A Study of Heuristic Search Algorithms for Planning in Artificial Intelligence

Frederico Messa, André G. Pereira

[1]Federal University of Rio Grande do Sul, Brazil

{frederico.messa,agpereira}@inf.ufrgs.br

Abstract. *Planning is a long-standing area of Artificial Intelligence that aims to solve goal-directed tasks. Planning tasks have compact descriptions that generate exponentially larger state-spaces, and heuristic search algorithms are the most effective methods to solve them. In this work, we study heuristic search algorithms for planning tasks. First, we propose a* hybrid memory-restricted heuristic search algorithm called PEA*+IDA* *that outperforms algorithms of the same class. Second, we present a complete* theoretical *and* experimental analysis *of memory-restricted algorithms helping to better understand the landscape of this class of algorithms. Finally, we propose a new* depth-first search algorithm *for* non-deterministic *planning tasks that outperforms comparable algorithms.*

1. Introduction

Planning is an Artificial Intelligence area that solves goal-directed problems. Instances of these problems, called *planning tasks*, are usually PSPACE-complete because they have compact descriptions that generate exponentially larger state-spaces and solutions. These state-spaces have an *initial state*, *goal states*, and deterministic *transitions* with non-negative cost. An (optimal) solution is a sequence of transitions that connects the initial state to one of the goal states (with minimal cost). Planning aims to develop algorithms capable of solving goal-directed tasks of several domains since many real-life problems can be modeled as planning tasks. The development of general planners is a cost-effective software engineering approach since one can model the problem of interest as a planning task and then select a general planner to solve it. If the problem changes, there is no need to change the planner, just the model of the task.

Since planning tasks have exponentially larger state-spaces, *heuristic functions* are required to produce effective search algorithms – heuristic functions guide the search to the most promising parts of the state-space first. However, in general, heuristic functions are imperfect and planners have to store all generated parts of the state-space. Therefore, planners struggle in memory-restricted scenarios, and algorithms that better use the available memory are required. Also, state-of-art planners use memory-based heuristic functions. Thus, algorithms that better use the available memory will result in better planners. Memory-restricted search is a sub-area that aims to develop fast search algorithms that do not fail due to memory restrictions.

In *non-deterministic* planning, transitions have non-deterministic outcomes. Thus, a solution for non-deterministic planning is a *policy* that takes into account all outcomes that might realistically occur and guarantees that all trajectories that follow the policy can always reach goal states. Available planners struggle with either task size or non-determinism, and an effective non-deterministic planner must be robust to both. Non-deterministic planners can be used to solve many formalisms of planning, including *general planning* and *temporal planning*. An important application of non-deterministic planners is to solve goal-directed *Markov Decision Processes* (MDPs) with compact descriptions (also known as *stochastic shortest path* (SSP) planning). These planners are, in general, more effective for goal-directed MDPs with compact descriptions than traditional reinforcement learning methods because they can provide a solution without evaluating the entire state-space [Hansen and Zilberstein 2001].

Contribution 1: ***A New Hybrid Memory-Restricted Heuristic Search Algorithm*** We present the algorithm PEA*+IDA* that runs Partial Expansion A* (PEA*) until the memory limit is reached, and then runs Iterative Deepening A* (IDA*). IDA* has an extremely low memory consumption. Thus, PEA*+IDA* does not fail by memory limits. We then compare it with a state-of-the-art memory-restricted algorithm, observing that our approach is faster and solves many more tasks. We theoretically analyze both algorithms and explain aspects that impact the algorithms' performance. **Published at the AAAI 2022, Qualis A1.**

Contribution 2: ***A Study of Memory-Restricted Heuristic Search Algorithm*** We extended our previous contribution by presenting formal proofs and a theoretical model of many search algorithms that explain their strengths and weakness in many memory-restricted scenarios. We experimentally show that our proposed algorithm PEA*+IDA* improves the state-of-the-art in practical settings. **To be submitted to a journal.**

Contribution 3: ***A Robust Heuristic Search Algorithm for Non-Deterministic Planning*** We propose a new iterative depth-first search (IDFS) algorithm with theoretical guarantees. IDFS outperforms other algorithms for non-deterministic planning, showing a robust performance concerning task size and non-determinism. **Published at the ICAPS 2022, Qualis A2.**

2. A New Hybrid Memory-Restricted Heuristic Search Algorithm.

Scope The undergrad student started and led this research during his undergraduate studies, resulting in his undergraduate final work.

Summary A* [Hart et al. 1968] is one of the most popular best-first heuristic search algorithms due to its capability to time-efficiently solve state-space tasks optimally while being intuitive and simple to understand. It expands first nodes with better estimates and stores all generated nodes until expanding and replacing the stored nodes with their children. Since the node estimates given by efficient heuristic functions are imperfect, A* often fails to solve challenging state-space tasks, even in scenarios with large memory limits. Iterative Deepening A* (IDA*) overcomes the memory limitations of A* [Korf 1985], as it is a heuristic search algorithm with low memory requirement, linear in the depth of the search. However, IDA* has no duplicate detection without using extra memory. Thus, it may frequently expand nodes with the same states. Also, it requires multiple re-expansions of the same nodes due to its iterative behavior, especially those close to the root node. Thus, pure IDA* needs, frequently, orders of magnitude more expansions than A* to solve optimality challenging state-space tasks.

Many algorithms were proposed to solve state-space tasks using less memory than A* and making fewer node expansions than IDA*, such as MREC [Sen and Bagchi 1989], SMA* [Russell 1992], AL* [Stern et al. 2010], and PEA* [Yoshizumi et al. 2000]. Some of them have a high polynomial-time overhead per node expansion or generation compared to A*. Some have the performance depending on the quality of hyper-parameter values that are hard to define, such as the AL* algorithm. Others like PEA* cannot be restricted to a specific memory limit. Finally, many are relatively difficult to understand or implement. Because of these issues, these algorithms are less frequently used in practice.

Bu and Korf [2019] presented a new algorithm combining A* and IDA* in a hybrid algorithm with two phases called A*+IDA*. Their new approach does not have the mentioned disadvantages since it is simple to understand, easy to implement, has low overhead per node, and limits the required memory. A*+IDA* achieves speed-ups of around five times over IDA* for specific domains. However, although A*+IDA* avoids failing due to memory limits, its second phase still has the drawbacks of the pure IDA* algorithm.

2.1. Background

A state-space task is a tuple $\Theta = \langle S, A, T, c, s_0, S_G \rangle$ [Sturtevant and Helmert 2019] where S is a finite set of states, A is a finite set of *actions*, $T : S \times A \rightharpoonup S$ is partial function of *transitions* between states, $c : A \to \mathbb{R}_{\geq 0}$ is a *cost function* that maps actions to non-negative real costs, $s_0 \in S$ is the *initial state* and $S_G \subseteq S$ the set of *goal states*. A state-space task Θ is the explicit representation obtained from the compact representation of a planning task Π. A *solution* of Θ is a path of transitions $\langle s_0 \xrightarrow{a_1} s_1, s_1 \xrightarrow{a_2} s_2, \ldots, s_{n-1} \xrightarrow{a_n} s_n \rangle$ with $s_n \in S_G$ and $T(s_{i-1}, a_i) = s_i, \forall i \in [1, n]$. It is optimal if its cost $\sum_{i=1}^{n} c(a_i)$ is minimal. A *heuristic function* $h : S \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ maps all states to their h-values. The h-value of a state s estimates the minimal cost path from s to any goal state. The *perfect* heuristic function h^* estimates that cost correctly for all states, assigning $h^*(s) = \infty$ to states s for which no such path exists. A heuristic is *admissible* if and only if $h(s) \leq h^*(s)$ for all $s \in S$. The f-value of a state s estimates the cost of a solution going through s and is defined as $f(s) = g(s) + h(s)$, where $g(s)$ is the current cost from s_0 to s. A *search node* n is a data structure that contains a state s, its g and f-values, and its parent node ($\perp$ for the root node). Function $\texttt{succ}(n)$ generates all nodes n' such that $n'.state$ is children of $n.state$ (i.e., nodes n' such that $T(n.state, a) = n'.state$). When $\texttt{succ}(n)$ is invoked, the node n is *expanded* and all its children are *generated*.

2.2. The PEA*+IDA* Algorithm

We propose the use of the Partial Expansion A* (PEA*) as the first phase algorithm, creating the PEA*+IDA* algorithm. Partial Expansion A* is an algorithm based on A* that avoids storing all generated children of expanded nodes, thus reducing its memory requirements. PEA*+IDA* is a new hybrid algorithm that is as simple and intuitive as A*+IDA*. With the trade-off of possibly having more expansions in the first phase, PEA*+IDA* generally reduces the number of IDA* iterations and expansions in the second phase. We now present a high-level description of PEA*+IDA* (Algorithm 1).

2.2.1. High-Level Description

First Phase (lines 3–21) PEA*+IDA* removes from Open first a node n with least F-value (line 4). The F-value of a node has the same purpose as the f-value, to estimate the cost of a solution going through the node, but it can be updated throughout the execution of the algorithm. When expanding the node n, the algorithm divides the generated children nodes from $\texttt{succ}(n)$ into two sets $\texttt{Children}_{\leq}$ and $\texttt{Children}_{>}$. The set $\texttt{Children}_{\leq}$ (line 7) stores nodes with F-values lower or equal to $n.F$. The set $\texttt{Children}_{>}$ (line 8) stores nodes F-values greater than $n.F$. PEA*+IDA* terminates the first phase (lines 9–11) if the memory (`Open` size) required to expand the node n is greater than the predetermined limit. Line 13 invokes the typical method of A* that processes generated nodes in $\texttt{Children}_{\leq}$. Lines 14–21 process $\texttt{Children}_{>}$. If there is more than one node in $\texttt{Children}_{>}$, they are discarded, and the node n is re-inserted in `Open` with F-value updated to the minimum F-value of the discarded nodes. Otherwise, no node children of n is discarded, and n is then closed.

Algorithm 1: PEA*+IDA*

```
1   Open := {make_root()}
2   Closed := ∅
    // First Phase:  Restricted PEA*
3   while Open ≠ ∅ do
4       Remove node $n$ from Open with minimum $n.F$
5       if is_goal($n$) then
6           return extract_path($n$)
7       $\texttt{Children}_{\leq}$ := $\{n' \mid n' \in \mathbf{succ(n)} \wedge n'.F \leq n.F\}$
8       $\texttt{Children}_{>}$ := $\{n' \mid n' \in \mathbf{succ(n)} \wedge n'.F > n.F\}$
9       if $|\texttt{Open}| + |\texttt{Children}_{\leq}| + \min(|\texttt{Children}_{>}|, 1) >$
          MEMORY_LIMIT then
10          Insert $n$ in Open
11          break loop
12      foreach $n' \in \texttt{Children}_{\leq}$ do
13          process_child($n'$)
14      if $|\texttt{Children}_{>}| = 0$ then
15          Insert $n$ in Closed
16      else if $|\texttt{Children}_{>}| = 1$ then
17          process_child($n'$) | $n' \in \texttt{Children}_{>}$
18          Insert $n$ in Closed
19      else
20          $n.F := \min\{n'.F \mid n' \in \texttt{Children}_{>}\}$
21          Insert $n$ in Open
    // Second Phase:  IDA*
22  while Open ≠ ∅ do
23      Remove node $n$ from Open with minimum $n.F$
24      solution_path, $new\_F\_limit$ := IDA*($n, n.F$)
25      if solution_path ≠ ⊥ then
26          return solution_path
27      if $new\_F\_limit = \infty$ then
28          Insert $n$ in Closed
29      else
30          $n.F := new\_F\_limit$
31          Insert $n$ in Open
32  return ⊥ // UNSOLVABLE
33  Method process_child ($n'$):
34      if $n'.state \neq n.state$ then
35          if $n'.state \in$ Open then
36              if $n'.g <$ Open(n'.$state$).g then
37                  Open($n'.state$).$update$($n'$)
38          else if $n'.state \in$ Closed then
39              if $n'.g <$ Closed(n'.$state$).g then
40                  Remove $n'.state$ from Closed
41                  Insert $n'$ in Open
42          else
43              Insert $n'$ in Open
```

Second Phase (lines 22–31) PEA*+IDA* again removes from `Open` first a node n using the same order from the first phase. Line 24 invokes a standard iteration of IDA* starting from the node n and using as F-limit its F-value. At the end of the iteration, if IDA* finds a solution, the algorithm returns it. If the new F-limit returned by the IDA* iteration is infinite, the node n is inserted in `Closed`. Otherwise, it is re-inserted in `Open` updating its F-value to the new F-limit defined by IDA*.

2.2.2. Empirical Analysis

In this subsection, we aim to better understand A*+IDA* and PEA*+IDA*. Thus, we compare them using three different memory limits, measuring time as the number of expanded nodes because it avoids differences that result from implementation details and memory consumption as the number of nodes stored in `Open` because it is the main source of these algorithms' memory consumption. We use the STRIPS optimal benchmark of 1877 tasks of the International Planning Competition (IPC) and remove from our experiments the

Table 1. Coverage and expansions of hybrid algorithms at three memory limits.

	10%		50%		90%		100%
	A*+IDA*	PEA*+IDA*	A*+IDA*	PEA*+IDA*	A*+IDA*	PEA*+IDA*	A*
Airport (2/50)	550	**188**	**203**	223	357	**223**	225
Blocks (10/35)	**240,000**	303,167	**80,140**	88,138	**68,379**	88,138	65,289
Data (5/20)	**8,337**	9,325	761	**354**	365	**354**	199
Depot (3/22)	**510,071,912**	586,146,642	13,285,410	**149,088**	5,919,570	**149,088**	97,433
Driverlog (7/20)	**353,027**	355,316	37,090	**3,870**	13,384	**3,870**	3,058
Floortile (5/40)	**9,305,927**	105,582,631	212,129	**52,701**	**38,791**	52,701	27,077
Ged (2/20)	**18,111,171**	55,838,776	9,253,283	**2,045,180**	3,707,632	**2,045,180**	2,166,322
Grid (1/5)	**331,728**	477,407	**89,609**	109,367	**83,421**	109,367	77,087
Logistics (3/63)	22,907,130	**951,837**	22,699,231	**373**	15,519,334	**373**	375
Miconic (93/150)	**168**	183	**168**	186	187	**186**	185
Mprime (7/35)	2,523	**1,538**	1,455	**940**	1,280	**940**	1,180
Mystery (3/30)	3,796	**2,571**	2,737	**2,412**	**1,609**	2,412	1,605
Nomystery (6/20)	**24,014**	45,142	8,657	**4,295**	5,164	**4,295**	3,605
Organic (6/40)	**3,793**	4,028	2,627	**2,444**	1,908	**1,769**	1,216
Parcprinter (11/50)	338	**235**	179	**63**	142	**63**	58
Parking (5/40)	**81,800**	143,550	38,201	**29,698**	**29,176**	29,698	24,404
Pipesworld (8/150)	**1,178,042**	1,334,954	187,688	**54,576**	143,275	**54,576**	43,619
Rovers (2/40)	**127,806,277**	404,719,675	9,457,675	**22,380**	4,270,004	**22,380**	19,372
Satellite (3/36)	416,021	**93,621**	130,167	**8,935**	53,908	**8,935**	7,999
Scanalyzer (10/50)	15	15	15	**14**	15	**14**	14
Sokoban (4/50)	15,289	15,289	4,392	**4,390**	1,262	1,262	458
Spider (2/20)	2,618,834	**1,627,121**	357,970	**101,673**	217,938	**101,673**	95,344
Storage (1/30)	**6,235,135**	13,994,290	857,116	**215,483**	447,060	**215,483**	155,763
Tidybot (8/40)	**344,327**	351,201	60,675	**49,727**	31,632	**27,813**	20,395
Trucks (3/30)	**396,319**	4,267,495	125,631	**14,434**	39,035	**14,434**	13,201
Visitall (5/40)	**1,040,799**	1,125,316	**269,693**	347,120	**212,046**	231,582	177,030
Woodworking (16/50)	349,931	**20,803**	82,080	**2,131**	39,288	**2,131**	1,564
Zenotravel (6/20)	623,588	**21,778**	411,994	**8,192**	189,333	**8,192**	8,628
Avg. Expansions	149,628.33	**132,823.20**	40,284.94	**7,815.38**	23,554.71	**7,133.72**	5,698.35
Coverage	247	**249**	267	**306**	275	**318**	

tasks that are either too "hard" or too "easy", i.e., not solved by pure A* with $h^{\texttt{LMCut}}$ in 10 minutes with 2 GB of memory, or solved by pure A* with blind heuristic function with the same limits. We use $h^{\texttt{LMCut}}$ in all the remaining experiments and compare the algorithms using the remaining 332 tasks, limiting the `Open` size to 10%, 50%, and 90% of the A*'s `Open` size peak to solve each task. We ran all experiments with a Ryzen 3900X, and all algorithms use as tie-breakers for `Open` first lower h-value followed by the greater depth and finally lower generation order. We use the Fast Downward [Helmert 2006] framework to implement all our algorithms. Our code is available to the community [1].

A*+IDA* vs. PEA*+IDA* We now compare the hybrid algorithms. In addition to the previously defined limits, the algorithms could not solve some tasks within six hours. For PEA*+IDA* the number of failures is respectively 83, 26, and 14 at 10%, 50%, and 90% memory limits, while for A*+IDA* is respectively 85, 65, and 57. Table 1 shows the coverage of both hybrid algorithms for the memory limits. Since both hybrid algorithms have a very similar cost per expansion in the first phase and the same cost per expansion in the second phase, the higher coverage of PEA*+IDA* shows that it is generally superior.

To compare expansions, we remove tasks that failed to be solved by any experiment, remaining 237 tasks. Table 1 shows as $({}^{x}/_{y})$ the total number y and the number x of the remaining tasks of each domain. Table 1 shows per-domain average expansions for each algorithm and memory limit. It also shows the expansions of pure A* for reference. We use a geometric mean in all average calculations since it avoids overweighting hard domains, reduces the effect that some domains have more remaining tasks than others. We deal with zero values by incrementing the values before the mean, and decrement-

[1] `https://github.com/Frederico-Messa/PEAstar-plus-IDAstar`

Table 2. Mean second phase information for hybrid algorithms, and A*+IDA$^{*\uparrow}$.

	A*+IDA*			A*+IDA$^{*\uparrow}$			PEA*+IDA*		
	10%	50%	90%	10%	50%	90%	10%	50%	90%
Avg. IDA* Phase Exp.	145,201.08	73,442.10	35,506.81	30,306.27	220.13	3.29	8,093.42	60.60	0.47
Avg. IDA* Iterations	1,062.78	722.83	210.76	938.31	46.48	1.25	265.14	18.63	0.18

ing afters. In per-domain expansions, at 10% memory limit, the hybrid algorithms are comparable. At 50% and 90% memory limits, PEA*+IDA* wins in almost all domains respectively 23 vs. 5 and 21 vs. 6.

PEA*+IDA* requires the second phase in few tasks at higher memory limits. Table 2 displays geometric mean second phase information over domains and tasks of Table 1 for the hybrid algorithms, showing that PEA*+IDA* dramatically reduces the average number of second phase expansions and iterations. Outliers often occur for A*+IDA*. In some tasks, it required more than $10{,}000$ times more expansions than PEA*+IDA*, while the opposite does not occurred.

We present lower bounds to the mean number of expansions in the 257 tasks that at least one of the two algorithms solved at all three memory limits. The lower bounds consider the number of expansions made up to the time limit of six hours. For the limits of 10%, 50%, and 90%, PEA*+IDA* has a respectively lower bound on the number of expansions of $207{,}149.27$, $11{,}794.65$, and $9{,}638.18$, while A*+IDA* has a respectively lower bound of $421{,}773.34$, $98{,}566.23$, and $55{,}575.52$. Thus, an estimate of the speed-up of PEA*+IDA* for the respective limits is 2.04, 8.36, and 5.77.

Better Open Composition PEA*+IDA* has a more homogeneous Open when memory reaches the limit and it also has a higher starting F-limit to the IDA* iterations. Thus, the higher starting F-limit could explain the better performance of PEA*+IDA*. However, PEA*+IDA*, besides reducing the number of IDA* iterations, also reduces (at 50% and 90%) the number of expansions of each iteration. The number of second phase expansions per iteration for PEA*+IDA* and the three limits is respectively 168.47, 2.63, and 2.61, while for A*+IDA* is 136.62, 32.30, and 30.53. Thus, the better Open composition of PEA*+IDA* is partially responsible for its performance.

Higher Initial F-Limit To evaluate the effect of the higher F-limit of PEA*+IDA*, we artificially modified A*+IDA* into what we call "A*+IDA$^{*\uparrow}$". A*+IDA$^{*\uparrow}$ runs A* as A*+IDA*, but, when memory reaches the limit and before the second phase begins, all nodes in Open with F-values lower than a value $F^\uparrow$ have their F-values updated to $F^\uparrow$. We define $F^\uparrow$ as the minimal F-value in the Open of PEA*+IDA* at its first IDA* iteration if it required the second phase to solve the task, and as $h^*(n_0.state)$, otherwise.

Table 2 shows that A*+IDA$^{*\uparrow}$ has a dramatic reduction of IDA* phase expansions and iterations when compared to A*+IDA* in all memory limits. This indicates that higher F-limits have a considerable impact on the second phase of the algorithm, although the last two layers of IDA* iterations dominate the number of expansions. We also used A*+IDA$^{*\uparrow}$ to measure the impact of the Open node composition of PEA*+IDA* against the one of A*+IDA* when memory reaches the limit. Since A*+IDA$^{*\uparrow}$ has a F-limit at first IDA* iteration greater or equal to PEA*+IDA*, and approximately the same Open size due to the memory limits, we could expect that the former would perform at least as better as the latter in the second phase. However, Table 2 shows otherwise, PEA*+IDA* is still superior concerning second phase expansions and IDA* iterations, due to its better Open composition.

3. A Study of Memory-Restricted Heuristic Search Algorithm.

Scope The undergrad student started, led this research during his undergraduate studies, and continues to work on this contribution.

Summary We present formal proofs of the soundness and completeness for PEA*+IDA* and a more in-depth theoretical model of five algorithms: PEA*+IDA*, A*+IDA*, pure A*, pure PEA*, and pure IDA*. This theoretical model shows how PEA*+IDA* can perform better than A*+IDA* although pure PEA* usually makes more expansions than pure A*. We also perform empirical experiments using other heuristic functions and compare our approach in practical settings, showing that it improves the state-of-the-art in planning. Here we present a simplified version of this contribution.

3.1. Soundness and Completeness

We show that PEA*+IDA* with an admissible heuristic function h always terminates returning an optimal solution if the task is solvable, and, terminates proving unsolvablity if the task is unsolvable. The main argument of the proof of the Theorem 1 is that PEA*+IDA* always has the possibility of choosing an a node n for expansion that is part of the an optimal solution.

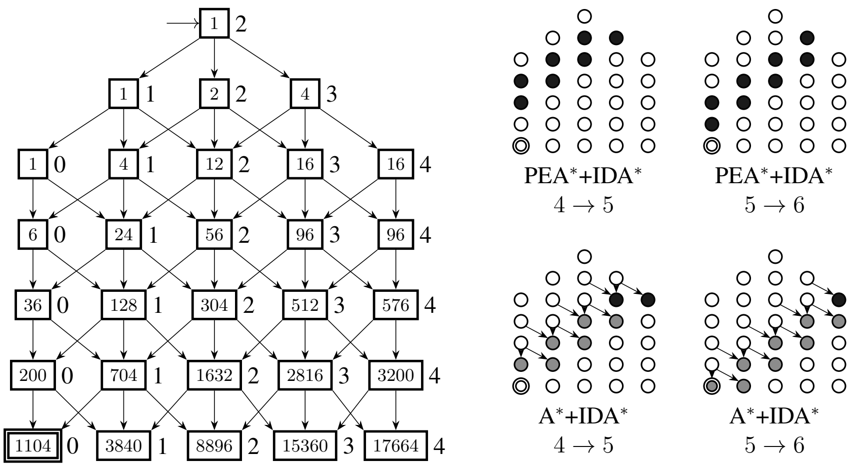
Theorem 1. *For a state-space task Θ, PEA*+IDA* with an admissible heuristic function h returns an optimal solution if one exists and $\perp$ otherwise.*

3.2. Open Size and Composition Model

We now present a simplified model of the size and node composition of the A*+IDA* and PEA*+IDA*'s Open lists when the minimum node F-value transitions from x to $x+1$. In this work, the F-value of A*+IDA* is always equal to the f-value. The model of Korf and Reid [1998] serves as inspiration for our model. In this model, h-values range from l^- to l^+, the root node has h-value equals to $h(n_0.state)$ and the transitions have unitary cost. In addition, a node n generates γ_1 non-duplicated children with h-value equals to $n.h-1$, γ_2 with h-value equals to $n.h$, and γ_3 with h-value equals to $n.h+1$. With the model, we can compute the number of nodes with g-value $\mathbf{g}$ and h-value $\mathbf{h}$ using Equation 1.

$$|N_{\mathbf{g},\mathbf{h}}| = \begin{cases} \gamma_1 \cdot |N_{\mathbf{g}-1,\mathbf{h}+1}| + \gamma_2 \cdot |N_{\mathbf{g}-1,\mathbf{h}}| + \gamma_3 \cdot |N_{\mathbf{g}-1,\mathbf{h}-1}| & \text{if } \mathbf{g} > 0 \wedge l^- \leq \mathbf{h} \leq l^+; \\ 1 & \text{if } \mathbf{g} = 0 \wedge \mathbf{h} = h(n_0.state); \text{ and} \\ 0 & \text{else.} \end{cases} \quad (1)$$

Suppose the hybrid algorithm does not require its IDA* phase yet. Then, we can determine what nodes are in Open at the instant of the transition of minimum node F-value, i.e., when Open starts to only contain nodes with F-values equal to at least $x+1$. For PEA*+IDA*, the nodes in Open are the ones with f-values (original F-values) at most x that have children nodes with f-values at least $x+1$, since nodes with f-values greater than x would still not be generated without being discarded, and since nodes without children nodes with f-values at least $x+1$ would have already been inserted in Closed. For the A*+IDA* algorithm, the nodes in Open are the ones with f-values at



(a) Example state-space task. (b) Illustration of the `Open` of the hybrid algorithms.

Figure 1. Each rectangle is a node-set that contains nodes with the same g and h-values. The number at the right of a node-set is its nodes h-value, while the number inside is its size. The depth of a node-set is its nodes g-value. Node sets with nodes g-values greater than 6 are omitted.

least $x + 1$ that are children of nodes with f-value at most x, since nodes with f-values lower than $x + 1$ would have already been expanded, and since children of nodes with f-values greater than x would have not been generated yet as their parents' nodes would have not been expanded. We can compute the number of nodes in `Open` for $x = \mathbf{x}$ using Equation 2 for PEA*+IDA* and Equation 3 for A*+IDA*.

$$\sum_{\mathbf{h}=l^-}^{l^+} \left(|N_{(\mathbf{x}-1)-\mathbf{h},\mathbf{h}}| + |N_{\mathbf{x}-\mathbf{h},\mathbf{h}}|\right) \quad (2)$$

$$\gamma_2 \cdot \sum_{\mathbf{h}=l^-}^{l^+} |N_{\mathbf{x}-\mathbf{h},\mathbf{h}}| + \gamma_3 \cdot \sum_{\mathbf{h}=l^-}^{l^+-1} \left(|N_{(\mathbf{x}-1)-\mathbf{h},\mathbf{h}}| + |N_{\mathbf{x}-\mathbf{h},\mathbf{h}}|\right) \quad (3)$$

Example Using the model we can create a state-space task that exemplifies the behavior of the hybrid algorithms. Figure 1a shows a task from a model with $l^- = 0$, $l^+ = 4$, $h(n_0.state) = 2$, $\gamma_1 = 1$, $\gamma_2 = 2$, and $\gamma_3 = 4$, i.e., a node n generates one non-duplicated child node with h-value one less than its h-value, two with h-value equals to its h-value, and four with h-value one more than its h-value. In this example, the optimal solution cost is 6. This example aims to emulate a space-state with a heuristic that maps few states to small h-values since generally few nodes are near goal states.

For $x = 4$, PEA*+IDA* has nodes in `Open` with f-values equal to 3 and 4, thus $(6+4+2)+(36+24+12+4) = 12+76 = 88$ nodes. Figure 1a shows these node-sets in the second and third diagonals. For $x = 5$, PEA*+IDA* has the nodes in `Open` with f-values equal to 4 and 5, thus $76+(200+128+56+16) = 76+400 = 476$ nodes. The Figure 1b shows in black, in the upper quadrants, the node-sets in PEA*+IDA*'s `Open` respectively for $x = 4$ and $x = 5$.

For $x = 4$, A*+IDA* has in `Open` the nodes with f-values 5 and 6 that are children of nodes with f-values equal to 3 and 4, thus $\gamma_3 \cdot 12 + (\gamma_2 + \gamma_3) \cdot 76 = 4 \cdot 12 + 6 \cdot 76 = 504$ nodes. For $x = 5$, A*+IDA* has in `Open` the nodes with f-values 6 and 7 that are children of nodes with f-values equal to 4 and 5, thus $\gamma_3 \cdot 76 + (\gamma_2 + \gamma_3) \cdot 400 = 2704$ nodes. Figure 1b, in the lower quadrants, respectively shows for $x = 4$ and $x = 5$, the node-sets entirely (in black) and partly (in gray) in A*+IDA*'s `Open`.

Note that, for a memory limit of 500 nodes in Open, A*+IDA* would run out of memory while still having a node with F-value equals to 4 in Open, while PEA*+IDA* would only run out of memory after having in Open only nodes with F-values at least 6. Thus, the IDA* phase of the former would have two more layers of iterations than the one of the latter, providing an intuition of why the PEA*+IDA* algorithm may overcome the A*+IDA* algorithm.

4. A Robust Heuristic Search Algorithm for Non-Deterministic Planning

Scope This contribution was a collaboration with the researchers Ramon F. Pereira and Giuseppe De Giacomo, from Sapienza University of Rome. The undergrad student led the theoretical analysis of the proposed algorithm and its theoretical proofs.

Summary Non-deterministic planning known as *Fully Observable Non-Deterministic* (FOND) planning models uncertainty through actions with *non-deterministic* outcomes. Existing FOND planners are effective and employ a wide range of techniques. However, such planners struggle with either task size or non-determinism, because they usually either do not use traditional sources of information for search or are designed to solve a problem that requires more guarantees, addressing the non-deterministic aspect in a non-explicit way. We developed a novel *iterative depth-first search algorithm, called IDFS*, that address the previously mentioned limitations, being robust to both increases on task size and non-determinism. We compared our proposed algorithm to well-known FOND planners [Muise et al. 2012, Geffner and Geffner 2018], and showed that it has robust performance over several types of FOND domains considering different metrics.

Theorem 2. *Given a FOND task Θ and an admissible heuristic function h. If Θ is* unsolvable*, then IDFS proves that. If Θ is* solvable*, then IDFS returns a* policy *by searching to a depth of at most* $\texttt{cv}^*(\Theta)$.

We introduced the concept of the *critical value* $\texttt{cv}^*(\Theta)$ to characterize the behavior of IDFS by the structure of policies of the FOND task Θ. We prove that if a FOND planning task Θ is *unsolvable*, IDFS identifies it correctly, and if Θ is *solvable*, IDFS returns a *policy* by searching to a depth of at most $\texttt{cv}^*(\Theta)$. The proofs related to Theorem 2 are extremely sophisticated including many nested proofs by induction. Besides characterizing IDFS, the proofs of the Theorem 2 help in understanding the structure of FOND policies and the research for the development of new FOND planning task algorithms.

5. Conclusion

Summary This work resulted in two publications.

- **Messa, F.**, Pereira, A. G. (2022). *PEA*+IDA*: An Improved Hybrid Memory-Restricted Algorithm.* **AAAI Conference on Artificial Intelligence (AAAI)**, Qualis A1.
- Pereira., R. F., Pereira, A. G., **Messa, F.**, De Giacomo, G. (2022). *Iterative Depth-First Search Algorithms for Fully Observable Non-Deterministic Planning.* **International Conference on Automated Planning and Scheduling (ICAPS)**, Qualis A2.

The undergrad student is first-author of the AAAI publication. AAAI is one of the two most important and selective conferences of artificial intelligence. ICAPS is the main conference of the automated planning and heuristic search research areas, being usually the chosen venue for novel contributions of researchers of these areas. We expect to submit Contribution 2 to the Artificial Intelligence Journal (AIJ) in the next month.

Impact In general, our contributions will help the community develop new memory-restricted algorithms by providing a better understanding of this class of algorithms. The experiments presented in Contribution 2 include the addition of our algorithm to one of the best planners of the IPC 2018, which resulted in significant improvement in its performance. Therefore, our contributions are also likely to impact many, practical, real-world applications that apply planning.

Traditional reinforcement learning algorithms solve goal-directed MDP tasks with compact descriptions by evaluating multiple times the entire state-space. IDFS and similar heuristic search algorithms can find policies evaluating only part of the state-space. IDFS is currently the most effective algorithm to solve FOND planning tasks. Thus, our algorithm IDFS will likely impact applications that solve goal-directed MDP tasks with compact descriptions.

Acknowledgments This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, UFRGS, CNPq and FAPERGS with projects 17/2551-0000867-7 and 21/2551-0000741-9.

References

Geffner, T. and Geffner, H. (2018). Compact policies for fully observable non-deterministic planning as SAT. In *International Conference on Automated Planning and Scheduling*.

Hansen, E. A. and Zilberstein, S. (2001). LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence*, 129(1-2).

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2).

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26.

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

Muise, C., McIlraith, S. A., and Beck, J. C. (2012). Improved Non-deterministic Planning by Exploiting State Relevance. In *International Conference on Automated Planning and Scheduling*.

Russell, S. (1992). Efficient Memory-Bounded Search Methods. In *European Conference on Artificial Intelligence*.

Sen, A. K. and Bagchi, A. (1989). Fast Recursive Formulations for Best-First Search That Allow Controlled Use of Memory. In *International Joint Conference on Artificial Intelligence*.

Stern, R., Kulberis, T., Felner, A., and Holte, R. (2010). Using Lookaheads with Optimal Best-First Search. In *AAAI Conference on Artificial Intelligence*.

Sturtevant, N. and Helmert, M. (2019). Exponential-Binary State-Space Search. arxiv.org/abs/1906.02912.

Yoshizumi, T., Miura, T., and Ishida, T. (2000). A* with Partial Expansion for Large Branching Factor Problems. In *AAAI Conference on Artificial Intelligence*.