Scaling-Up LAO* in FOND Planning: An Ablation Study

Ramon Fraga Pereira

University of Manchester, UK
ramon.fragapereira@manchester.ac.uk

Abstract.

The use of multi-queue heuristic search and tie-breaking strategies has shown to be very effective for *satisficing* planning in the *Classical Planning* setting. However, to the best of our knowledge, the use of such techniques has never been studied and employed in heuristic search algorithms for Fully Observable Non-Deterministic (FOND) Planning. In this paper, we adapt existing satisficing techniques for scaling-up an AND/OR heuristic search algorithm for FOND Planning. Namely, we employ multi-queue heuristic search, dead-end detection, and tie-breaking strategies in LAO* for improving the extraction of strong-cyclic policies. We assess the efficiency of our techniques in LAO* through an extensive ablation study over two different FOND Planning benchmarks. Empirical results show that our techniques improve the performance of LAO* in terms of coverage, expanded nodes, and planning time compared to a well-known planner based on vanilla LAO*. Indeed, the best configuration of our techniques is competitive with the current state-of-the-art in FOND Planning.

1 Introduction

Fully Observable Non-Deterministic (FOND) planning is a very important class of planning domain models that aims to handle the uncertainty of the execution of plans [16]. In FOND planning, states are fully observable and actions may have non-deterministic effects (i.e., a set of possible successor states). FOND planning is relevant for solving other related planning problems, such as *stochastic planning* (stochastic shortest path problems (SSP), MDP, etc) [5], *planning for temporally extended goals* [40, 14, 15, 13, 18, 11], and *generalised planning* [29, 10, 6], *qualitative numeric planning* [8]. As formally characterized by Cimatti et al. [16], there are three distinct types of FOND planning tasks and solutions: *weak planning*, in which solutions (i.e., simple plans) have a chance to achieve the goal state; *strong planning*, in which solution policies guarantee to achieve the goal state in a finite number of steps by never visiting the same state twice; and *strong cyclic planning*, in which solution policies achieve the goal state and every reachable state can reach the goal using this policy.

One of the first algorithms to solve FOND planning problems was developed by Cimatti et al. [16], a *model-checking* planner based on Binary Decision Diagrams (BDDs) called MBP. Kissmann and Edelkamp [30] also developed a FOND planner based on BDDs, but more efficient than MBP. Other FOND planning algorithms in the literature rely on *Classical* planners, such as NDP [33], FIP [20], and PRP [39]. The latter (PRP) is one of the most efficient FOND planners in the literature. Heuristic search-based algorithms for FOND planning employ the use of AND/OR graphs that represent the semantics of non-deterministic transition systems [35, 24] to solve FOND planning problems (AO* for strong planning, and LAO* for strong cyclic planning), such as MYND [36] and GRENDEL [42]. Geffner and Geffner [22] developed FONDSAT, an iterative SAT-based FOND planner that is capable to extract very compact (strong and strong cyclic) solutions for FOND planning. Most recently, Pereira et al. [41] have developed iterative depth-first search algorithms akin to IDA* [32] that have shown to be very effective for solving FOND planning tasks.

Multi-queue heuristic search (along with alternation methods) and tie-breaking strategies have shown to be very effective for *satisficing* planning in the *Classical Planning* setting [45, 44]. However, to the best of our knowledge, the use of such techniques has never been studied and employed in heuristic search algorithms for FOND Planning. In this paper, we adapt existing satisficing techniques for scaling-up an AND/OR heuristic search algorithm for FOND Planning, i.e., we employ such techniques in an LAO* variant [24] for FOND planning introduced by Mattmüller in [37]. Namely, we employ multi-queue heuristic search with alternation queues, dead-end detection and avoidance, and (simple) tie-breaking strategies [2, 1, 17] in LAO* for improving the extraction of strong-cyclic policies in FOND planning.

We assess the efficiency of our techniques in LAO* through an extensive ablation study over two different FOND Planning benchmarks: a benchmark set from the 6th International Planning Competition (IPC) [12] and [39]; and a benchmark set that contains a set of more interesting and complex FOND planning tasks, proposed and used by Geffner and Geffner in [22]. Our empirical results show that the techniques we developed and adapted from other planning settings can improve the performance of LAO* in terms of coverage, expanded nodes, and planning time compared to a well-known planner based on vanilla LAO* (MYND [36]). Indeed, the best configuration of our techniques yields a FOND planner that is competitive with the current state-of-the-art planner in FOND planning.

2 Background and Related Work

2.1 FOND *Planning*

We define *Fully Observable Non-Deterministic* (FOND) planning tasks by adopting the SAS^+ formalism for non-deterministic planning [36, 39]. Formally, a FOND planning task is a tuple $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A} \rangle$, where $\mathcal{V}$ is a set of *state variables*, and each variable $v \in \mathcal{V}$ has a finite domain D_v. A *partial state* is a function s on a subset $\mathcal{V}_s$ of $\mathcal{V}$, such that $s[v] \in \mathrm{D}_v$ if $v \in \mathcal{V}_s$, and $s[v] = \perp$ otherwise. If $\mathcal{V}_s = \mathcal{V}$, then s is a *complete state* (or simply "state"). s_0 is a complete state representing the *initial* state, whereas s_* is a partial state representing the *goal condition*. A complete 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 $a = \langle \mathrm{PRE}, \mathrm{EFF} \rangle$, where $\mathrm{PRE}(a)$ is the *preconditions* of a, and $\mathrm{EFF}(a)$ is a set of non-deterministic *effects* of

a. $\text{PRE}(a)$ is a partial state *condition* under which a may be executed, and $\text{EFF}(a)$ is the possible outcomes of a. A non-deterministic *action* $a \in \mathcal{A}$ is applicable in a state s iff $s \vDash \text{PRE}(a)$. The *application* of a non-deterministic effect $eff_a \in \text{EFF}(a)$ to a state s is a resulting state $s' = \delta(s, eff_a)$ (δ is a transition function), and the application of $\text{EFF}(a)$ to a state s is a set of successor states $\text{SUCC}(s, \text{EFF}(a)) = \{\delta(s, eff_a) \mid eff_a \in \text{EFF}(a)\}$.

A solution to a FOND planning task Π is called *policy* [21]. A policy is usually denoted as π, and we formally define a policy π as a partial function $\pi : \mathcal{S} \mapsto \mathcal{A} \cup \{\bot\}$, where π maps states $\mathcal{S}$ of a FOND planning task Π into applicable actions that eventually achieve the goal state s_* from the initial state s_0. Cimatti el al. [16] define three types of solutions to FOND planning, as follows. A *weak solution* is a policy π that is a simple sequence (path) of actions that achieves a goal state from the initial state s_0 under at least one selection of action outcomes, namely, such solution will have some chance of achieving a goal state. A *strong solution* is a policy π that is guaranteed to achieve a goal state from the initial state s_0 regardless of the non-determinism of the environment. This type of solution guarantees to achieve a goal state in a finite number of steps while never visiting the same state twice. A *strong-cyclic solution* is a policy π that guarantees to achieve a goal state from the initial state s_0 only under the assumption of *fairness*. The fairness assumption defines that all action outcomes in a given state will occur infinitely often [16] and may revisit states, so it is not guaranteed to achieve a goal state in a fixed number of steps.

2.2 Semantics of AND/OR Graphs in **FOND** Planning

AND/OR graphs can represent the semantics of non-deterministic transition systems [35, 24]. As defined by [36], the semantics of a FOND planning task Π can be specified as AND/OR graphs over the set of states $\mathcal{S}$ and the non-deterministic actions $\mathcal{A}$, where an AND/OR graph $\mathcal{G} = \langle N, C \rangle$ comprises a set of *nodes* N, and a set of *connectors* C, such that a connector c is a pair $\langle n, M \rangle$ that connects a *parent* node $n \in N$ to a non-empty set of children $M \subseteq N$. We denote a connector $c = \langle n, M \rangle$ as a non-deterministic "node transition" (or simply "transition"), using the notation $c = n \rightarrow M$. $\mathcal{G}$ contains a unique *initial node* $n_0 \in N$ and a set of *goal nodes* $N_* \in N$. Nodes in an AND/OR graph $\mathcal{G}$ can have associated costs with respect to their estimated minimum distance to any goal node and the cost of the best path so far from n_0. We denote these two costs, respectively, as *estimated cost* $h(n)$ (computed by a heuristic function h), and *accumulated cost* $g(n)$, where n is a node $n \in N$ [21].

We define *closedness*, *properness*, and *acyclicity* of an AND/OR graph as follows [37]: $\mathcal{G}$ is *closed*, if for all non-goal nodes $n \in N \setminus N_*$, there is exactly one outgoing connector $c = \langle n, M \rangle \in C$; $\mathcal{G}$ is *proper*, if for all non-goal nodes $n \in N \setminus N_*$, there is a path in $\mathcal{G}$ from n to a goal node $n_* \in N_*$; and $\mathcal{G}$ is *acyclic*, if it contains no path from n to n with positive length for any $n \in N$. An AND/OR graph induced by Π is a tuple $\mathcal{G}_\Pi = \langle N, C \rangle$, where N is the set of states $\mathcal{S}$, and C is the set of actions applications. A transition $c = n \rightarrow M = \langle n, M \rangle \in C$ is analogous to $\langle s, \text{SUCC}(s, \text{EFF}(a)) \rangle$, such that a is an applicable action in s that non-deterministically leads to a set of successor states $\text{SUCC}(s, \text{EFF}(a))$ of s. A sub-graph $\mathcal{G}' = \langle N', C' \rangle$ of $\mathcal{G}$ (denoted $\mathcal{G}' \subseteq \mathcal{G}$) is an AND/OR graph with $N' \subseteq N$, initial node n'_0 equals to the initial node n_0 of $\mathcal{G}$, and a set of goal nodes $N'_* = N_* \cap N'$. A sub-graph $\mathcal{G}'_\Pi$ of the AND/OR graph $\mathcal{G}_\Pi$ induced by a FOND planning task Π can be seen as a *solution* (i.e., a policy π) to Π, and $\mathcal{G}'_\Pi$ is a strong-cyclic solution if it is closed and proper. A strong-cyclic solution is a strong solution if it is acyclic, whereas a sub-graph is a weak solution if it contains a path from n_0 to a goal node $n_* \in N_*$.

2.3 Heuristics for **FOND** Planning

Mattmüller [37] shows that *delete-relaxation* heuristics can be effectively used in FOND planning by "simplifying" and "relaxing" the non-determinism of the actions in a FOND planning task. Such simplification and relaxation are known as *determinisation*, and it has been used by several FOND planners in the literature, such as PRP [39] and FIP [20]. Formally, a determinisation of a FOND planning task Π yields a deterministic FOND planning task $\Pi^{\text{DET}} = \langle \mathcal{V}, s_0, s_*, \mathcal{A}^{\text{DET}} \rangle$, where $\mathcal{A}^{\text{DET}}$ is a simple modification of $\mathcal{A}$ such that every action in $\mathcal{A}^{\text{DET}}$ is deterministic. Namely, $\mathcal{A}^{\text{DET}}$ is formally created by an *all-outcomes determinisation* that creates a new (deterministic) action for every outcome in EFF of all non-deterministic actions in $\mathcal{A}$.

By combining this determinisation process of a FOND planning task with delete-relaxation (ignoring negative effects of all actions in $\mathcal{A}^{\text{DET}}$) it is possible to use any off-the-shelf delete-relaxation heuristic [9, 27] from the literature for computing approximate goal distances in FOND planning. Other types of heuristics for FOND planning have been proposed in the literature, such as *pattern-database* heuristics [36] and heuristics based on *symmetry-reduction* [46]. We evaluate our techniques in LAO* using determinised delete-relaxation heuristics [9, 27] for FOND planning, and the *pattern-database* heuristic of [36].

2.4 **FOND** Planners

To the best of our knowledge, MBP (Model-Based Planner) is one of the first FOND planners in the literature, developed by Cimatti et al. [16]. MBP solves FOND planning tasks via *model-checking*, and it is built upon Binary Decision Diagrams (BDDs). GAMER [30], the winner of the FOND track at IPC [12], is also based on BDDs, but GAMER has shown to be much more efficient than MBP. NDP [33] makes use of off-the-shelf *Classical* planners to solve FOND planning tasks. MYND [36] is a FOND planner that performs heuristic-search on AND/OR graphs to extract either strong (using AO*) or strong-cyclic solutions (using LAO*) from FOND planning tasks. FIP [20] works similarly to NDP, though it is more efficient by avoiding exploring already explored/solved states. PRP [39] is one of the most efficient FOND planners in the literature, and it works as a "re-planner" and it is built upon improvements over the state relevance techniques (avoiding dead-ends states, etc) proposed by FIP. GRENDEL [42] is a FOND planner that combines regression with a symbolic fixed-point computation for extracting strong-cyclic solutions. Geffner and Geffner [22] developed FONDSAT, an iterative SAT-based FOND planner that is capable to extract very compact (strong and strong-cyclic) solutions for FOND planning tasks. FONDSAT is the only FOND planner that is apt to solve *dual* FOND planning tasks, namely, FOND planning tasks in which some non-deterministic actions are assumed to be fair (e.g., probabilistic) and others unfair (e.g., adversarial). Pereira et al. [41] developed a set of iterative depth-first search algorithms in a planner called PALADINUS, and they have shown that such algorithms are very effective for solving more complex FOND planning tasks, resulting in one of the current state-of-the-art algorithms for FOND planning. Recently, Messa and Pereira [38] introduced novel heuristics along with a *Best-First Search* algorithm for FOND planning called AND*.

3 LAO* in FOND Planning

LAO* is an AND/OR graph heuristic search algorithm originally developed by Hansen and Zilberstein in [23, 24] for extracting cyclic solution policies in MDP problems. LAO* generalises AO* [35] to extract cyclic solution policies (i.e., policies with loops) in MDP

problems without exploring the entire state-space. Mattmüller [37] has adapted AO* and LAO* for extracting both strong and strong-cycling policies in FOND planning, and this is the focus of our paper.

Algorithm 1 outlines a pseudocode based on the LAO* algorithm developed by Mattmüller in [37]. In order to clearly show how our techniques improve the performance of LAO*, we describe LAO* for FOND planning in two different stages: *Expansion Stage* and *Cost Revision and Labelling Stage*.

Given an AND/OR graph $\mathcal{G}_\Pi$ induced by a FOND planning task Π, LAO* starts the searching process from the initial node n_0, and gradually, in each step, the algorithm extracts a partial solution $\mathcal{G}'_\Pi$ by tracing the most promising connectors and unexpanded non-goal nodes in $\mathcal{G}'_\Pi$ (Line 4, using the function TRACEMARKEDCONNECTORSUNODES[1]), and after that, it returns a set of candidates non-goal nodes to be expanded, denoted as Z, and then it proceeds to expand one or more of the unexpanded non-goal nodes in Z (Lines 7–12) using the function APPLICABLETRANSITIONS to compute the applicable transitions $\mathcal{T}_n$ for every $n \in Z$. The applicable transitions computed by APPLICABLETRANSITIONS represent new nodes and connectors that are added as part of the AND/OR graph that the algorithm builds during the searching process from a node n. The algorithm evaluates nodes using an estimated evaluation cost function f (Line 12) that relies on a heuristic function h. We refer to this part in Algorithm 1 as *Expansion Stage* (Lines 4–12).

After tracing down connectors and the nodes to be expanded (*Expansion Stage*), the LAO* algorithm outlined in Algorithm 1 starts the labelling process (Line 13—SOLVELABELLING) in $\mathcal{G}'_\Pi$ and performs a cost revision (Line 14) on the heuristic estimates of the nodes in Z by marking the best outgoing connectors on the nodes in Z based on the information from the last expansion in $\mathcal{G}'_\Pi$ via dynamic programming, using either *Policy Iteration* [28, 4] or *Value Iteration* [4]. In essence, the SOLVELABELLING function marks nodes and prunes the state-space. Connectors with *solved* nodes do not need to be traced during the searching process, and a node is marked as *solved* if it is a goal node or if there is an applicable transition that may achieve a *solved* node. The use of a *dynamic programming* algorithm allows LAO* to update and propagate node estimates in order to find solutions with loops (strong-cyclic policies), as formally introduced by Hansen and Zilberstein in [23, 24]. Here, we make use of *Value Iteration* (Line 14) as a dynamic programming algorithm for performing *cost revision*. We refer to this part in Algorithm 1 as *Cost Revision and Labelling Stage* (Lines 13–14).

We can see that the LAO* algorithm outlined in Algorithm 1 terminates either when the initial node n_0 is *solved* during the searching process, returning a strong-cyclic policy π (Line 16), or if there are no unexpanded non-goal nodes in $\mathcal{G}'_\Pi$, and the initial node n_0 is still marked as *unsolved*, the algorithm then returns NO SOLUTION FOUND (Line 18).

4 Scaling-Up LAO* in FOND Planning

We now adapt techniques from other planning settings (*satisficing* planning, etc) for scaling-up LAO* in FOND planning. Namely, we employ *multi-queue heuristic search*, *tie-breaking* strategies, and *dead-end detection* as part of the *Expansion Stage* in LAO*, and develop a new function for mapping connectors to node estimates to be used in the *Cost Revision and Labelling Stage*.

[1] T represents a function that traces down the most promising connectors in $\mathcal{G}'_\Pi$ from n_0, and is used in Algorithm 1 in Lines 4, 14, and 16.

Algorithm 1: LAO*($\mathcal{G}_\Pi$)

```
1  G'_Π ≔ ⟨N', C'⟩, N' ≔ {n_0}, C' ≔ {}
2  f(n_0) = h(n_0) and mark n_0 as solved if n_0 ∈ N_* or h(n_0) = 0
3  while n_0 unsolved do
       /* Expansion Stage                                              */
4      Z ≔ TRACEMARKEDCONNECTORSUNODES(G_Π, T(G'_Π))
5      if Z = ∅ then Z ≔ {n ∈ N' | n ∉ N_* and n is unexpanded}
6      if Z = ∅ then break
7      for n ∈ Z do
8          T_n ≔ APPLICABLETRANSITIONS(n)
9          for (c = n → M) ∈ T_n do
10             add M to N' and c to C'
11             for m ∈ M do
12                 f(m) ≔ { 0,    if m ∈ N_*
                          { h(m), otherwise
       /* Cost Revision and Labelling Stage                            */
13     SOLVELABELLING(G'_Π)
14     DYNAMICPROGRAMMING(Z, G_Π, T(G'_Π))
15 if n_0 solved then
16     return T(G'_Π)
17 else
18     return NO SOLUTION FOUND
```

4.1 Multi-Queue Heuristic Search and Alternation

Multi-queue heuristic search has been widely used in planning to improve the performance of automated planners [26, 44, 45]. The central idea of using *multi-queue heuristic search* is that separate priority queues can use different heuristics to exploit their strengths in different parts of the state-space during the searching process. We maintain separate priority queues for a set of used heuristics, and states (nodes) are always evaluated with respect to all used heuristics, with their successor states (nodes) being added to all priority queues and ordered in the priority queues according to their respective heuristic. During the searching process, when selecting which state (node) will be expanded, the algorithm chooses and removes the "best" state (node) of the current queue, and it also removes the chosen state (node) of the other priority queues. The algorithm alternates between the different priority queues at every step during searching process according to some criteria, and we refer this alternation between the different priority queues as *alternation*.

We employ *multi-queue heuristic search* and *alternation* as part of the *Expansion Stage* in LAO* (Algorithm 1). Specifically, we use multiple priority queues to sort the non-goal nodes to be expanded Z returned by TRACEMARKEDCONNECTORSUNODES (Line 4). During the tracing process to trace the most promising connectors in $\mathcal{G}'_\Pi$ from n_0, we order the set of candidates non-goal nodes to be expanded according to the some criteria. The criteria we use here in this paper are the following: *minimum* f value, FIFO (*first-in-first-out*, oldest nodes first), and LIFO (*last-in-first-out*, most recent inserted nodes first). As for the *minimum* f value criterion, we order the set of candidate non-goal nodes to be expanded according to their minimum f value, much like other heuristic search algorithms in the literature, e.g., A* [25]. With respect to FIFO and LIFO, we order the nodes *first-in-first-out*—the oldest nodes first, and *last-in-first-out*—most recent inserted nodes first, respectively. We denote such priority queues as $[f]$, [FIFO], and [LIFO]. We formally denote the use of multiple priority queues as $\langle []_0, []_1, []_2, ..., []_n \rangle$. Example 1 exemplifies how we formalise multiple sets of priority queues for LAO* search.

Example 1. *Let us consider that we have the following multiple set of priority queues:* $\langle [f = h_X], [f = h_Y], [\text{FIFO}], [\text{LIFO}] \rangle$*, where the first priority queue employs the minimum* f *using a heuristic* h_X*, the second next priority queue employs the minimum* f *using a heuristic* h_Y*, and so on for the next queues* [FIFO] *and* [LIFO]*, respectively.*

Our LAO* implementation employs an *alternation* technique that alternates between the priority queues at every step during searching process, and it alternates between the priority queues as *circular queue*. Essentially, it alternates between the priority queues by selecting nodes from the first priority queue, and at next step it selects nodes from the second queue, and so on until the LAO* terminates.

4.2 Tie-Breaking

In *Planning*, *tie-breaking* refers to the process of selecting the "most promising" state (node) among possible nodes that have the same level of desirability according to an algorithm's evaluation function [2, 1, 17]. When multiple states (nodes) have the same evaluation function value, a tie-breaking strategy is then used to determine which state (node) should be chosen and selected to be expanded. The choice of an appropriate tie-breaking strategy can have a significant impact on the performance and effectiveness of a planning algorithm. Therefore, selecting an appropriate tie-breaking strategy is an important consideration in designing an effective automated planner [45].

In this paper, we employ simple (but effective) tie-breaking strategies in LAO*. In essence, we define two types of tie-breaking strategies: (1) we use a secondary heuristic to break ties when the main heuristic is not able to provide an estimate to select solely one node; and (2) we define a default tie-breaking strategy that is used when no secondary heuristic is defined as a tie-breaker, and this default strategy breaks ties using FIFO, so it gives preferences to the *oldest* nodes in the AND/OR graph.

We formally denote the use of tie-breaking strategies by following the work of Asai and Fukunaga [2], as follows. LAO* without any explicitly tie-breaking strategy with the evaluation function f (and h_X as the main heuristic) is denoted as $\langle[f = h_X]\rangle$, whereas f with h_X as the main heuristic and h_Y as secondary heuristic for tie-breaking is denoted as $\langle[f = h_X, h_Y]\rangle$. We note $\langle[f = h_X]\rangle$ means that there is no explicitly tie-breaking strategy defined, so it uses the default tie-breaker FIFO for breaking ties[2]. In Section 5, we empirically show that the use a secondary heuristic for tie-breaking can improve significantly the results of LAO* in several different FOND domain models. In Example 2, we provide an example on how we break ties when non-goal nodes to be expanded have the same f value.

Example 2. *Figure 1a illustrates a partial AND/OR graph with 5 nodes (n_0, n_1, n_2, n_3, and n_4) and 2 connectors (**a** and **b**). Every node has estimated values using a primary heuristic h and a secondary heuristic h'. Consider that the non-goal nodes to be expanded and added to the priority queue are: n_1 and n_2. We can see that they have the same estimated value using h, which is $h = 5$. For breaking ties, we use their estimated values for the secondary heuristic h', which is $h'(n_1) = 4$ and $h'(n_2) = 3$. In this case, the node n_2 is the top-priority node in the priority queue, preceding n_1, that comes next.*

4.3 Dead-End Detection and Pruning

Dead-ends in *Planning* are defined as states from which the goal state is not possible to be achieved [21, 34]. Different types of planning settings efficiently employ dead-end detection and avoidance, such as Classical Planning [34], SSP Planning [31], and FOND Planning [39]. In FOND Planning, Muise, McIlraith, and Beck in [39] have employed dead-end detection and avoidance to develop PRP.

[2] We use FIFO as a default tie-breaker because it has yielded better results not only for our planner but also for MYND [36]

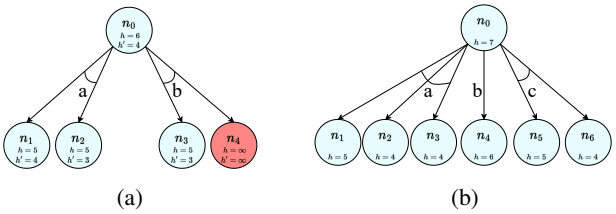


Figure 1: LAO* AND/OR sub-graph examples.

In this paper, we develop a simple but effective dead-end detection that prunes actions and nodes that lead to dead-end nodes. We discard applicable transitions $n \to M \in C$ from a node n if at least one of the child nodes $n' \in M$ have $h(n') = \infty$, i.e., we discard transitions that lead to known *dead-end* nodes. We use this dead-end detection to prune actions and nodes in LAO*.

More specifically, we use this dead-end detection and pruning in APPLICABLETRANSITIONS (Line 8 in Algorithm 1) so LAO* avoids evaluation nodes and uses connectors that lead to dead-end nodes. We denote the use of this dead-end detection and pruning when computing transitions as APPLICABLETRANSITIONS$_{\mathcal{D}^\infty}$, or simply $\mathcal{D}^\infty$. Example 3 exemplifies our dead-end detection works for pruning states and actions that lead to dead-ends.

Example 3. *Consider the partial AND/OR graph in Figure 1a. For computing the applicable transitions of n_0, APPLICABLETRANSITIONS$_{\mathcal{D}^\infty}(n_0)$ would return $\mathcal{T}_{n_0}$, which would not consider the connector **b** and its child nodes n_3 and n_4 as part of $\mathcal{T}_{n_0}$ because n_4 is a dead-end node. Thus, our function discards the entire connector and its child nodes, returning in this case the transition with the connector **a** and its children n_1 and n_2.*

4.4 Updating Functions for Cost Revision

Dynamic programming for MDP problems aims to find the evaluation function f that satisfies the Bellman formulation [3] by improving through updates the evaluation function f via dynamic programming updates in the state-space. Two dynamic programming algorithms can be used for solving MDP problems: *Policy Iteration* and *Value Iteration*. The seminal LAO* developed by Hansen and Zilberstein [24] uses dynamic programming for SSP problems In FOND planning [36, 37], LAO* uses dynamic programming to propagate heuristic estimates for a set of particular nodes (Z in Algorithm 1) and mark the best connectors for such nodes, as we stated in Section 3.

Mattmüller et al. [36] define two updating functions (Equations 1 and 2) to update the evaluation function f over the non-goal nodes Z during the dynamic programming step (Line 14 in Algorithm 1), as follows:

$$\mathcal{F}_{\text{AVG}}(n) = \begin{cases} 0 & \text{if } n \in N_* \\ 1 + \min\limits_{\langle n, M\rangle \in C} \left(\frac{1}{|M|} \sum\limits_{m \in M} h(m) \right) & \text{otherwise} \end{cases} \quad (1)$$

$$\mathcal{F}_{\text{MAX}}(n) = \begin{cases} 0 & \text{if } n \in N_* \\ 1 + \min\limits_{\langle n, M\rangle \in C} \left(\max\limits_{m \in M} h(m) \right) & \text{otherwise} \end{cases} \quad (2)$$

Equation 1 represents the (minimum) average expected value over the child nodes along the connectors of n, whereas Equation 2 represents the (minimum) maximum value ("pessimistic") over the child nodes along the connectors of n. Therefore, for every $n \in Z$ the evaluation function $f(n)$ can be updated using either $\mathcal{F}_{\text{AVG}}$ or $\mathcal{F}_{\text{MAX}}$, namely, $f(n) \coloneqq \mathcal{F}_{\text{AVG}}(n)$ or $f(n) \coloneqq \mathcal{F}_{\text{MAX}}(n)$.

As an alternative to the functions presented above, we define a new updating function that is "considerably pessimistic" by selecting the

maximum value between the heuristic value of n and the average expected value over the child nodes along the connectors of n. This pessimistic rationale of our new updating function has also been used before in [7] for MDPs and in [41] for FOND planning, inspiring the development and employment of our new updating function. Our new updating function is formalised in Equation 3.

$$\mathcal{F}_{\text{PESS}}(n) = \begin{cases} 0 & \text{if } n \in N_* \\ 1 + \min\limits_{\langle n, M\rangle \in C} \left(\max(h(n), \frac{1}{|M|} \sum\limits_{m \in M} h(m)) \right) & \text{otherwise} \end{cases} \quad (3)$$

The DYNAMICPROGRAMMING function (Line 14 in Algorithm 1) performs a cost revision for every $n \in Z$ using either *Policy Iteration* or *Value Iteration*, selecting an outgoing connector of n as the best connector that minimises $f(n)$. This is a very important step for LAO* to extract strong-cyclic policies in FOND planning. In the ablation study in Section 5, we show that the appropriate use of the updating function can improve the results of LAO* in FOND planning not only in terms of planning time but also in terms of coverage and expanded nodes. Example 4 describes an example on how the updating functions in Equations 1,2, and 3 work to update the evaluation function f.

Example 4. *Figure 1b depicts an AND/OR graph with 7 nodes ($n_0, n_1, n_2, n_3, n_4, n_5$, and n_6) and 3 connectors (**a**, **b**, and **c**). When using the updating function $\mathcal{F}_{\text{AVG}}$ for n_0, we have the following updated f value: $\mathcal{F}_{\text{AVG}}(n_0) = 4.3$, which is the average value over the child nodes of **a**, the minimum average over all connectors from n_0. The updated f value for n_0 when using $\mathcal{F}_{\text{MAX}}$ is: $\mathcal{F}_{\text{MAX}}(n_0) = 6$, which is the maximum value over the child nodes of all possible connectors, in this case, the h value of the child node (n_4) of **b**. As for using $\mathcal{F}_{\text{PESS}}$ for n_0, the updated f value is: $\mathcal{F}_{\text{PESS}}(n_0) = 7$, and in this case, f is not updated because the $h(n_0)$ is greater than the average of the child nodes of the connectors of n_0.*

5 Experiments and Evaluation

We now present an extensive set of experiments and an ablation study for evaluating the efficiency of our techniques in LAO* for solving FOND planning tasks. We implemented the techniques we present in Section 4 by modifying the LAO* implementation of MYND [36], and as a result, we have a new FOND planner that we name as JADIS[3]. Apart from the ablation study, which shows how our techniques affect the efficiency of LAO*, we also compare our best planner configuration with state-of-the-art FOND planners in the literature, such as PRP [39], MYND [36], FONDSAT [22], and PALADINUS [41].

5.1 Benchmarks and Setup

We empirically evaluate JADIS using two distinct FOND planning benchmark sets. One of the benchmark sets, denoted as IPC-FOND, contains 379 planning tasks over 12 FOND planning domains from the IPC [12] and [39], such as ACROBATICS, BEAM-WALK, BLOCKS-WORLD, FAULTS, FIRST-RESP, TRIANGLE-TIREWORLD. The other benchmark set was proposed and used in [22], denoted as NEW-FOND, which includes FOND planning domains that are more challenging and complex than IPC-FOND. NEW-FOND contains 211 planning tasks over 5 FOND planning domains, such as introduced (DOORS, ISLANDS, MINER, TW-SPIKY, and TW-TRUCK). We note that 25 out of 590 planning tasks have no solution, namely, 25 tasks of FIRST-RESP (a domain part of the IPC-FOND benchmarks).

We have run all experiments using a single-core of a 24-core Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.0GHz with 32GB of RAM, with a memory usage limit of 4GB, and set a 5-minute (300 seconds) time-out per planning task. We use the following metrics: *number of solved tasks – Coverage* (C), *planning time – Time* (T) in seconds, *average policy size* ($|\pi|$). In the supplementary material, we provide a more detailed evaluation of how our techniques affect the performance of LAO*, such as detailed results for each planning task in the benchmarks (the *number of expanded nodes*, etc), and the performance of the other FOND planners when using different heuristics.

As heuristic functions, we use three different heuristics from the FOND planning literature: we use two determinised delete-relaxation heuristics [9, 27] for FOND planning [37], the *additive* heuristic h^{ADD} and the *fast-forward* heuristic h^{FF}; and the *pattern-database* heuristic developed by Mattmüller et al. in [36], denoted as h^{PDBS}.

5.2 Ablation Study

We now present the ablation study to evaluate the impact[4] of our techniques (presented in Section 4) in LAO* for FOND planning. The idea of this ablation study is to "play" with our techniques and different heuristics in LAO* by adding/removing them in order to access their impact on the LAO* performance in FOND planning. We have tried all combinations of our techniques, but we report here only the best results as they provide a clear enough picture of the overall performance, and leave such an extensive evaluation for future work.

Single-Queue Search Comparison. The first part of our ablation study is analysing the use of single-queue search with the three updating functions $\mathcal{F}_{\text{AVG}}$, $\mathcal{F}_{\text{MAX}}$, and $\mathcal{F}_{\text{PESS}}$, with and without dead-end detection and pruning $\mathcal{D}^{\infty}$. Table 1 shows the results for an ablation study for single-queue search, and there are two important things that we can see from these results: (1) our dead-end detection/pruning technique $\mathcal{D}^{\infty}$ increases the coverage by solving more tasks, see the results at second-half in Table 1; and (2) it is also possible to see that the use of $\mathcal{F}_{\text{PESS}}$ improves the results when using and not using $\mathcal{D}^{\infty}$. We can also see a gradual increase in terms of coverage when using the heuristics, in which h^{PDBS} has the better results (especially for solving tasks in the NEW-FOND benchmarks, which is the "most" complex and difficult FOND planning benchmark set), then h^{FF}, and then h^{ADD}, and we will see this "pattern" over the ablation study.

Table 1: Single-queue results. *Coverage* results are reported as: (IPC-FOND *solved tasks* + NEW-FOND *solved tasks*) = *Total solved tasks*.

	Coverage	*Time*
$\langle \mathcal{F}_{\text{AVG}} \rangle$		
$\langle [f = h^{\text{ADD}}] \rangle$	(230 + 30) = 260	9.17
$\langle [f = h^{\text{FF}}] \rangle$	(225 + 31) = 256	14.00
$\langle [f = h^{\text{PDBS}}] \rangle$	(193 + 101) = 294	16.85
$\langle \mathcal{F}_{\text{MAX}} \rangle$		
$\langle [f = h^{\text{ADD}}] \rangle$	(218 + 30) = 248	7.56
$\langle [f = h^{\text{FF}}] \rangle$	(239 + 31) = 270	12.94
$\langle [f = h^{\text{PDBS}}] \rangle$	(190 + 101) = 291	16.12
$\langle \mathcal{F}_{\text{PESS}} \rangle$		
$\langle [f = h^{\text{ADD}}] \rangle$	(236 + 32) = 268	7.31
$\langle [f = h^{\text{FF}}] \rangle$	(260 + 34) = 294	6.32
$\langle [f = h^{\text{PDBS}}] \rangle$	**(205 + 100) = 305**	**19.74**
$\langle \mathcal{F}_{\text{AVG}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}] \rangle$	(223 + 53) = 276	7.97
$\langle [f = h^{\text{FF}}] \rangle$	(221 + 55) = 276	12.04
$\langle [f = h^{\text{PDBS}}] \rangle$	(185 + 133) = 318	16.15
$\langle \mathcal{F}_{\text{MAX}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}] \rangle$	(214 + 54) = 268	11.80
$\langle [f = h^{\text{FF}}] \rangle$	(236 + 57) = 293	14.27
$\langle [f = h^{\text{PDBS}}] \rangle$	(184 + 133) = 317	19.68
$\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}] \rangle$	(234 + 52) = 286	10.79
$\langle [f = h^{\text{FF}}] \rangle$	(257 + 50) = 307	10.62
$\langle [f = h^{\text{PDBS}}] \rangle$	**(195 + 133) = 328**	**18.82**

[3] JADIS source code: https://github.com/ramonpereira/jadis

[4] The best results are highlighted in **bold** in Tables 1, 2, 3, and 4.

Multi-Queue Search Comparison: f and FIFO/LIFO. The second part of our ablation study is employing multi-queue search in LAO*. Since $\mathcal{D}^{\infty}$ has shown to improve the results in terms of coverage (see Table 1), from now on, all the reported results for LAO* use $\mathcal{D}^{\infty}$. Table 2 shows the results using multi-queue search with alternation between f and FIFO/LIFO. Note that combining f and LIFO yields better results than using a single-queue search. However, we can see that the alternation between f and FIFO has better results in overall, it has a significant improvement in terms of coverage when employing multi-queue search compared to single-queue search, with approximately 12% of improvement.

Table 2: Multi-queue results with f and FIFO/LIFO, and $\mathcal{D}^{\infty}$. *Coverage* results are reported as: (IPC-FOND *solved tasks* + NEW-FOND *solved tasks*) = *Total solved tasks*.

	Coverage	*Time*
$\langle \mathcal{F}_{\text{AVG}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [\text{FIFO}] \rangle$	(251 + 59) = 310	15.93
$\langle [f = h^{\text{FF}}], [\text{FIFO}] \rangle$	(245 + 60) = 305	15.76
$\langle [f = h^{\text{PDBS}}], [\text{FIFO}] \rangle$	(207 + 136) = 343	23.93
$\langle \mathcal{F}_{\text{MAX}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [\text{FIFO}] \rangle$	(246 + 59) = 305	15.40
$\langle [f = h^{\text{FF}}], [\text{FIFO}] \rangle$	(267 + 60) = 327	16.08
$\langle [f = h^{\text{PDBS}}], [\text{FIFO}] \rangle$	(203 + 136) = 339	25.95
$\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [\text{FIFO}] \rangle$	(267 + 56) = 323	14.37
$\langle [f = h^{\text{FF}}], [\text{FIFO}] \rangle$	(283 + 56) = 339	20.34
$\langle [f = h^{\text{PDBS}}], [\text{FIFO}] \rangle$	**(231 + 137) = 368**	**29.44**
$\langle \mathcal{F}_{\text{AVG}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [\text{LIFO}] \rangle$	(249 + 57) = 306	8.85
$\langle [f = h^{\text{FF}}], [\text{LIFO}] \rangle$	(229 + 60) = 289	18.53
$\langle [f = h^{\text{PDBS}}], [\text{LIFO}] \rangle$	**(188 + 134) = 322**	**22.97**
$\langle \mathcal{F}_{\text{MAX}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [\text{LIFO}] \rangle$	(234 + 57) = 291	9.38
$\langle [f = h^{\text{FF}}], [\text{LIFO}] \rangle$	(237 + 59) = 296	14.17
$\langle [f = h^{\text{PDBS}}], [\text{LIFO}] \rangle$	(183 + 134) = 317	19.28
$\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [\text{LIFO}] \rangle$	(265 + 32) = 297	13.24
$\langle [f = h^{\text{FF}}], [\text{LIFO}] \rangle$	(255 + 33) = 288	5.38
$\langle [f = h^{\text{PDBS}}], [\text{LIFO}] \rangle$	(197 + 90) = 287	22.18

Table 3: Multi-queue results with f and FIFO/LIFO – $\mathcal{D}^{\infty}$, with a secondary heuristic for tie-breaking. *Coverage* is reported as: (IPC-FOND *solved tasks* + NEW-FOND *solved tasks*) = *Total solved tasks*.

	Coverage	*Time*
$\langle \mathcal{F}_{\text{AVG}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}, h^{\text{FF}}], [\text{FIFO}] \rangle$	(252 + 56) = 308	26.07
$\langle [f = h^{\text{FF}}, h^{\text{ADD}}], [\text{FIFO}] \rangle$	(245 + 57) = 302	18.90
$\langle [f = h^{\text{PDBS}}, h^{\text{ADD}}], [\text{FIFO}] \rangle$	(208 + 136) = 344	27.27
$\langle [f = h^{\text{PDBS}}, h^{\text{FF}}], [\text{FIFO}] \rangle$	(208 + 136) = 344	24.20
$\langle \mathcal{F}_{\text{MAX}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}, h^{\text{FF}}], [\text{FIFO}] \rangle$	(250 + 55) = 305	27.76
$\langle [f = h^{\text{FF}}, h^{\text{ADD}}], [\text{FIFO}] \rangle$	(270 + 55) = 325	16.36
$\langle [f = h^{\text{PDBS}}, h^{\text{ADD}}], [\text{FIFO}] \rangle$	(204 + 136) = 340	21.55
$\langle [f = h^{\text{PDBS}}, h^{\text{FF}}], [\text{FIFO}] \rangle$	(203 + 136) = 339	21.76
$\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}, h^{\text{FF}}], [\text{FIFO}] \rangle$	(264 + 51) = 315	21.28
$\langle [f = h^{\text{FF}}, h^{\text{ADD}}], [\text{FIFO}] \rangle$	(282 + 51) = 333	24.77
$\langle [f = h^{\text{PDBS}}, h^{\text{ADD}}], [\text{FIFO}] \rangle$	(218 + 136) = 354	29.89
$\langle [f = h^{\text{PDBS}}, h^{\text{FF}}], [\text{FIFO}] \rangle$	**(250 + 137) = 387**	**31.97**
$\langle \mathcal{F}_{\text{AVG}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}, h^{\text{FF}}], [\text{LIFO}] \rangle$	(247 + 58) = 305	9.37
$\langle [f = h^{\text{FF}}, h^{\text{ADD}}], [\text{LIFO}] \rangle$	(219 + 61) = 280	17.49
$\langle [f = h^{\text{PDBS}}, h^{\text{ADD}}], [\text{LIFO}] \rangle$	(176 + 134) = 310	19.10
$\langle [f = h^{\text{PDBS}}, h^{\text{FF}}], [\text{LIFO}] \rangle$	(179 + 133) = 312	15.63
$\langle \mathcal{F}_{\text{MAX}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}, h^{\text{FF}}], [\text{LIFO}] \rangle$	(231 + 57) = 288	11.27
$\langle [f = h^{\text{FF}}, h^{\text{ADD}}], [\text{LIFO}] \rangle$	(237 + 62) = 299	13.28
$\langle [f = h^{\text{PDBS}}, h^{\text{ADD}}], [\text{LIFO}] \rangle$	(186 + 135) = 321	14.00
$\langle [f = h^{\text{PDBS}}, h^{\text{FF}}], [\text{LIFO}] \rangle$	(181 + 134) = 315	16.58
$\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}, h^{\text{FF}}], [\text{LIFO}] \rangle$	(262 + 57) = 319	21.42
$\langle [f = h^{\text{FF}}, h^{\text{ADD}}], [\text{LIFO}] \rangle$	(251 + 56) = 307	10.61
$\langle [f = h^{\text{PDBS}}, h^{\text{ADD}}], [\text{LIFO}] \rangle$	**(209 + 133) = 342**	**15.66**
$\langle [f = h^{\text{PDBS}}, h^{\text{FF}}], [\text{LIFO}] \rangle$	(199 + 133) = 332	16.83

Multi-Queue Search Comparison: f with Tie-breaking and FIFO/LIFO. We could see in Table 2 that the use of multi-queue search can improve LAO*. We now show the results of LAO* by combining multi-queue search with tie-breaking. Table 3 shows the results of using for multi-queue search with alternation between f (with a second heuristic for breaking ties) and FIFO/LIFO. Here, we can see again a significant improvement in the LAO* performance, the use of a second heuristic for breaking ties has provided a great improvement in terms of coverage, especially by combining h^{PDBS} with h^{ADD} or h^{FF} as secondary heuristic. Multi-queue search with tie-breaking has approximately 6% of improvement compared to the previous multi-queue search results.

Table 4: Multi-queue results with two queues using f, $\mathcal{D}^{\infty}$, and no tie-breaking. *Coverage* is reported as: (IPC-FOND *solved tasks* + NEW-FOND *solved tasks*) = *Total solved tasks*.

	Coverage	*Time*
$\langle \mathcal{F}_{\text{MAX}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [f = h^{\text{PDBS}}] \rangle$	(180 + 53) = 233	17.14
$\langle [f = h^{\text{ADD}}], [f = h^{\text{FF}}] \rangle$	(224 + 51) = 275	10.64
$\langle [f = h^{\text{FF}}], [f = h^{\text{PDBS}}] \rangle$	(203 + 55) = 258	21.42
$\langle [f = h^{\text{FF}}], [f = h^{\text{ADD}}] \rangle$	(233 + 54) = 287	9.28
$\langle [f = h^{\text{PDBS}}], [f = h^{\text{ADD}}] \rangle$	(180 + 133) = 313	21.62
$\langle [f = h^{\text{PDBS}}], [f = h^{\text{FF}}] \rangle$	**(181 + 133) = 314**	**23.44**
$\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$		
$\langle [f = h^{\text{ADD}}], [f = h^{\text{PDBS}}] \rangle$	(195 + 50) = 245	35.11
$\langle [f = h^{\text{ADD}}], [f = h^{\text{FF}}] \rangle$	(231 + 52) = 283	11.09
$\langle [f = h^{\text{FF}}], [f = h^{\text{PDBS}}] \rangle$	(217 + 51) = 268	26.09
$\langle [f = h^{\text{FF}}], [f = h^{\text{ADD}}] \rangle$	(258 + 51) = 309	14.96
$\langle [f = h^{\text{PDBS}}], [f = h^{\text{ADD}}] \rangle$	(206 + 134) = 340	19.31
$\langle [f = h^{\text{PDBS}}], [f = h^{\text{FF}}] \rangle$	**(208 + 134) = 342**	**20.45**

Multi-Queue Search Comparison: Dual-queue with f. The last part of our ablation study shows results for multi-queue search by using dual-queue (alternation) with f and no tie-breaking. The idea of this study is to see how the alternation between two different heuristics can affect the results of LAO*. Table 4 shows the results of using dual-queue (alternation) with f for the updating functions $\mathcal{F}_{\text{MAX}}$ and $\mathcal{F}_{\text{PESS}}$, which are the functions that overall yield better results (in terms of coverage). Here, and as we have shown, we can see that $\mathcal{F}_{\text{PESS}}$ using h^{PDBS} with h^{ADD} or h^{FF} is the configuration that yields better results in terms of coverage balance for solving tasks in both benchmark sets (IPC-FOND and NEW-FOND). However, dual-queue with f has not shown to be better than multi-queue search f with tie-breaking and FIFO/LIFO (results of Table 3).

5.3 *Comparison with other* **FOND** *Planners*

We conclude our empirical study by comparing our best planner configuration ($\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$: $\langle [h^{\text{PDBS}}, h^{\text{FF}}], [\text{FIFO}] \rangle$) for JADIS with other FOND planners from the literature. Here, access the efficiency of our techniques in JADIS LAO* with the best configuration of the other FOND planners, such as PALADINUS (IDFS+P (MAX,h^{ADD})), MYND (h^{ADD})[5], PRP (h^{FF}), and FONDSAT.

Figure 2 shows a comparison with respect to the expanded nodes of our best LAO* configuration against the best LAO* configuration of MYND. We can see that our best configuration has expanded fewer nodes (in overall) compared to MYND, especially for the NEW-FOND benchmarks, and it empirically shows that our techniques can improve the performance of LAO* in terms of node expansion.

Table 5 shows the results over all 17 domains (for both benchmarks) of best LAO* configuration against the other FOND planners. This comparison takes into account the intersection of T and $|\pi|$ among the solved task. We can see that JADIS ($\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$: $\langle [h^{\text{PDBS}}, h^{\text{FF}}], [\text{FIFO}] \rangle$) is competitive with the state-of-art planners (PALADINUS and PRP) in FOND planning, outperforming MYND

[5] LAO* implementation of MYND uses single-queue search with no dead-detection and FIFO as a tie-breaker.

Table 5: Comparison with other FOND planners. C is coverage, T is time (in seconds), and average policy size $|\pi|$.

	JADIS $\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$ $\langle [h^{\text{PDBS}}, h^{\text{FF}}], [\text{FIFO}] \rangle$			PALADINUS IDFS+P (MAX,h^{ADD})			PRP (h^{FF})			MYND (h^{ADD})			FONDSAT		
#	C	T	$\lvert\pi\rvert$	C	T	$\lvert\pi\rvert$	C	T	$\lvert\pi\rvert$	C	T	$\lvert\pi\rvert$	C	T	$\lvert\pi\rvert$
DOORS (#15)	13	1.63	1159.14	13	0.91	1159.14	12	0.19	17.0	10	31.19	1159.14	11	40.27	17.0
ISLANDS (#60)	60	0.93	6.23	60	0.1	6.23	27	0.07	7.23	13	20.11	6.62	47	4.08	7.23
MINER (#51)	51	-	-	51	-	-	9	-	-	0	-	-	32	-	-
TW-SPIKY (#11)	1	0.84	25.0	10	0.13	25.0	1	19.55	23.0	1	0.33	25.0	4	97.07	23.0
TW-TRUCK (#74)	12	21.3	13.73	44	2.97	21.27	16	20.34	19.36	12	12.94	13.82	67	4.51	12.18
Summary (#211)	137	4.94	301.03	178	0.82	302.91	65	8.03	16.65	36	16.14	301.14	161	29.19	14.85
ACROBATICS (#8)	3	0.82	8.33	8	0.05	8.33	8	9.43	9.33	8	0.02	8.33	3	3.51	9.33
BEAM-WALK (#11)	3	0.81	11.0	11	0.02	11.0	11	0.86	12.0	10	0.02	11.0	2	1.94	12.0
BW-ORIG (#30)	16	1.01	12.6	27	0.1	12.2	30	0.06	11.7	15	0.1	11.6	10	15.12	11.1
BW-2 (#15)	7	1.66	22.8	15	0.12	13.2	15	0.08	14.4	6	0.23	17.6	5	25.17	12.2
BW-NEW (#40)	11	0.82	9.5	21	0.08	8.33	40	0.05	7.83	9	0.08	8.5	6	15.18	7.5
CHAIN (#10)	10	-	-	10	-	-	10	-	-	10	-	-	0	-	-
EARTH-OBS (#40)	34	-	-	25	-	-	40	-	-	21	-	-	0	-	-
ELEVATORS (#15)	12	1.37	17.29	8	0.07	19.43	15	0.05	17.71	10	1.11	18.57	7	20.3	15.86
FAULTS (#55)	48	19.04	82.9	55	0.14	120.66	55	0.06	11.48	53	0.95	67.55	29	38.43	11.48
FIRST-RESP (#100)	66	6.34	11.0	46	32.79	105.39	75	0.64	10.0	54	4.96	10.64	44	22.34	9.39
TRI-TW (#40)	35	0.57	25.0	8	0.08	22.0	32	0.1	23.0	40	0.04	34.0	3	52.42	16.0
ZENO (#15)	5	53.73	28.0	8	1.01	27.0	15	0.13	23.67	5	0.44	22.67	3	139.31	16.33
Summary (#379)	250	7.18	57.1	242	2.87	86.88	346	0.96	35.28	241	0.66	52.62	112	33.37	30.3
Total (#590)	387	6.52	358.13	420	2.27	389.8	411	3.04	51.93	277	4.53	353.76	273	31.98	45.15

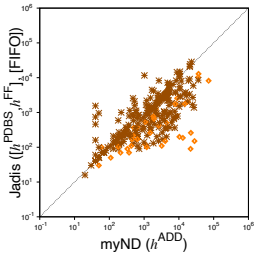


Figure 2: Expanded nodes comparison between JADIS ($\langle \mathcal{F}_{\text{PESS}}, \mathcal{D}^{\infty} \rangle$ and $\langle [h^{\text{PDBS}}, h^{\text{FF}}], [\text{FIFO}] \rangle$) and MYND ($h^{\text{ADD}}$). Orange diamonds are results for the NEW-FOND benchmark set, and brown asterisks are results for the IPC-FOND benchmark set.

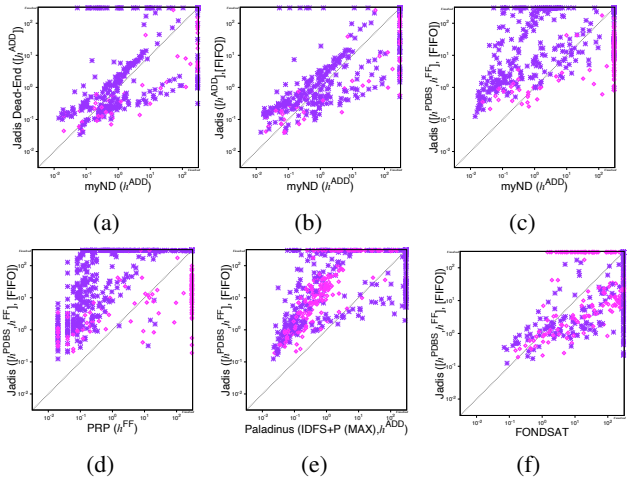
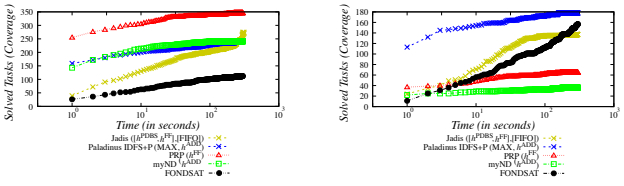


Figure 3: Planning time (in seconds) comparison per FOND planning task. Pink diamonds are results for the NEW-FOND benchmark set, and purple asterisks are results for the IPC-FOND benchmark set.



(a) IPC-FOND benchmarks. (b) NEW-FOND benchmarks.

Figure 4: Solved tasks throughout the range of planning time.

and FONDSAT in total coverage. It is also possible to see that JADIS outperforms PALADINUS in the IPC-FOND planning benchmark set.

In terms of planning time per task, our best LAO* configuration is not as fast as MYND, PRP and PALADINUS, see Figures 3c, 3d, and 3e, but it is quite faster than FONDSAT (see Figure 3f). We note that our best LAO* configuration may perform "slower" than the other planners due to the fact that it uses two heuristic functions (h^{PDBS} and h^{FF}) during the planning process. For a more fair comparison with MYND (h^{ADD}), we show a planning time comparison with our best results using h^{ADD} as the heuristic estimator, see Figures 3a and 3b. The planning time results per task empirically show that JADIS is slightly faster (much faster for NEW-FOND benchmark set) than MYND using the same heuristic function.

Figure 4 depicts the number of solved tasks throughout the range of planning run-time for our best configuration against PALADINUS, PRP, MYND, and FONDSAT. JADIS is the second best at the end of the plot for IPC-FOND (Figure 4a) As for the comparison over NEW-FOND (Figure 4b), we can see that JADIS is overall the second/third best in terms of solved tasks throughout the range of run-time.

6 Conclusions

In this paper, we have employed and adapted well-known techniques from other planning settings for scaling-up LAO* in FOND planning. We have empirically shown in an extensive ablation study that our techniques significantly improves the performance of the vanilla LAO* by scaling it up in terms of planning time, coverage, and expanded nodes. The resulting JADIS planner has shown to be competitive with the existing state-of-the-art FOND planners.

As future work, we aim to investigate other techniques for improving LAO* for FOND planning, so we intend to look into other updating functions, use and adapt modern FOND planning heuristics [38], design explicitly tie-breaking strategies for LAO*, and employ other satisficing planning techniques in LAO*, such as *preferred operators* detection and selection [43], *deferred evaluation* [43], *Pareto* [45], *look-ahead* strategies for delete-relaxation heuristics [19], and etc.

References

[1] Masataro Asai and Alex Fukunaga, 'Tie-breaking strategies for cost-optimal best first search', *Journal of Artificial Intelligence Research*, **58**, 67–121, (2017).

[2] Masataro Asai and Alex S. Fukunaga, 'Tiebreaking strategies for a* search: How to explore the final frontier', in *AAAI*, (2016).

[3] Richard Bellman, *Dynamic Programming*, Dover Publications, 1957.

[4] Dimitri Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA, 1995.

[5] Dimitri P. Bertsekas and John N. Tsitsiklis, 'An analysis of stochastic shortest path problems', *Mathematics of Operations Research*, **16**(3), (1991).

[6] Blai Bonet, Giuseppe De Giacomo, Hector Geffner, Fabio Patrizi, and Sasha Rubin, 'High-level programming via generalized planning and LTL synthesis', in *KR*, (2020).

[7] Blai Bonet and Hector Geffner, 'An algorithm better than ao*?', in *AAAI*, (2005).

[8] Blai Bonet and Hector Geffner, 'Qualitative numeric planning: Reductions and complexity', *Journal of Artificial Intelligence Research*, **69**, 923–961, (2020).

[9] Blai Bonet and Héctor Geffner, 'Planning as heuristic search', *Artificial Intelligence*, **129**, 5–33, (2001).

[10] Blai Bonet, Giuseppe De Giacomo, Hector Geffner, and Sasha Rubin, 'Generalized planning: Non-deterministic abstractions and trajectory constraints', in *IJCAI*, (2017).

[11] R. Brafman and G. De Giacomo, 'Planning for LTLf/LDLf goals in non-markovian fully observable nondeterministic domains', in *IJCAI*, (2019).

[12] Daniel Bryce and Olivier Buffet, '6th International Planning Competition: Uncertainty Part', *International Planning Competition (IPC)*, (2008).

[13] A. Camacho, J. Baier, C. Muise, and S. McIlraith, 'Finite LTL synthesis as planning', in *ICAPS*, (2018).

[14] A. Camacho, E. Triantafillou, C. Muise, J. Baier, and S. McIlraith, 'Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces', in *AAAI*, (2017).

[15] Alberto Camacho and Sheila A. McIlraith, 'Strong fully observable non-deterministic planning with LTL and ltlf goals', in *IJCAI*, (2019).

[16] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso, 'Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking', *Artificial Intelligence*, **147**(1-2), (2003).

[17] Augusto B. Corrêa, André Grahl Pereira, and Marcus Ritt, 'Analyzing tie-breaking strategies for the a* algorithm', in *IJCAI*, (2018).

[18] Giuseppe De Giacomo and Sasha Rubin, 'Automata-theoretic foundations of fond planning for ltlf and ldlf goals', in *IJCAI*, (2018).

[19] Maximilian Fickert, 'A novel lookahead strategy for delete relaxation heuristics in greedy best-first search', in *ICAPS*, (2020).

[20] Jicheng Fu, Vincent Ng, Farokh B. Bastani, and I-Ling Yen, 'Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems', in *IJCAI*, (2011).

[21] Hector Geffner and Blai Bonet, *A Concise Introduction to Models and Methods for Automated Planning*, Morgan & Claypool Publishers, 2013.

[22] Tomas Geffner and Hector Geffner, 'Compact policies for fully observable non-deterministic planning as SAT', in *ICAPS*, (2018).

[23] Eric A. Hansen and Shlomo Zilberstein, 'Heuristic search in cyclic AND/OR graphs', in *AAAI*, (1998).

[24] Eric A. Hansen and Shlomo Zilberstein, 'Lao*: A heuristic search algorithm that finds solutions with loops', *Artificial Intelligence*, **129**(1-2), 35–62, (2001).

[25] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics*, (1968).

[26] Malte Helmert, 'The fast downward planning system', *Journal of Artificial Intelligence Research*, **26**, 191–246, (2006).

[27] Jörg Hoffmann and Bernhard Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

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

[29] Yuxiao Hu and Giuseppe De Giacomo, 'Generalized planning: Synthesizing plans that work for multiple environments', in *IJCAI*, (2011).

[30] Peter Kissmann and Stefan Edelkamp, 'Solving fully-observable non-deterministic planning problems via translation into a general game', in *KI Advances in AI*, volume 5803, pp. 1–8, (2009).

[31] Andrey Kolobov, Mausam, Daniel S. Weld, and Hector Geffner, 'Heuristic search for generalized stochastic shortest path mdps', in *ICAPS*, (2011).

[32] Richard E. Korf, 'Depth-first iterative-deepening: An optimal admissible tree search', *Artificial Intelligence*, **27**(1), 97–109, (1985).

[33] Ugur Kuter, Dana S. Nau, Elnatan Reisner, and Robert P. Goldman, 'Using classical planners to solve nondeterministic planning problems', in *ICAPS*, (2008).

[34] Nir Lipovetzky, Christian J. Muise, and Hector Geffner, 'Traps, invariants, and dead-ends', in *ICAPS*, (2016).

[35] Alberto Martelli and Ugo Montanari, 'Additive AND/OR graphs', in *IJCAI*, (1973).

[36] Robert Mattmüller, Manuela Ortlieb, Malte Helmert, and Pascal Bercher, 'Pattern database heuristics for fully observable nondeterministic planning', in *ICAPS*, (2010).

[37] Robert Mattmüller, *Informed Progression Search for Fully Observable Nondeterministic Planning*, Ph.D. dissertation, Albert-Ludwigs-Universität Freiburg, 2013.

[38] Frederico Messa and André Grahl Pereira, 'A Best-First Search Algorithm for FOND Planning and Heuristic Functions to Optimize Decompressed Solution Size', in *ICAPS*, (2023).

[39] Christian Muise, Sheila A McIlraith, and J Christopher Beck, 'Improved non-deterministic planning by exploiting state relevance', in *ICAPS*, (2012).

[40] Fabio Patrizi, Nir Lipovetzky, and Hector Geffner, 'Fair LTL synthesis for non-deterministic systems using strong cyclic planners', in *IJCAI*, (2013).

[41] Ramon Fraga Pereira, André Grahl Pereira, Frederico Messa, and Giuseppe De Giacomo, 'Iterative Depth-First Search for FOND Planning', in *ICAPS*, (2022).

[42] Miquel Ramírez and Sebastian Sardiña, 'Directed fixed-point regression-based planning for non-deterministic domains', in *ICAPS*, (2014).

[43] Silvia Richter and Malte Helmert, 'Preferred operators and deferred evaluation in satisficing planning', in *ICAPS*, (2009).

[44] Silvia Richter and Matthias Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks', *Journal of Artificial Intelligence Research*, **39**, 127–177, (2010).

[45] Gabriele Röger and Malte Helmert, 'The more, the merrier: Combining heuristic estimators for satisficing planning', in *ICAPS*, (2010).

[46] Dominik Winterer, Martin Wehrle, and Michael Katz, 'Structural symmetries for fully observable nondeterministic planning', in *IJCAI*, (2016).