A Formalism for Optimal Search with Dynamic Heuristics

Remo Christen, Florian Pommerening, Clemens Büchner, Malte Helmert
University of Basel, Switzerland
{remo.christen, florian.pommerening, clemens.buechner, malte.helmert}@unibas.ch

Abstract

While most heuristics studied in heuristic search depend only on the state, some accumulate information during search and thus also depend on the search history. Multiple existing approaches use such dynamic heuristics in A*-like algorithms and appeal to classic results for A* to show that they return optimal solutions. However, doing so disregards the intricacies of searching with a mutable heuristic. We treat dynamic heuristics formally and propose a framework that defines how the information dynamic heuristics rely on can be modified. We use these transformations in a generic search algorithm and an instantiation that models A* with dynamic heuristics, allowing us to provide general conditions for optimality. We show that existing approaches fit our framework and apply our results. Doing so for future applications of dynamic heuristics may simplify formal arguments for optimality.

Introduction

Heuristic search with A* is a canonical approach to finding optimal solutions in transition systems. Heuristic functions evaluate states to guide the search and typically map states to numeric values. In this work, we consider the more general class of *dynamic heuristics* that additionally depend on information procured during search (e.g., Gelperin 1977; Mérő 1984; Koyfman et al. 2024).

Classic optimality results for A* (e.g., Hart, Nilsson, and Raphael 1968; Dechter and Pearl 1985) cannot be directly applied to search with such dynamic heuristics, as their proofs assume static heuristics. Nevertheless, existing approaches to classical planning that use dynamic heuristics, such as LM-A* (Karpas and Domshlak 2009), LTL-A* (Simon and Röger 2015), or online abstraction refinement (Eifler and Fickert 2018; Franco and Torralba 2019), are claimed to return optimal solutions by referring to static notions of admissibility and thereby implicitly relying on classic results, without taking the interaction between search and heuristic into account. This is further complicated by discussions of additional properties such as monotonically increasing heuristic values and A* with re-evaluation, a modification that re-inserts states popped from the open list if their heuristic value improved while they where queued. Connections to consistency and reopening are implied, but their impact on the optimality of solutions is not formally proven.

Koyfman et al. (2024) show more formally that their A*-based approach guarantees optimal solutions while using a dynamic heuristic. They achieve this result by showing that their heuristic has a property called path-dynamic admissibility (PDA). However, PDA has strong requirements about the behavior of the search. Therefore the complexity inherent to the interaction between a dynamic heuristic and the search must be addressed while showing that the heuristic has the PDA property, making it difficult to translate the result to other approaches.

We aim to prove these results in a more general fashion by first formalizing the information that dynamic heuristics rely on as a mutable object together with functions to transform that object. We then define a generic algorithm framework for heuristic forward search that models when information can be transformed, define an instantiation of the framework based on A*, and show under what conditions it guarantees optimal solutions. These conditions rely on extensions of classic heuristic properties to the dynamic case that do not depend on search behavior. Additionally, we show conditions where no reopening occurs in the algorithm.

Finally, we apply these results to classical planning approaches that use dynamic heuristics, such as online abstraction refinement where abstractions are improved while the search is running. We also briefly consider other search strategies that do not use dynamic heuristics as such, but whose behavior can be modeled by one. This includes ways to consider multiple heuristics (e.g., Zhang and Bacchus 2012; Domshlak, Karpas, and Markovitch 2012; Tolpin et al. 2013), pathmax (Mérő 1984), path-dependent f-values (Dechter and Pearl 1985), deferred evaluation (Helmert 2006), and partial expansion A* (Yoshizumi, Miura, and Ishida 2000).

Transition Systems

We consider transition systems $\mathcal{T} = \langle S, L, c, T, s_0, S_* \rangle$, where S is a finite set of *states*; L is a finite set of *labels*; $c\colon L \to \mathbb{R}_{\geq 0}$ is a *cost function* assigning each label a cost; the set $T \subseteq S \times L \times S$ contains labeled *transitions* $\langle s, \ell, s' \rangle$; $s_0 \in S$ is the *initial state*; and $S_* \subseteq S$ are *goal states*. For a transition $\langle s, \ell, s' \rangle \in T$, we call s the *origin*, ℓ the *label*, and s' the *target* of the transition, and call s' a *successor* of s.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

$$\begin{tikzcd} {} \arrow[r] & \boxed{A} \arrow[r, "x"] \arrow[rr, bend right, "y"'] & \boxed{B} \arrow[r, "y"] & \boxed{C} \arrow[r, "x"] & \boxed{\boxed{D}} \end{tikzcd}$$

Figure 1: Transition system of our running example.

A *path* from s to s' is a sequence of transitions $\pi = \langle t_1, \dots t_n \rangle$ where the origin of t_1 is s, the origin of t_{i+1} is the target of t_i for $1 \leq i < n$, and the target of t_n is s'. A path to s' is a path from s_0 to s' and a *solution* (for s) is a path (from s) to some state in S_*. The cost of a path $\pi = \langle t_1, \dots t_n \rangle$ with $t_i = \langle s_i, \ell_i, s'_i \rangle$ is $c(\pi) = \sum_{i=1}^{n} c(\ell_i)$. A path (from s) to s' is *optimal*, if it has a minimal cost among all paths (from s) to s'. We denote the cost of an optimal path to s by $g^*(s)$. We say a state is *reachable* if a path to it exists and that $\mathcal{T}$ is *solvable* if it has a solution.

We assume that the transition system is encoded in a compact form (e.g., a planning task), where we can access the initial state, can generate the successors of a given state, and can check whether a given state is a goal state.

A (static) *heuristic* is a function $h\colon S \to \mathbb{R}_{\geq 0} \cup \{\infty\}$. The *perfect heuristic* h^* maps each state s to the cost of an optimal path from s to a goal state or to ∞ if no such path exists. A heuristic h is *admissible* if $h(s) \leq h^*(s)$ for all $s \in S$, and *consistent* if $h(s) \leq c(\ell) + h(s')$ for all $\langle s, \ell, s' \rangle \in T$. A heuristic is *safe* if $h(s) = \infty$ implies $h^*(s) = \infty$ and it is *goal-aware* if $h(s) = 0$ for all goal states $s \in S_*$.

Running Example In the following, we give intuitions for definitions and concepts based on the transition system $\mathcal{T} = \langle \{A, B, C, D\}, \{x, y\}, \{x \mapsto 1, y \mapsto 2\}, T, A, \{D\} \rangle$ with $T = \{\langle A, x, B \rangle, \langle A, y, C \rangle, \langle B, y, C \rangle, \langle C, x, D \rangle\}$. Figure 1 visualizes $\mathcal{T}$. We also need the concept of *landmarks* in our examples (e.g., Hoffmann, Porteous, and Sebastia 2004). In our context, a landmark for a state s is a label that occurs in every solution for s. For example, x is a landmark for C, while x and y are landmarks for A.

Dynamic Heuristics

Dynamic heuristics depend on information gained while searching for a solution. In our running example, this information is a partial function mapping states $s \in \{A, B, C, D\}$ to sets of landmarks from $2^{\{x,y\}}$. We call the set of all such functions the *information space* of our landmark example. In general, we consider dynamic heuristics to depend on information from an arbitrary information space.

Koyfman et al. (2024) introduced the notion of dynamic heuristics but in their definition, the information can change arbitrarily between heuristic evaluations. We extend their definition by considering more structured sources of information where the possible modifications of information are limited.

Definition 1. *An* information source *src for a transition system with states S and transitions T consists of an information space $\mathcal{I}_{src}$, and the following constant and functions:*

$$\text{initial-info}_{src} \in \mathcal{I}_{src},$$
$$\text{update}_{src} \colon \mathcal{I}_{src} \times T \to \mathcal{I}_{src} \text{ and}$$
$$\text{refine}_{src} \colon \mathcal{I}_{src} \times S \to \mathcal{I}_{src}.$$

Information is initialized to initial-info$_{src}$ and then modified with update$_{src}$ and refine$_{src}$. In our example, we could start in a situation where we only know the landmarks of the initial state: $\mathit{info}_0 = \text{initial-info}_{LM} = \{A \mapsto \{x, y\}\}$. This information can be updated by considering transition $t = \langle A, y, C \rangle$, thereby gaining information about C: while x must also be a landmark for C, the same is not true for y because the label of t is y. We can express this by $\mathit{info}_u = \text{update}_{LM}(\mathit{info}_0, t) = \{A \mapsto \{x, y\}, C \mapsto \{x\}\}$. Alternatively, we could refine info_0, for example by computing landmarks for B: $\mathit{info}_r = \text{refine}_{LM}(\mathit{info}_0, B) = \{A \mapsto \{x, y\}, B \mapsto \{x, y\}\}$. We consider expansion-based search algorithms, which can only refine on known states and update along transitions starting in known states. Such an algorithm can never refine info_0 on B without first updating on a transition to B. The following definition formalizes this.

Definition 2. *Let src be an information source for a transition system with states S and transitions T. An information object $\mathit{info}_n \in \mathcal{I}_{src}$ is called* reachable *if there are $e_1, \dots, e_n \in S \cup T$ and $\mathit{info}_0, \dots, \mathit{info}_n \in \mathcal{I}_{src}$ such that $\mathit{info}_0 = \text{initial-info}_{src}$ and for all $0 < i \leq n$*

- *if $e_i = s \in S$ then $\mathit{info}_i = \text{refine}_{src}(\mathit{info}_{i-1}, s)$,*
- *if $e_i = t \in T$ then $\mathit{info}_i = \text{update}_{src}(\mathit{info}_{i-1}, t)$, and*
- *states $e_i \in S$ and the origins of transitions $e_i \in T$ are either s_0 or the target of a transition e_j with $j < i$.*

In our example above, info_0 and info_u are reachable. For info_r the sequence with $e_1 = B$ does not show that it is reachable because B is not a target of a transition earlier in the sequence. Information info_r could still be reachable if the same information can be produced along a different path, say first updating info_0 along $\langle A, x, B \rangle$ and then refining the result on B. For the resulting information the sequence with $e_1 = \langle A, x, B \rangle$ and $e_2 = B$ shows reachability.

Dynamic heuristics are heuristics that depend on an information object in addition to a state.

Definition 3. *A* dynamic heuristic *over a transition system $\mathcal{T}$ with states S depends on an information source src for $\mathcal{T}$ and is a function*

$$h\colon S \times \mathcal{I}_{src} \to \mathbb{R}_{\geq 0} \cup \{\infty\}.$$

In our landmark example, a dynamic heuristic could estimate the cost of each state by the sum of the label cost of its known landmarks or by 0 if no landmarks are known. For example, $h_{LM}(A, \mathit{info}_0) = 3$ and $h(s, \mathit{info}_0) = 0$ for all $s \in \{B, C, D\}$. After updating the information along $\langle A, y, C \rangle$ the heuristic becomes more informed: $h_{LM}(A, \mathit{info}_1) = 3$, $h_{LM}(C, \mathit{info}_1) = 1$, and $h_{LM}(B, \mathit{info}_1) = h_{LM}(D, \mathit{info}_1) = 0$.

Static heuristics can be seen as the special case for a constant information source, i.e., one where update and refine do not modify the information. Let us now define the dynamic counterparts of common static heuristic properties.

Definition 4. *Let h be a dynamic heuristic over a transition system with states S and transitions T and over information source src. We say h is*

DYN-safe *if $h(s, \mathit{info}) = \infty$ implies $h^*(s) = \infty$ for all $s \in S$ and all reachable $\mathit{info} \in \mathcal{I}_{src}$;*

DYN-admissible *if $h(s, \mathit{info}) \leq h^*(s)$ for all $s \in S$ and all reachable $\mathit{info} \in \mathcal{I}_{src}$;*

DYN-consistent *if $h(s, \mathit{info}) \leq c(\ell) + h(s', \mathit{info})$ for all $\langle s, \ell, s' \rangle \in T$ and all reachable $\mathit{info} \in \mathcal{I}_{src}$; and*

DYN-goal-aware *if $h(s, \mathit{info}) = 0$ for all $s \in S_*$ and all reachable $\mathit{info} \in \mathcal{I}_{src}$.*

As in the static case, a DYN-goal-aware and DYN-consistent heuristic is DYN-admissible, and a DYN-admissible heuristic is DYN-safe and DYN-goal-aware. This can easily be shown by considering the static heuristics that result from fixing the individual reachable $\mathit{info} \in \mathcal{I}_{src}$.

If update$_{LM}$ and refine$_{LM}$ guarantee that all stored labels are landmarks for their states, then h_{LM} is DYN-admissible (and thus DYN-safe and DYN-goal-aware). It is not DYN-consistent because for example $h_{LM}(A, \mathit{info}_0) = 3 > 2 + 0 = c(y) + h_{LM}(C, \mathit{info}_0)$.

We also define a new property that describes that heuristic values can only increase over time.

Definition 5. *A dynamic heuristic h over a transition system with states S and transitions T and over information source src is* DYN-monotonic *if $h(s, \mathit{info}) \leq h(s, \text{update}(\mathit{info}, t))$ and $h(s, \mathit{info}) \leq h(s, \text{refine}(\mathit{info}, s'))$ for all reachable $\mathit{info} \in \mathcal{I}_{src}$, all $s, s' \in S$, and all $t \in T$.*

If update$_{LM}$ and refine$_{LM}$ guarantee that landmarks are never removed for any state, then h_{LM} is DYN-monotonic.

Monotonicity is particularly desirable for any DYN-admissible heuristic h as increasing values brings it closer to h^*. Fortunately, we can easily ensure monotonicity for h by using $h'(s, \mathit{info}) = \max(h(s, \mathit{info}), h(s, \mathit{info}'))$ where info' is the information encountered so far that has lead to the highest h-value for state s. Similar notions of monotonicity have previously been described by Eifler and Fickert (2018) and Franco and Torralba (2019). Note that this monotonicity notion is different from the monotonicity introduced by Pohl (1977), which is equivalent to consistency (Pearl 1984).

Progression-Based Heuristics

In our running example, information is only stored per state. While this is not a general requirement of dynamic heuristics, it is an interesting special case because it allows for simpler definitions, local reasoning, and, in practical implementations, efficient storage of information. All our theoretical results apply to dynamic heuristics in general but using this special case makes it more natural to talk about several existing techniques like LTL trajectory constraints (Simon and Röger 2015) or landmark progression (Büchner et al. 2023), which our running example is based on.

We first define how information for a single state is modified on a low level (Definition 6), and then use these transformations in the definition of a special type of information source (Definition 8).

Definition 6. *A* progression source *ps consists of an information space $\mathcal{I}_{ps}$, and the following constant and functions:*

$$\text{initial-state-info}_{ps} \in \mathcal{I}_{ps},$$
$$\text{progress}_{ps} \colon \mathcal{I}_{ps} \times T \to \mathcal{I}_{ps} \text{ and}$$
$$\text{merge}_{ps} \colon \mathcal{I}_{ps} \times \mathcal{I}_{ps} \to \mathcal{I}_{ps}.$$

For our running example we can define the progression source *LM* where $\mathcal{I}_{LM} = 2^{\{x,y\}}$ is the information space for a state and the initial state A is assigned initial-state-info$_{LM} = \{x, y\}$. The progression of a set of landmarks L for a state s along a transition $t = \langle s, \ell, s' \rangle$ are the labels progress$_{LM}(L, t) = L \setminus \{\ell\}$ because they are guaranteed to be landmarks for s' (while ℓ is not). If we get two sets of landmarks L_1, L_2 for the same state, their union merge$_{LM}(L_1, L_2) = L_1 \cup L_2$ is guaranteed to contain only landmarks for that state.

Beside our running example, we can also view two fundamental concepts in search, namely *g-values* and *parent pointers* of states, as a progression source.

Definition 7. *The* parent source *p is a progression source with $\mathcal{I}_p = \mathbb{R}_{\geq 0} \times (T \cup \{\bot\})$ and*

$$\text{initial-state-info}_p = \langle 0, \bot \rangle,$$
$$\text{progress}_p(\langle g, t \rangle, \langle s, \ell, s' \rangle) = \langle g + c(\ell), \langle s, \ell, s' \rangle \rangle, \text{ and}$$
$$\text{merge}_p(\langle g, t \rangle, \langle g', t' \rangle) = \begin{cases} \langle g, t \rangle & \text{if } g \leq g' \\ \langle g', t' \rangle & \text{otherwise.} \end{cases}$$

We now define information sources that update per-state information based on this procedure of progressing and merging. This special case does not allow refinement because we want it to be as restrictive as possible while still being useful.

Definition 8. *Let ps be a progression source. The* progression-based information source *src$_{ps}$ is an information source for a transition system with states S and transitions T where $\mathcal{I}_{src_{ps}}$ is the set of partial functions $\mathit{info} \colon S \rightharpoonup \mathcal{I}_{ps}$, and*

$$\text{initial-info} = \{s_0 \mapsto \text{initial-state-info}_{ps}\}$$
$$\text{update}(\mathit{info}, t) = \mathit{info}_t \quad \text{for all } \mathit{info} \in \mathcal{I}_{src_{ps}} \text{ and } t \in T$$
$$\text{refine}(\mathit{info}, s) = \mathit{info} \quad \text{for all } \mathit{info} \in \mathcal{I}_{src_{ps}} \text{ and } s \in S$$

where info_t for a transition $t = \langle s, \ell, s' \rangle$ is the same as info for all states other than s' and maps s' to

$$\begin{cases} \text{progress}(\mathit{info}(s), t) & \text{if } \mathit{info}(s') \text{ undefined} \\ \text{merge}(\text{progress}(\mathit{info}(s), t), \mathit{info}(s')) & \text{otherwise.} \end{cases}$$

With this definition we get *src$_{LM}$*, the progression-based information source built on the progression source *LM*. Progressing the initial set of landmarks $\{x, y\}$ along $\langle A, x, B \rangle$ gives the landmarks $\{y\}$ for B. Progressing this information along $\langle B, y, C \rangle$ gives the empty set of landmarks for C, but if we then also progress $\{x, y\}$ along $\langle A, y, C \rangle$ this gives new information $\{x\}$ for C which has to be merged with the existing information, so the information stored for C after the progression is $\emptyset \cup \{x\} = \{x\}$. The application section provides details on the full landmark progression technique.

Similarly, we also get src_p, a progression-based information source for g-values and parent pointers. Note that for a reachable parent information *info* with $\mathit{info}(s) = \langle g, t\rangle$, we can show by induction that following parent pointers back to $\bot$ yields a path to s with cost of at most g. This will be useful later.

Dynamic heuristics based on progression sources base the heuristic value of a state on its stored information.

Definition 9. *A* progression-based heuristic *is a dynamic heuristic with a progression-based information source src_{ps} where $h(s, \mathit{info}) = 0$ if $\mathit{info}(s)$ is undefined, and where $\mathit{info}(s) = \mathit{info}'(s)$ implies $h(s, \mathit{info}) = h(s, \mathit{info}')$ for all pairs $\mathit{info}, \mathit{info}' \in \mathcal{I}_{\mathit{src}_{ps}}$ and all states $s \in S$.*

Because of the way progression-based information sources are updated, progression-based heuristics are always 0 for states not yet discovered in the search, i.e., states s where $\mathit{info}(s)$ is undefined. These default values can easily violate DYN-consistency and we therefore consider a heuristic h to be *partially* DYN-*consistent* if $h(s, \mathit{info}) \le c(\pi) + h(s', \mathit{info})$ for all paths π from s to s' for which $\mathit{info}(s)$ and $\mathit{info}(s')$ are defined.

Dynamic Heuristic Search Framework

A typical approach to finding solutions in a transition system is to explore it sequentially starting from the initial state s_0. Algorithm 1 implements such a forward search and maintains information during the exploration. Besides the search direction, Algorithm 1 leaves choices such as the expansion order open. This captures a wide range of instantiations that may vary in how often they use, update or refine the information within the search.

The algorithm maintains one information object $\mathit{infos}[\mathit{src}]$ for each source *src*, initializes it at the start, updates it when exploring transitions, and optionally refines it for known states. By keeping track of known states in S_{known} it guarantees that exactly the reachable information according to Definition 2 can occur in the algorithm. If at some point a goal state is known, the problem is solvable. If not, and no new states can be reached, then it is unsolvable. A simple induction shows that S_{known} contains reachable states.

Lemma 1. *Every state in S_{known} in Algorithm 1 is reachable.*

We use this result to show that Algorithm 1 is sound.

Theorem 1. *Any instantiation $\mathcal{A}$ of Algorithm 1 is* sound.

Proof. Instantiation $\mathcal{A}$ only returns `solvable` if S_{known} contains a goal state s_*. Then Lemma 1 implies that s_* is reachable and the underlying problem is solvable.

Further, $\mathcal{A}$ only returns `unsolvable` if GENERATE UNKNOWN is not applicable (line 24) and thus that S_{known} contains all reachable states. Since no goal state is in S_{known} (line 23), this means no goal is reachable. □

We would also like instantiations of Algorithm 1 to terminate in a finite number of steps.

Theorem 2. *An instantiation $\mathcal{A}$ of Algorithm 1 is* complete *if all loops over* GENERATE KNOWN *and* REFINE *are finite.*

Algorithm 1: Dynamic Heuristic Search Framework

```
Input: transition system $\langle S, L, c, T, s_0, S_*\rangle$; set of information sources $\mathit{Srcs}$
Output: solvable or unsolvable
1  $\mathit{infos} \leftarrow \{\mathit{src} \mapsto \text{initial-info}_{\mathit{src}} \mid \mathit{src} \in \mathit{Srcs}\}$
2  $S_{\text{known}} \leftarrow \{s_0\}$
3  loop
4      choose applicable operation
5      switch (operation)
6      case GENERATE UNKNOWN:
7          choose $t = \langle s, \ell, s'\rangle \in T$ such that $s \in S_{\text{known}}$ and
               $s' \notin S_{\text{known}}$
8          for each $\mathit{src} \in \mathit{Srcs}$ do
9              $\mathit{infos}(\mathit{src}) \leftarrow \text{update}_{\mathit{src}}(\mathit{infos}(\mathit{src}), t)$
10         $S_{\text{known}} \leftarrow S_{\text{known}} \cup \{s'\}$
11     case GENERATE KNOWN:
12         choose $t = \langle s, \ell, s'\rangle \in T$ such that $s, s' \in S_{\text{known}}$
13         for each $\mathit{src} \in \mathit{Srcs}$ do
14             $\mathit{infos}(\mathit{src}) \leftarrow \text{update}_{\mathit{src}}(\mathit{infos}(\mathit{src}), t)$
15     case REFINE:
16         choose $s \in S_{\text{known}}$
17         for each $\mathit{src} \in \mathit{Srcs}$ do
18             $\mathit{infos}(\mathit{src}) \leftarrow \text{refine}_{\mathit{src}}(\mathit{infos}(\mathit{src}), s)$
19     case DECLARE SOLVABLE:
20         ensure that $S_{\text{known}}$ contains a goal state
21         return solvable
22     case DECLARE UNSOLVABLE:
23         ensure that $S_{\text{known}}$ does not contain a goal state
24         ensure that there is no $\langle s, \ell, s'\rangle \in T$ such that
               $s \in S_{\text{known}}$ and $s' \notin S_{\text{known}}$
25         return unsolvable
```

Proof. Whenever $\mathcal{A}$ chooses an operation, either GENERATE UNKNOWN, DECLARE SOLVABLE, or DECLARE UNSOLVABLE is applicable: if a transition $\langle s, \ell, s'\rangle \in T$ with $s \in S_{\text{known}}$ and $s' \notin S_{\text{known}}$ exists, then GENERATE UNKNOWN is applicable, otherwise either DECLARE SOLVABLE or DECLARE UNSOLVABLE is applicable depending on whether a goal state is contained in S_{known} or not.

Further, the number of times operation GENERATE UNKNOWN can be applied is bounded by $|S|$ as it requires $s' \notin S_{\text{known}}$ and adds s' to S_{known} when applied.

Since no infinite loops over GENERATE KNOWN and REFINE occur in $\mathcal{A}$, and because one of the remaining operations must be applicable, $\mathcal{A}$ must eventually terminate with DECLARE SOLVABLE or DECLARE UNSOLVABLE. □

Dynamic A*

Algorithm 2 instantiates the dynamic heuristic search framework as DYN-A*, a generalization of A* for dynamic heuristics that, as we will see, covers many existing techniques.

The algorithm depends on the progression-based information source src_p, i.e. the parent source p, to track g-values and parent pointers as well as on an information source src_h used by a dynamic heuristic h. Like regular A*, DYN-A* maintains *open*, a priority queue of states ordered by $f = g + h$, and *closed*, a hash set of states.

Since information changes over time, we use the counters i for iterations and j for steps within the iteration, to reference specific *times* $t = \langle i, j\rangle$. We then refer to information

Algorithm 2: DYN-A* with optional re-evaluation.

```
Input: transition system $\langle S, L, c, T, s_0, S_* \rangle$; set of $Srcs = \{src_p, src_h\}$; dynamic heuristic $h$ over $src_h$; flag re-eval
Output: solution or unsolvable
 1  $infos \leftarrow \{src \mapsto \text{initial-info}_{src} \mid src \in Srcs\}$
 2  $i \leftarrow 0$; $j \leftarrow 0$ ▷ counters for iterations and steps within them
 3  $S_{\text{known}} \leftarrow \{s_0\}$; $closed \leftarrow \{\}$
 4  $open \leftarrow$ new priority queue with elements
        $\langle \text{state}, g^{val}, h^{val} \rangle$, prioritizing lower $f = g^{val} + h^{val}$
 5  if $h^{i,j}(s_0) < \infty$ then
 6      insert $\langle s_0, g^{i,j}(s_0), h^{i,j}(s_0) \rangle$ into open
 7  while open is not empty do
 8      $i \leftarrow i + 1$; $j \leftarrow 0$
 9      $\langle s, g^{val}, h^{val} \rangle \leftarrow$ pop highest priority element from open
10      if $s \in closed$ then
11          continue
12      for each $src \in Srcs$ do
13          $infos(src) \leftarrow \text{refine}_{src}(infos(src), s)$
14      $j \leftarrow 1$
15      if re-eval and $h^{val} < h^{i,j}(s)$ then
16          if $h^{i,j}(s) < \infty$ then
17              insert $\langle s, g^{i,j}(s), h^{i,j}(s) \rangle$ into open
18          continue
19      $closed \leftarrow closed \cup \{s\}$
20      if $s$ is a goal state then
21          return extracted solution according to $src_p$
22      for each $t = \langle s, \ell, s' \rangle \in T$ do
23          if $s' \in S_{\text{known}}$ then
24              $old\text{-}g \leftarrow g^{i,j}(s')$
25          else
26              $old\text{-}g \leftarrow$ undefined
27          for each $src \in Srcs$ do
28              $infos(src) \leftarrow \text{update}_{src}(infos(src), t)$
29          $j \leftarrow j + 1$
30          $S_{\text{known}} \leftarrow S_{\text{known}} \cup \{s'\}$
31          if $h^{i,j}(s') = \infty$ then
32              continue
33          if $old\text{-}g =$ undefined then               ▷ first path to $s'$
34              insert $\langle s', g^{i,j}(s'), h^{i,j}(s') \rangle$ into open
35          else if $old\text{-}g > g^{i,j}(s')$ then        ▷ cheaper path to $s'$
36              if $s' \in closed$ then
37                  $closed \leftarrow closed \setminus \{s'\}$
38              insert $\langle s', g^{i,j}(s'), h^{i,j}(s') \rangle$ into open
39  return unsolvable
```

available to the algorithm at a particular time t by $infos^t$, which is unambiguous since i or j always change immediately after *infos* is modified. Similarly, we use $g^t(s)$ to refer to the g-value stored in $infos^t(src_p)$ for state s, and $h^t(s)$ to refer to $h(s, infos^t(src_h))$.

For example, the initial state s_0 is inserted into *open* in line 6 with $g^{0,0}(s_0)$ and $h^{0,0}(s_0)$, then popped in line 9 in the first iteration at time $\langle 1, 0 \rangle$. If its first successor s' is not pruned, it is inserted in line 34 with $g^{1,2}(s')$ and $h^{1,2}(s')$. We say $\langle i, j \rangle < \langle i', j' \rangle$ if $i < i'$ or if $i = i'$ and $j < j'$.

Entries in *open* are sorted by their f-value at time of insertion, not the current f-values of the contained states. Put differently, if a heuristic improves during the search, this does not directly affect entries already on *open*.

DYN-A* implements *delayed duplicate detection*, a technique often used in practice, e.g. in Fast Downward (Helmert 2006), that allows multiple entries for a state on *open* and later eliminates duplicates in line 10. The alternative is to detect duplicate states early (at the time of insertion) and update open list entries when a cheaper path is found. However, the optimal time complexity of this version relies on an efficient *decrease-key* operation which practical implementations of priority queues like binary heaps do not support.

Optionally, states whose heuristic value improved since being inserted into *open* can be *re-inserted* with their new heuristic value in line 17, an idea used by systems relying on dynamic heuristics (e.g., Karpas and Domshlak 2009; Zhang and Bacchus 2012; Eifler and Fickert 2018). We call this modification *re-evaluation* and study the algorithm with and without it. States that are neither detected as duplicates nor re-evaluated are *expanded*. If a goal state is expanded, a plan is constructed from the parent pointers stored in src_p, otherwise all successors are considered.

Successors with a heuristic value of ∞ are skipped (line 32) and unknown successors are inserted into *open* (line 34). If we find a new path to a known successor s', then s' is only placed on *open* if this new path is cheaper than the one we already knew (line 35). In case s' was already closed, it is *reopened* (line 37); We will later show that, with some restrictions on the heuristic, states are never reopened.

DYN-A* simulates A* if heuristic h is static, i.e., $h(s, info)$ is independent of *info*. Note that the value of *re-eval* has no effect on the algorithm in this case: re-evaluation never occurs because $h^{val} = h^{i,j}(s)$ in line 15.

Let us now connect DYN-A* to the generic framework.

Theorem 3. *The algorithm* DYN-A* *using a* DYN-*safe dynamic heuristic is an instantiation of the dynamic heuristic search framework.*

Proof. We first show that DYN-A* only modifies *infos* and S_{known} in ways allowed by the framework. The initialization in line 1 is identical. The loop in lines 12–13 corresponds to a REFINE operation. We know $s \in S_{\text{known}}$ at this time because only known states are added to *open*. The loop in lines 27–28 corresponds to GENERATE KNOWN if $s' \in S_{\text{known}}$ in line 23, and to GENERATE UNKNOWN otherwise, in which case s' is inserted into S_{known} in line 30.

Next, we have to show that DYN-A* only terminates in case the framework can terminate. Line 21 corresponds to DECLARE SOLVABLE which is applicable due to the condition in line 20 and because s was inserted into S_{known} before inserted into *open* (and now popped from there).

Lastly, in cases where line 39 returns `unsolvable`, we have to show that (a) no goal state is in S_{known} and that (b) there is no transition leaving S_{known}.

Condition (a) holds because every state in S_{known} was inserted into *open* at least once. Given that *open* is empty in line 39, all entries have been popped in line 9. If there were a goal state in S_{known}, a DYN-safe heuristic assigns the state a finite heuristic value and popping it would either re-evaluate it (putting it back on *open* without closing it) or terminate the search with a solution.

Condition (b) does not necessarily hold since DYN-A* can prune states with infinite heuristic values in lines 5, 16, and 31. Executing line 39 does thus not directly correspond to DECLARE UNSOLVABLE in the framework. Instead, we show that it corresponds to repeated uses of GENERATE UNKNOWN followed by DECLARE UNSOLVABLE. Since the heuristic is DYN-safe, no goal state is reachable from all pruned states. We can thus use GENERATE UNKNOWN to fully explore the state space reachable from s and will not add a goal state to S_{known}. For all states that were not pruned, we can see with the same argument as above that they were added to *open* and eventually expanded, adding all their unpruned successors to S_{known}. □

DYN-A* using a DYN-safe heuristic is sound due to Theorems 1 and 3 and we can also show that it can be complete.

Theorem 4. DYN-A* *is complete if chains of* REFINE *operations converge, i.e., for all reachable info*$_0$*, states s, and info*$_i$ = REFINE$_{src_h}$(*info*$_{i-1}$, s) *with* $i > 0$*, there is an* $n \geq 0$ *such that* $h(s, \mathit{info}_n) = h(s, \mathit{info}_{n+1})$.

Proof. Using Theorem 2, we only need to show that there are no infinite loops of REFINE and GENERATE KNOWN operations. The former cannot happen because REFINE operations converge. The latter holds because known transitions are only considered if we find a cheaper path and after generating a first path to a state there are only finitely many values that can occur as cheaper path costs. □

Optimality of Solutions

We first show that DYN-A* returns optimal solutions when using a DYN-admissible heuristic. The proof generally follows the same line of reasoning as the one for A* with a static admissible heuristic. Our situation is more complicated because using delayed duplicate detection with dynamic heuristics means that there can be states on *open* where both g-values and h-values are outdated because we discovered both a cheaper path to and a higher heuristic value for those states before popping them from the open list. Additionally, we want to show optimality both with and without re-evaluation which adds an additional source of changes to the open list.

As in the static case (e.g., Hart, Nilsson, and Raphael 1968), our general strategy is to show that at any time before termination, there is an entry on *open* that represents a prefix of an optimal plan, and the algorithm can only terminate by completing one of these prefixes. To state this formally, we first define some auxiliary notation.

Definition 10. *A state* s *is called* settled *in iteration* i *if it was expanded in some iteration* $i' < i$ *with* $g^{i',0}(s) = g^*(s)$.

Once a state is settled, we know a cheapest path to it and have seen its successors along this path. If we consider an optimal path, some prefix of it will be settled and the next state along this path will be on *open* with an optimal g-value.

Lemma 2. *Consider* DYN-A* *with a* DYN-*safe dynamic heuristic in iteration* i*. Let* s *be a state settled in iteration* i *and let* s' *be a state where* $h^*(s') < \infty$ *such that* s *is a predecessor of* s' *and there is an optimal path* $\langle t_1, \dots, t_n \rangle$ *to* s' *with* $t_n = \langle s, \ell, s' \rangle$*, i.e.,* s' *can be reached optimally through* s*. Then open contains an entry* $\langle s', g^{val}, \cdot \rangle$ *in iteration* i *and* $g^{val} = g^*(s')$*, or* s' *is settled in iteration* i*.*

Proof. Consider states s and s' such that the conditions of the lemma hold in iteration A. Because s is settled in iteration A, it must have been expanded in an iteration $B < A$ with $g^{B,0}(s) = g^*(s)$.

Because s was expanded, iteration B reached line 19. Also, DYN-A* did not terminate in line 21 in iteration B, because otherwise iteration A would not exist. Thus $\langle s, \ell, s' \rangle$ was considered at a time $\langle B, j \rangle$, we found an optimal path to s', and $g^{B,j}(s') = g^*(s')$.

Let $\langle C, k \rangle$ be the first time that the g-value of s' is reduced to $g^*(s')$. (This happens at $\langle B, j \rangle$ at the latest but could also have happened earlier.) At this time, either s' was not known or a cheaper path to it was found. Because $h^*(s') < \infty$ and h is DYN-safe, we have $h^{C,k}(s') < \infty$ and s' was inserted into *open* with a g-value of $g^{C,k}(s') = g^*(s')$. Any entry for s' added to *open* after $\langle C, k \rangle$ also has a g-value of $g^*(s')$.

If the entry $\langle s', g^*(s'), \cdot \rangle$ inserted in iteration C is still on *open* in iteration A, then the first alternative of the lemma's consequent is satisfied. Otherwise, one or more entries for s' were popped from *open* after C and (a) re-evaluated, (b) ignored because s' was in *closed*, or (c) expanded. In case (a), a new entry $\langle s', g^*(s'), \cdot \rangle$ is added to *open* and the argument restarts at the beginning of this paragraph, replacing C with the iteration of this reinsertion. Case (b) can only happen if s' was expanded after C because Algorithm 2 ensures that states inserted into *open* are not closed[1] and only expanding s' can close it. It follows that s' was expanded (case (c)) with g-value $g^*(s')$ in an iteration between C and A and thus that s' is settled in A, satisfying the second alternative of the lemma's consequent. □

With DYN-admissible heuristics, this result implies that there always is an entry with an f-value of at most the optimal solution cost.

Lemma 3. *Consider* DYN-A* *with a* DYN-*admissible dynamic heuristic* h *for a solvable transition system with initial state* s_0*. Then open contains an entry* $\langle \cdot, g^{val}, h^{val} \rangle$ *with* $g^{val} + h^{val} \leq h^*(s_0)$ *at the beginning of each iteration.*

Proof. Consider an optimal solution $\pi = \langle t_1, \dots, t_n \rangle$ through states $s_0, \dots, s_n$, i.e., $t_i = \langle s_{i-1}, \cdot, s_i \rangle$. Let s_k be the first state that is not settled in an iteration i. Note that s_n could not have been settled without DYN-A* terminating.

In case $k = 0$, we know that s_0 is settled in iteration 1, so i must be 0. Since h is DYN-admissible and hence $h^*(s_0) < \infty$, the entry $\langle s_0, 0, h^{0,0}(s_0) \rangle$ was inserted into *open* and is the only entry at the beginning of iteration i. The lemma then follows from the admissibility of $h^{0,0}$.

In case $k > 0$, we know that s_{k-1} is settled in iteration i. Moreover, $\langle t_{k+1}, \dots, t_n \rangle$ is a path from s_k to a goal state showing that $h^*(s_k) < \infty$ and $\langle t_1, \dots, t_k \rangle$ is an optimal path to s_k. We can therefore use Lemma 2 in iteration i with $s = s_{k-1}$ and $s' = s_k$. Because s_k is not settled in iteration i, we know that there must be an entry $\langle s_k, g^{val}, h^{val} \rangle$ on *open* where $g^{val} = g^*(s_k)$ and $h^{val} = h^{ins}(s_k)$ for some time ins. With the admissibility of h^{ins}, we get $g^{val} + h^{val} = g^*(s_k) + h^{ins}(s_k) \leq g^*(s_k) + h^*(s_k) = h^*(s_0)$. □

[1]This is explicit except in line 34 where s' cannot be closed because it was just discovered.

In the context of graphs with unknown obstacles, Koyfman et al. (2024) show that a variant of A* returns optimal solutions when using a dynamic heuristic that is PDA. As PDA requires that a prefix of an optimal solution is closed and its continuation exists on *open*, Lemma 3 directly follows. Showing that a heuristic is PDA thus requires reasoning about *open* and depends on the search behavior. Our Lemmas 2 and 3 only depend on the heuristic being DYN-admissible, which is independent of the search algorithm.

While there is an entry with f-value of at most $h^*(s_0)$ on *open*, DYN-A* cannot yield a solution of cost $c > h^*(s_0)$.

Theorem 5. DYN-A* *with a* DYN-*admissible dynamic heuristic returns optimal solutions.*

Proof. Assume the algorithm terminated with a solution of cost $c > h^*(s_0)$ after popping $\langle s, g^{ins}(s), h^{ins}(s)\rangle$ from *open* at time pop. Then $g^{ins}(s) + h^{ins}(s) = g^{ins}(s) + 0 \geq g^{pop}(s) = c > h^*(s_0)$. This contradicts Lemma 3 because DYN-A* pops the entry in *open* with minimal f-value. □

The results in this chapter also apply to A* as a special case. A corollary is that A* with an admissible static heuristic is also optimal when using delayed duplicate detection.

Reopening

A classic result for A* is that states are never reopened when the heuristic is consistent. One way to prove this is by showing that the sequence of f-values of popped open list entries is monotonically increasing. If we then consider the projection of this sequence to a single state, we can use the fact that the heuristic value $h(s)$ of a static heuristic is constant to see that multiple copies of a state are expanded in order of increasing g-value. If the lowest g-value of a state is expanded first, the state will never be reopened.

Interestingly, we can show that the intermediate result of monotonically increasing f-values also holds for dynamic heuristics with the right properties, but reopening may still occur. Since the intermediate result is not useful in this setting, we refer to the technical report for the full proof (Christen et al. 2025).

Theorem 6. *The sequence of f-values popped by* DYN-A* *from open is non-decreasing if it uses a* DYN-*admissible,* DYN-*monotonic, and* DYN-*consistent dynamic heuristic.*

Proof sketch. We show that any entry inserted into *open* has an f-value at least as high as the value that was popped in the same iteration. Since the popped element had the minimal f-value at the time, this is sufficient to show that future pops cannot have lower f-values. □

Without re-evaluation DYN-A* may reopen states even if the heuristic satisfies the conditions of Theorem 6. In the example in Figure 2, states are expanded in the order A, B, C, D, E, F. When A is expanded, F is added to the open list with the suboptimal f-value of $8+0$ and D is added with a suboptimal value of $6 + 0$. While B and C are added to the open list, their heuristic values increase from 0 to 1. Next B is expanded and discovers a new path to D along which we find information to improve the heuristic value of

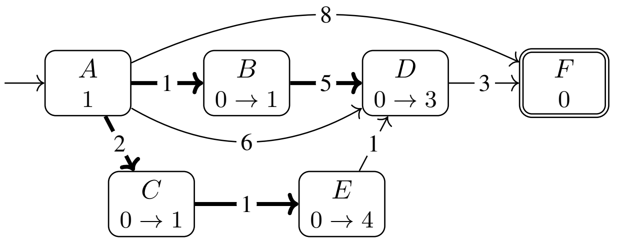

Figure 2: Example state space, showing that reopening can occur with DYN-consistent heuristics. Edge costs are written on the edges, heuristic values are written inside the states. The heuristic is dynamic and the heuristic value of a state with label $x \to y$ changes from x to y once it is reached along the bold incoming edge.

D to 3. This path is not cheaper so D remains on *open* with a value of $6 + 0$. After expanding C and increasing the heuristic value of E, *open* is $\langle\langle D, 6, 0\rangle, \langle E, 3, 4\rangle, \langle F, 8, 0\rangle\rangle$. State D is expanded first but adds nothing to *open* because F is reached through a more expensive path than the one found previously. Then E is expanded and finds the optimal path to D. If we do not reopen D at this point, the algorithm terminates with the suboptimal path $\langle\langle A, \cdot, F\rangle\rangle$.

Note that f-values of popped states increase monotonically, the heuristic is monotonically increasing, admissible and consistent throughout the search, and only updated along transitions (i.e., progression-based).

At the time D was expanded, its heuristic value was higher than the one we used to insert it. If we would have re-evaluated D at that time, it would have been re-inserted into *open* with a value of $6 + 3$ and we would have expanded E first, finding a cheaper path to D before expanding it. In that case, no reopening happens. We now show that this is the case in general with re-evaluation.

Theorem 7. DYN-A* *with re-eval enabled and using a* DYN-*monotonic,* DYN-*consistent dynamic heuristic does not reopen states.*

Proof. Assume that a state s_n is reopened at time $reop$. Consider the path $\langle t_1, \dots, t_n\rangle$ to s_n where $t_i = \langle s_{i-1}, \ell_i, s_i\rangle$ is the parent pointer of s_i at time $reop$ for all $0 < i \leq n$. State s_n must have been added to *closed* at some earlier time $cls < reop$. Since $g^{cls}(s_0) = c(\langle\rangle) = 0$ and $g^{cls}(s_n) > c(\langle t_1, \dots, t_n\rangle)$, there must be a smallest j such that s_{j+1} satisfies $g^{cls}(s_{j+1}) > c(\langle t_1, \dots, t_{j+1}\rangle)$. For all s_i with $i \leq j$, we then know $g^{cls}(s_i) = c(\langle t_1, \dots, t_j\rangle)$.

Note that $h^{cls}(s_n) < \infty$ and thus $h^{cls}(s_i) < \infty$ for all $i < n$ due to DYN-consistency. Also, since h is DYN-monotonic, $h^t(s_i) < \infty$ for all times $t < cls$.

At some time $exp < cls$ the g-value of s_j was set to $c(\langle t_1, \dots, t_j\rangle)$ and at that time, state s_j must have been added to *open*. Between exp and cls the algorithm could not have expanded s_j because that would explore transition t_{j+1} and contradict $g^{cls}(s_{j+1}) > c(\langle t_1, \dots, t_{j+1}\rangle)$. Any re-evaluation of s_j that occurs between exp and cls will insert s_j back into *open* because $h^{exp}(s_j) < \infty$. Thus at time *cls* there is an entry $\langle s_j, g^{re}(s_j), h^{re}(s_j)\rangle$ on *open* that was added at a time *re* with $exp \leq re < cls$.

At time cls an entry $\langle s_n, g^{ins}(s_n), h^{ins}(s_n)\rangle$ is popped and expanded that was added earlier at time $ins < cls$.

$$\begin{aligned}
& g^{ins}(s_n) + h^{cls}(s_n) \\
\leq\ & g^{ins}(s_n) + h^{ins}(s_n) && (1)\\
\leq\ & g^{re}(s_j) + h^{re}(s_j) && (2)\\
=\ & c(\langle t_1, \dots, t_j\rangle) + h^{re}(s_j) && (3)\\
\leq\ & c(\langle t_1, \dots, t_j\rangle) + h^{cls}(s_j) && (4)\\
\leq\ & c(\langle t_1, \dots, t_j\rangle) + c(\langle t_{j+1}, \dots, t_n\rangle) + h^{cls}(s_n) && (5)\\
\leq\ & g^{reop}(s_n) + h^{cls}(s_n) && (6)
\end{aligned}$$

Step (1) holds because s_n was expanded and not re-evaluated at time *cls*; step (2) holds because $\langle s_j, g^{re}(s_j), h^{re}(s_j)\rangle$ was on *open* but did not have minimal f-value at time *cls*; step (3) holds because the g-value of s_j matches $c(\langle t_1, \dots, t_j\rangle)$ for all times between exp and *cls*; step (4) holds by DYN-monotonicity; step (5) holds by DYN-consistency; and step (6) holds because the cost of the path defined through parent pointers is bounded by the stored g-values.

In summary, we have $g^{ins}(s_n) + h^{cls}(s_n) \leq g^{reop}(s_n) + h^{cls}(s_n)$, so $g^{ins}(s_n) \leq g^{reop}(s_n)$. This contradicts that s_n is reopened at time *reop*. □

Note that this proof only relies on DYN-consistency between known states so this result also holds for partially DYN-consistent heuristics. Furthermore, Koyfman et al. (2024) define the property OPTEX for algorithms where every expanded state has an optimal g-value. If the conditions of Theorem 7 are satisfied, DYN-A* is OPTEX.

Applications

We now show how existing approaches, primarily from classical planning, fit our framework. We argue more formally for the first application and go into less detail for the others.

Interleaved Search

Franco and Torralba (2019) consider *interleaved search*, where search and computation of an abstraction heuristic are alternated in progressively longer time slices. They refer to a generic function `HeuristicImprovement` (`HI`) that returns an improved abstraction within a given time limit, potentially using additional information gained during search.

We represent this idea as a dynamic heuristic by first defining the information source IS. Given a function `HI` that takes a time limit and information from an information source ADD, the information space $\mathcal{I}_{IS}$ contains tuples $\langle t, l, \alpha, D, \mathit{info}\rangle$, where t tracks search time, l is the current time limit, α the current abstraction with abstract distance table D, and $\mathit{info} \in \mathcal{I}_{ADD}$ is additional information. The constant initial-info$_{IS}$ is $\langle 0, l_0, \alpha_0, D_0, \text{initial-info}_{ADD}\rangle$ where $\texttt{HI}(l_0, \text{initial-info}_{ADD})$ returned α_0 and D_0. Calling update$_{IS}$ just replaces *info* with update$_{ADD}$(*info*) and leaves the other components unchanged. Calling refine$_{IS}(\langle t, l, \alpha, D, \mathit{info}_{ADD}\rangle, s)$ adds the time passed since the last call to refine$_{IS}$ (or since program start) to t. If $t > l$, it sets α and D to the result of $\texttt{HI}(l, \mathit{info}_{ADD})$, sets l to $2 \cdot l$, and finally sets t to 0. The dynamic heuristic h_{IS} then calculates the heuristic value of a state s as $D(\alpha(s))$.

The properties of h_{IS} naturally depend on `HI`. Franco and Torralba describe an implementation based on symbolic pattern databases (PDBs; Edelkamp 2002), let us call it h_{IS}^{FT}, that samples states from the open list to guide the pattern selection. We can approximate this with a nested information source ADD that tracks sampled states in update$_{ADD}$. They enforce DYN-monotonicity by maximizing over known PDBs, DYN-admissibility and DYN-consistency are also guaranteed as each improvement call returns either a partial PDB (Anderson, Holte, and Schaeffer 2007) or a full PDB, making the resulting heuristics admissible and consistent.

Theorem 8. *Interleaved search as described by Franco and Torralba (2019) is optimal and does not reopen states.*

Proof. DYN-A* with *re-eval* using h_{IS}^{FT} with the `HI` function defined by Franco and Torralba models the behavior of interleaved search. Optimality then follows from Theorem 5 together with the DYN-admissibility of h_{IS}^{FT}. Finally no states are reopened due to Theorem 7 together with the DYN-monotonicity and DYN-consistency of h_{IS}^{FT}. □

This confirms Franco and Torralba's conjecture that re-evaluation and monotonicity (plus DYN-admissibility and DYN-consistency) ensure that no states are reopened.

Online Cartesian Abstraction Refinement

Eifler and Fickert (2018) refine additive Cartesian abstractions (Seipp and Helmert 2014), combined via cost partitioning, during search. Their refinement strategy guarantees DYN-admissibility, DYN-consistency, and DYN-monotonicity. It is called for states popped from *open* when a Bellman optimality equation detects a local error. The resulting heuristic is then used in A* with re-evaluation. We can define a suitable information source in a way analogous to *IS* and show a result similar to Theorem 8.

Theorem 9. A* *with re-evaluation using online Cartesian abstraction refinement as described by Eifler and Fickert (2018) is optimal and does not reopen states.*

Landmark Progression

We have already seen a simplified version of landmark progression as our running example. In general landmarks are not limited to single operators but denote properties that hold along all plans of a given task. Since landmarks cannot be computed efficiently in general (Hoffmann, Porteous, and Sebastia 2004), applications generate a set of landmarks only for the initial state and then progress this information (e.g., Richter and Westphal 2010; Karpas and Domshlak 2009; Domshlak et al. 2011).

Büchner et al. (2023) formalize landmark progression in the landmark best-first search (LM-BFS) framework, which describes how landmark information is represented, initialized, progressed, and merged. These components can be directly cast as a progression source and thus define a progression-based information source. We can then define a dynamic heuristic based on this landmark information. LM-BFS can thus be viewed as an instantiation of DYN-A* with re-evaluation, also making our framework applicable to its instantiations, e.g. LM-A* (Karpas and Domshlak 2009).

Theorem 10. *An instantiation of LM-BFS using a* DYN-*admissible landmark heuristic and a priority queue ordered by* $f = g + h$ *is optimal.*

LTL-A*

Simon and Röger (2015) describe an A* search where each state s is annotated with a linear temporal logic (LTL; Pnueli 1977) formula representing a *trajectory constraint* that must be satisfied by any optimal path from s to a goal state.

We can define a progression source over LTL formulas and associated g-values, where progression uses the rules by Bacchus and Kabanza (2000). Simon and Röger define the merging of two formulas φ and ψ as $\varphi \wedge \psi$ if they were reached with the same g-value, and as the formula with lower g-value otherwise. We can do so by tracking g-values when progressing and merging.

While this gives us a progression-based information source, the heuristic described by Simon and Röger only guarantees path-admissibility (Karpas and Domshlak 2012) for some given information, from which neither DYN-admissibility nor PDA follow directly.

A* with Lazy Heuristic Evaluation

Zhang and Bacchus (2012) introduce a modification of A* that uses two heuristics, say h_C and h_A, such that h_C is cheap and h_A is accurate. When first inserting a state into *open*, it is evaluated using h_C. When a state is popped, it is only expanded if its assigned h-value was calculated by h_A, otherwise it is evaluated using h_A and re-inserted into *open*.

We represent this idea as a dynamic heuristic by first defining the information source *lazy* over the space of functions $\mathit{info}\colon S \to \{C, A\}$ as initial-info$_{lazy} = \{s \mapsto C \mid s \in S\}$, update$_{lazy}(\mathit{info}, t) = \mathit{info}$, and refine$_{lazy}(\mathit{info}, s) = \mathit{info}'$ such that $\mathit{info}' = \mathit{info}$ except $\mathit{info}'(s) = A$.

The *lazy evaluation heuristic* depending on *lazy* then maps a state s and information *info* to $h_{\mathit{info}(s)}(s)$.

Theorem 11. A* *using lazy heuristic evaluation with two admissible heuristics is optimal.*

Proof. DYN-A* with *re-eval* using the lazy evaluation heuristic models the behavior of A* with lazy heuristic evaluation. DYN-admissibility is given as the two static heuristics are admissible. Applying Theorem 5 concludes the proof. □

The instantiation discussed by Zhang and Bacchus (2012) using LM-Cut and MAXSAT is optimal as it satisfies the conditions of the Theorem. We expect that the same result can be shown for similar approaches such as selective max (Domshlak, Karpas, and Markovitch 2012) or rational lazy A* (Tolpin et al. 2013). Note that using consistent heuristics does not guarantee DYN-consistency of the resulting lazy evaluation heuristic, thus reopening may be necessary.

Path-Dependent f-Values

Dechter and Pearl (1985) investigate the optimality of best-first search algorithms where the f-value of a state can depend on the path to that state. This is similar to our notion of dynamic heuristics although they consider static heuristics and allow path-dependent evaluation functions $f(\pi)$ other than $f = g + h$. Such an evaluation function $f(\pi)$ can be interpreted in our framework as a dynamic heuristic $h(s, \pi) = f(\pi) - g(s)$ where paths are stored in an information source that is updated with newly discovered paths.

Dynamic heuristics can accumulate information based on all paths leading to a state, even more expensive ones. This is not possible in their framework. In contrast, their framework covers cases like weighted A* (Pohl 1970) that fit our general framework but not DYN-A*. While we could encode them into a dynamic heuristic as discussed above, it might not be useful to investigate the algorithm's behavior in such cases. Generalizing their results to dynamic heuristics is an interesting line of future work.

Future Work

There are more applications that likely fit our framework.

Pathmax (Mérő 1984) is a technique to propagate h-values from a transition's origin to its target and vice versa. Originally defined for the algorithm B', Zhang et al. (2009) show how pathmax can be applied to A*, alongside the more powerful bidirectional pathmax by Felner et al. (2005).

Deferred evaluation (Helmert 2006) assigns the heuristic value of the state being expanded to its successors upon generating them. Once a successors is expanded itself, its true heuristic value is calculated and forwarded to its successors.

Partial expansion A* (Yoshizumi, Miura, and Ishida 2000) avoids inserting too many nodes into the open list by only doing so for promising successors. Unpromising successors are represented by re-inserting their parent node with updated priority. This technique does not match the DYN-A* instantiation but it probably fits our general framework.

Conclusion

We investigated dynamic heuristics and explicitly modeled information they are based on. We have shown soundness and completeness results for a dynamic heuristic search framework tracking such information and looked into DYN-A*, an instantiation extending A* with dynamic heuristics. DYN-A* is optimal with and without re-evaluation when using a dynamic heuristic that is DYN-admissible, a natural extension of admissibility. With re-evaluation, it further does not reopen states when using a heuristic that monotonically improves its values and is the dynamic equivalent to consistent. These results extend the classic results for static heuristics that many existing approaches appeal to despite using dynamic heuristics. We have seen that such arguments are not always valid as re-opening can occur even with a DYN-consistent heuristic. By showing that existing approaches fit our framework, we provide a formal basis for such claims.

In addition to the use cases discussed previously, studying optimal efficiency results of DYN-A* would be interesting.

Acknowledgments

This research was supported by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA).

References

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial Pattern Databases. In Miguel, I.; and Ruml, W., eds., *Proceedings of the 7th International Symposium on Abstraction, Reformulation and Approximation (SARA 2007)*, volume 4612 of *Lecture Notes in Artificial Intelligence*, 20–34. Springer-Verlag.

Bacchus, F.; and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 116(1–2): 123–191.

Büchner, C.; Eriksson, S.; Keller, T.; and Helmert, M. 2023. Landmark Progression in Heuristic Search. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, 70–79. AAAI Press.

Christen, R.; Pommerening, F.; Büchner, C.; and Helmert, M. 2025. A Formalism for Optimal Search with Dynamic Heuristics: Technical Report. arXiv:2504.21131 [cs.AI].

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *Journal of the ACM*, 32(3): 505–536.

Domshlak, C.; Helmert, M.; Karpas, E.; Keyder, E.; Richter, S.; Röger, G.; Seipp, J.; and Westphal, M. 2011. BJOLP: The Big Joint Optimal Landmarks Planner. In *IPC 2011 Planner Abstracts*, 91–95.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2012. Online Speedup Learning for Optimal Planning. *Journal of Artificial Intelligence Research*, 44: 709–755.

Edelkamp, S. 2002. Symbolic Pattern Databases in Heuristic Search Planning. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 274–283. AAAI Press.

Eifler, R.; and Fickert, M. 2018. Online Refinement of Cartesian Abstraction Heuristics. In Bulitko, V.; and Storandt, S., eds., *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 46–54. AAAI Press.

Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R. C. 2005. Dual Lookups in Pattern Databases. In Kaelbling, L. P.; and Saffiotti, A., eds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 103–108. Professional Book Center.

Franco, S.; and Torralba, Á. 2019. Interleaving Search and Heuristic Improvement. In Surynek, P.; and Yeoh, W., eds., *Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCS 2019)*, 130–134. AAAI Press.

Gelperin, D. 1977. On the Optimality of A*. *Artificial Intelligence*, 8(1): 69–76.

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.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22: 215–278.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733. AAAI Press.

Karpas, E.; and Domshlak, C. 2012. Optimal Search with Inadmissible Heuristics. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 92–100. AAAI Press.

Koyfman, D.; Shperberg, S. S.; Atzmon, D.; and Felner, A. 2024. Minimizing State Exploration While Searching Graphs with Unknown Obstacles. In Dastani, M.; Sichman, J. S.; Alechina, N.; and Dignum, V., eds., *Proceedings of the Twenty-Third International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, 1038–1046. IFAAMAS/ACM.

Mérő, L. 1984. A Heuristic Search Algorithm with Modifiable Estimate. *Artificial Intelligence*, 23(1): 13–27.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pnueli, A. 1977. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, 46–57.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1: 193–204.

Pohl, I. 1977. Practical and Theoretical Considerations in Heuristic Search Algorithms. In Elcock, E. W.; and Michie, D., eds., *Machine Intelligence 8*, 55–72. Ellis Horwood.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Seipp, J.; and Helmert, M. 2014. Diverse and Additive Cartesian Abstraction Heuristics. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.

Simon, S.; and Röger, G. 2015. Finding and Exploiting LTL Trajectory Constraints in Heuristic Search. In Lelis, L.; and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 113–121. AAAI Press.

Tolpin, D.; Beja, T.; Shimony, S. E.; Felner, A.; and Karpas, E. 2013. Towards Rational Deployment of Multiple Heuristics in A*. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 674–680. AAAI Press.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with Partial Expansion for Large Branching Factor Problems. In Kautz, H.; and Porter, B., eds., *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, 923–929. AAAI Press.

Zhang, L.; and Bacchus, F. 2012. MAXSAT Heuristics for Cost Optimal Planning. In Hoffmann, J.; and Selman, B., eds., *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012)*, 1846–1852. AAAI Press.

Zhang, Z.; Sturtevant, N. R.; Holte, R.; Schaeffer, J.; and Felner, A. 2009. A* Search with Inconsistent Heuristics. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 634–639. AAAI Press.