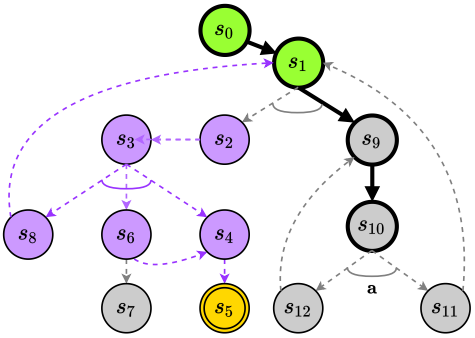


Figure 1: Current path of analysis is in bold. Goal state in yellow. π is in purple. $\mathcal{Z}_*$ is in green.

Consider again the FOND planning task example of Figure 1. Assume that IDFS$_R$ reaches the point to evaluate the successor states s_{11} and s_{12} of s_{10}. Note that for s_{10} the set of A primary target states is $\{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_8\}$, and the set B of secondary target states is $\{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_8, s_9\}$. IDFS$_R$ aims to produce a policy π' such that A is reachable from s_{10} in π', and π' is *sinking* to B. To ensure that, IDFS$_R$ must find a trajectory from s_{10} to a state in A that does not include a state of $B \setminus A$. Thus, IDFS$_R$ analyzes all successors of s_{10} to find such a trajectory. Before finding this trajectory, the arguments $\pi' \coloneqq \pi$, $\mathcal{Z}' \coloneqq \mathcal{Z} \cup \{s\}$ and $\mathcal{Z}'_* \coloneqq \mathcal{Z}_*$ passed to IDFS$_R$ when evaluating states s_{11} and s_{12} remaining unchanged.

Suppose the first recursive call evaluates s_{12}, thus the primary targets states for s_{12} are A. Since s_9 is an ancestor of s_{12} and $s_9 \notin A$, there is no trajectory from s_{12} to a state of A that does not includes a state of $B \setminus A$. Thus, the recursive call will fail and return UNSOLVED. Next, IDFS$_R$ will proceed to the other successor state of s_{10}, namely s_{11}. If the recursive call on s_{11} fails because of the bound, the algorithm will have analyzed all successors of s_{10} without having any progress, as the set of successors states "already solved" $\mathcal{M}_*$ would not have changed, and thus a fixed-point would be reached, resulting in the action being discarded.

Suppose the recursive call on state s_{11} does not fail, and it returns SOLVED. Then, the returned policy is a partial strong cyclic policy from s_{11} for the set of primary states A' and the set of secondary states B'. Since $A' = A$ and $B' = B \cup \{s\}$. IDFS$_R$ now evaluates s_{12} again, but now with a modified A'. Since we already have a trajectory from s_{10} to A, now A' includes also s_9, s_{10} and s_{11}, and $B' = A'$. The recursive call on s_{12} returns SOLVED because there is a trajectory to A'. The new policy extended with $s_{10} \mapsto \mathbf{a}$ is a partial strong cyclic policy from s_{10} for A and B, and can be returned with the flag SOLVED.

IDFS$_R$ End (Line 36) In case none of the actions $a \in \text{APPLICABLEACTIONS}(s)$ are able to generate a partial strong cyclic from s to A and B, IDFS$_R$ returns UNSOLVED.

IDFS Pruning We now present an extended version of IDFS called IDFS *Pruning* (IDFSP). Algorithm 2 presents the pseudo-code of IDFSP. In essence, IDFSP is similar to

Algorithm 2: IDFS Pruning (IDFSP)

```
1  IDFSP(s_0):
2      bound := h(s_0), nextBound := ∞, X := ∅
3      while bound ≤ |S| do
4          flag, π := IDFSP_R(s_0, ∅, ∅, ∅)
5          if flag = SOLVED then
6              return π
7          bound := nextBound, nextBound := ∞, X := ∅
8      return UNSOLVED
9  IDFSP_R(s, Z, Z_*, π):
       ← Lines 11-14 of Algorithm 1.
10     if s ∈ X then
11         return UNSOLVED, π
12     PROMISING := FALSE
13     for a ∈ APPLICABLEACTIONS(s) do
           ← Lines 16-23 of Algorithm 1.
           // Fixed Point.
14         Z'_* := Z_*, π' := π, M := SUCCS(s, a), M_* := ∅
15         repeat
16             REACHEDFIXEDPOINT := TRUE
17             for s' ∈ (M \ M_*) do
18                 flag, π' := IDFSP_R(s', Z ∪ {s}, Z'_*, π')
19                 if M ∩ X ≠ ∅ then
20                     break
21                 if flag = SOLVED then
22                     M_* := M_* ∪ {s'}
23                     Z'_* := Z ∪ {s}
24                     REACHEDFIXEDPOINT := FALSE
25             if M ∩ X ≠ ∅ then
26                 break
27             if REACHEDFIXEDPOINT then
28                 PROMISING := TRUE
29         until REACHEDFIXEDPOINT
           ← Lines 34-36 of Algorithm 1.
30     if PROMISING = FALSE then
31         X := X ∪ {s}
32     return UNSOLVED, π
```

IDFS, and the main difference is that it *prunes* states during the searching process. IDFSP_R considers that a state s is *promising* if $s \in A$ or at least one of its applicable actions a reaches the fixed point when evaluating the set of successor states $\mathcal{M} := \text{SUCCS}(s, a)$. If the state s is not promising, IDFSP_R adds the state s into the global set $\mathcal{X}$. IDFSP sets $\mathcal{X}$ to empty before each iteration. IDFSP_R has one additional base case that returns UNSOLVED if state $s \in \mathcal{X}$ (Lines 10–11). During the fixed-point computation, IDFSP_R verifies if at least one of the states in $\mathcal{M} := \text{SUCCS}(s, a)$ is in $\mathcal{X}$ and stops the fixed-point computation if it is. This pruning method helps the search because it avoids repeated evaluation of states that generate successors that can not be part of the policy with the current bound.

Minimal Critical-Value in FOND Planning

We now introduce key properties about the set of strong cyclic policies of a FOND task Π that are important to characterize the behavior of IDFS. A FOND planning task Π has

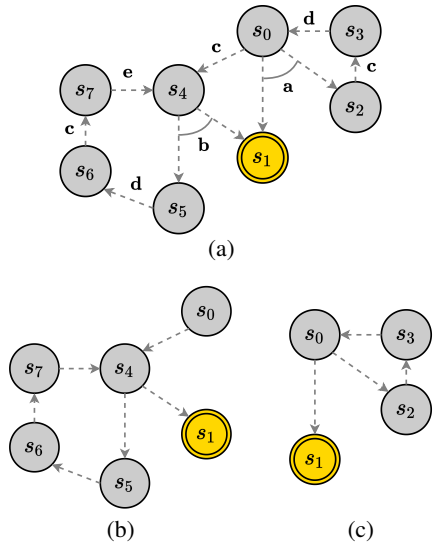


Figure 2: *Minimal Critical-Value* example.

a set of strong cyclic policies $\mathcal{P}(\Pi)$ – if Π is unsolvable, then $\mathcal{P}(\Pi) = \varnothing$. Figure 2a shows the state-space of a FOND planning task Π, with eight states, three deterministic actions, and two non-deterministic actions – namely $\{a, b\}$. This task has only two strong cyclic policies, $\pi_0 = \{s_0 \mapsto \mathbf{c}, s_4 \mapsto \mathbf{b}, s_5 \mapsto \mathbf{d}, s_6 \mapsto \mathbf{c}, s_7 \mapsto \mathbf{e}\}$ and $\pi_1 = \{s_0 \mapsto \mathbf{a}, s_2 \mapsto \mathbf{c}, s_3 \mapsto \mathbf{d}\}$. Figures 2b and 2c, show respectively the part of the state-space reachable from s_0 using each policy. We use these state-spaces to present the concept of *critical-values* of policies (Definition 2).

Definition 2. *The **critical-value** $\mathbf{cv}(\pi)$ of a policy π is the value of the length of the longest π-trajectory $\langle s^1, s^2, \ldots, s^k \rangle$ with $s^1 = s_0$ and no $i < j \leq k - 1$ with $s^i = s^j$.*

The $\mathbf{cv}$ of π_0 is generated by $\langle s_0, s_4, s_5, s_6, s_7, s_4 \rangle$, therefore $\mathbf{cv}(\pi_0) = 5$. The $\mathbf{cv}$ of π_1 is generated by $\langle s_0, s_2, s_3, s_0 \rangle$, therefore $\mathbf{cv}(\pi_1) = 3$. Definition 3 introduces the concept of minimal critical-value $\mathbf{cv}^*$ of a FOND planning task Π.

Definition 3. *The **minimal critical-value** $\mathbf{cv}^*$ of a* FOND *planning task Π is equal to $\min_{\pi \in \mathcal{P}(\Pi)} \mathbf{cv}(\pi)$.*

Thus, the $\mathbf{cv}^*$ of the FOND planning task Π in the Figure 2a is $\mathbf{cv}^*(\Pi) = \min\{\mathbf{cv}(\pi_0), \mathbf{cv}(\pi_1)\} = \min\{5, 3\} = 3$. Definition 3 considers all strong cyclic policies of the task Π, which are, in general, unavailable. Therefore, we usually do not know the value of $\mathbf{cv}^*(\Pi)$. Nevertheless, we prove that if IDFS uses $\mathcal{F}_{\min}$ and an admissible heuristic function for the deterministic version of the task Π, IDFS will search to a depth of at most $\mathbf{cv}^*$. Therefore, if Π is *solvable*, IDFS will return a strong cyclic policy before the search starts evaluating states at a depth greater than $\mathbf{cv}^*$.

Theoretical Properties

In this section, we present a proof idea that shows that if a FOND planning task Π is *solvable*, IDFS returns a *strong cyclic policy* by searching to a depth of at most $\mathbf{cv}^*(\Pi)$, and if Π is *unsolvable*, IDFS identifies it correctly. Theorem 1

	IDFS ($\mathcal{F}_{\min}$, h^{BLIND})					IDFS ($\mathcal{F}_{\min}$, h^{MAX})				
Domain (#)	C	T	$\lvert\pi\rvert$	b_I/b_F	i	C	T	$\lvert\pi\rvert$	b_I/b_F	i
DOORS (#15)	11	30.1	1486.7	0.0/8.0	8.0	11	10.9	1486.7	7.0/8.0	2.0
ISLANDS (#60)	29	18.7	4.9	0.0/4.9	4.9	60	0.1	4.9	4.9/4.9	1.0
MINER (#51)	0	-	-	-/-	-	40	-	-	-/-	-
TW-SPIKY (#11)	4	18.5	26.0	0.0/22.0	22.0	9	3.7	25.0	8.0/22.0	15.0
TW-TRUCK (#74)	13	23.4	13.8	0.0/10.8	10.8	26	0.6	13.2	4.2/10.8	7.7
Sub-Total (#211)	57	22.6	382.5	0.0/11.4	11.4	146	3.8	382.4	6.1/11.4	6.4
ACROBATICS (#8)	4	2.3	14.0	0.0/14.0	14.0	8	0.1	14.0	3.8/14.0	11.3
BEAM-WALK (#11)	8	29.2	254.0	0.0/254.0	254.0	8	9.5	254.0	127.5/254.0	127.5
BW-ORIG (#30)	10	17.4	13.5	0.0/7.5	7.5	10	4.2	12.4	2.8/7.5	5.7
BW-2 (#15)	5	38.0	14.4	0.0/9.4	9.4	5	4.9	14.2	2.8/9.4	7.6
BW-NEW (#40)	6	26.8	8.0	0.0/5.5	5.5	6	2.5	8.0	2.2/5.5	4.2
CHAIN (#10)	2	62.8	42.0	0.0/28.0	28.0	10	0.1	42.0	28.0/28.0	1.0
EARTH-OBS (#40)	8	3.9	19.3	0.0/9.6	9.6	9	0.4	18.3	4.4/9.6	6.0
ELEVATORS (#15)	4	44.0	12.0	0.0/11.3	11.3	5	1.5	11.3	4.8/11.3	7.5
FAULTS (#55)	18	14.8	28.4	0.0/7.3	7.3	19	7.7	21.6	2.0/7.3	6.3
FIRST-RESP (#100)	20	22.5	5.7	0.0/5.7	5.7	23	3.7	6.3	2.6/5.7	4.0
TRI-TW (#40)	3	23.2	22.0	0.0/15.0	15.0	3	13.4	22.0	4.0/15.0	12.0
ZENO (#15)	0	-	-	-/-	-	3	-	-	-/-	-
Total (#590)	145	25.0	131.0	0.0/27.5	27.5	255	3.7	130.3	13.9/27.5	14.6

Table 1: IDFS comparison with $\mathcal{F}_{\min}$: h^{BLIND} vs h^{MAX}.

bounds the behavior of the IDFS algorithm by the structure of the FOND planning task Π. The complete proofs are available in (Pereira et al. 2022).

Theorem 1. *Given a* FOND *planning task* Π*, an admissible heuristic function* h *for a* deterministic *version of* Π*, and* IDFS *using* $\mathcal{F}_{\min}$*. If* Π *is* solvable*, then* IDFS *returns a* strong cyclic policy π *by searching to a depth of at most* $\mathbf{cv}^*(\Pi)$*. If* Π *is* unsolvable*, then* IDFS *returns* UNSOLVABLE.

Proof Idea. If Π is *solvable*, then there is a strong cyclic policy π which has $\mathbf{cv}(\pi) = \mathbf{cv}^*(\Pi)$. Suppose state s is part of the policy π, IDFS$_R$ analyzes all actions applicable on s, including the action that is part of the policy $\pi(s)$, with incremental search depths and using $\mathcal{F}_{\min}$ and heuristic h when possible. Since s is in the policy π, IDFS$_R$ can, by the construction of the algorithm, find a *policy* that includes s searching to a depth of at most $\mathbf{cv}^*$. Because task Π is *solvable* and s_0 is in any policy including policy π, IDFS returns a *strong cyclic policy* by searching to a depth of at most $\mathbf{cv}^*$. IDFS$_R$ only returns SOLVED for a state s using action a if all its successors in $\text{SUCCS}(s, a)$ return SOLVED. Thus, if a FOND planning task Π is *unsolvable*, IDFS returns UNSOLVABLE. IDFS always terminates because the state-space size limits the number of iterations of the main loop.

Experiments and Evaluation

We now present the set of experiments we have conducted to evaluate the efficiency of our IDFS algorithm for solving FOND planning tasks. We compare our algorithm to state-of-the-art FOND planners, such as PRP (Muise, McIlraith, and Beck 2012), MYND (Mattmüller et al. 2010), and FOND-SAT (Geffner and Geffner 2018). We have implemented our algorithm using part of the source code of MYND. We use the delete relaxation heuristic functions for the deterministic version of the planning task as proposed by Mattmüller (2013). As a result, we have a FOND planner called PALADINUS[2].

We empirically evaluate IDFS using two distinct benchmark sets: IPC-FOND and NEW-FOND. IPC-FOND contains 379 planning tasks over 12 FOND domains from the IPC (2008) and (Muise, McIlraith, and Beck 2012). The NEW-FOND benchmark set (Geffner and Geffner 2018) introduces FOND planning tasks that contain several trajectories to goal states that are not part of any strong cyclic policy. NEW-FOND contains 211 tasks over five FOND domains, namely DOORS, ISLANDS, MINER, TW-SPIKY, and TW-TRUCK. Note that 25 out of 590 tasks are *unsolvable*, namely, 25 FOND planning tasks of FIRST-RESP– a domain of IPC-FOND.

We have run all experiments using a single core of a 12 core Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz with 16GB of RAM, with a memory limit of 4GB, and set a 5 minute (300 seconds) time-out per planning task. We evaluate the planners, when applicable, using the following metrics: *number of solved tasks*, i.e., *coverage* (C), *time to solve* (T) in seconds, *average policy size* ($|\pi|$), *initial bound* and the *final bound* (respectively, b_I and b_F), and the *number iterations* (i). Apart from the coverage (C), all results shown in Tables 1, 2, 3, and 5 are calculated over the intersection of the tasks solved by all planners in the respective table.

IDFS with Admissible Heuristic Functions We start our evaluation by presenting a comparison of IDFS using h^{BLIND} and h^{MAX} with the evaluation function $\mathcal{F}_{\min}$. This comparison evaluates how useful the information of the heuristic function is for IDFS concerning search efficiency. We evaluate this with the following metrics: the number of solved tasks, the time to solve, and the number of iterations required to solve the task – fewer iterations mean that IDFS reaches faster the depth where it finds a strong cyclic policy. Table 1 summarizes the results for all 17 FOND domains of the used benchmark

[2]PALADINUS code: https://github.com/ramonpereira/paladinus

	IDFS ($\mathcal{F}_{\min}$, h^{ADD})					IDFS ($\mathcal{F}_{\max}$, h^{ADD})				
Domain (#)	C	T	$\lvert\pi\rvert$	b_I/b_F	i	C	T	$\lvert\pi\rvert$	b_I/b_F	i
DOORS (#15)	11	14.1	1486.7	26.7/35.7	1.9	11	10.7	1486.7	26.7/120.5	2.8
ISLANDS (#60)	60	0.5	7.0	7.0/7.0	1.0	60	0.6	7.0	7.0/7.0	1.0
MINER (#51)	51	1.1	23.2	39.6/39.9	1.2	51	1.8	23.2	39.6/39.9	1.2
TW-SPIKY (#11)	9	14.9	25.0	8.0/22.0	15.0	6	39.8	25.0	8.0/24.0	17.0
TW-TRUCK (#74)	26	19.3	14.3	3.9/11.1	8.2	21	30.0	14.4	3.9/12.1	9.2
Sub-Total (#211)	157	9.9	311.2	17.1/23.1	5.4	149	16.6	311.2	17.1/40.7	6.2
ACROBATICS (#8)	8	1.6	126.5	63.8/126.5	63.8	8	0.6	126.5	63.8/748.1	73.9
BEAM-WALK (#11)	11	0.5	453.2	453.2/453.2	1.0	9	12.0	453.2	453.2/39176.7	113.6
BW-ORIG (#30)	15	10.5	23.2	14.6/15.2	1.6	25	0.6	17.3	14.6/22.4	2.0
BW-2 (#15)	7	1.6	21.6	15.4/17.0	2.1	14	0.8	21.3	15.4/22.6	2.0
BW-NEW (#40)	10	3.3	20.0	12.3/12.8	1.4	19	0.6	17.4	12.3/19.3	2.0
CHAIN (#10)	10	0.3	162.0	161.0/161.8	1.8	10	0.3	162.0	161.0/161.8	1.8
EARTH-OBS (#40)	18	0.2	47.4	22.9/23.9	1.7	19	0.5	38.8	22.9/25.8	2.5
ELEVATORS (#15)	10	0.1	19.6	21.1/21.7	1.6	8	0.2	19.6	21.1/21.9	1.7
FAULTS (#55)	22	19.5	26.8	5.9/7.9	3.0	23	16.2	26.8	5.9/9.1	2.6
FIRST-RESP (#100)	57	0.3	13.7	12.6/12.6	1.0	31	28.7	14.3	12.6/13.8	2.2
TRI-TW (#40)	3	13.2	22.0	4.0/15.0	12.0	3	9.8	22.0	4.0/15.0	8.0
ZENO (#15)	7	13.4	29.5	30.0/31.2	2.2	6	17.8	29.5	30.0/31.2	2.2
Total (#590)	335	6.7	148.3	53.1/59.7	7.1	324	10.1	147.4	53.1/2380.7	14.5

Table 2: IDFS algorithm using h^{ADD} with $\mathcal{F}_{\min}$ and $\mathcal{F}_{\max}$, without pruning.

	IDFSP ($\mathcal{F}_{\min}$, h^{ADD})					IDFSP ($\mathcal{F}_{\max}$, h^{ADD})				
Domain (#)	C	T	$\lvert\pi\rvert$	b_I/b_F	i	C	T	$\lvert\pi\rvert$	b_I/b_F	i
DOORS (#15)	13	2.3	1486.7	26.7/35.7	1.9	13	1.7	1486.7	26.7/120.5	2.8
ISLANDS (#60)	60	0.3	7.0	7.0/7.0	1.0	60	0.5	7.0	7.0/7.0	1.0
MINER (#51)	51	0.8	23.2	39.6/39.9	1.2	51	0.9	23.2	39.6/39.9	1.2
TW-SPIKY (#11)	10	2.5	1409.3	8.0/20.0	13.0	10	3.4	1409.3	8.0/22.0	15.0
TW-TRUCK (#74)	55	0.2	17.4	3.9/12.8	9.9	44	2.1	19.8	3.9/17.3	14.4
Sub-Total (#211)	189	1.2	588.7	17.1/23.1	5.4	178	1.7	589.2	17.1/41.3	6.8
ACROBATICS (#8)	8	0.1	126.5	63.8/63.8	1.0	8	0.6	126.5	63.8/749.8	75.5
BEAM-WALK (#11)	11	0.4	453.2	453.2/453.2	1.0	11	0.5	453.2	453.2/39176.7	113.6
BW-ORIG (#30)	16	7.5	23.8	14.6/17.6	3.9	29	0.3	16.9	14.6/22.6	2.2
BW-2 (#15)	10	1.0	17.4	15.4/18.6	3.4	15	0.3	19.1	15.4/22.9	2.3
BW-NEW (#40)	12	1.6	14.4	12.3/14.9	3.5	21	0.2	17.4	12.3/19.3	2.0
CHAIN (#10)	10	0.3	162.0	161.0/161.8	1.8	10	0.3	162.0	161.0/161.8	1.8
EARTH-OBS (#40)	19	1.4	40.9	22.9/28.1	4.3	25	0.1	35.6	22.9/27.9	3.9
ELEVATORS (#15)	9	0.1	19.4	21.1/21.7	1.6	8	0.1	19.4	21.1/21.9	1.7
FAULTS (#55)	55	0.1	47.5	5.9/7.5	1.8	55	0.1	43.1	5.9/8.9	2.3
FIRST-RESP (#100)	60	0.3	19.7	12.6/12.6	1.1	46	5.9	109.5	12.6/13.8	2.3
TRI-TW (#40)	36	0.1	26.0	4.0/13.3	10.3	8	0.1	22.0	4.0/15.0	8.0
ZENO (#15)	6	12.8	29.7	30.0/31.2	2.2	8	12.9	29.7	30.0/31.2	2.2
Total (#590)	411	1.9	230.8	53.1/56.4	3.7	422	1.8	235.3	53.1/2381.1	14.8

Table 3: IDFS algorithm using h^{ADD} with $\mathcal{F}_{\min}$ and $\mathcal{F}_{\max}$, with pruning.

sets, showing the performance of IDFS when using $\mathcal{F}_{\min}$ with h^{MAX} and h^{BLIND}, denoted as IDFS ($\mathcal{F}_{\min}$, h^{MAX}) and IDFS ($\mathcal{F}_{\min}$, h^{BLIND}), respectively.

IDFS ($\mathcal{F}_{\min}$, h^{MAX}) solves in total 255 tasks, whereas IDFS ($\mathcal{F}_{\min}$, h^{BLIND}) solves 145 tasks. Both IDFS ($\mathcal{F}_{\min}$, h^{MAX}) and IDFS ($\mathcal{F}_{\min}$, h^{BLIND}) identified the 25 tasks of FIRST-RESP as *unsolvable*. IDFS ($\mathcal{F}_{\min}$, h^{BLIND}) exceeded the time limit to solve all tasks of MINER and ZENO. Table 1 shows that IDFS ($\mathcal{F}_{\min}$, h^{MAX}) always uses fewer iterations to solve the same tasks when compared to IDFS ($\mathcal{F}_{\min}$, h^{BLIND}), and it also shows that, in general, IDFS ($\mathcal{F}_{\min}$, h^{MAX}) is much faster even considering the cost of computing the heuristic function.

Figure 3a shows the planning time comparison between IDFS ($\mathcal{F}_{\min}$, h^{MAX}) and IDFS ($\mathcal{F}_{\min}$, h^{BLIND}). Overall, IDFS ($\mathcal{F}_{\min}$, h^{MAX}) outperforms IDFS ($\mathcal{F}_{\min}$, h^{BLIND}) with respect to planning time among most planning tasks, especially over the NEW-FOND benchmarks (blue diamond in Figure 3a). Thus, we conclude that, in general, IDFS benefits from using the information of the heuristic function.

IDFS vs. IDFS Pruning We now evaluate our IDFS algorithm using h^{ADD} with $\mathcal{F}_{\max}$ and $\mathcal{F}_{\min}$. We also compare the versions of IDFS with and without pruning. Tables 2 and 3 show the results the four variations of IDFS with h^{ADD}. Note

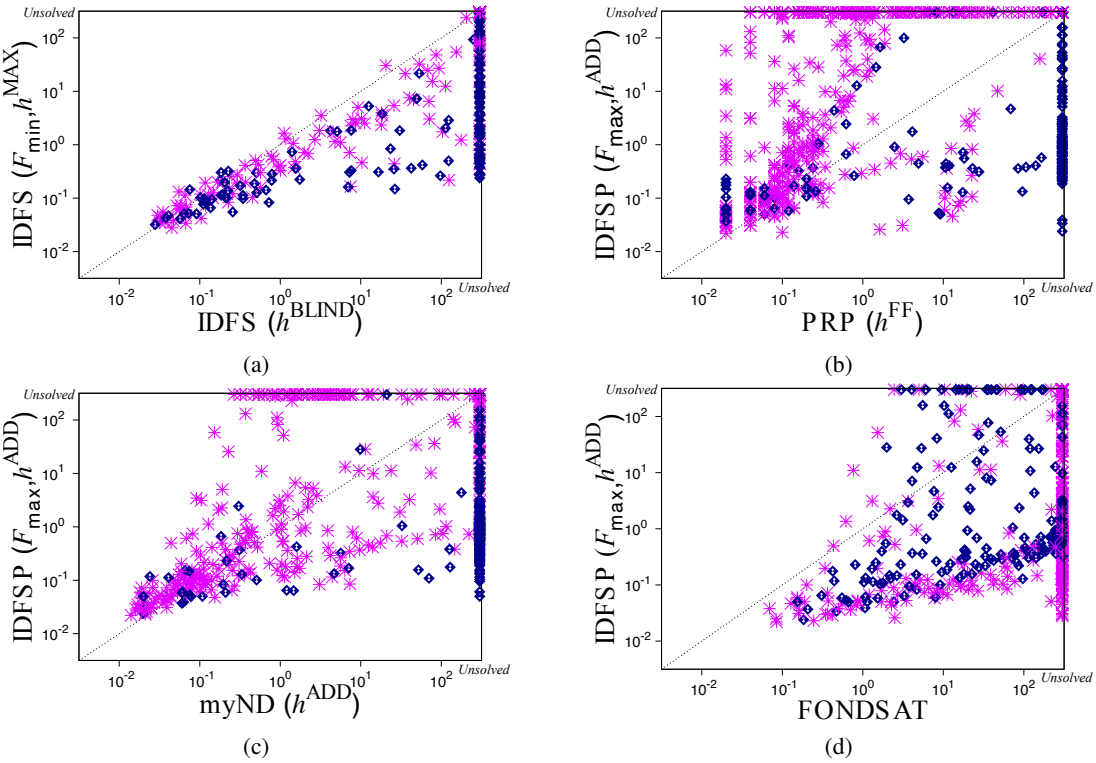


Figure 3: Planning time (in seconds) per planning task. Blue diamonds: results for the NEW-FOND benchmark set; Dark-pink asterisks: results for the IPC-FOND benchmark set.

Planner	*Solved Tasks (#590)*
PALADINUS IDFSP ($\mathcal{F}_{\min}$, h^{MAX})	337
PALADINUS IDFSP ($\mathcal{F}_{\min}$, h^{FF})	406
PALADINUS IDFSP ($\mathcal{F}_{\min}$, h^{ADD})	411
PALADINUS IDFSP ($\mathcal{F}_{\max}$, h^{MAX})	334
PALADINUS IDFSP ($\mathcal{F}_{\max}$, h^{FF})	380
PALADINUS IDFSP ($\mathcal{F}_{\max}$, h^{ADD})	**422**
FONDSAT	276
PRP (h^{MAX})	292
PRP (h^{FF})	412
PRP (h^{ADD})	389
MYND (h^{MAX})	180
MYND (h^{FF})	265
MYND (h^{ADD})	289

Table 4: Overall coverage results.

that all four variations of IDFS with h^{ADD} solved more tasks than both IDFS ($\mathcal{F}_{\min}$, h^{MAX}) and IDFS ($\mathcal{F}_{\min}$, h^{BLIND}). Such empirical results show that using a more informative heuristic has a significant impact on the results. IDFS ($\mathcal{F}_{\min}$, h^{ADD}) solved 80 tasks more than IDFS ($\mathcal{F}_{\min}$, h^{MAX}). IDFSP ($\mathcal{F}_{\max}$, h^{ADD}) outperforms the other variants in terms of coverage and planning time. However, the average final bound b_F, and the average number of iterations i for the intersection of the solved tasks are higher for variations with $\mathcal{F}_{\max}$ compared to the variations with $\mathcal{F}_{\min}$. Also, the pruning variants (IDFSP) are far superior to the variants without pruning.

Comparison with other FOND Planners Finally, we conclude our evaluation by comparing the best variation of our algorithm (IDFSP ($\mathcal{F}_{\max}$, h^{ADD})) with the state-of-the-art in FOND planning, i.e., the PRP, MYND, and FONDSAT planners. Table 4 shows the coverage results of IDFSP with both $\mathcal{F}_{\min}$ and $\mathcal{F}_{\max}$ using using different heuristic functions (h^{MAX}, h^{FF}, and h^{ADD}) against the other FOND planners over both benchmark sets. Note that IDFSP solved more tasks than the other planners. Namely, by comparing IDFSP with PRP and MYND, note that IDFSP with h^{ADD} outperforms PRP and MYND (in terms of solved tasks) with any of the three used heuristics.

Table 5 shows a detailed comparison between the best-evaluated variation of our algorithm against the best-evaluated variations of PRP, MYND, and FONDSAT. IDFSP ($\mathcal{F}_{\max}$, h^{ADD}) outperforms all the other FOND planners in terms of solved tasks and planning time. Our best algorithm performed better than PRP and MYND over the NEW-FOND benchmarks. FONDSAT also performed well for solving FOND planning tasks over the NEW-FOND benchmarks, as Geffner and Geffner (2018) have shown. When comparing the FOND planners in terms of policy size ($|\pi|$), on average, FONDSAT is the planner that returns more compact policies. However, we note that our algorithm and MYND do not compact the policies using partial states, whereas PRP and FONDSAT do. Apart from some tasks for DOORS, FAULTS, and FIRST-RESP, IDFSP ($\mathcal{F}_{\max}$, h^{ADD}) has returned policies that are as compact as the ones returned by PRP and FONDSAT, see $|\pi|$ in Table 5.

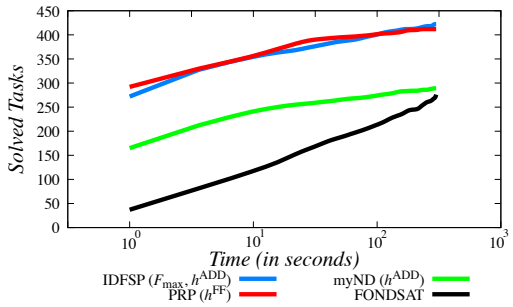
Figures 3b, 3c, and 3d show a comparison among the FOND

	IDFSP ($\mathcal{F}_{\max}$, h^{ADD})			PRP (h^{FF})			MYND (h^{ADD})			FONDSAT		
Domain (#)	C	T	$\lvert\pi\rvert$	C	T	$\lvert\pi\rvert$	C	T	$\lvert\pi\rvert$	C	T	$\lvert\pi\rvert$
DOORS (#15)	13	0.34	670.0	12	0.13	16.0	9	6.77	670.0	10	23.48	16.0
ISLANDS (#60)	60	0.10	6.5	27	0.08	7.5	12	11.06	6.83	46	4.38	7.5
MINER (#51)	51	-	-	9	-	-	0	-	-	28	-	-
TW-SPIKY (#11)	10	0.13	25.0	1	17.4	23.0	1	0.33	25.0	3	97.07	23.0
TW-TRUCK (#74)	44	2.97	21.27	17	20.34	19.36	12	12.94	13.82	67	4.51	12.18
Sub-Total (#211)	178	0.88	180.69	66	9.49	16.47	36	7.77	178.91	154	32.36	14.67
ACROBATICS (#8)	8	0.05	8.33	8	9.43	9.33	8	0.02	8.33	3	3.04	9.33
BEAM-WALK (#11)	11	0.02	11.0	11	0.86	12.0	10	0.02	11.0	2	1.37	12.0
BW-ORIG (#30)	29	0.10	12.2	30	0.06	11.7	15	0.10	11.6	10	15.02	11.1
BW-2 (#15)	15	0.12	13.2	15	0.08	14.4	6	0.23	17.6	5	24.71	12.2
BW-NEW (#40)	21	0.08	8.33	40	0.05	7.83	9	0.08	8.5	6	14.85	7.5
CHAIN (#10)	10	0.05	27.0	10	0.1	28.0	10	0.07	27.0	1	218.39	28.0
EARTH-OBS (#40)	25	-	-	40	-	-	25	-	-	0	-	-
ELEVATORS (#15)	8	0.07	19.43	15	0.05	17.71	10	1.11	18.57	7	19.01	15.86
FAULTS (#55)	55	0.14	120.66	55	0.06	11.48	53	0.95	67.55	29	38.05	11.48
FIRST-RESP (#100)	46	34.68	103.16	75	0.62	10.22	58	8.65	10.95	44	27.86	9.57
TRI-TW (#40)	8	0.08	22.0	32	0.1	23.0	40	0.04	34.0	3	51.42	16.0
ZENO (#15)	8	1.01	27.0	15	0.13	23.67	5	0.44	22.67	3	137.64	16.33
Total (#590)	422	2.38	93.12	412	2.91	13.84	289	4.77	90.17	276	45.22	13.03

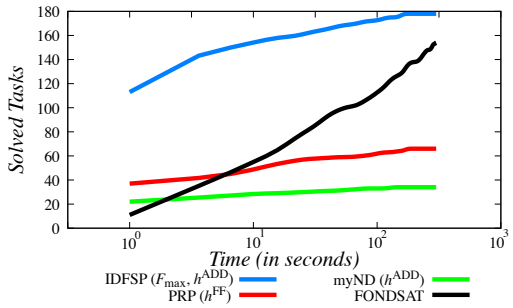
Table 5: Comparison with PRP, MYND, and FONDSAT.

planners with respect to planning time over all planning tasks for both benchmark sets. Planning tasks that *timed out* are at the limit of x-axis and y-axis (300 seconds). Figure 3b shows that our algorithm is slower than PRP for a substantial number of tasks, but PRP *timed out* for more tasks (most for the NEW-FOND benchmark set shown as blue diamond). When comparing our algorithm with MYND (Figure 3c), it is overall faster than MYND and *timed out* for fewer tasks. Figure 3d shows the planning time comparison between our algorithm and FONDSAT. Our algorithm is faster and solves more tasks than FONDSAT.

Figure 4 shows the number of solved tasks throughout the range of run-time for our algorithm (IDFSP ($\mathcal{F}_{\max}$, h^{ADD})) against PRP, MYND, and FONDSAT. When comparing the FOND planners over all benchmark sets (Figure 4a), IDFSP ($\mathcal{F}_{\max}$, h^{ADD}) has more solved tasks than MYND and FONDSAT throughout all the range of run-time and is competitive with PRP. Our algorithm (light-blue line) surpasses PRP (red line) in terms of solved tasks after ≈ 200 seconds of planning time. Over the NEW-FOND benchmark set, Figure 4a shows that our algorithm IDFSP ($\mathcal{F}_{\max}$, h^{ADD}) outperforms all the other FOND planners throughout all the range of run-time.



(a) All benchmarks.



(b) NEW-FOND benchmarks.

Figure 4: Solved tasks throughout the range of run-time.

Conclusions

We have developed a novel *iterative depth-first search* algorithm that efficiently solves FOND planning tasks. It considers more explicitly the non-determinism aspect of FOND planning, and uses heuristic functions to guide the searching process. We empirically show that our algorithm can outperform existing planners concerning planning time and coverage.

As future work, we intend to investigate how to use the information gathered during previous iterations to make the following iterations of the searching more efficient. We also aim to investigate how to design more informed heuristic functions for FOND planning. We aim to study the problem of designing algorithms to extract *dual policy* solutions, when *fairness* is not a valid assumption (Camacho and McIlraith 2016; Geffner and Geffner 2018; Rodriguez et al. 2021). We also aim to investigate how to design domains and FOND planning tasks that better capture the most significant characteristics of FOND planning. These domains and tasks can be used to evaluate new planners.

Acknowledgments

André acknowledges support from FAPERGS with projects 17/2551-0000867-7 and 21/2551-0000741-9, and Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior (CAPES), Brazil, Finance Code 001. Frederico acknowledges UFRGS, CNPq and FAPERGS for partially funding his research. Ramon and Giuseppe acknowledge support from the ERC Advanced Grant WhiteMech (No. 834228) and the EU ICT-48 2020 project TAILOR (No. 952215). Giuseppe also acknowledges the JPMorgan AI Research Award 2021.

References

Bertsekas, D. P.; and Tsitsiklis, J. N. 1991. An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research*, 16(3).

Bonet, B.; De Giacomo, G.; Geffner, H.; Patrizi, F.; and Rubin, S. 2020. High-level Programming via Generalized Planning and LTL Synthesis. In *KR*.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129: 5–33.

Bonet, B.; Giacomo, G. D.; Geffner, H.; and Rubin, S. 2017. Generalized Planning: Non-Deterministic Abstractions and Trajectory Constraints. In *IJCAI*.

Brafman, R.; and De Giacomo, G. 2019. Planning for LTLf/LDLf goals in non-markovian fully observable nondeterministic domains. In *IJCAI*.

Bryce, D.; and Buffet, O. 2008. 6th International Planning Competition: Uncertainty Part. *International Planning Competition (IPC)*.

Camacho, A.; Baier, J.; Muise, C.; and McIlraith, S. 2018. Finite LTL Synthesis as Planning. In *ICAPS*.

Camacho, A.; and McIlraith, S. A. 2016. Strong-Cyclic Planning when Fairness is Not a Valid Assumption. In *IJCAI Workshop on Knowledge-Based techniques for Problem Solving*.

Camacho, A.; and McIlraith, S. A. 2019. Strong Fully Observable Non-Deterministic Planning with LTL and LTLf Goals. In *IJCAI*.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J.; and McIlraith, S. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. In *AAAI*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147(1-2).

De Giacomo, G.; and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals. In *IJCAI*.

Fu, J.; Ng, V.; Bastani, F. B.; and Yen, I. 2011. Simple and Fast Strong Cyclic Planning for Fully-Observable Nondeterministic Planning Problems. In *IJCAI*.

Geffner, T.; and Geffner, H. 2018. Compact Policies for Fully Observable Non-Deterministic Planning as SAT. In *ICAPS*.

Hansen, E. A.; and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2): 35–62.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Hu, Y.; and Giacomo, G. D. 2011. Generalized Planning: Synthesizing Plans that Work for Multiple Environments. In *IJCAI*.

Kissmann, P.; and Edelkamp, S. 2009. Solving Fully-Observable Non-Deterministic Planning Problems via Translation into a General Game. In *KI Advances in AI*, volume 5803, 1–8.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27(1): 97–109.

Kuter, U.; Nau, D. S.; Reisner, E.; and Goldman, R. P. 2008. Using Classical Planners to Solve Nondeterministic Planning Problems. In *ICAPS*.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern Database Heuristics for Fully Observable Nondeterministic Planning. In *ICAPS*.

Mattmüller, R. 2013. *Informed Progression Search for Fully Observable Nondeterministic Planning*. Ph.D. thesis, Albert-Ludwigs-Universität Freiburg.

Muise, C.; Belle, V.; and McIlraith, S. A. 2014. Computing Contingent Plans via Fully Observable Non-Deterministic Planning. In *AAAI*.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-deterministic Planning by Exploiting State Relevance. In *ICAPS*.

Muise, C.; McIlraith, S. A.; and Belle, V. 2014. Non-Deterministic Planning With Conditional Effects. In *ICAPS*.

Patrizi, F.; Lipovetzky, N.; and Geffner, H. 2013. Fair LTL Synthesis for Non-Deterministic Systems using Strong Cyclic Planners. In *IJCAI*.

Pereira, R. F.; Pereira, A. G.; Messa, F.; and Giacomo, G. D. 2022. Iterative Depth-First Search for Fully Observable Non-Deterministic Planning. *arXiv preprint*.

Ramírez, M.; and Sardiña, S. 2014. Directed Fixed-Point Regression-Based Planning for Non-Deterministic Domains. In *ICAPS*.

Rodriguez, I. D.; Bonet, B.; Sardiña, S.; and Geffner, H. 2021. Flexible FOND Planning with Explicit Fairness Assumptions. In *ICAPS*.

Winterer, D.; Alkhazraji, Y.; Katz, M.; and Wehrle, M. 2017. Stubborn Sets for Fully Observable Nondeterministic Planning. In *ICAPS*.

Winterer, D.; Wehrle, M.; and Katz, M. 2016. Structural Symmetries for Fully Observable Nondeterministic Planning. In *IJCAI*.