Probably Approximately Correct Heuristic Search

Roni Stern   Ariel Felner
Information Systems Engineering
Ben Gurion University
Beer-Sheva, Israel 85104
roni.stern@gmail.com   felner@bgu.ac.il

Robert Holte
Computing Science Department
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
holte@cs.ualberta.ca

Abstract

A* is a best-first search algorithm that returns an optimal solution. w-admissible algorithms guarantee that the returned solution is no larger than w times the optimal solution. In this paper we introduce a generalization of the w-admissibility concept that we call *PAC search*, which is inspired by the PAC learning framework in Machine Learning. The task of a *PAC search algorithm* is to find a solution that is w-admissible with high probability. In this paper we formally define PAC search, and present a framework for PAC search algorithms that can work on top of any search algorithm that produces a sequence of solutions. Experimental results on the 15-puzzle demonstrate that our framework activated on top of Anytime Weighted A* (AWA*) expands significantly less nodes than regular AWA* while returning solutions that have almost the same quality.

Introduction

Consider a standard search problem of finding a path in a state space from a given initial state s to a goal state g. Throughout the paper we use the standard search notation: $g(n)$ is the sum of the edge costs from s to n and $h(n)$ is an admissible heuristic that estimates the cost of getting from n to g. The optimal solution can be found using the A* algorithm (Hart, Nilsson, and Raphael 1968), a best-first search algorithm that uses a cost function of $f(n) = g(n) + h(n)$. Furthermore, if $h^*(s)$ is the cost of the optimal solution then all the nodes with $g + h < h^*(s)$ must be expanded in order to verify that no better path exist (Dechter and Pearl 1985).

While developing accurate heuristics can greatly reduce the number of nodes with $g + h < h^*(s)$, it has been shown that in many domains even with an almost perfect heuristic expanding all the nodes with $g + h < h^*(s)$ is not feasible within reasonable computing resources. (Helmert and Röger 2008)

When finding an optimal solution is not feasible, a range of search algorithms have been proposed that return suboptimal solutions. In particular, when an algorithm is guaranteed to return a solution that is at most w times the optimal solution we say that this algorithm is w-admissible. Weighted A* (Pohl 1970), A^*_ϵ (Pearl and Kim 1982), Anytime Weighted A* (Hansen and Zhou 2007) and Optimistic

Search (Thayer and Ruml 2008) are known examples of w-admissible algorithms. In general, w-admissible search algorithms achieve w-admissibility by using an admissible heuristic to obtain a lower bound on the optimal solution. When the ratio between the *incumbent solution* (i.e., the best solution found so far) and this lower bound is below w, then w-admissibility is guaranteed. Efficient w-admissible algorithms often introduce a natural tradeoff between solution quality and search runtime. When w is high, solutions are returned quickly but have poorer quality.[1] By contrast, setting low w values will increase the search runtime but solutions of higher quality will be returned.

In this paper we argue that it is possible to develop a search algorithm that will run much faster than traditional w-admissible algorithm by relaxing the strict w-admissibility requirement. Specifically, search problems can be solved much faster if one allows the returned solution to be w-admissible in *most* of the cases instead of *always*.

For example, consider Weighted A* (WA*), a best-first search algorithm which uses the cost $f(n) = g(n) + w \cdot h(n)$. It has been proven that WA* with weight w is w-admissible. For example, if the requirement is that a solution must be 1.25-admissible, then w will be set to 1.25. However, it is often the case that setting a higher weight may often return a solution that is also 1.25-admissible with high probability. For example, on the standard 100 random 15-puzzle instances (Korf 1985), we have found that running WA* with $w = 1.5$ always returns a solution that is 1.25-admissible. Importantly, running WA* with $w = 1.5$ often returns a solution faster than WA* with $w = 1.25$. Consequently, if one requires a 1.25-admissible solution with high probability for a 15-puzzle instance, running WA* with $w = 1.5$ would be a better choice.

Inspired by the Probably Approximately Correct (PAC) learning framework from Machine Learning (Valiant 1984), we formalize the notion of finding a w-admissible solution with high probability. We call this concept *Probably Approximately Correct Heuristic Search*, or *PAC search* in short. A PAC search algorithm is given two parameters, ϵ and δ, and is required to return a solution that is at most $1 + \epsilon$ times the optimal solution, with probability higher than

---

[1]In some domains, increasing w above some point also degrades the search runtime.

$1 - \delta$. We call $1 + \epsilon$ the *desired suboptimality*, and $1 - \delta$ the *required confidence*.

We first introduce and formally define what is PAC search. Then, we present a general framework for a PAC search algorithm. This framework can be built on top of any algorithm that produces a sequence of solutions. When the search algorithm produces a solution that achieves the desired suboptimality with the required confidence, the search halts and the *incumbent solution* (= the best solution found so far by the search algorithms) is returned. This results in a PAC search algorithm that can accept any positive value ϵ and δ and return a solution within these bounds.

Empirical evaluation were performed on the 15-puzzle using our PAC search algorithm framework on top of Anytime Weighted A* (Hansen and Zhou 2007). The results show that decreasing the required confidence (1-δ) indeed yields a reduction in the number of expanded nodes. For example, setting $\delta = 0.05$ decreases the number of expanded nodes by a factor of 3 when compared to regular Anytime Weighted A* (where $\delta = 0$) for various values of ϵ.

Related Work

Ernandes and Gori (2004) have previously pointed out the possible connection between the PAC learning framework and heuristic search. They used an artificial neural network (ANN) to generate a heuristic function $\hat{h}$ that is only *likely admissible*, i.e., admissible with high probability. They showed experimentally that A* with $\hat{h}$ as its heuristic can solve the 15 puzzle quickly and return the optimal solutions in many instances. This can be viewed as a special case of the PAC search concept presented in this paper, where $\epsilon = 0$ and δ is allowed values larger than zero.

In addition, they bound the quality of the returned solution to be a function of two parameters: 1) $P(\hat{h} \uparrow)$, which is the probability that $\hat{h}$ is overestimating the optimal cost, and 2) d, the length (number of hops) of the optimal path to a goal. Specifically, the probability that the path found by A* with $\hat{h}$ as a heuristic is optimal is given by $(1 - P(\hat{h} \uparrow))^d$. Unfortunately, this formula is only given as a theoretical observation. In practice, the length of the optimal path to a goal d is not known until the problem is solved optimally, and thus this bound cannot be used to identify whether a solution is probably optimal in practice.

Other algorithms that use machine learning techniques to generate accurate heuristics have also been proposed (Samadi, Felner, and Schaeffer 2008; Jabbari Arfaee, Zilles, and Holte 2010). The resulting heuristics are not guaranteed to be admissible but were shown to be very effective. No theoretical analysis of the amount of suboptimality was performed for these algorithms.

The PAC framework has been borrowed from machine learning to other fields as well. For example, a data mining algorithm that is probably approximately correct (Cox, Fu, and Hansen 2009) has been proposed. Also, a framework for probably approximately optimal strategy selection was also proposed, for the problem of finding an efficient execution order of sequence of experiments with probabilistic outcomes (Greiner and Orponen 1990). To our knowledge,

---

Algorithm 1: PAC search algorithm framework

Input: $1 + \epsilon$, The required suboptimality
Input: $1 - \delta$, The required confidence
1   $U \leftarrow \infty$
2   **while** *Improving U is possible* **do**
3      $NewSolution \leftarrow$ search for a solution
4   **if** $NewSolution < U$ **then**
5         $U \leftarrow NewSolution$
6   **if** U *is a PAC solution* **then return** U

---

the PAC framework has not yet been adapted to heuristic search.

PAC Heuristic Search

PAC Learning is a framework for analyzing machine learning algorithms and the complexity of learning classes of concepts. A learning algorithm is said to be a *PAC learning algorithm* if it generates with probability higher than $1 - \delta$ a hypothesis with an error rate lower than ϵ, where δ and ϵ are parameters of the learning algorithm. Similarly, we say that a search algorithm is a *PAC search algorithm*, if it returns with high probability $(1 - \delta)$ a solution that has the desired suboptimality (of $1 + \epsilon$).

Formal Definition

We now define formally what is a *PAC search algorithm*. Let $\mathcal{M}$ be the set of all possible start states in a given domain, and let D be a distribution over $\mathcal{M}$. Correspondingly, we define a random variable S, to be a state drawn randomly from $\mathcal{M}$ according to distribution D. For a search algorithm A and a state $s \in \mathcal{M}$, we denote by $cost(A, s)$ the cost of the solution returned by A given s as a start state. We denote by $h^*(s)$ the cost of the optimal solution for state s. Correspondingly, $cost(A, S)$ is a random variable that consists of the cost of the solution returned by A for a state randomly drawn from $\mathcal{M}$ according to distribution D. Similarly, $h^*(S)$ is a random variable that consists of the cost of the optimal solution for a random state S.

Definition 1 *[PAC search algorithm]*
An algorithm A is a PAC search algorithm iff

$$Pr(cost(A, S) \leq (1 + \epsilon) \cdot h^*(S)) > 1 - \delta$$

Classical search algorithms can be viewed as special cases of a PAC search algorithm. Algorithms that always return an optimal solution, such as A* and IDA*, are simply PAC search algorithms that set both ϵ and δ to zero. w-admissible algorithm are PAC search algorithms where $w = 1 + \epsilon$ and $\delta = 0$. In this work we propose a general framework for a PAC search algorithm that is suited for any non-negative value of ϵ and δ.

Framework for a PAC Heuristic Search Algorithm

We now introduce a general *PAC search framework* (PAC-SF) to obtain PAC search algorithms. The main idea of PAC-

SF, is to generate better and better solutions and halt whenever the incumbent solution has the desired suboptimality $(1+\epsilon)$ with the required confidence $(1-\delta)$. The pseudo code for PAC-SF is listed in Algorithm 1. The incumbent solution U is initially set to be ∞ (line 1). In every iteration of PAC-SF, a search algorithm is run until a solution with cost lower than U is found (line 3), or it is verified that such a solution does not exist (line 2). When the incumbent solution U is a PAC solution, i.e., it is $(1+\epsilon)$-admissible with high probability (above $1-\delta$), the search halts and U is returned (line 6).

Implementing PAC-SF introduces two challenges:

1. **Choosing a search algorithm (line 3).**
   This is the fundamental challenge in a search problem: how to find a solution.

2. **Identifying when to halt (line 6).**
   This is the challenge of identifying when the incumbent solution U ensures that the desired suboptimality has been achieved with the required confidence.

Anytime search algorithms can address the first challenge. Anytime algorithms are: "algorithms whose quality of results improves gradually as computation time increases" (Zilberstein 1996). After the first solution is found, an anytime search algorithm continues to run, finding solutions of better qualities.[2] This is exactly what is needed for PAC-SF (line 3 in Algorithm 1). Prominent examples of anytime search algorithms are Anytime Weighted A* (Hansen and Zhou 2007), Beam-Stack Search (Zhou and Hansen 2005) and Anytime Window A* (Aine, Chakrabarti, and Kumar 2007). Note that the incumbent solution U can be passed to an anytime search algorithm for pruning purposes, e.g., pruning all the nodes with $g + h \geq U$ with an admissible h.

Some anytime algorithms are guaranteed to eventually find the optimal solution. Consequently, if PAC-SF is built on top of such an algorithm, then it is guaranteed to return a solution for any values of ϵ and δ. This is because whenever the optimal solution is found, then the incumbent solution U is optimal and can be safely returned for any ϵ and δ. For simplicity, we assume hereinafter that the search algorithm used in PAC-SF is an anytime search algorithm that converges to the optimal solution.

Identifying a PAC Solution

We now turn to address the second challenge in PAC-SF, which is how to identify when the incumbent solution U is a PAC solution, and the search can terminate (line 6 in Algorithm 1).

Definition 2 *[Sufficient PAC Condition]*
A sufficient PAC condition is a termination condition for PAC-SF that ensures that PAC-SF will return a solution for

---

[2]For some anytime algorithms, it is not guaranteed that a returned solution is necessarily of higher quality than a previously returned solution. However, this can be easily remedied by continuing to run such anytime algorithms, until a solution that is better than all of the previously returned solutions is found.

a randomly drawn state that is $(1+\epsilon)$-admissible with probability of at least $1 - \delta$.

PAC-SF with an anytime search algorithm that converges to the optimal solution is guaranteed to be a PAC search algorithm (as defined in Definition 1) iff it halts (and returns a solution) when a *sufficient PAC condition* has been met.

How do we recognize when a *sufficient PAC condition* has been met? For a given start state s, a solution of cost U is $(1 + \epsilon)$-admissible if the following equation holds.

$$U \leq h^*(s) \cdot (1 + \epsilon) \tag{1}$$

However, Equation 1 cannot be used in practice as a *sufficient PAC condition* in PAC-SF, because $h^*(s)$ is known only when an optimal solution has been found. Next, we present several practical *sufficient PAC conditions*, that are based on Equation 1.

Trivial PAC Condition

Recall that S denotes a randomly drawn start state. The following condition is a sufficient PAC condition.

$$Pr(U \leq h^*(S) \cdot (1 + \epsilon)) > 1 - \delta \tag{2}$$

The key idea here is to assume we know nothing about a given start state s except that it was drawn from the same distribution as S (i.e., drawn from $\mathcal{M}$ according to distribution D). With this assumption, the random variable $h^*(S)$ can be used in place of $h^*(s)$ in Equation 1.

To use the sufficient PAC condition depicted in Equation 2, the distribution of $h^*(S)$ is required. $Pr(h^*(S) \geq X)$ can be estimated in a preprocessing stage by randomly sampling states from S. Each of the sampled states is solved optimally, resulting in a set of h^* values. The cumulative distribution function $Pr(h^*(S) \geq X)$ can then be estimated by simply counting the number of instances with $h^* \geq X$, or using any statistically valid curve fitting technique. A reminiscent approach was used in the KRE formula (Korf, Reid, and Edelkamp 2001) for predicting the number of nodes generated by IDA*, where the state space was sampled to estimate the probability that a random state has a heuristic value $h \leq X$.

Note that the procedure used to sample the state space should be designed so that the distribution of the sampled states will be as similar as possible to the real distribution of start states. In some domains this may be difficult, while in other domains sampling states from the same distribution is easy. For example, sampling 15-puzzles instances from a uniform distribution over the state space can be done by generating a random permutation of the 15 tiles and verifying mathematically that the resulting permutation represents a solvable 15-puzzle instance (Johnson 1879). Sampling random states can also be done in some domains by performing a sequence of random walks from a set of known start states.

Clearly, a *sufficient PAC condition* based on $Pr(h^*(S) > X)$ is very crude, as it ignores all the state attributes of the initial state s. For example, if for a given start state s we have $h(s) = 40$ and h is admissible then $h^*(s)$ cannot be below 40. However, if one of the randomly sampled states has h^* of 35 then we will have $Pr(h^*(S) < 40) > 0$.

Ratio-based PAC Condition

Next, we propose an alternative sufficient PAC condition, that regards the heuristic value of the start state s. Instead of considering the distribution of $h^*(S)$, consider the distribution of the ratio between h^* and h for a random start state S. We denote this as $\frac{h^*}{h}(S)$. Similarly, the cumulative distribution function $Pr(\frac{h^*}{h}(S) > Y)$ is the probability that a random start state S (i.e., drawn from from $\mathcal{M}$ according to distribution D) has $\frac{h^*}{h}$ larger than a value Y. This allows the following sufficient PAC condition.

$$Pr(\frac{h^*}{h}(S) \geq \frac{U}{h(s) \cdot (1 + \epsilon)}) > 1 - \delta \qquad (3)$$

Equation 3 can be seen as a simple extension of Equation 2, where both sides are divided by the heuristic estimate of the random state, and given the heuristic value of the specific start state s. It is easy to see that Equation 3 is indeed a sufficient PAC condition. The benefit of this condition is that the heuristic estimate of the given start state ($h(s)$) is considered.

Estimating $Pr(\frac{h^*}{h}(S) \geq Y)$ in practice can be done in a similar manner that was described above for estimating $Pr(h^*(S) \geq X)$. First, random problem instances are sampled. Then, collect $\frac{h^*}{h}$ values instead of h^* values, and generate the corresponding cumulative distribution function $Pr(\frac{h^*}{h}(S) \geq Y)$. Note that if h is admissible then $Pr(\frac{h^*}{h}(S) \geq 1) = 1$.

Our experiments (detailed below) were performed on the 15-puzzle domain - a standard search benchmark, using the additive 7-8 PDB as a heuristic (Korf and Felner 2002; Felner, Korf, and Hanan 2004). We use these experiments to demonstrate how the sampling process described above can be done. First, the distribution of $\frac{h^*}{h}$ for the 15-puzzle and the additive 7-8 PDB heuristic has been learned as follows. The standard 1,000 random 15-puzzle instances (Felner, Korf, and Hanan 2004) were solved optimally using A*. The ratio $\frac{h^*}{h}$ was calculated for the start state of every instance. Figure 1 presents the resulting cumulative and probability distribution functions. The x-axis displays values of $\frac{h^*}{h}$. The blue bars which correspond to the left y-axis show
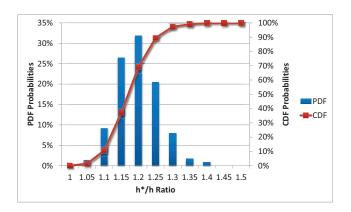

Figure 1: $\frac{h^*}{h}$ distribution for the additive 7-8 PDB heuristic.

the probability of a problem instance having a specific $\frac{h^*}{h}$ value. In other words the blue bars show the probability distribution function (PDF) of $\frac{h^*}{h}$ which is $Pr(\frac{h^*}{h} = X)$. The red curve, which corresponds to the right y-axis, shows the cumulative distribution function (CDF) of $\frac{h^*}{h}$, i.e., given X the curve shows $Pr(\frac{h^*}{h} \leq X)$.

Next, assume that we are given as input the start state s, $\epsilon = 0.1$ and $\delta = 0.1$. Now assume that $h(s) = 50$ and a solution of cost 60 has been found (i.e., $U = 60$). According to sufficient PAC condition depicted in Equation 3, the search can halt when:

$$Pr(\frac{h^*}{h}(S) \geq \frac{U}{h(s) \cdot (1 + \epsilon)}) > 1 - \delta$$

Setting U=60, $h(s)$=50, ϵ=0.1 and δ=0.1, we have:

$$Pr(\frac{h^*}{h}(S) \geq 1.09) > 0.9$$

The probability that $\frac{h^*}{h}(S) \geq 1.09$ can be estimated with the CDF displayed in Figure 1. As indicated by the red dot above the 1.1 point of the x-axis (according to the right y-axis), $Pr(\frac{h^*}{h}(S) < 1.09) \leq 0.1$ and consequently $Pr(\frac{h^*}{h}(S) \geq 1.09) > 0.9$. Therefore, the sufficient PAC condition from Equation 3 is met and the search can safely return the incumbent solution (60) and halt. By contrast, if the incumbent solution were 70, then $\frac{U}{h(s) \cdot (1+\epsilon)} = 1.27$, and according to the CDF in Figure 1 $\frac{h^*}{h}$ is lower than 1.27 with probability that is higher than 90%. Therefore, in this case the sufficient PAC condition is not met, and the search will continue, seeking for a better solution than 70.

It is important to note that the process of obtaining the distribution of $Pr(\frac{h^*}{h}(S) \geq X)$ is done in a preprocessing stage, as it requires solving a set of instances optimally. This is crucial since optimally solving a set of instances may be computationally expensive (if finding optimal solutions is easy there is no need for PAC search). This expensive preprocessing stage is done only once per domain and heuristic. By contrast, the actual search is performed per problem instance. Usually one implements a search algorithm to be used for many problem instances. Therefore the cost of expensive preprocessing stage should be amortized over the gain achieved for all instances that will be solved by the implemented algorithm.

Experimental Results

Next, we demonstrate empirically the benefits of PAC-SF on the 15-puzzle, which is a standard search benchmark. A solution was returned when the sufficient PAC condition described in Equation 3 was met, using the distribution of $\frac{h^*}{h}$ shown in Figure 1. For producing solution (line 3 in Algorithm 1) we have used Anytime Weighted A* (Hansen and Zhou 2007), which is an anytime variant of Weighted A* (Pohl 1970). Weighted A* (WA*) is a w-admissible best-first search algorithm which uses $f(n) = g(n) + w \cdot h(n)$ for its cost function. Anytime Weighted A* (AWA*) differs from WA* in the termination condition. While WA* halts when a goal is expanded, AWA* continues to search,

			AWA*+PAC			
$1+\epsilon$	$1-\delta$	A*	w=1.1	w=1.25	w=1.5	w=2.0
1	0.5	37,688	21,048 (0.98)	21,913 (0.92)	26,989 (0.90)	43,573 (0.89)
	0.8		21,988 (0.99)	23,194 (0.98)	28,367 (0.98)	44,949 (0.98)
	0.9		21,995 (1.00)	23,213 (1.00)	28,391 (1.00)	44,980 (1.00)
	0.95		22,005 (1.00)	23,230 (1.00)	28,409 (1.00)	45,002 (1.00)
	1		22,007 (1.00)	23,236 (1.00)	28,416 (1.00)	45,008 (1.00)
1.1	0.5			4,841 (0.99)	7,143 (0.89)	18,826 (0.84)
	0.8			14,761 (1.00)	18,345 (0.96)	33,031 (0.93)
	0.9			19,020 (1.00)	23,504 (0.98)	38,926 (0.97)
	0.95			20,619 (1.00)	25,466 (1.00)	41,558 (0.99)
	1		9,369 (1.00)	23,165 (1.00)	28,335 (1.00)	44,935 (1.00)
1.25	0.5				1,043 (1.00)	1,637 (0.90)
	0.8				1,257 (1.00)	2,566 (0.93)
	0.9				1,496 (1.00)	5,046 (0.97)
	0.95				1,786 (1.00)	5,961 (0.99)
	1			1,865 (1.00)	7,079 (1.00)	17,715 (1.00)
1.5	0.5					533 (0.99)
	0.8					566 (1.00)
	0.9					582 (1.00)
	0.95					603 (1.00)
	1				993 (1.00)	782 (1.00)

Table 1: Average number of nodes expanded until PAC-SF returned a solution.

returning better and better solutions. Eventually, AWA* will converge to the optimal solution and halt.

In the experiments we varied the following parameters:

- **Weights** (w for AWA*): 1.1, 1.25, 1.5 and 2.
- **Desired suboptimality** ($1 + \epsilon$): 1, 1.1, 1.25 and 1.5.
- **Required confidence** ($1 - \delta$): 0.5, 0.8, 0.9, 0.95 and 1.

Table 1 shows the number of nodes expanded until PAC-SF returned a solution. The data in every cell of the table is the average over the standard 100 random 15-puzzle instances (Korf 1985). For reference, the average number of nodes expanded by A*, which is 37,688, is given the in the column denoted by A*. Note that these 100 instances are not included in the standard 1,000 random instances (Felner, Korf, and Hanan 2004) that were used to estimate $Pr(\frac{h^*}{h}(S) > X)$. This was done to separate the training set from the test set. The values in brackets show the ratio of instances where the solution returned indeed achieved the desired suboptimality, i.e, when the cost of the solution was no more than $1 + \epsilon$ times the optimal solution.

Until the first solution has been found, AWA* behaves exactly like Weighted A*. Therefore it is guaranteed that the first solution found by AWA* with weight w is not larger than w times the optimal solution (Pohl 1970). For example, if the desired suboptimality is 1.25, 1.5 or 2, the number of nodes expanded by AWA* with $w = 1.25$ is exactly the same, and the confidence that the found solution achieves the desired suboptimality is 1.0. Therefore, for $w = 1 + \epsilon$ we report only results with confidence 1.0 (for reference) and omit the results for $w < 1 + \epsilon$ since as explained above they are exactly the same as the results for $w = 1 + \epsilon$.

As can be seen, all the values in the brackets exceed the required confidence significantly. Therefore, in this do-main PAC-SF succeeds in returning solutions that achieve the desired suboptimality with confidence higher than the required confidence. Note that in this domain PAC-SF with AWA* is very conservative. For example, AWA* with w=1.5 achieved a suboptimality of 1.25 for all the 100 instances, even when the required confidence was only 0.5 ($1 - delta = 0.5$). This suggests that it may be possible to further improve the proposed PAC identification technique in future work.

Now, consider the number of nodes expanded when $w > 1 + \epsilon$. Clearly, for every value of suboptimality (i.e., every value of $1 + \epsilon$), decreasing the required confidence ($1 - \delta$) reduced the number of nodes expanded. This means that relaxing the required confidence indeed allowed returning solutions of the desired quality faster. For example, consider AWA* with $w = 1.5$, where the desired suboptimality is 1.25 (1+ε=1.25). If the required confidence is 1.0, then AWA* with $w = 1.5$ expanded 7,079 nodes. On the other hand, by relaxing the required confidence to 95% (i.e., 1-δ=0.95), AWA* expanded only 1,786 nodes. Interestingly, even when the required confidence is set to 95%, AWA* with $w = 1.5$ was able to find a 1.25-admissible solution in 100% of the instances (see the value in the brackets).

Conclusion and Future Work

In this paper we adapt the *probably approximately correct* concept from machine learning to heuristic search. A PAC heuristic search algorithm, denoted as PAC search algorithm, is defined as an algorithm that returns with high probability a solution with cost that is w-admissible. A general framework for a PAC search algorithm is presented. This framework can use any algorithm that returns a sequence of solutions, and obtain a PAC search algorithm. A major chal-

lenge in PAC search is to identify when a found solution is good enough. We propose an easy-to-implement technique for identifying such a solution, based on sampling and estimating the ratio between the optimal path and the heuristic estimate of the start state.

Empirical evaluation on the 15-puzzle demonstrate that the proposed PAC search algorithm framework is able to indeed find solutions with the desired suboptimality with probability higher than the required confidence. Furthermore, by allowing AWA* to halt when the desired suboptimality is reached with high probability (but not 100%), AWA* is able to find solutions faster (i.e., expanding less nodes) than regular AWA*, which halts when the desired suboptimality is guaranteed.

We currently plan to extend this work in several directions. One of the directions that we are currently perusing is to develop better *sufficient PAC conditions* that exploit the knowledge gained during the search (e.g., exploiting the openlist in a best-first search). Another research direction is to obtain a more accurate $\frac{h^*}{h}$ distribution by using an abstraction of the state space, similar to the *type system* concept used to predict the number of states generated by IDA* in the CDP formula (Zahavi et al. 2010). States in the state space will be grouped into types, and each type will have a corresponding $\frac{h^*}{h}$ distribution. A third research direction is how to adapt the choice of which node to expand next to incorporate the value of information gained by expanding each node.

Acknowledgments

References

Aine, S.; Chakrabarti, P. P.; and Kumar, R. 2007. AWA* - a window constrained anytime heuristic search algorithm. In *IJCAI*, 2250–2255.

Cox, I.; Fu, R.; and Hansen, L. 2009. Probably approximately correct search. In *Advances in Information Retrieval Theory*, volume 5766 of *Lecture Notes in Computer Science*. Springer. 2–16.

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery* 32(3):505–536.

Ernandes, M., and Gori, M. 2004. Likely-admissible and sub-symbolic heuristics. In *European Conference on Artificial Intelligence (ECAI)*, 613–617.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)* 22:279–318.

Greiner, R., and Orponen, P. 1990. Probably approximately optimal satisficing strategies. *Artificial Intelligence* 82:21–44.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 28:267–297.

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* SSC-4(2):100–107.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *AAAI*, 944–949.

Jabbari Arfaee, S.; Zilles, S.; and Holte, R. C. 2010. Bootstrap learning of heuristic functions. In *Symposium on Combinatorial Search (SoCs)*.

Johnson, W. W. 1879. Notes on the "15" Puzzle. *American Journal of Mathematics* 2(4):397–404.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1-2):9–22.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artificial Intelligence* 129(1-2):199–218.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible treesearch. *Artificial Intelligence* 27(1):97–109.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4(4):392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193 – 204.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In *AAAI*, 357–362.

Thayer, J. T., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 355–362.

Valiant, L. G. 1984. A theory of the learnable. *Communications of the ACM* 27:1134–1142.

Zahavi, U.; Felner, A.; Burch, N.; and Holte, R. C. 2010. Predicting the performance of IDA* using conditional distributions. *Journal of Artificial Intelligence Research (JAIR)* 37:41–83.

Zhou, R., and Hansen, E. A. 2005. Beam-stack search: Integrating backtracking with beam search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 90–98.

Zilberstein, S. 1996. Using anytime algorithms in intelligent systems. *AI Magazine* 17(3):73–83.