Search-Aware Conditions for 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

The notion of finding a solution that is approximately optimal with high probability was recently introduced to the field of heuristic search, formalized as *Probably Approximately Correct Heuristic Search*, or PAC search in short. A big challenge when constructing a PAC search algorithm is to identify when a given solution achieves the desired sub-optimality with the required confidence, allowing the search to halt and return the *incumbent solution*. In this paper we propose two novel methods for identifying when a PAC search can halt. Unlike previous work, the new methods provided in this paper become more knowledgeable as the search progresses. This can speedup the search, since the search can halt earlier with the proposed methods and still keeping the desired PAC solution quality guarantees. Experimental results indeed show a substantial speedup of the search in comparison to the previous approach for PAC search.

1 Introduction

Many Artificial Intelligence applications and algorithms employ search algorithms to solve optimization problems. *Optimal search algorithms* are search algorithms that are guaranteed to return optimal solutions. A* (Hart, Nilsson, and Raphael 1968), IDA* (Korf 1985) and RBFS (Korf 1993) are examples of optimal search algorithms. In practice and in theory, finding an optimal solution with an optimal search algorithm is often intractable, even if one is given an extremely accurate heuristic to guide the search (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*. Such algorithms are also referred to as *bounded-suboptimal* algorithms. Weighted A* (Pohl 1970), A^*_ϵ (Pearl and Kim 1982), Optimistic Search (Thayer and Ruml 2008) and Skeptical Search (Thayer, Dionne, and Ruml 2011) are known examples of w*-admissible* algorithms.

Recently (Stern, Felner, and Holte 2011), it has been shown that it is possible to develop search algorithms that will run much faster than traditional w*-admissible* algorithms by allowing the search algorithm to return a solution that is w-admissible in *most* of the cases instead of *always*. Inspired by the Probably Approximately Correct (PAC) learning framework from Machine Learning (Valiant 1984), the notion of finding a w-admissible solution with high probability was formalized as *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 $1 - \delta$. The parameters $1 + \epsilon$ and $1 - \delta$ are referred to as the *desired suboptimality* and *required confidence*, respectively.

A big challenge when constructing a PAC search algorithm is to identify when a given solution achieves the desired suboptimality with the required confidence, allowing the search to halt and return the *incumbent solution* (=the best solution found so far). This type of condition is called a *PAC condition*. Previous work (Stern, Felner, and Holte 2011) has addressed this challenge by considering the heuristic of the start state (the value assigned to the start state by the heuristic function) and a probability distribution of the ratio between the heuristic and the true cost. While shown to be effective, the resulting *PAC condition*s ignore the knowledge gained throughout the search. In this paper we propose two novel *PAC conditions*. These new methods become more knowledgeable as the search progresses, and can identify more accurately when to halt. This results in substantial speedup of the search, and still keeping the desired solution quality.

2 PAC Heuristic Search

Next, we provide a formal description of a PAC search algorithm, and introduce relevant notation.

Let $\mathcal{M}$ be the set of all possible start states in a given domain, and let D be a probability 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

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

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 paper we aim at the more general case, where ϵ and δ may both be non-zero.

3 PAC Search and δ-Risk Admissibility

The concept of PAC Search is reminiscent of the *δ-risk-admissibility* concept defined in the seminal work on semi-admissible heuristic search by Pearl and Kim (1982). To explain what is *δ-risk-admissibility* and its relation to PAC search, we first explain the notion of *risk*, as defined by Pearl and Kim.

Assume that a search algorithm finds a solution of cost C. If the search algorithm cannot guarantee that the cost of the optimal (i.e., lowest cost) solution is C, then returning C holds a *risk*, that a better solution of cost lower than C exists. Pearl and Kim (1982) defined that this risk is quantified by a *risk function*, denoted by $R(C)$. Given a risk function $R(C)$, they defined a δ-risk-admissible algorithm as follows.

Definition 2 [δ-risk-admissibility]
An algorithm is said to be δ-risk-admissible if it always terminates at a solution cost C such that $R(C) \leq \delta$ for each node left in OPEN.

It is important to note that the risk function, as defined by Pearl and Kim, is a function of *C for a given node* in OPEN. They denoted the risk function by $R(C)$, assuming the node is understood from the context. One of the risk functions that was proposed by Pearl and Kim is the probability that a node n has $f^*(n) < C$.

Consider the difference between a PAC search algorithm with desired suboptimality zero ($\epsilon = 0$) and a δ-risk-admissible algorithm with such a risk function $R(C)$, which is the probability of a node n having $f^*(n) < C$. A δ-risk-admissible algorithm must verify that:

$$\forall n \in OPEN \qquad Pr(f^*(n) < C) < \delta$$

By contrast, a PAC search algorithm must verify that:

$$Pr(\bigvee_{n \in OPEN} f^*(n) < C) < \delta$$

In other words, a PAC search algorithm must verify that δ is larger than the joint probability of having a node in OPEN being part of a solution of cost smaller than C.

Hence, there is a crucial difference between PAC search and δ-risk-admissibility, demonstrated by the following example. Assume that a solution of cost C has been found,

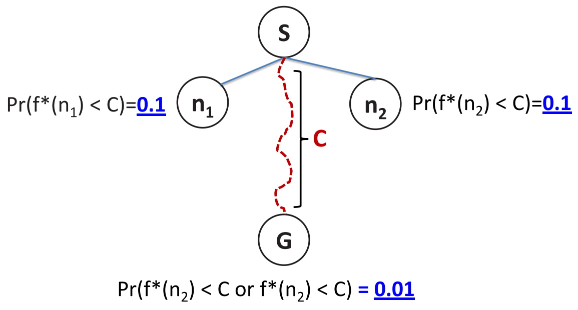

Figure 1: Example of δ-risk-admissible vs. PAC search

$\delta = 0.1$, and there are two nodes in OPEN, n_1 and n_2. Also, assume that the probability that $f^*(n_1) < C$ is 0.1, and similarly the probability that $f^*(n_2) < C$ is also 0.1. Figure 1 illustrates this example. Clearly, a δ-risk-admissible algorithm can return the found solution (of cost C). However, a PAC search algorithm can return C only if the joint probability of $f^*(n_1) < C$ or $f^*(n_2) < C$ is smaller than or equal to 0.1. For example, if the joint probability $Pr(f^*(n_1) < C$ or $f^*(n_2) < C) = 0.15$, then a PAC search algorithm will not be satisfied with the solution C, and will need to search for a better solution.

Our definition of PAC search is motivated by considering the client of the search algorithm. A client of a search algorithm is interested in the quality of the solution that is returned, and less with bounds on particular nodes in the search tree. As shown in the example above (Figure 1), there are many cases where a δ-risk-admissible algorithm will return a solution that is not optimal with probability higher than δ.

Note that Pearl and Kim also proposed a δ-risk-admissible algorithm, called R^*. The R^* algorithm is a best-first search that expands nodes according to their risk function ($R(C)$). In order to calculate the risk function $R(C)$, the state space is sampled (as a preprocessing stage), and a probability distribution function of $h^*(n)$ is obtained. R^* returns a solution when a goal node is expanded. While Pearl and Kim have shown that R^* is indeed δ-risk-admissible, it is easy to see that it is not a PAC search algorithm. Note that in the rest of this paper we follow Pearl and Kim by assuming that the state space can be sampled in a representative manner.

4 PAC Conditions

One can view a PAC search algorithm as having two components. The first component is an *anytime search algorithm*, i.e., a search algorithm "whose quality of results improves gradually as computation time increases" (Zilberstein 1996). Note that the quality of results in a search algorithm usually corresponds to the cost of the solution that was found. The second component identifies when to halt the first component and return the incumbent solution. It is the responsibility of the second component to ensure that the desired suboptimality $(1 + \epsilon)$ has been achieved by the incumbent solution with the required confidence $(1 - \delta)$. This second component is called a *sufficient PAC condition* or simply *PAC condition*, defined as follows.

Definition 3 *[Sufficient PAC Condition]*
A sufficient PAC condition is a termination condition for a search algorithm ensuring that for a randomly drawn state, this search algorithm will return a solution that is $(1+\epsilon)$-admissible with probability of at least $1-\delta$.

Given an anytime search algorithm and a *PAC condition*, one can construct a PAC search algorithm by running the anytime search algorithm, and halting when the PAC condition has been met. While many anytime search algorithms exist (Hansen and Zhou 2007; Aine, Chakrabarti, and Kumar 2007; Likhachev et al. 2008), only two simple *PAC conditions* have been proposed (Stern, Felner, and Holte 2011). Next, we describe the existing *PAC conditions* and point out their shortcomings. Then, we provide new, more accurate *PAC conditions*, to address these shortcomings.

4.1 Blind PAC Condition

For a given start state s, a solution of cost U is $(1+\epsilon)$-admissible if the following equation holds.

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

Clearly, Equation 1 cannot be used in practice as a *PAC condition*, because $h^*(s)$ is known only when an optimal solution has been found. Previous work proposed the two following *PAC conditions*.

All the PAC conditions consider the distribution of a randomly drawn state S. The first PAC condition that was introduced is called the *blind* PAC condition, as it assumes we know nothing about the 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, resulting in the blind PAC condition depicted in Equation 2.

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

To use the *blind* PAC condition, 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 (Korf, Reid, and Edelkamp 2001) and CDP (Zahavi et al. 2010) and ϵ-truncation (Lelis, Zilles, and Holte 2011) formulas 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 $h \leq X$. Similar sampling is also proposed by Pearl and Kim (1982) in the δ-risk admissibility work mentioned above.

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 (Haslum et al. 2007; Domshlak, Karpas, and Markovitch 2010).

A major drawback of the *blind* PAC condition is that it ignores all the state attributes of the initial state s. One such attribute is the heuristic function. 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 $h^* = 35$ then we will have $Pr(h^*(S) < 40) > 0$. The rest of the PAC conditions described in this paper indeed consider the values of the heuristic function.

4.2 Ratio-based PAC Condition (RPAC)

The ratio-based PAC condition (Stern, Felner, and Holte 2011), denoted as RPAC, that is described next, considers the heuristic value of the start state s. Instead of considering the distribution of $h^*(S)$, the distribution of the ratio between h^* and h for a random start state S is considered. We denote this as $\frac{h^*}{h}(S)$. Similarly, the cumulative distribution function $Pr(\frac{h^*}{h}(S) > X)$ 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 X. This allows the following PAC condition.

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

Using Equation 3 requires estimating $Pr(\frac{h^*}{h}(S) \geq Y)$. This can be done as follows. First, random problem instances are sampled. Then, collect $\frac{h^*}{h}$ values, and generate the corresponding cumulative distribution function $Pr(\frac{h^*}{h}(S) \geq X)$. Note that if h is admissible then $Pr(\frac{h^*}{h}(S) \geq 1) = 1$.

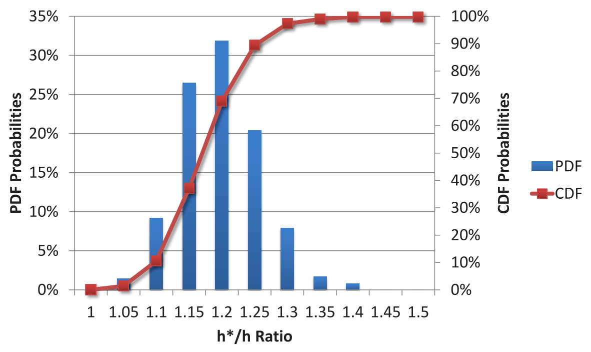

Figure 2: $\frac{h^*}{h}$ distribution for the additive 7-8 PDB heuristic.

As an example of the sampling process described above, we estimated the distribution of $\frac{h^*}{h}$ for the additive 7-8 PDB heuristic (Felner, Korf, and Hanan 2004) for the standard search benchmark of the 15-puzzle, as follows. The standard 1,000 random 15-puzzle instances (Felner, Korf, and Hanan 2004) were solved optimally using A*. Then, the ratio $\frac{h^*}{h}$

was calculated for the start state of every instance. Figure 2 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 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 $Pr(\frac{h^*}{h} \leq X)$.

As an example, assume that we are given as input the start state s, $\epsilon = 0.1$ and $\delta = 0.1$. Also assume that $h(s) = 50$ and that 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:

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

Setting U=60, $h(s)$=50, ϵ=0.1 and δ=0.1, we have that the search can halt if:

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

The probability that $\frac{h^*}{h}(S) \geq 1.09$ can be estimated with the CDF displayed in Figure 2. As indicated by the red dot above the 1.1 point of the x-axis (according to the right y-axis), $Pr(\frac{h^*}{h}(S) < 1.09)$ is slightly smaller than 0.1 and consequently $Pr(\frac{h^*}{h}(S) \geq 1.09)$ is slightly larger than 0.9. Therefore, the sufficient PAC condition from Equation 3 is satisfied 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 2 $\frac{h^*}{h}$ is smaller than 1.27 with probability that is higher than 90%. Therefore, in this case the PAC condition is not met, and the search will continue, seeking 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. Expensive preprocessing is very common in supervised learning, where a training set is obtained and learning algorithms are applied to construct a classifier from the training set (Mitchell 1997). In addition, preprocessing is also common in the heuristic search community. For example, it is used in the construction of pattern databases heuristics (Culberson and Schaeffer 1998; Korf 1997; Felner, Korf, and Hanan 2004; Holte et al. 2006; Haslum et al. 2007), heuristics for 2D-pathfinding problems (Sturtevant et al. 2009; Pochter et al. 2010; Goldenberg et al. 2011), learning-based heuristics (Ernandes and Gori 2004; Samadi, Felner, and Schaeffer 2008; Jabbari Arfaee, Zilles, and Holte 2011), search effort prediction formulas (Korf, Reid, and Edelkamp 2001; Zahavi et al. 2010; Lelis, Zilles, and Holte 2011) and search cost prediction formulas (Lelis, Stern, and Jabbari Arfaee 2011; Lelis et al. 2012).

In some cases it is not possible to solve problems optimally even in a preprocessing stage. In this paper we restrict the discussion to domains where such preprocessing is possible. However, In such cases one may use the recently developed solution cost prediction algorithm (Lelis, Stern, and Jabbari Arfaee 2011; Lelis et al. 2012) instead of solving instances optimally, to obtain an accurate approximation of the optimal solution for the sampled problems.

5 Search-Aware PAC Conditions

The two PAC conditions described in the previous section (and in our previous work), can be used regardless of the search algorithm that is used in the actual search. Such a search algorithm finds a solution, and the blind PAC condition or RPAC are used to identify if the found solution can be returned and the search may halt.

While very general, this decoupling of the PAC condition from the search algorithm means that these PAC conditions will only consider halting the search when a new incumbent solution is found. Finding a better incumbent solution can be very time consuming when searching in combinatorially large state spaces, and it seems wasteful to ignore all the knowledge gathered during the search.

Next, we present two novel PAC conditions that are aware of the underlying search, and consider the knowledge it gathers about the searched state space. Specifically, we focus on the case where the search algorithm used is *anytime best-first search* algorithm, e.g., AWA* (Hansen and Zhou 2007), ARA* (Likhachev et al. 2008) and APTS (Stern, Puzis, and Felner 2011). [1] Anytime best-first search algorithms are anytime algorithms that run a best-first search.

5.1 Lower-Bounded Ratio-Based PAC Condition

Best-first search algorithms maintain an open-list (denoted hereinafter as OPEN), that contain all the nodes that have been generated but not expanded. These nodes are the frontier of the search, and therefore any optimal solution contains at least of one of these nodes. Several previous papers (Likhachev et al. 2008; Hansen and Zhou 2007) have exploited this fact to obtain a lower bound on the optimal cost, as follows. Let $g(n)$ be the sum of the edge costs from the start to node n and let $h(n)$ be an admissible heuristic estimation of the cost from node n to a goal. Then $f_{min} = \min_{n \in OPEN}(g(n) + h(n))$ is a lower bound on the optimal solution. Note that f_{min} may change after every node expansion, and it is always a lower bound on the optimal solution. As such, the maximal f_{min} seen during the search, denoted by $maxf_{min}$, is also a lower bound on the optimal solution. Clearly, $\frac{maxf_{min}}{h(s)} \leq \frac{h^*(s)}{h(s)}$. Therefore, the PAC condition in Equation 3 can be refined, allowing a PAC search to return a solution faster. This refined PAC condition is as follows:

Corollary 1 *[Lower bounded ratio-based PAC Condition] The following equation is a sufficient PAC condition:*

$$Pr(\frac{maxf_{min}}{h(s)} \leq \frac{h^*}{h}(S) < \frac{U}{h(s) \cdot (1+\epsilon)}) < \delta$$

[1] APTS is also known as ANA* (van den Berg et al. 2011).

Proof: Recall the PAC condition in Equation 3:

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

Clearly $\frac{maxf_{min}}{h(s)} \leq \frac{h^*(s)}{h(s)}$, since $maxf_{min}$ is a lower bound on the optimal cost. Thus, we can consider only the probability that $Pr(\frac{maxf_{min}}{h(s)} \leq \frac{h^*}{h}(S) < \frac{U}{h(s)\cdot(1+\epsilon)})$ as the probability that the desired suboptimality was not achieved.$\square$

Consider the difference between RPAC and the lower-bounded ratio-based PAC condition, denoted as RPAC+LB. For RPAC, if the PAC condition has not been met yet, it could only be met after a new incumbent solution was found. By contrast, the RPAC+LB can be met even if no new incumbent solution has been found. As the search progresses, $maxf_{min}$ increases, and the condition in Corollary 1 can be satisfied even if a new incumbent solution has not been found yet, unlike the previously presented PAC conditions. Therefore, a PAC solution might be identified faster using RPAC+LB than when using RPAC.

Note that the heuristic $h(n)$ used in both RPAC and RPAC+LB can be inadmissible. RPAC+LB only needs an admissible heuristic, $h_a(n)$, to maintain $maxf_{min}$. Thus, it is possible to calculate two heuristic functions for each state: an admissible $h_a(n)$, for maintaining $maxf_n$ and an inadmissible $h(n)$ to be used in Corollary 1 (and of course to order the search). This approach of having two heuristics for a state was previously proposed in w-admissible search algorithms such as Optimistic search (Thayer and Ruml 2008) and Explicit Estimation Search (Thayer and Ruml 2011).

RPAC+LB requires that $maxf_{min}$ be calculated. To do so efficiently, one is required to calculate f_{min} fast. With a consistent heuristic or by correcting the heuristic of generated nodes with Pathmax (Mero 1984), it is guaranteed that f_{min} is monotonic non-decreasing, and thus the current f_{min} will always be $maxf_{min}$ as well. While maintaining f_{min} incurs some overhead (e.g., by maintaining an additional priority queue which is ordered by f-values), it has been used in previous search algorithms (Hansen and Zhou 2007; Thayer and Ruml 2008; 2011).

5.2 Open-based PAC Condition

The PAC condition in Corollary 1 can be satisfied either when a better incumbent solution is found (decreasing U), or when the lower bound on the optimal cost increases (increasing $maxf_{min}$). However, $maxf_{min}$ will only increase after all the nodes with $g+h \leq maxf_{min}$ are expanded. In large combinatorial state spaces with a heuristic that is not perfect, there may be an exponential number of such nodes. Thus, an exponential number of nodes may be expanded without even considering any of the previously described PAC conditions.

To overcome this shortcoming, another PAC condition is presented next, named the *Open-based PAC condition*, or RPAC+OPEN in short. RPAC+OPEN is based on the knowledge gained from all the nodes in OPEN and can be satisfied after every single node is expanded, even before $maxf_{min}$ increases or U decreases. To describe RPAC+OPEN, several definitions are needed.

Definition 4 *[Reject]*
In a PAC search, a node n is said to reject *a cost U with respect to ϵ if*

$$g(n) + h^*(n)\cdot(1+\epsilon) < U$$

Intuitively, a node n *rejects* a cost U if the cost of the shortest path from the initial state to a goal state that passes through node n is small enough to reject the hypothesis that U has the desired suboptimality of $1+\epsilon$.

Lemma 2 *In a best-first search, if the optimal solution has not been found and every node n in OPEN does not reject a solution cost U, then U achieves the desired suboptimality (i.e., U is larger than 1+ϵ times the optimal solution).*

Proof: Proof by contradiction. Assume that U does not achieve the desired suboptimality. In other words, U is not $(1+\epsilon)$-admissible. This means that the optimal solution $h^*(s)\times(1+\epsilon)$ is smaller than U. Let $g^*(n)$ denote the optimal path from the initial state s to a node n. It is well-known that in a best-first search, as long as the optimal solution has not been found there exists a node m in OPEN that is part of the optimal solution, and whose g-value is the cost of the optimal path from the initial state s to that node.[2] In other words, $h^*(s) = g(m) + h^*(m)$. Since every node in OPEN does not *reject* the cost U, it holds that for node m:

$$h^*(s) = g(m) + h^*(m)\cdot(1+\epsilon) \geq U$$

This contradicts the assumption that U does not achieve the desired suboptimality, i.e.,

$$h^*(s)\cdot(1+\epsilon) \leq U \quad \square$$

When a node n is in OPEN, the value of $h^*(n)$ is not known. Thus, determining if a node n rejects a cost U is not possible in practice. However, it is possible to obtain the probability that a randomly drawn node with a given g and h values will reject U, by applying the same arguments used for RPAC (Equation 3).

Corollary 3 *The probability that a randomly drawn node with h value h_v and g value g_v will reject a cost U is:*

$$Pr(\frac{h^*}{h}(S) < \frac{1}{h_v}\cdot(\frac{U}{1+\epsilon} - g_v))$$

Proof: According to Definition 4, a randomly drawn node S with $g(S) = g_v$ and $h(S) = h_v$ will reject a cost U if:

$$(g_v + h^*(S))\cdot(1+\epsilon) < U$$
$$h^*(S) < \frac{U}{1+\epsilon} - g_v$$

Since $h(S) = h_v$, the probability that the above inequality will hold is given by $Pr(\frac{h^*}{h}(S) < \frac{1}{h_v}(\frac{U}{1+\epsilon} - g_v))$$\square$

[2]This is proved for A* in Lemma 1 of (Hart, Nilsson, and Raphael 1968), but the same proof holds for any best-first search.

The value $Pr(\frac{h^*}{h}(S) < \frac{1}{h_v}(\frac{U}{1+\epsilon} - g_v))$ is denoted by $P(U, h_v, g_v, \epsilon, \delta)$ or simply $P(U, h_v, g_v)$ when ϵ and δ are clear from the context. For a given node m, the value $P(U, h(m), g(m))$ will be denoted as $P(U, m)$. $P(U, m)$ can be viewed as the probability that node m will reject the cost U. Finally, the *Open-based PAC condition* can be presented.

Corollary 4 *[Open-based PAC condition]*
If for any $n_1, n_2 \in OPEN$ it holds that $P(U, n_1)$ and $P(U, n_2)$ are not negatively correlated, then the following is a sufficient PAC condition:

$$\sum_{n \in OPEN} \log(1 - P(U, n)) \geq \log(1 - \delta)$$

Proof: The shortest path from s to the goal must pass through one of the nodes in OPEN. Consequently, if all of the nodes in OPEN do not reject the cost U then U is a PAC solution. Given that all the knowledge available about a node in OPEN is its g value and its h value, the probability that a node $n \in OPEN$ does not reject the cost U is given by 1-P(U,n) according to Corollary 3. How to calculate the combined probability of all the nodes in OPEN not rejecting the cost U, depends on the correlation between these events. If these events are either independent or positively correlated, i.e., they are not negatively correlated as required in Corollary 4. In such cases, the probability that every node in OPEN do not reject U is lower bounded by $\prod_{n \in OPEN}(1 - P(U, n))$. Thus, once this expression is above 1-δ, a PAC condition is satisfied. A logarithm is applied to both sides to avoid precision issues,[3] resulting in the expression displayed in Corollary 4. □

The complexity of checking whether the *open-based PAC condition* is satisfied consists of calculating the expression $\sum_{n \in OPEN} \log(1-P(U, n))$, and comparing it to $\log(1-\delta)$ (see Corollary 4). Let $\hat{P}(U)$ denote this expression, i.e., $\hat{P}(U) = \sum_{n \in OPEN} \log(1 - P(U, n))$. Since $\log(1 - \delta)$ is constant, the complexity of checking the *open-based PAC condition* is dominated by the complexity of calculating $\hat{P}(U)$. Calculating $\hat{P}(U)$ can be done in an incremental manner efficiently after every node expansion. When a node in expanded, it exits OPEN, and its children are inserted to OPEN. Thus, when a node n is expanded, the value of $\hat{P}(U)$ should decrease by $\log(1 - P(U, n))$ and increase by $\log(1 - P(U, n'))$ for every child n' of n.[4] Note that, to reduce the number of logarithm calculations, one can cache logarithm values.

It is easy to see that updating $\hat{P}(U)$ as described above for a non-goal node can be done in $O(1)$ for every node generated. However, when a goal node is expanded and a better incumbent solution is found, U decreases. Consequently, when calculating $\hat{P}(U)$, the value $\log(1 - P(U, n))$ must be updated for every node n in OPEN. This requires an overhead of $O(|OPEN|)$ operations. However, this occurs only when a new incumbent solution is found. If the number of times the incumbent solution is updated is D, then the overhead of updating $\hat{P}(U)$ can be amortized over the cost of generating each node in OPEN, incurring an additional D operations per generated node.

6 Refined Heuristic Distribution

The distribution of $\frac{h^*}{h}(S)$ that are used by all the PAC conditions except RPAC+OPEN are used specifically for the start state. Thus, a representative distribution of $\frac{h^*}{h}(S)$ can be obtained by sampling random start states. Such a distribution is called the "overall" distribution of states in the state space (Holte et al. 2006). However, the open-based PAC condition requires calculating the distribution of $\frac{h^*}{h}(S)$ for every state generated during the search. This includes states that are very close to the goal an having low h-values. Holte et. al.(2006) have shown that the distribution of states seen during the search, called the "runtime" distribution, can be different than the distribution of states seen in by the "overall" distribution. Furthermore, in many domains the heuristic function is more accurate as one gets closer to the goal. This phenomenon has been observed previously, where the accuracy of a heuristic is improved as h becomes smaller (Stern, Puzis, and Felner 2011). Thus, obtaining a probability distribution of $\frac{h^*}{h}(S)$ by sampling only random states will not be accurate enough.

To overcome this we store a set of heuristic distributions, having a different distribution for different ranges of heuristic values. When the value of $Pr(\frac{h^*}{h}(S) > X)$ was required for RPAC+OPEN for a generated state n, a single heuristic distribution was chosen according to the value of $h(n)$. For example, states with $h(n) \in [1, 5]$ had one heuristic distribution, while states with $h(n) \in [6, 10]$ used a different distribution. This set of heuristic distributions was obtained by performing random walks of different length backwards from a goal node. This follows the state space sampling done by Zahavi et. al.(2010). In the following experimental results we used this *composite* heuristic for the Open-based PAC conditions.

7 Experimental Results

Next, we demonstrate empirically the benefits of the new PAC conditions on the 15-puzzle, which is a standard search benchmark. For simplicity, we used the Anytime Weighted A* (Hansen and Zhou 2007) algorithm as our search algorithm. Anytime Weighted A* (AWA*) is an anytime variant of Weighted A* (Pohl 1970). While WA* halts when a goal is expanded, AWA* continues to search, returning better and better solutions. Eventually, AWA* will converge to the optimal solution and halt. The experiments in this section were run on random 15-puzzle instance with the additive 7-8 PDB heuristic function, and using Anytime Weighted A* to produce solutions.

[3]Applying logarithm to use summation of very small negative numbers in stead of a product of fractions is a is commonly used technique, e.g., in likelihood calculations.

[4]If a child n' of n is already in OPEN, then $\hat{P}(U))$ should increase by the difference between $\log(1 - P(U, n'))$ and the value added for n' to $\hat{P}(U)$ when it was previously generated. Note that this value may differ from the current value of $\log(1 - P(U, n'))$, since $g(n')$ may have changed.

$1-\delta$	0.8	0.9	0.95	0.99	1
	$1+\epsilon$=1.00, AWA* w=1.2				
RPAC	21,876 (0.99)	21,911 (1.00)	21,924 (1.00)	21,930 (1.00)	21,930 (1.00)
RPAC+LB	18,745 (0.97)	21,453 (0.98)	21,745 (0.98)	21,930 (1.00)	21,930 (1.00)
RPAC+OPEN	18,402 (0.96)	21,385 (0.98)	21,744 (0.99)	21,927 (1.00)	21,929 (1.00)
	$1+\epsilon$=1.00, AWA* w=1.3				
RPAC	28,163 (0.98)	28,185 (1.00)	28,204 (1.00)	28,210 (1.00)	28,210 (1.00)
RPAC+LB	27,310 (0.96)	27,892 (0.99)	28,176 (0.99)	28,210 (1.00)	28,210 (1.00)
RPAC+OPEN	26,730 (0.97)	27,519 (0.99)	27,730 (1.00)	27,962 (1.00)	28,109 (1.00)
	$1+\epsilon$=1.10, AWA* w=1.2				
RPAC	8,019 (1.00)	9,257 (1.00)	9,697 (1.00)	10,125 (1.00)	10,125 (1.00)
RPAC+LB	3,617 (1.00)	5,581 (1.00)	6,165 (1.00)	6,306 (1.00)	6,327 (1.00)
RPAC+OPEN	3,340 (1.00)	3,377 (1.00)	3,857 (1.00)	4,269 (1.00)	4,344 (1.00)
	$1+\epsilon$=1.10, AWA* w=1.3				
RPAC	17,692 (0.96)	22,115 (0.98)	23,834 (1.00)	25,917 (1.00)	25,917 (1.00)
RPAC+LB	10,826 (0.96)	15,458 (0.98)	18,377 (1.00)	19,271 (1.00)	19,759 (1.00)
RPAC+OPEN	6,946 (0.96)	8,670 (1.00)	10,076 (1.00)	12,042 (1.00)	12,819 (1.00)
	$1+\epsilon$=1.20, AWA* w=1.3				
RPAC	1,921 (0.97)	2,672 (0.98)	3,594 (1.00)	5,669 (1.00)	5,669 (1.00)
RPAC+LB	1,882 (0.97)	2,318 (0.98)	2,970 (1.00)	3,307 (1.00)	3,899 (1.00)
RPAC+OPEN	1,545 (0.97)	2,080 (1.00)	2,216 (1.00)	2,449 (1.00)	2,907 (1.00)

Table 1: Performance of different PAC conditions. Nodes expanded (success rate)

In the following experiments the following parameters were varied: **(1)** weight of AWA* (1.2 and 1.3), **(2)** the desired suboptimality $1+\epsilon$ (1, 1.1 and 1.2) and **(3)** required confidence $1-\delta$ (0.8, 0.9, 0.95, 0.99 and 1).

Table 1 presents a comparison between the three sufficient PAC conditions that are based on the $\frac{h^*}{h}$ cumulative distribution: **(1)** RPAC, **(2)** RPAC+LB, and **(3)** RPAC+OPEN. Every data cell presents the average number of nodes expanded until the search was halted and the incumbent solution was returned. The values in brackets are the *success rate*, i.e., the percentage of instances where the returned solution indeed achieved the desired suboptimality. This was measured offline by comparing the returned solution with the known optimal solution.

As can be seen by the values in the brackets, the required confidence was always achieved for all of the sufficient PAC conditions. That is, the success rate of being within the desired suboptimality $(1+\epsilon)$ was always larger than the required confidence $1-\delta$ (shown in the top of the columns).

In terms of expanded nodes, it is clear that RPAC+LB outperforms RPAC (i.e., the ratio-based PAC condition in Equation 3), and that RPAC+OPEN outperforms all of the other PAC condition. When the desired suboptimality is 1 ($1+\epsilon=1$), the advantage of RPAC+OPEN over the other PAC conditions is minor. However, for higher values of ϵ the advantage of RPAC+OPEN over the other sufficient PAC conditions is substantial. For example, consider the number of nodes expanded with RPAC, RPAC+LB and RPAC+OPEN, for $1+\epsilon$=1.10 and $1-\delta$=0.99 using AWA* with w=1.2. Using RPAC, a solution (within the ϵ and δ requirements) was found after expanding 10,125 nodes on average, while RPAC+LB required only 6,306 nodes and RPAC+OPEN required 4,269. In the same setting, by decreasing $1-\delta$ to 0.9, RPAC expanded 9,257 nodes, RPAC+LB expanded 5,581 nodes and RPAC+OPEN expanded only 3,377 nodes.

Exact timing results are not provided because the CPU time of the different algorithms was very similar. The reason is that our implementation of the 15-puzzle is based on Korf's well-known, highly optimized 15-puzzle solver (Korf 1985). Thus, at each node, the vast majority of the CPU time was spent on accessing the large data structures such as the PDBs. Our algorithms differ only in simple variable bookkeeping and these did not influence the overall time.

8 Conclusions and Future Work

Previous work has adapted the *probably approximately correct* concept from machine learning to heuristic search, and proposed two simple conditions to identify when a search algorithm finds a probably approximately correct solution. These conditions only consider the heuristic of the start state and the value of the incumbent solution, but ignore all other knowledge that is gained by the search.

In this paper we presented two novel PAC conditions that take advantage of the states expanded during the search. Specifically, the first method considers the lower bound on the optimal solution that is obtained by the lowest f-value found during the search. The second method considers all the nodes that are in OPEN. We show the correctness of these conditions theoretically, and demonstrate empirically that using these conditions can yield substantial speedup.

There are several directions for future work and many open research questions. One research direction is to develop more effective *sufficient PAC conditions*. 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; Lelis, Stern, and Jabbari Arfaee 2011). 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. One possible way is to use a best-first search according to the *reject* probability, expanding in every iteration the node with the highest probability to reject the incumbent solution. This is similar to the R^* algorithm (Pearl and Kim 1982) described in Section 3, but instead of halting when a solution is found (like R^*), the search will continue until a PAC condition is met.

9 Acknowledgments

This research was supported by the Israel science foundation (ISF) under grant #305/09 to Ariel Felner.

References

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

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To max or not to max: Online learning for speeding up optimal planning. In *AAAI*.

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.

Goldenberg, M.; Sturtevant, N. R.; Felner, A.; and Schaeffer, J. 2011. The compressed differential heuristic. In *AAAI*.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *J. Artif. Intell. Res. (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.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, 1007–1012.

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

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artif. Intell.* 170(16-17):1123–1136.

Jabbari Arfaee, S.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artif. Intell.* 175(16-17):2075–2098.

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

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

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

Korf, R. E. 1993. Linear-space best-first search. *Artif. Intell.* 62(1):41–78.

Korf, R. E. 1997. Finding optimal solutions to rubik's cube using pattern databases. In *AAAI/IAAI*, 700–705.

Lelis, L.; Stern, R.; Zilles, S.; Holte, R.; and Felner, A. 2012. Predicting optimal solution cost with bidirectional stratified sampling. In *ICAPS*.

Lelis, L.; Stern, R.; and Jabbari Arfaee, S. 2011. Predicting solution cost with conditional probabilities. In *SOCS*.

Lelis, L.; Zilles, S.; and Holte, R. C. 2011. Improved prediction of IDA* performance via ϵ-truncation. In *SoCS*.

Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; and Thrun, S. 2008. Anytime search in dynamic graphs. *Artif. Intell.* 172:1613–1643.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *Artif. Intell.* 23(1):13 – 27.

Mitchell, T. M. 1997. *Machine Learning*. New York: McGraw-Hill.

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

Pochter, N.; Zohar, A.; Rosenschein, J. S.; and Felner, A. 2010. Search space reduction using swamp hierarchies. In *AAAI*.

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

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

Stern, R.; Felner, A.; and Holte, R. 2011. Probably approximately correct heuristic search. In *SoCS*.

Stern, R.; Puzis, R.; and Felner, A. 2011. Potential search: A bounded-cost search algorithm. In *ICAPS*.

Sturtevant, N. R.; Felner, A.; Barrer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. In *IJCAI*, 609–614.

Thayer, J. T., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS*, 355–362.

Thayer, J. T., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

Thayer, J.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *ICAPS*.

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

van den Berg, J.; Shah, R.; Huang, A.; and Goldberg, K. Y. 2011. Anytime nonparametric A*. In *AAAI*.

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.

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