Beliefs We Can Believe in: Replacing Assumptions with Data in Real-Time Search

Maximilian Fickert,[*1] Tianyi Gu,[*2] Leonhard Staut,[*1] Wheeler Ruml,[2]
Jörg Hoffmann,[1] Marek Petrik[2]
[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2]Department of Computer Science, University of New Hampshire, USA

Abstract

Suboptimal heuristic search algorithms can benefit from reasoning about heuristic error, especially in a real-time setting where there is not enough time to search all the way to a goal. However, current reasoning methods implicitly or explicitly incorporate assumptions about the cost-to-go function. We consider a recent real-time search algorithm, called Nancy, that manipulates explicit beliefs about the cost-to-go. The original presentation of Nancy assumed that these beliefs are Gaussian, with parameters following a certain form. In this paper, we explore how to replace these assumptions with actual data. We develop a data-driven variant of Nancy, DDNancy, that bases its beliefs on heuristic performance statistics from the same domain. We extend Nancy and DDNancy with the notion of persistence and prove their completeness. Experimental results show that DDNancy can perform well in domains in which the original assumption-based Nancy performs poorly.

Introduction

When an agent has to react to its environment in real time, there often is not enough planning time to find a complete plan. Instead, partial commitments must be made based on partial exploration. Real-time heuristic search approaches this problem by exploiting the information contained in a heuristic function. Such a search can be viewed as starting from an initial state and then trying to reach the goal via a series of search episodes. Within each episode the agent performs an amount of lookahead search limited by runtime or by the number of expanded nodes, guided by the heuristic function. The agent commits to the most promising action, or subsequence of actions, given its lookahead, and iterates until it reaches a goal state.

One key element in such an arrangement is learning: the agent needs to revise its heuristic estimation of state values. Otherwise it can easily get stuck in cycles; in particular, it will never escape a local minimum unless its lookahead is large enough to see beyond the minimum's exits. A natural form of learning is the backpropagation step proposed by Korf (1990), updating state value estimates backwards from the lookahead frontier at the end of each iteration. The resulting three-phase paradigm – lookahead, update, execute – has become standard, with perhaps LSS-LRTA* (Koenig and Sun 2008) being its most popular exemplar.

In this paper, we emphasize additional ways in which a search can reason about heuristic error. Such concerns are central to real-time planning, for the agent must commit to action decisions while the leaf nodes of the search are still open and it has no immediate information about them other than their heuristic values. Various methods for reasoning about heuristic error have been proposed for purposes different from real-time search (Stern et al. 2014; Rose, Burns, and Ruml 2011; Burns and Ruml 2013), that we outline in some detail further below. For real-time search, proposals in this direction have a long history in AI research (e.g. (Mutchler 1986; Pemberton and Korf 1994)). Here we follow up on a recent proposal: the Nancy algorithm (Mitchell et al. 2019). Adapting earlier ideas by Pemberton and Korf (1994), Nancy manipulates cost-to-go beliefs, i.e., distributions of cost-to-go. Nancy backs up such belief distributions instead of just heuristic values, thereby explicitly maintaining state-value uncertainty. The belief distributions are used to guide expansions towards the most informative states, replacing the conventional A* expansion policy with one that attempts to minimize risk, i.e., the regret if the action committed to is in fact not the optimal choice.

The key question then becomes where these cost-to-go beliefs actually come from. Instead of a heuristic value, Nancy requires such distributions at each leaf node. Mitchell et al. (2019) address this by adopting *assumptions*. They obtain the desired beliefs as a function of heuristic value, assuming that (1) the beliefs are Gaussian and (2) their variance decreases as search gets closer to the goal. While these assumptions seem plausible, and while Nancy has been shown to outperform LSS-LRTA* in certain situations, the question arises if it is possible to replace the assumption-based beliefs with actual *data*. This is what we explore in this paper.

We develop a data-driven variant of Nancy, DDNancy, that gathers its beliefs from statistics. Prior to the real-time search on instances of some domain, we solve example instances of that domain and store pairs of heuristic value h

[*]These authors contributed equally to this work.

vs. the real cost-to-go h^*. The belief distribution for heuristic value h is then the set of h^* values associated with h in the data. We furthermore develop an extension to Nancy, and transitively to DDNancy, which makes the algorithm *persistent* in its exploration of promising heuristic values and thus renders it complete, i.e., yields the guarantee that eventually a goal state will be reached in a dead-end free state space. The ideas used in our proof can be transferred to LSS-LRTA*, for which we give a similar completeness proof that, in contrast to prior work, does not require the heuristic to be consistent or admissible. Finally, running experiments in classical planning as well as single-agent search benchmarks, we show that DDNancy is able to perform well in domains where assumption-based Nancy performs poorly.

Previous Work

There are many suboptimal heuristic search algorithms that make assumptions about how the heuristic behaves. Most of these search algorithms work very well on the domain problems that follow the assumptions. For example, potential search (Stern et al. 2014) is a cost-bounded suboptimal search that expands the open node with the highest potential value. The potential value is given by $\frac{C-g(n)}{h(n)}$, which is based on a linear relative heuristic error model, that assumes the error of the heuristic can be modeled as a random variable multiplied by the heuristic value. Stern et al. prove that, under this assumption, potential search expands the node most likely to lead to a solution within the cost bound. IDA*$_{IM}$ (Burns and Ruml 2013) learns an incremental model of heuristic error and uses it to choose bounds for an IDA* search. Skeptical search (Thayer, Dionne, and Ruml 2011) learns a simple model of heuristic error online and uses it to guide a bounded-suboptimal search. However, all these works consider only the heuristic value itself, without the uncertainty estimates that we will exploit here.

Real-time Heuristic Search

In real-time search, there are three phases: lookahead, decision making, and learning.

In the lookahead phase, the algorithm performs a certain amount of lookahead search, e.g., select the best node in the open list, expand it, push all the child nodes into open list, then close the node. There are different ways to sort the open list. LSS-LRTA* follows the A* convention to order nodes by f value, which is cost-to-come g plus the lower bound estimate on cost-to-go h. However, as pointed out by Mutchler (1986), expanding the frontier node with lowest f is not necessary the optimal way to make use of a limited number of node expansions because f does not take any heuristic error into account. A better alternative is to sort the open list by $\hat{f}$, which denotes an estimate of the expected value of f^*, rather than a lower bound (Kiesel, Burns, and Ruml 2015). This better matches the principle of rationality, which stipulates minimizing expected cost.

In the decision making phase, the algorithm has to commit to the best action given the current lookahead search information. LSS-LRTA* follows the A* convention to move toward the frontier node with the lowest f. However, this is not necessarily the optimal decision for on-line real-time planning (Pemberton and Korf 1994). Taking into account the uncertainty that an agent with limited lookahead has about the values that will be revealed by future search, it can sometimes be a better choice to commit to a action that would lead us toward multiple good-looking options rather than one that offers a single great-looking course of action.

In the learning phase, the algorithm backs up the information in the search frontier towards the LSS, supporting the decision making and the expansion strategy for following iterations. LSS-LRTA* backs up only the lower bound estimate h, since that is all the information it needs from the frontier.

Nancy with Gaussian Beliefs

Recently, real-time search has been interpreted as decision-making under uncertainty. Mitchell et al. introduced a new algorithm called Nancy, which bases its expansion decisions on *belief distributions* that characterize the agent's uncertainty about the possible true values of the action instead of scalar heuristic values. In the lookahead phase, Nancy explores nodes in order to minimize the *regret* that the agent would experience if the top-level action that is currently deemed best were in fact not the best action to follow. The regret is calculated by considering the *post-expansion belief* for each top-level action, i.e., by considering how the belief is expected to change if an expansion were made under each action. Since the heuristic error is expected to decrease when moving closer to the goal, the post-expansion belief is computed by moving the mass of the (pre-expansion) belief towards the mean, reducing its variance. As we will explain in detail in the next section, Nancy's decision making and learning phases are similar to LSS-LRTA*'s, with modifications for using belief distributions instead of scalar heuristic values.

As presented by Mitchell et al., Nancy makes use of at least four assumptions. First, she assumes beliefs about the true h^* values of frontier nodes that are Gaussian, centered around an $\hat{h}$ estimate derived from h and an error model estimated on-line during search. The error model assumes a constant underestimate by h for every edge in the state space graph and requires an estimate of the number of edges-to-go. The edges-to-go are estimated by h in unit-cost domains or a similar function d in domains with non-uniform costs (see the original paper for details). The variance was assumed to be proportional to the edges-to-go as well.

Second, Nancy back-propagates only the belief of the successor with the lowest expected value. As Mitchell et al. explain, this assumes that no more information will become available to distinguish between sibling nodes. And if multiple successors look promising, they should each be allowed to influence our belief about their parent.

Third, the regret calculation requires estimating the change in belief that would be caused by expanding nodes under a top-level action. While this is likely complex, Nancy assumes it to be a simple reduction in variance of the current belief, proportional to the estimated reduction in the number of edges-to-go. In fact, done properly, the post-expansion belief should be a distribution over possible beliefs, and just

Algorithm 1: Nancy

1 $s \coloneqq s_{start}$
2 $\pi_{curr} \coloneqq \langle\rangle$
3 **while** s *is not a goal state* **do**
4 $\quad s_f \coloneqq riskLookahead(s)$
5 $\quad$ **if** s_f *is a goal state* **then**
6 $\quad\quad$ **return** path to s_f
7 $\quad$ **if** $\hat{f}(s_f) < \hat{f}(s[\![\pi_{curr}]\!])$ **then**
8 $\quad\quad \pi_{curr} \coloneqq$ shortest known plan to s_f
9 $\quad$ let $a_0, \ldots, a_n$ be the action sequence of π_{curr}
10 $\quad s \coloneqq s[\![a_0]\!]$
11 $\quad \pi_{curr} \coloneqq \langle a_1, \ldots, a_n \rangle$
12 $\quad$ update $\hat{h}$ on the states expanded in the lookahead

as with the backups, multiple successors should be allowed to influence the estimate.

And fourth, in her regret calculation, Nancy does not consider the combinatorial space of outcomes, but just considers two top-level actions at a time and assumes independence of subtrees.

While some of these assumptions may be necessary for computational efficiency and model simplicity, in this paper we begin the process of understanding how to weaken them by showing how the first assumption, regarding the leaf belief distributions, can be replaced by data.

Generalizing Nancy

We introduce a new variant of Nancy that uses data generated in an offline training phase for the leaf belief distributions. Since most of our proofs and theoretical discussion apply to both the original Nancy and our data-driven variant, we refer to the general framework as Nancy, while referring to the specific variants as "assumption-based Nancy" and "data-driven Nancy" (DDNancy for short). We begin by explaining the general Nancy framework in detail and the differences between the two variants. First, we introduce some of the notation that is used in the pseudo-code and later in the proofs. We denote the set of applicable actions in a state s by $A(s)$. The successor of the state s with action a is the state $s[\![a]\!]$. We overload this notation for action sequences π as $s[\![\pi]\!]$. The cost of an action a is denoted by $c(a)$, which we also overload for paths, where the cost is the sum of the cost of its actions. In Nancy, each state s has an associated belief $\mathcal{B}(s)$ representing a probability distribution over the cost-to-go from s. Its expected value is denoted by $\hat{h}(s)$. The post-expansion belief $\mathcal{B}_{post}(s)$ represents an estimated updated belief if s were to be expanded.

Algorithm 1 shows the pseudo-code for Nancy. The lookahead phase using risk-based exploration returns the best frontier state according to $\hat{f}$ (Alg. 1, line 4), breaking ties by $\hat{h}$. In the original description of (assumption-based) Nancy, the search would then commit to the first action on the best plan towards s_f found in the lookahead. However, this difference in guidance (using risk for the expansion, but

Algorithm 2: Risk-Based Lookahead

Input: s
Output: frontier state with minimal $\hat{f}$
1 let *TLAs* be the actions applicable in s
2 **while** *lookahead limit is not reached* **do**
3 $\quad$ do a Nancy backup on *TLAs* (update $\mathcal{B}$ and $\mathcal{B}_{post}$)
4 $\quad \alpha \coloneqq \arg\min_{TLA \in TLAs} \hat{f}(TLA.\text{open.min}())$
5 $\quad currentTLA \coloneqq \arg\min_{TLA \in TLAs} risk(TLA, \alpha, TLAs)$
6 $\quad s' \coloneqq currentTLA.\text{open.pop}()$
7 $\quad$ **if** s' *is a goal state* **then**
8 $\quad\quad$ **return** s'
9 $\quad$ **for** $a \in A(s')$ **do**
10 $\quad\quad currentTLA.\text{open.push}(s'[\![a]\!])$
11 *selectedTLA* $\coloneqq \arg\min_{TLA \in TLAs} \hat{f}(TLA.\text{open.min}())$
12 **return** *selectedTLA*.open.min()

Algorithm 3: Risk Value of a Top-Level Action *TLA*

Input: *TLA*, *TLAs*, α
Output: Expected loss of α not being the action that minimizes $\hat{f}$ after expanding under *TLA*
1 $risk \coloneqq 0$
2 $\mathcal{B}_\alpha \coloneqq \begin{cases} \mathcal{B}_{post}(\alpha) & \text{if } TLA = \alpha \\ \mathcal{B}(\alpha) & \text{if } TLA \neq \alpha \end{cases}$
3 **for** $\beta \in TLAs \setminus \{\alpha\}$ **do**
4 $\quad \mathcal{B}_\beta \coloneqq \begin{cases} \mathcal{B}_{post}(\beta) & \text{if } TLA = \beta \\ \mathcal{B}(\beta) & \text{if } TLA \neq \beta \end{cases}$
5 $\quad$ **for** *every pair of distribution samples* $x_\alpha \in \mathcal{B}_\alpha, x_\beta \in \mathcal{B}_\beta$ **do**
6 $\quad\quad$ **if** $cost(x_\alpha) > cost(x_\beta)$ **then**
7 $\quad\quad\quad p \coloneqq prob(x_\alpha) * prob(x_\beta)$
8 $\quad\quad\quad risk \coloneqq risk + p * (cost(x_\alpha) - cost(x_\beta))$
9 **return** *risk*

$\hat{f}$ for the decision making) can cause the search to be incomplete: it may happen that after making a step towards a promising frontier node with low $\hat{f}$ value, the lookahead in the following iteration does not generate that node anymore, which would cause the search to move away and lose progress. Hence, we introduce a slight change from the original Nancy framework: we keep track of the best known state so far and the plan towards it π_{curr}. If the lookahead does not yield a strictly better state, Nancy will keep moving along π_{curr}. This change makes Nancy *persistent*, and as we show in the theoretical analysis section, this version of Nancy is indeed complete.

For the lookahead, Nancy uses her first expansion to generate the top level actions (Alg. 1, line 1). From that point forward, Nancy expands nodes such that an approximation of risk is minimized, until the expansion or time limit of the

lookahead runs out. Algorithm 2 shows the pseudo-code for the risk-based lookahead. Each top-level action has an associated open list (denoted by TLA.open in the pseudo-code) that is ordered by $\hat{f}$. For each expansion, a risk assessment is used to decide under which TLA the next element of its associated open list is expanded. Before each expansion, the associated beliefs $\mathcal{B}$ and post-expansion beliefs $\mathcal{B}_{post}$ of all top-level actions (TLAs) are updated first, such that they are pointing to the best frontier node following the last expansion (Alg. 2, line 3), such that, e.g., for a top-level action TLA, $\mathcal{B}(TLA) = \mathcal{B}(TLA.\text{open.min}())$. The TLA α is the one with the lowest expected cost $\hat{f}$, i.e., the TLA that would be executed according to the Nancy backup if the lookahead were to end at this moment (Alg. 2, line 4). Nancy then performs the risk calculation (see Algorithm 3) and chooses the TLA under which the next expansion is performed accordingly. Recall that the risk is the expected loss that would be incurred if α were to turn out not to be the best action to commit to, i.e. the probability that some other TLA leads to a better plan, weighted by the expected cost difference of that cheaper plan. The risk value is calculated by performing a numerical integration over every sample (i.e., pairs of cost and corresponding probability) in $\mathcal{B}_\alpha$ and every sample in $\mathcal{B}_\beta$, thereby increasing the risk whenever a sample in $\mathcal{B}_\beta$ has cost less than that of $\mathcal{B}_\alpha$ (Alg. 3, line 8).

Once the expected risk that would result from expanding each node has been estimated, the node with the lowest post-expansion risk estimate is actually expanded (Alg. 2, line 10). This process is then repeated until the expansion or time limit is reached, or a goal state is selected for expansion. Once the lookahead phase ends, the search performs Nancy backups and executes the TLA with the lowest expected cost (Alg. 1, line 10). In the learning phase, the beliefs $\mathcal{B}$ and post-expansion beliefs $\mathcal{B}_{post}$ of all nodes within the local search space are updated (Alg. 1, line 12). This learning process performs a dynamic programming-like learning step to update the $\hat{h}$-values of the expanded states (like LSS-LRTA*).

Nancy using Experience

Given the general Nancy framework, a central question is how to obtain the belief $\mathcal{B}(s)$ about the cost-to-go associated for each state s. In DDNancy, we construct the distributions out of data generated in an offline phase. The data takes the form of h, h^* pairs, yielding a distribution of h^* values for each h value that DDNancy looks up during search.

Gathering these h, h^* pairs is done in the offline training phase. First, we run an initial search on a set of representative training instances. Each state expanded during this search is then solved optimally, and its h and h^* value are recorded. The set of all h^* values seen for a particular h makes up the distribution used later in the online phase. As the algorithm for the initial search, we used weighted A* with a weight of 2 and the same heuristic function that will be used online. The motivation for weighted A* as the data-gathering tool is that both weighted A* and real-time heuristic search are sub-optimal heuristic search algorithms. In that sense, the visited states during the weighted A* search could be typical, and thus make the data more informative for online real-time search. The data generation process is done for each domain individually such that the set of states seen during this process serves as a more accurate approximation of the states that DDNancy will see in this domain. This requires new data to be generated for each domain DDNancy is supposed to run on. However, this way we account for different kinds of heuristic error in different benchmark domains and obtain more representative and informed data.

In assumption-based Nancy, updating the beliefs of the expanded states is done by updating the $\hat{h}$-value by propagating the heuristic value of their successor with lowest expected cost. DDNancy instead propagates the belief distribution of the successor, shifting all samples by the cost of the action.

Theoretical Analysis

In this section, we show that Nancy is complete. We assume that: (A1) action costs are greater than 0, (A2) for every state, there is a goal reachable from it, (A3) all initial beliefs have finite expected value, (A4) the state space is finite. Our proof follows the style of Korf's (1990) proof for RTA* and Bulitko and Sampley's (2016) proof for Weighted Lateral LRTA* (wbLRTA*): we first prove that incompleteness implies that there must exist a subset of states within which Nancy circulates forever. Then we prove that there cannot exist such a set due to the updates made by the learning rule of Nancy.

Definition 1 A subset of states $S_\circ$ is called a *circulating set* if there exists a time $t_\circ$ after which the agent will visit only states $s \in S_\circ$ and visit each one an infinite number of times.

Lemma 1 Under assumptions (A2) and (A4), if a real-time search algorithm is incomplete, it must have a circulating set $S_\circ$.

Proof. Since a goal is reachable from all states (A2)[1], a real-time search only terminates when it reaches a goal state, so incompleteness means that the search never terminates. Because the state space is finite (A4), there must exist a subset of non-goal states $S_\circ$ such that the agent will re-visit each of the states in $S_\circ$ an infinite number of times after some initial time t. Let S be the set of non-goal states. If there exist states $s \in S$ that are not visited an infinite number of times, let the last time such a state s is visited be t_s. Then $t_\circ := \max(t, t_s)$ satisfies the claim. □

Definition 2 A real-time search is called *goal aware* if, upon generating a goal state in its lookahead, it commits to the path towards it.

Nancy is goal aware (Algorithm 1, line 6) and so is LSS-LRTA* (Koenig and Sun 2008, Figure 5, line 32).

Lemma 2 Under assumptions (A2) and (A4), if a goal-aware real-time search algorithm has a circulating set $S_\circ$, then a) there exists a finite set of non-goal states $S_\infty \supseteq S_\circ$ that are expanded infinitely often, b) every successor state

[1]Each state having at least one successor would be sufficient for this argument. This is implied by assumption (A2), which we use instead for simplicity.

$s' \in S_F := \{s[\![a]\!] \mid s \in S_\infty, a \in A(s)\} \setminus S_\infty$ appears infinitely often in the frontier of lookaheads from states in $S_\circ$, c) there is a time t_1 after which no $s' \in S_F$ is expanded, and d) S_F is non-empty.

Proof. The lookaheads from all states $s \in S_\circ$ are performed infinitely often with a fixed number of expansions, so for each $s \in S_\circ$ there must be a set of states $S^s_\infty \supseteq \{s\}$ that are expanded infinitely often, and so is their union S_∞. S_∞ cannot contain a goal, as a goal-aware search would head towards it, breaking the circulation. The neighbor states S_F are obviously generated infinitely often from S_∞; as they are not in S_∞, they are expanded only a finite number of times so the claimed time t_1 exists. S_F is non-empty because S_∞ does not contain a goal state, but the goal is reachable from all states (A2), so S_F must contain at least one state on a path to the goal. All of these sets must be finite since the state space is finite (A4). □

Definition 3 A learning algorithm is called *dynamic programming-like* if it updates the heuristic values of the states S expanded in the local search space such that afterwards the heuristic values of all $s \in S$ satisfy

$$\hat{h}(s) = \min_{a \in A(s)} (c(a) + \hat{h}(s[\![a]\!]))$$

A standard result is that, if updates of the form $\hat{h}(s) := \min_{a \in A(s)}(c(a) + \hat{h}(s[\![a]\!]))$ are performed infinitely often on a finite state-space graph with positive action costs, starting from arbitrary finite initial values the state values will eventually converge, i.e., all satisfy that equation (Bertsekas and Tsitsiklis 1996, Proposition 2.3).

Lemma 3 Under assumptions (A1)-(A4) with S_∞ as in Lemma 2, if a real-time search algorithm that performs dynamic programming-like learning has a circulating set $S_\circ$, then there exists a time t_2 after which, for every $s \in S_\infty$, we have $\hat{h}(s) = \min_{a \in A(s)}(c(a) + \hat{h}(s[\![a]\!]))$.

Proof. Consider the state space sub-graph S' induced by $S_\infty \cup S_F$. The $\hat{h}$ values of states $s_f \in S_F$ are never updated, since they are never expanded. All update operations performed on S' are well defined, i.e., consider successors contained in S'. By construction, the update procedure is called infinitely often on every state $s \in S_\infty$. Due to the convergence of the dynamic programming-like learning procedure with positive action costs (A1) and bounded initial $\hat{h}$ values (A3), the $\hat{h}$- values of these states will eventually converge to a solution of the state-update equation as claimed. □

In the following, we will use f_s and g_s to denote the f value and g value, respectively, of a state in a lookahead search space with respect to the current root state s of the lookahead.

Definition 4 A real-time search algorithm is called *persistent* if, whenever it generates a state s_f on the lookahead frontier from the current state s with $\hat{f}_s(s_f) < \hat{f}_s(s)$ or $\hat{f}_s(s_f) = \hat{f}_s(s)$ and $\hat{h}(s_f) < \hat{h}(s)$, it will eventually select s_f itself for expansion, or a state s' where $\hat{f}_s(s') < \hat{f}_s(s_f)$ or $\hat{f}_s(s') = \hat{f}_s(s_f)$ and $\hat{h}(s') < \hat{h}(s_f)$.

Lemma 4 Under assumptions (A1)-(A4), a persistent and goal-aware real-time search algorithm that does dynamic programming-like learning cannot have a circulating set.

Proof. By contradiction. Assume that the search algorithm does have a circulating set $S_\circ$. Then there must be sets S_∞ and S_F as per Lemma 2. Due to Lemma 3, we know that $\hat{h}$ will eventually be consistent on all states $s \in S_\infty$ at some point in time t_2. Let $s_f = \arg\min_{s \in S_F} \hat{h}(s)$ be a never-expanded frontier node with minimal $\hat{h}$ value among such frontier nodes. Since s_f is generated infinitely often, it will be generated in some lookahead at time $t_3 > t_2$. Since heuristic values are converged on S_∞, all states in that lookahead have the same $\hat{f}_s$ value, and s_f has the minimal $\hat{h}$ value among those states. Since the search is persistent, it will eventually select s_f for expansion, or a state s' with $\hat{f}_s(s') < \hat{f}_s(s_f)$ or $\hat{f}_s(s') = \hat{f}_s(s_f)$ and $\hat{h}(s') < \hat{h}(s_f)$. Observe that, for every $s \in S_\infty$, we have $\hat{h}(s) > \hat{h}(s_f)$ due to convergence and positive action costs (A1). Therefore, the state s' cannot be contained in S_∞.

Thus the search must eventually select a state not in S_∞ for expansion, implying that it must eventually select a state in S_F for expansion, in contradiction to its definition. □

Theorem 1 Under assumptions (A1)-(A4), a persistent and goal-aware real-time search with dynamic programming-like learning will eventually reach a goal.

Proof. If the search never reaches a goal, it has a circulating set by Lemma 1, which by Lemma 4 is not possible. □

Lemma 5 Under assumption (A4), Nancy is a persistent real-time search algorithm.

Proof. If the lookahead from a state s provides a state s' with $\hat{f}_s(s') < \hat{f}_s(s)$ or $\hat{f}_s(s') = \hat{f}_s(s)$ and $\hat{h}(s') < \hat{h}(s)$, Nancy will start moving towards that state, unless the search is already moving towards a state s'' with $\hat{f}_s(s'') < \hat{f}_s(s')$ or $\hat{f}_s(s'') = \hat{f}_s(s')$ and $\hat{h}(s'') < \hat{h}(s')$ (see Algorithm 1, line 7). Let s_{best} be the state that Nancy is currently moving towards. There are two cases. In the first case, s_{best} is eventually selected for expansion and therefore Nancy is persistent. For the second, assume that s_{best} is never selected for expansion. Then $\hat{h}(s_{best})$ can never change, since the learning phase only updates $\hat{h}$ values of expanded states. By line 7 of the Nancy algorithm, the only way s_{best} would not be selected for expansion is if a different and strictly better state s'_{best} were to be discovered. Because the state space is finite (A4), Nancy can discover and switch to such a better state only a finite number of times, so eventually Nancy will reach and select the state she is heading toward for expansion, thereby satisfying persistence. □

Corollary 1 Under assumptions (A1)-(A4), Nancy will eventually reach a goal.

Proof. Nancy is a persistent and goal-aware real-time search algorithm (Lemma 5) with dynamic programming-like learning, and is thereby complete under assumptions (A1)-(A4) via Theorem 1. □

We can also adapt these ideas to show that LSS-LRTA* is complete even for heuristics that are not consistent or admissible (a consistent heuristic is the only case proven in the original paper). One avenue is to modify the notion of persistence to only hold over the circulating set, and show LSS-LRTA* has that behavior. However, it seems simpler to proceed without the notion of persistence.

Theorem 2 Under assumptions (A1)-(A4), if LSS-LRTA* breaks ties deterministically, it is complete.

Proof. For the purpose of contradiction, assume LSS-LRTA* has a circulating set $S_\circ$. Then there must be sets S_∞ and S_F as per Lemma 2 at a time $t_1 \geq t_\circ$. Let $s_f = \arg\min_{s_\circ \in S_\circ, s \in S_F} f_{s_\circ}(s)$, breaking ties by h and then deterministically following our assumption. Since s_f is generated infinitely often, it will be generated in some lookahead from a state $s \in S_\circ$ at time $t' > t_1$. Let $s, s_0, \ldots, s_n$ be the states expanded in that lookahead in order of expansion. Since the heuristic is consistent on the expanded states (Lemma 3), there are no duplicates in $s, s_0, \ldots, s_n$, as each state is only expanded once during the lookahead. Let $\pi = \langle a_0, \ldots, a_n \rangle$ be the shortest path to s_f. Since s_f has minimal f_s (states in S_∞ must have greater or equal f_s due to consistency) and ties are broken in its favor, after the lookahead, the agent will move to $s[\![a_0]\!]$. In the learning phase, nothing changes as $\hat{h}$ is already converged. In the next lookahead, the f-value of all states along π will be reduced by $c(a_0)$. Observe that the f-value of all states $s, s_0, \ldots, s_n$ can not decrease by more than $c(a_0)$, because otherwise that cheaper path would have been found in the lookahead from s. Let s_p be the predecessor of s_f in the lookahead from s (i.e. the second-to-last state on the state sequence induced by π). Since $f_{s[\![a_0]\!]}(s_p) = f_s(s_p) - c(a)$ and there is no other state whose f-value decreased by more than $c(a)$, by the assumption of deterministic tie-breaking, s_p can only move to an earlier position in the expansion order. Furthermore, there cannot be a state s' that was not expanded in the lookahead from s but would now be expanded before s_p, because then it must have $f_s(s') + c(a) \leq f_{s[\![a_0]\!]}(s) \leq f_{s[\![a_0]\!]}(s_p)$, and since that implies $f_s(s') \leq f_s(s_p)$, it would have been expanded before s_p in the previous lookahead as well with the assumption of deterministic tie breaking. Thus, s_p is expanded again in the lookahead from $s[\![a_0]\!]$, generating s_f. Since s_f has minimal f and ties are deterministically broken in its favor, LSS-LRTA* will move towards s_f. This reasoning applies repeatedly. The only way s_f would not eventually be expanded is that a state with even lower f-value (or equal f but lower h) would be generated. However, such a state must be outside $S_\infty \cup S_F$, because the states on the path toward s_f have equal and minimal f-value, and s_f has minimal h among those due to consistency. Therefore, a state in S_F will eventually be expanded, a contradiction to its construction. Thus, by Lemma 1, LSS-LRTA* must be complete. □

Experimental Results

We evaluated our new variants of Nancy and compared them against LSS-LRTA* as a baseline. We present results in both classical planning domains and sliding-tile puzzles. In all tested scenarios, we vary the size of the lookahead for each algorithm. This constrains how much information each algorithm can possibly obtain in a single lookahead phase and tests their effectiveness for different limits thereof. Our performance metric is the total solution cost to reach the goal.

We test three variants of Nancy against LSS-LRTA*: (1) Nancy: the original assumption-based Nancy algorithm (Mitchell et al. 2019), (2) Nancy (pers.): the persistent variant of the original algorithm, and (3) Nancy (DD): our persistent data-driven variant that replaces the assumption-based belief distributions with data.

Classical Planning

To evaluate our new algorithms on classical planning benchmark domains, we extended Fast Downward (Helmert 2006) to facilitate real-time search algorithms bounded by a number of expansions and implemented LSS-LRTA* and our new Nancy variants in this framework. Since we need to gather data for each specific domain that we want to evaluate, we chose to focus our experiments on a few selected domains, Blocksworld, Transport, Elevators, where preliminary experiments showed assumption-based Nancy to be relatively ineffective compared to LSS-LRTA*. For Transport and Elevators, we include their unit-cost versions, and we omit the original-cost version of Elevators as it has zero-cost actions, making the considered algorithms incomplete. We ran the experiments on a cluster of Intel Xeon E5-2660 machines using the lab framework (Seipp et al. 2017). We choose $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009) due to it being a popular admissible heuristic that achieves competitive performance without requiring the tuning of many parameters, and used the standard limits of 30 minutes and 4 GB memory in all experiments.

The data for the data-driven variant was generated on the same number of instances (185) using the same size parameters as the IPC instances. On these instances we computed h^* for all states that were expanded by weighted A*. The set of h^*-values forms a distribution for each h-value which is looked up by DDNancy during search. Distributions for h-values that were not seen during training and can therefore not be looked up are extrapolated from the next lower h-value.

Table 1 shows the results of these experiments. We test lookaheads of size 100, 300, and 1,000 nodes. As a general trend, solution cost tends to decrease with larger lookahead for each algorithm, since a larger lookahead usually leads to more informed decisions. Comparing the individual algorithms with each other, we take LSS-LRTA* as the baseline. The original Nancy variant performs comparatively worse in almost all tested scenarios. The persistent assumption-based Nancy variant improves on the original Nancy algorithm across all domains, sometimes dramatically. Compared to LSS-LRTA*, however, it still performs poorly in Transport. The data-driven Nancy variant manages to perform well in all test cases. It stays competitive with all other algorithms and frequently has the lowest solution cost overall, even in transport where the assumption-based variant faltered.

Domain	L	LSS-LRTA*	Nancy	Nancy (pers.)	Nancy (DD)
Blocksw. (35)	100	46	67	**33**	38
	300	36	46	**30**	34
	1000	30	44	32	**27**
Transport (60)	100	631	1116	615	**496**
	300	519	705	559	**485**
	1000	499	607	567	**422**
Transport (60) (unit-cost)	100	48	79	40	**31**
	300	47	43	**30**	34
	1000	35	36	29	**27**
Elevators (30) (unit-cost)	100	50	55	**35**	39
	300	32	40	**29**	30
	1000	34	31	27	**26**

Table 1: Geometric means of the solution cost on instances solved by all algorithms. The limit on the number of expanded nodes in the lookahead is denoted by L.

Sliding Tiles

We also evaluate our algorithms on the classic 100 15-puzzle instances published by Korf (1985). We used two variations: uniform-cost, in which every actions costs one, and heavy, in which the action cost is equal to the label of the moved tile. We use the Manhattan distance heuristic with weighted A* during data-gathering and the final online real-time search. Weighted A* was run on 500 random tile instances to record all visited states. For each observed h-value, we found the 200 most frequently visited states, solved them optimally, and used the resulting h^* values to form the h^* belief distributions for that h-value. A* with Manhattan distance cannot solve arbitrary heavy-tile puzzle instances within reasonable time and memory limits, so we implemented two 6-tile pattern databases (0,4,5,6,7,8,9 and 0,10,11,12,13,14,15) and used them with IDA*$_{CR}$ (Sarkar et al. 1991) to solve these instances and gather the h^* values.

Figure 1 shows mean solution cost relative to LSS-LRTA* on uniform-cost 15-puzzle, with error bars indicating 95% confidence intervals. We used lookahead limits of 30, 100, 300, and 1,000 expanded nodes. The persistent variant of the original assumption-based Nancy performs very similarly to the original version, while DDNancy does not perform as well as assumption-based Nancy. We conjecture that this is because, due to the popularity of the 15-puzzle as a heuristic search benchmark, the assumptions embodied in Nancy were likely heavily informed by search behavior in this domain, the Gaussian assumption on the heuristic error is not seriously violated in this domain, and the smoothness of its model may in fact aid the algorithm's performance.

Figure 2 shows corresponding results from the heavy-cost 15-puzzle. All three variants of Nancy are clearly better than LSS-LRTA* for lookaheads 30, 100, and 300. DDNancy performs better than assumption-based Nancy at lookaheads of 300 and higher, and still beats LSS-LRTA* at a lookahead bound of 1000 expansions.

Discussion

Our experimental results show that search algorithm performance can be heavily influenced by its assumptions and that performance can sometimes be greatly enhanced by using data instead. This enables the algorithm to use more accurate information to support its reasoning and decision making. In this way, the algorithm can be adapted to perform well in scenarios where the assumptions turn out to be inaccurate. However, this places the burden on the data to be more informative, and inaccurate data can lead to undesirable behavior just like wrong assumptions can. Furthermore, it is not obvious how to replace certain assumptions. For example, our variant of DDNancy continues to use the assumption of reduced variance to estimate its post-expansion belief $\mathcal{B}_{post}$. Developing methods to gather more informative data to replace additional assumptions is an area that requires further attention. The current greedy one-step lookahead belief

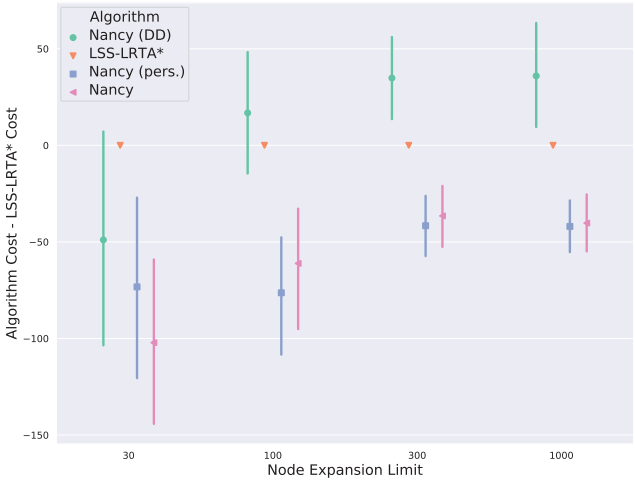

Figure 1: Solution cost compared to LSS-LRTA* on the uniform-cost variant of the 15-puzzle.

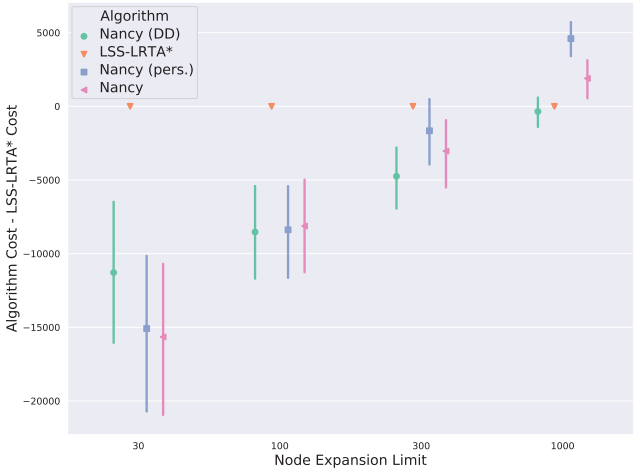

Figure 2: Solution cost compared to LSS-LRTA* on the heavy-cost variant of the 15-puzzle.

$\mathcal{B}_{post}$ could also be generalized to an n-steps post-search approach, which might yield more accurate risk values.

A limiting factor when using data is the number of features used to learn it. DDNancy maps a single heuristic value to a distribution of h^* values. It is possible that the heuristic may over-generalize certain states which can make the data misleading. How to use additional features like a second heuristic or state variables effectively to make the data more informative is also subject to further research. Previous work on learning heuristic functions, such as that of Arfaee, Zilles, and Holte (2011) for example, may prove helpful in this context.

Conclusion

We advanced the use of explicit beliefs about heuristic error in heuristic search by extending the real-time Nancy algorithm to handle arbitrary belief distributions gathered from data. We proved that our generalized version of Nancy is complete, given only minimal assumptions (e.g., the beliefs have finite expected value). Our experiments suggest that DDNancy can perform well, even in domains where assumption-based Nancy falls short, demonstrating the potential benefit of replacing assumptions with actual data. While we anticipate that assumptions will continue to be useful in situations where it is hard to gather representative data, we also expect that replacing additional assumptions will result in further insights and even higher performance.

Acknowledgments. Maximilian Fickert was funded by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science).

References

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 2075–2098.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena Scientific.

Bulitko, V., and Sampley, A. 2016. Weighted lateral learning in real-time heuristic search. In *Ninth Annual Symposium on Combinatorial Search*.

Burns, E., and Ruml, W. 2013. Iterative-deepening search with on-line tree size prediction. *Annals of Mathematics and Artificial Intelligence* 69(2):183–205.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

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

Kiesel, S.; Burns, E.; and Ruml, W. 2015. Achieving goals quickly using real-time search: experimental results in video games. *Journal of Artificial Intelligence Research* 54:123–158.

Koenig, S., and Sun, X. 2008. Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and Multi-Agent Systems* 18(3):313—341.

Korf, R. E. 1985. Iterative-deepening-A*: An optimal admissible tree search. In *Proceedings of IJCAI-85*, 1034–1036.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Mitchell, A.; Ruml, W.; Spaniol, F.; Hoffmann, J.; and Petrik, M. 2019. Real-time planning as decision-making under uncertainty. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*.

Mutchler, D. 1986. Optimal allocation of very limited search resources. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence (AAAI-86)*.

Pemberton, J. C., and Korf, R. E. 1994. Incremental search algorithms for real-time decision making. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*.

Rose, K.; Burns, E.; and Ruml, W. 2011. Best-first search for bounded-depth trees. In *Fourth Annual Symposium on Combinatorial Search*.

Sarkar, U. K.; Chakrabarti, P. P.; Ghose, S.; and De Sarkar, S. 1991. Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence* 50(2):207–221.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Stern, R.; Felner, A.; van den Berg, J.; Puzis, R.; Shah, R.; and Goldberg, K. 2014. Potential-based bounded-cost search and anytime non-parametric a*. *Artificial Intelligence* 214:1–25.

Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of the Twenty-first International Conference on Automated Planning and Scheduling (ICAPS-11)*.