# PEA*+IDA*: An Improved Hybrid Memory-Restricted Algorithm

**Frederico Messa, André Grahl Pereira**

Federal University of Rio Grande do Sul, Brazil
{frederico.messa, agpereira}@inf.ufrgs.br

## Abstract

It is well-known that the search algorithms A* and Iterative Deepening A* (IDA*) can fail to solve state-space tasks optimally due to time and memory limits. The former typically fails in memory-restricted scenarios and the latter in time-restricted scenarios. Therefore, several algorithms were proposed to solve state-space tasks optimally using less memory than A* and less time than IDA*, such as A*+IDA*, a hybrid memory-restricted algorithm that combines A* and IDA*. In this paper, we present a hybrid memory-restricted algorithm that combines Partial Expansion A* (PEA*) and IDA*. This new algorithm has two phases, the same structure as the A*+IDA* algorithm. The first phase of PEA*+IDA* runs PEA* until it reaches a memory limit, and the second phase runs IDA* without duplicate detection on each node of PEA*'s `Open`. First, we present a model that shows how PEA*+IDA* can perform better than A*+IDA* although pure PEA* usually makes more expansions than pure A*. Later, we perform an experimental evaluation using three memory limits and show that, compared to A*+IDA* on classical planning domains, PEA*+IDA* has higher coverage and expands fewer nodes. Finally, we experimentally analyze both algorithms and show that having higher $F$-limits and better priority-queue composition given by PEA* have a considerable impact on the performance of the algorithms.

## Introduction

A* (Hart, Nilsson, and Raphael 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 optimally using less memory than A* and making fewer node expansions than IDA*, such as MREC (Sen and Bagchi 1989), MA* (Chakrabarti et al. 1989), SMA* (Russell 1992), SMAG* (Kaindl and Khorsand 1994), BAI (Kaindl et al. 1995), BIDA* (Manzini 1995), AL* (Stern et al. 2010; Bu et al. 2014), and PEA* (Yoshizumi, Miura, and Ishida 2000). Some of them have a high polynomial-time overhead per node expansion or generation compared to A*, such as MA*, SMA*, and SMAG*. 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* (Yoshizumi, Miura, and Ishida 2000) 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 memory required. A*+IDA* can achieve speed-ups around five times over IDA* for specific domains. Bu and Korf explain that the main advantage of A*+IDA* is that it starts performing IDA* iterations from nodes with higher depth than the pure IDA* algorithm (which starts from the root node). They assert that avoiding some IDA* iteration is less impactful since the last two layers of IDA* iterations dominate the search. Although the A*+IDA* algorithm avoids failing due to memory limits, its second phase still has the drawbacks of the pure IDA* algorithm.

In this paper, we propose the use of the Partial Expansion A* (PEA*) as the first phase algorithm (instead of A*), 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 present a model that shows how PEA*+IDA* can perform better than A*+IDA*. We compare the PEA*+IDA* algorithm with the A*+IDA* algorithm on several domains of the International Planning Competition (IPC) at three different memory limits. The experiments show a reduction in the total number of expansions and an increase in coverage. We also analyze which aspects can yield speed-ups of PEA*+IDA* over A*+IDA* and we found that the $F$-values of the nodes in Open and its node composition are important aspects. Our analysis generally improves the understanding of hybrid memory-restricted algorithms and presents new research directions for efficient hybrid algorithms.

## Background

**State-Space Search** 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 a partial function of *transitions* between states, $c : A \rightarrow \mathbb{R}_{\geq 0}$ is a *cost function* that maps actions to non-negative real costs, $s_0 \in S$ is the *initial state* and $S_G \subseteq S$ is the set of *goal states*. A *solution* of $\Theta$ is a path of transitions $\pi = \langle \langle s_0, a_0, s_1 \rangle, \langle s_1, a_1, s_2 \rangle, \ldots, \langle s_{n-1}, a_{n-1}, s_n \rangle \rangle$ with $s_n \in S_G$ and $T(s_i, a_i) = s_{i+1}, \forall i \in [0, n-1]$. It is optimal if its cost $\sum_{i=0}^{n-1} c(a_i)$ is minimal. A *heuristic function* $h : S \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ 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). We assume the search algorithms have access to the state-space task through a *black-box* interface, i.e., they do not have access to a declarative representation of the task. The *black-box* interface provides the following methods: `make_root()` generates a node $n_0$ with the initial state $s_0$, `is_goal(n)` tests if $n$ contains a goal state, `extract_path`$(n)$ generates the path of transitions from $s_0$ to $n.state$ and `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 `succ`$(n)$ is invoked, the node $n$ is *expanded* and all its children are *generated*.

**A* Algorithm** The A* algorithm (Hart, Nilsson, and Raphael 1968) processes first nodes in Open with least $f$-value. It initializes the Open list with the root node $n_0$ and repeatedly removes Open nodes until it removes a node that contains a goal state. At each iteration, it removes a node $n$, generates the nodes children of $n$, and adds $n$ with its $g$-value to Closed. For each generated child $n'$, if $n'.state \notin$ Open and $n'.state \notin$ Closed, then $n'$ is inserted in Open. If $n'.state \in$ Open and $n'.g <$ Open$(n.state).g$, then its $g$-value and parent are updated. If $n'.state \in$ Closed and $n'.g <$ Closed$(n.state).g$, then $n'.state$ is removed from Closed and $n'$ is inserted in Open.

**Iterative Deepening A* Algorithm** The Iterative Deepening A* (IDA*) (Korf 1985) algorithm performs iterations bounded by an increasing $f$-limit. At each iteration, starting from the root node, IDA* expands nodes recursively discarding generated nodes with $f$-values greater than the current $f$-limit. If a node containing a goal state with $f$-value equal to the $f$-limit is generated during an iteration, the algorithm terminates finding a solution. At the end of the iteration, the minimal $f$-value among generated discarded nodes is set to be the next $f$-limit (if there is at least one lower than $\infty$, otherwise the search ends by task unsolvability). IDA* is a linear-space search algorithm. The trade-off is that it may frequently expand nodes with the same states and requires multiple re-expansions of the same nodes. Transposition Tables (TTs) (Reinefeld and Marsland 1994; Akagi, Kishimoto, and Fukunaga 2010) and other methods use extra memory to reduce the IDA* number of re-expansions aiming to approximate the performance of A*. We assume IDA* prunes cycles in expanded paths and that it process children sorted by lower $f$-value and lower $h$-value.

**A*+IDA* Algorithm** The A*+IDA* algorithm (Bu and Korf 2019) has two phases. The first phase runs A* until it finds a solution or reaches a memory limit. If A* reaches a memory limit, A*+IDA* starts the second phase. The second phase removes a node $n$ from Open, using the A* order, and performs an IDA* iteration starting from $n$ and using as $f$-limit $n.f$. If the iteration finds a solution, the search ends. Otherwise, the node $n$ is inserted in Open with $f$-value updated. The new $f$-value of $n$ is the new $f$-limit returned by IDA*. This process repeats until a solution is found. A*+IDA* is a memory-restricted algorithm that finds optimal solutions for states-space tasks using specific memory limits. Unfortunately, A*+IDA*'s second phase has the drawbacks of being an IDA* search. However, Bu and Korf reported that A*+IDA* is empirically superior to other methods such as TT for specific domains.

**Partial Expansion A* Algorithm** Partial Expansion A* (PEA*) (Yoshizumi, Miura, and Ishida 2000) is an algorithm that reduces the A*'s Open memory consumption with a trade-off of possibly requiring multiple re-expansions of nodes. PEA* processes first nodes with least $F$-value instead of a node with least $f$-value. The $F$-value of a node is defined to be equal to its $f$-value until it is updated to another value. When expanding a node $n$, the algorithm discards all its children that have $F$-values greater than its $n.F$. PEA* re-inserts $n$ in Open with $F$-value updated to the minimal finite $F$-value of the discarded children if there is at least one. Otherwise, the node $n$ is inserted in Closed. The original version of PEA* allows defining a parameter $C$, such that PEA* only discards children of a node $n$ that have $F$-values greater than $n.F + C$. We assume $C = 0$, which is most frequently used. When $C = 0$, no node with $f$-value greater than $h^*(s_0)$ is ever stored. Goldenberg et al. (2014) improved PEA* by making it avoid generating the children nodes that will be discarded, when using specific heuristics or solving specific domains. In this paper, we don't evaluate the enhanced version of PEA* (EPEA*) because we assume the *black-box* interface model and a general heuristic.

**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 return extract_path(n)
6  |  Children_≤ := {n' | n' ∈ succ(n) ∧ n'.F ≤ n.F}
7  |  Children_> := {n' | n' ∈ succ(n) ∧ n'.F > n.F}
8  |  if |Open| + |Children_≤| + min(|Children_>|, 1) >
   |    MEMORY_LIMIT then
9  |  |  Insert n in Open
10 |  |  break
11 |  foreach n' ∈ Children_≤ do process_child(n')
12 |  n.F := min{n'.F | n' ∈ Children_>}
13 |  if n.F = ∞ then Insert n in Closed
14 |  else if |Children_>| = 1 then
15 |  |  process_child(n') | n' ∈ Children_>
16 |  |  Insert n in Closed
17 |  else Insert n in Open
   /* Second Phase: IDA*                             */
18 while Open ≠ ∅ do
19 |  Remove node n from Open with minimum n.F
20 |  solution_path, new_F_limit := IDA*(n, n.F)
21 |  if solution_path ≠ ⊥ then return solution_path
22 |  if new_F_limit = ∞ then Insert n in Closed
23 |  else
24 |  |  n.F := new_F_limit
25 |  |  Insert n in Open
26 return ⊥ // UNSOLVABLE
27 Method process_child(n'):
28 |  if n'.state ≠ n.state then
29 |  |  if n'.state ∈ Open then
30 |  |  |  if n'.g < Open(n'.state).g then
31 |  |  |  |  Open(n'.state).update(n'.parent, n'.g, n'.F)
32 |  |  else if n'.state ∈ Closed then
33 |  |  |  if n'.g < Closed(n'.state).g then
34 |  |  |  |  Remove n'.state from Closed
35 |  |  |  |  Insert n' in Open
36 |  |  else Insert n' in Open
```

## PEA*+IDA* Algorithm

In this section, we introduce the PEA*+IDA* hybrid memory-restricted algorithm. We show its high-level description and how to modify it to use as A*+IDA*, or as the pure algorithms PEA*, A*, or IDA*. We then present a proof sketch of its soundness and completeness. Finally, we present a model that shows how PEA*+IDA* can perform better than A*+IDA*. The PEA*+IDA* algorithm has two phases. The first phase runs PEA* until it reaches a memory limit. Our aim to use PEA* as the first phase algorithm is to reduce the drawbacks of its IDA* phase. Using PEA* instead of A* may extend the first phase, since PEA* reduces the Open size of A*.

### High-Level Description

Algorithm 1 shows PEA*+IDA* with its two phases: PEA* (lines 3–17) and IDA* (lines 18–25).

**First Phase (lines 3–17)** PEA*+IDA* removes from Open first a node $n$ with least $F$-value (line 4) and not least $f$-value. Note that the $F$-value of a node can be updated through the execution of the algorithm. When expanding the node $n$, the algorithm divides the generated children nodes from succ($n$) into two sets $\texttt{Children}_{\leq}$ and $\texttt{Children}_{>}$. The set $\texttt{Children}_{\leq}$ (line 6) stores nodes with $F$-values lower or equal to $n.F$. The set $\texttt{Children}_{>}$ (line 7) stores nodes $F$-values greater than $n.F$. PEA*+IDA* terminates the first phase (lines 8–10) if the memory (Open size) required to expand the node $n$ is greater than the predetermined limit. Line 11 invokes the typical method of A* that processes generated nodes in $\texttt{Children}_{\leq}$. Lines 12–17 process $\texttt{Children}_{>}$, if there is no child with finite $F$-value greater than $n.F$, then the node $n$ is inserted in Closed. Otherwise, if there is more than one node in $\texttt{Children}_{>}$ the node $n$ is re-inserted in Open with $F$-value equals to the minimum finite $F$-value of nodes in $\texttt{Children}_{>}$. We propose a minor modification of the original PEA* that reduces expansions of PEA*+IDA* in our experiments: if $\texttt{Children}_{>}$ has only one child node $n'$, then it is processed as a child node with $F$-value lower or equal to $n.F$, and the node $n$ is then inserted in Closed. We do that because in this case processing $n'$ and closing $n$ completes the expansion without changing the overall Open size.

**Second Phase (lines 18–25)** PEA*+IDA* again removes from Open first a node $n$ using the same order from the first phase. Line 20 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.

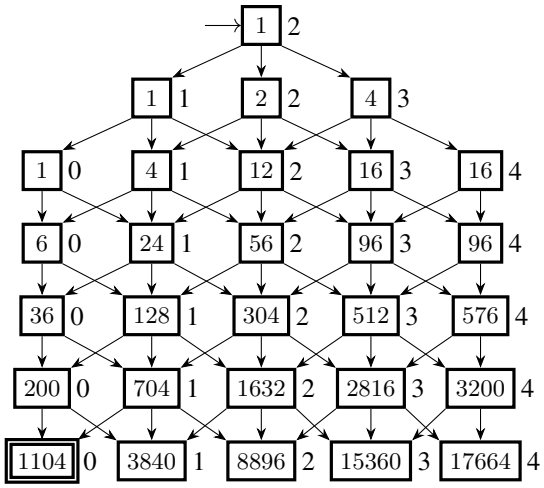
**Obtaining Other Algorithms** We can obtain other algorithms by performing minor changes in PEA*+IDA*. To obtain IDA*, we can set MEMORY_LIMIT to zero since PEA*+IDA* would fail to make an expansion in the first phase, going then straight to the second phase. To obtain PEA*, we can set MEMORY_LIMIT to $\infty$ since it would never go to the second phase. To obtain the A*+IDA* algorithm, it is sufficient to insert all children nodes into $\texttt{Children}_{\leq}$, instead of splitting them into $\texttt{Children}_{\leq}$ and $\texttt{Children}_{>}$. Lastly, to obtain the A* algorithm is sufficient to simultaneously perform both the conversion procedures to obtain PEA* and obtain A*+IDA*.

### Soundness and Completeness

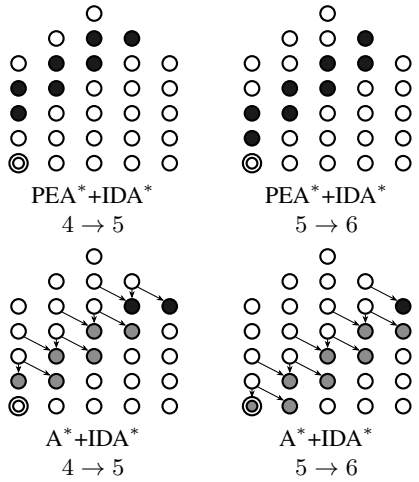
Here we present a proof sketch of the soundness and completeness of the PEA*+IDA* algorithm. It does not consider all effects of having the Open and Closed duplicate detection due to space constraints, although we could prove it does not harm the algorithm soundness and completeness.

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

*Proof sketch.* Before any Open node removal, there is a node $n$ in Open such that there is an optimal solution including $n$ with cost at least $n.F$. This is initially valid be-




(a) Example state-space task.




(b) Illustration of the Open of the hybrid algorithms.

Figure 1: On the left: an example of a state-space task. 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. On the right: an illustration of the node composition of the A*+IDA* and PEA*+IDA*'s Open lists.

cause at the start $n_0$ is in Open with $n_0.F = n_0.f$. The property remains valid until $n$ is removed from Open. When $n$ is removed in the first phase, if $n$ is a goal, the search successfully ends. Otherwise, $n$ has a child $n'$ with an optimal solution including $n'$. If $n'.F = n'.f > n.F$, then $n$ will be re-inserted with $n.F$ increased to some value at most $n'.f$, and since $n'.f$ is a lower bound to the optimal solution cost, the new $n.F$ also would be, maintaining the property. If $n'.F = n'.f \leq n.F$, $n'$ would be simply inserted in Open, thus also maintaining the property. If $n$ is removed in the second phase and a solution is not found, the *new-F-limit* assigned to $n.F$ is still a lower bound to the optimal solution cost, maintaining the property. So, the property remains true during the whole search. It remains to be proved that the algorithm terminates. The first phase terminates because eventually all states with finite $h$-value will be stored in Closed and because the number of times that the $F$-value of a node can be updated is finite. The second phase terminates because nodes that form a cycle are pruned. Thus each iteration of IDA* terminates expanding at most nodes with depth $|S|$. ■

## Open Size and Composition Model

We now present a simplified model (due to space constraints) 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 paper, 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 $\gamma_1$ non-duplicated children with $h$-value equals to $n.h-1$, $\gamma_2$ with $h$-value equals to $n.h$, and $\gamma_3$ 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.

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

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 $x = \mathbf{x}$ using Equation 2 we can compute the number of nodes in PEA*+IDA*'s Open.

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

For the A*+IDA* algorithm, the nodes in Open are the ones with $f$-values at 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. For $x = \mathbf{x}$ using Equation 3 we can compute the number of nodes in A*+IDA*'s Open.

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

**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.

Therefore, 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 shows in black, in the lower quadrants, the node-sets in A*+IDA*'s Open respectively for $x = 4$ and $x = 5$, and in gray, the node-sets partly in A*+IDA*'s Open also respectively for $x = 4$ and $x = 5$.

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.

## Empirical Analysis

In this section, we aim to understand better A*+IDA* and PEA*+IDA*. Thus, we compare them using three different memory limits. We measure time as the number of expanded nodes because it avoids differences that result from implementation details. Moreover, since the memory consumption of these algorithms is mainly given by their Open and Closed lists, but the relative cost of each one depends on implementation details, we measure memory consumption using the number of nodes stored in Open instead of real memory. Taking also into account the Closed size would increase the advantage of PEA*+IDA* since, as we will show, it typically has a significantly smaller Closed than A*+IDA*. In addition, Open usually grows faster and consumes more memory per node than Closed.

We use the STRIPS (Nilsson and Fikes 1971) optimal benchmark of 1877 tasks of the International Planning Competition (IPC). We obtain the memory limits by solving tasks using pure A* with $h^{\text{LMCut}}$ (Helmert and Domshlak 2009) saving the peak of number of nodes in A*'s Open for each solved task. We remove from our experiments tasks that are either too "hard" or too "easy", i.e., not solved by pure A* with $h^{\text{LMCut}}$ in 10 minutes with 2 GB of memory, or solved by pure A* with blind heuristic function with the same limits. 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.

A* with $h^{\text{LMCut}}$ does not solve 927 tasks, 123 failed by memory, 800 by time, and four by being unsolvable, fully removing *Agricola* and *Childsnack* domains both with 20 tasks. The 800 tasks that failed by time in pure A* should not be solved by any other algorithm (unless tie-breakers luckily benefit some algorithm in some task). The other algorithms may solve the 123 tasks that failed by memory, but we removed them because we do not have the Open size peak information. A* with the blind heuristic solves 618 tasks, and fails by memory in 332 of the 950 remaining tasks, removing the domains (with their respective number of tasks in parenthesis): *Barman* (34), *Gripper* (20), *Hiking* (20), *Movie* (30), *Openstacks* (100), *Pegsol* (50), *Snake* (20), and *Tetris* (17).

We compare the algorithms using the remaining 332 tasks limiting the Open size to 10%, 50%, and 90% of the A*'s Open size peak. We use $h^{\text{LMCut}}$ in all the remaining experiments. Note that tie-breakers may cause A* to have more expansions than some of the hybrid algorithms. Also, note that the hybrid algorithms' Open second phase node removals are not expansions and that the total number of expansions of the hybrid algorithms account for all expansions made during each IDA* iteration.

**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. At 10% there are nine tasks that only PEA*+IDA* failed to solve, and 11 tasks that only A*+IDA* failed to solve. At 50% and 90%, only PEA*+IDA* failed respectively on four and zero tasks, while only A*+IDA* failed respectively on 43 and 43 tasks. 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, and removing the domains (with their number of tasks in parenthesis) *Elevators* (50), *Freecell* (80), *Pathways* (30), *Petri* (20), *PSR* (50), *Termes* (20), *TPP* (30) and *Transport* (70). Table 1 shows as $(x/y)$ the total number $y$ and the number $x$ of the remaining tasks of each domain after all filtering. The experiments further made did not result in any new failure or removal. Table 1 shows per-domain geometric mean expansions for each algorithm and memory limit. It also shows the expansions of pure A* for reference. To compute the expansions, we increment by one the values before doing

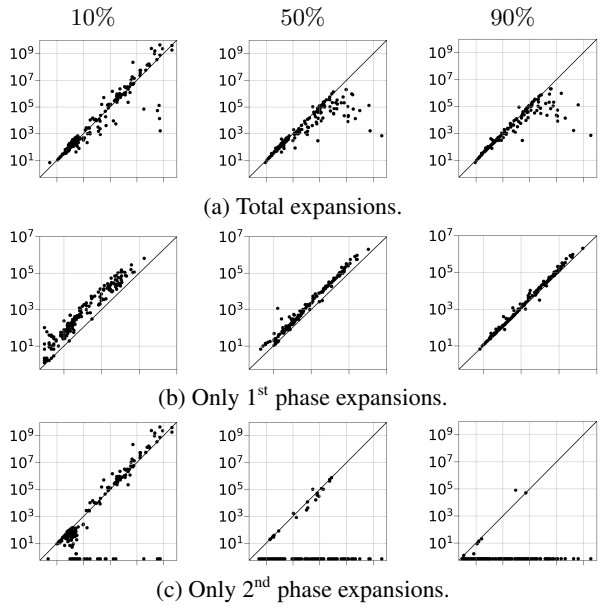
| | 10% | | 50% | | 90% | | 100% |
|---|---|---|---|---|---|---|---|
| | A*+IDA* | PEA*+IDA* | A*+IDA* | PEA*+IDA* | A*+IDA* | PEA*+IDA* | A* |
| *Airport* (2/50) | 550 | **188** | **203** | 223 | 357 | **223** | 225 |
| *Blocks* (10/35) | **240,000** | 303,167 | **80,140** | 88,138 | **68,379** | 88,138 | 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 |
| **Avg. Node Generations** | 1,814,141.82 | **1,608,987.07** | 483,384.55 | **92,846.75** | 280,722.73 | **85,429.31** | 66,580.69 |
| **Avg. `Closed` Size Peak** | 429.05 | **120.63** | 2,387.17 | **368.76** | 4,853.33 | **415.08** | 5,683.18 |
| **Coverage** | 247 | **249** | 267 | **306** | 275 | **318** | |

Table 1: Coverage, expansions, node generations and `Closed` size for hybrid algorithms at three memory limits.

the mean (and decremented by one after) to deal with zero values of IDA* expansions. 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. 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.

Figure 2 shows the number of expansions for each task of Table 1. It shows that for most tasks, the first phase of PEA*+IDA* is, as expected, extended, especially at 10% where memory is critical, and that few tasks require the second phase at higher memory limits. Figure 2 also shows that outliers often occur for A*+IDA*, having much more total expansions. In some tasks, it requires more than 10,000 times more expansions than PEA*+IDA*. We believe that tasks which A*+IDA* failed to solve would have significantly more expansions than the tasks that PEA*+IDA* failed to solve. However, since those tasks were removed from all Figures and Tables, and running them to the end could be prohibitive, 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.

Table 1 also shows the mean number of nodes generated by the hybrid algorithm at the three memory limits, and their `Closed` size peaks. On average PEA*+IDA* has a smaller `Closed` size peak supporting the claim that the advantage of PEA*+IDA* would increase if we were to consider also the `Closed` memory consumption. PEA*+IDA* has also great advantage regarding node generations, which would be even greater if EPEA* was used, due to its capability of reducing PEA*'s node generations. Table 2 displays geometric mean




(a) Total expansions.

(b) Only 1st phase expansions.

(c) Only 2nd phase expansions.

Figure 2: PEA*+IDA* (vertical axis) vs. A*+IDA* (horizontal axis) number of expansions for each task. Note that the horizontal scales are equal to the vertical scales.

second phase information over domains and tasks of Table 1 for the hybrid algorithms, showing that PEA*+IDA* dramatically reduces the number of second phase expansions and iterations.

**Better Open Composition** We now analyze information about $F$-values of nodes in Open when memory reaches the limit (requiring the second phase). We focus on the memory limit of 10% since it provides the highest number of tasks that both algorithms solve reaching the memory limit. The minimum, mean, maximum $F$-values in Open for A*+IDA* are respectively 37.18, 40.28, and 45.03 while for PEA*+IDA* are 39.34, 41.14, and 44.90. The mean cost the solution of those tasks is 42.40, and the mean solution length is 28.12 for A*+IDA* and 28.37 for PEA*+IDA*. The percentage of nodes with minimum $F$-values for A*+IDA* is 18% and for PEA*+IDA* is 30%. Therefore, 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.

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

Table 2: Mean second phase number of expansions and iterations for hybrid algorithms, and A*+IDA*$^{\uparrow}$.

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.

## Conclusion and Future Work

In this paper, we proposed an improved hybrid memory-restricted algorithm combining PEA* and IDA*. We showed that we could increase the minimum $F$-value in the Open at fixed memory limits by using PEA* instead of A* as the first phase algorithm. Our experiments show that PEA*+IDA* reduces the number of IDA* iterations and expansions, generally reducing the number of total expansions and increasing the coverage. Our analysis shows that higher minimum $F$-values do not entirely explain the improvement obtained by the algorithm and that the Open composition is also an important aspect. We plan to refine our model to understand better each component of hybrid memory-restricted algorithms in the future. Also, we plan to investigate further the role of the Open composition in the performance of IDA* iterations, and how to improved the second phase with techniques such as Transposition Tables.

## Acknowledgments

André G. Pereira acknowledges support from FAPERGS with projects 17/2551-0000867-7 and 21/2551-0000741-9. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. We thank UFRGS, CNPq and FAPERGS for partially funding this research.

## References

Akagi, Y.; Kishimoto, A.; and Fukunaga, A. 2010. On Transposition Tables for Single-Agent Search and Planning: Summary of Results. In *Symposium on Combinatorial Search*, 2–9.

Bu, Z.; and Korf, R. E. 2019. A*+IDA*: A Simple Hybrid Search Algorithm. In *International Joint Conference on Artificial Intelligence*, 1206–1212.

Bu, Z.; Stern, R.; Felner, A.; and Holte, R. C. 2014. A* with Lookahead Re-Evaluated. In *Symposium on Combinatorial Search*, 15–17.

Chakrabarti, P. P.; Ghose, S.; Acharya, A.; and de Sarkar, S. 1989. Heuristic Search in Restricted Memory. *Artificial Intelligence*, 41(2): 197–221.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; C. Holte, R.; and Schaeffer, J. 2014. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research*, 50: 141–187.

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): 100–107.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *International Conference on Automated Planning and Scheduling*, 162–169.

Kaindl, H.; Kainz, G.; Leeb, A.; and Smetana, H. 1995. How to Use Limited Memory in Heuristic Search. In *International Joint Conference on Artificial Intelligence*, 236–242.

Kaindl, H.; and Khorsand, A. 1994. Memory-bounded bidirectional search. In *AAAI Conference on Artificial Intelligence*, 1359–1364.

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

Korf, R. E.; and Reid, M. 1998. Complexity Analysis of Admissible Heuristic Search. In *AAAI Conference on Artificial Intelligence*, 305–310.

Manzini, G. 1995. BIDA*: An Improved Perimeter Search Algorithm. *Artificial Intelligence*, 75(2): 347–360.

Nilsson, N. J.; and Fikes, R. E. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Reinefeld, A.; and Marsland, T. 1994. Enhanced Iterative-Deepening Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7): 701–710.

Russell, S. 1992. Efficient Memory-Bounded Search Methods. In *European Conference on Artificial Intelligence*, 1–5.

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*, 297–302.

Stern, R.; Kulberis, T.; Felner, A.; and Holte, R. 2010. Using Lookaheads with Optimal Best-First Search. In *AAAI Conference on Artificial Intelligence*, 185–190.

Sturtevant, N.; and Helmert, M. 2019. Exponential-Binary State-Space Search. arxiv.org/abs/1906.02912. arXiv: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*, 923–929.