# Analytical Results on the BFS vs. DFS Algorithm Selection Problem. Part I: Tree Search

Tom Everitt and Marcus Hutter

Australian National University, Canberra, Australia

**Abstract.** BFS and DFS are the two most fundamental search algorithms. We derive approximations of their expected runtimes in complete trees, as a function of tree depth and probabilistic goal distribution. We also demonstrate that the analytical approximations are close to the empirical averages for most parameter settings, and that the results can be used to predict the best algorithm given the relevant problem features.

## 1 Introduction

A wide range of problems in artificial intelligence can be naturally formulated as *search problems* (Russell and Norvig, 2010; Edelkamp and Schrödl, 2012). Examples include planning, scheduling, and combinatorial optimisation (TSP, graph colouring, etc.), as well as various toy problems such as Sudoku and the Towers of Hanoi. Search problems can be solved by exploring the space of possible solutions in a more or less systematic or clever order. Not all problems are created equal, however, and substantial gains can be made by choosing the right method for the right problem. Predicting the best algorithm is sometimes known as the *algorithm selection problem* (Rice, 1975).

A number of studies have approached the algorithm selection problem with machine learning techniques (Kotthoff, 2014; Hutter et al., 2014). While demonstrably a feasible path, machine learning tend to be used as a *black box*, offering little insight into *why* a certain method works better on a given problem. On the other hand, most existing analytical results focus on worst-case big-O analysis, which is often less useful than average-case analysis when selecting algorithm. An important worst-case result is Knuth's (1975) simple but useful technique for estimating the depth-first search tree size. Kilby et al. (2006) used it for algorithm selection in the SAT problem. See also the extensions by Purdom (1978), Chen (1992), and Lelis et al. (2013). Analytical IDA* runtime predictions based on problem features were obtained by Korf et al. (2001) and Zahavi et al. (2010). In this study we focus on *theoretical analysis* of *average runtime* of BFS and DFS. While the IDA* results can be interpreted to give rough estimates for average BFS search time, no similar results are available for DFS.

To facilitate the analysis, we use a probabilistic model of goal distribution and graph structure. Currently no method to automatically estimate the model parameters is available. However, the analysis still offers important theoretical insights into BFS and DFS search. The parameters of the model can also be interpreted as a *Bayesian prior belief* about goal distribution. A precise understanding of BFS and DFS performance is likely to have both practical and theoretical value: Practical, as BFS and DFS are both widely employed; theoretical, as BFS and DFS are two most fundamental ways to search, so their properties may be useful in analytical approaches to more advanced search algorithms as well.

Our main contributions are estimates of average BFS and DFS runtime as a function of tree depth and goal distribution (goal quality ignored). This paper focuses on the performance of *tree search* versions of BFS and DFS that do *not* remember visited nodes. Graph search algorithms are generally superior when there are many ways to get to the same node. In such cases, tree search algorithms may end up exploring the same nodes multiple times. On the other hand, keeping track of visited nodes comes with a high prize in memory consumption, so graph search algorithms are not always a viable choice. BFS tree search may be implemented in a memory-efficient way as iterative-deepening DFS (ID-DFS). Our results are derived for standard BFS, but are only marginally affected by substituting BFS with ID-DFS. The technical report (Everitt and Hutter, 2015a) gives further details and contains all omitted formal proofs. Part II of this paper (Everitt and Hutter, 2015b) provides a similar analysis of the *graph search* case where visited nodes are marked. Our main analytical results are developed in Section 3 and 4, and verified experimentally in Section 5. Part II of this paper offers a longer discussion of conclusions and outlooks.

## 2 Preliminaries

*Breadth-first search (BFS)* and *depth-first search (DFS)* are two standard methods for uninformed graph search. Both BFS and DFS assume oracle access to a *neighbourhood function* and a *goal check function* defined on a *state space*. BFS explores increasingly wider neighbourhoods around the start node. DFS follows one path as long as possible, and *backtracks* when stuck. The *tree search* variants of BFS and DFS do not remember which nodes they have visited. This has no impact when searching in trees, where each node can only be reached through one path. One way to understand tree search behaviour in general graphs is to say that they still *effectively* explore a tree; branches in this tree correspond to paths in the original graph, and copies of the same node $v$ will appear in several places of the tree whenever $v$ can be reached through several paths. DFS tree search may search forever if there are cycles in the graph. We always assume that path lengths are bounded by a constant $D$. Figure 1 illustrates the BFS and DFS search strategies, and how they (initially) focus on different parts of the search space. Please refer to (Russell and Norvig, 2010) for details.

The *runtime* or *search time* of a search method (BFS or DFS) is the number of nodes explored until a first goal is found (5 and 6 respectively in Figure 1). This simplifying assumption relies on node expansion being the dominant operation, consuming similar time throughout the tree. If no goal exists, the search method

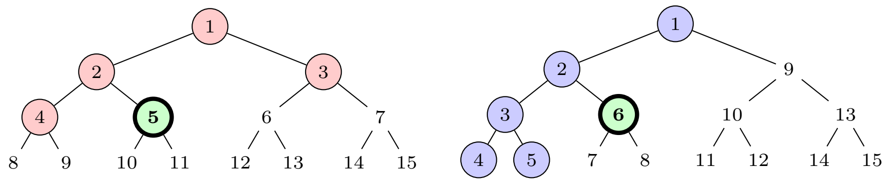

Fig. 1: The difference between BFS (left) and DFS (right) in a complete binary tree where a goal is placed in the second position on level 2 (the third row). The numbers indicate traversal order. Circled nodes are explored before the goal is found. Note how BFS and DFS explore different parts of the tree. In bigger trees, this may lead to substantial differences in search performance.

will explore all nodes before halting. In this case, we define the runtime as the number of nodes in the search problem plus 1 (i.e., $2^{D+1}$ in the case of a binary tree of depth $D$). Let $\Gamma$ be the event that a goal exists, $\Gamma_k$ the event that a goal exists on level $k$, and $\bar{\Gamma}$ and $\bar{\Gamma}_k$ their complements. Let $F_k = \Gamma_k \cap (\bigcap_{i=0}^{k-1} \bar{\Gamma}_i)$ be the event that level $k$ has the *first* goal.

A random variable $X$ is *geometrically distributed* $\text{Geo}(p)$ if $P(X = k) = (1-p)^{k-1}p$ for $k \in \{1, 2, \dots\}$. The interpretation of $X$ is the number of trials until the first success when each trial succeeds with probability $p$. Its cumulative distribution function (CDF) is $P(X \leq k) = 1 - (1-p)^k$, and its *average* or *expected value* $\mathbb{E}[X] = 1/p$. A random variable $Y$ is *truncated geometrically distributed* $X \sim \text{TruncGeo}(p, m)$ if $Y = (X \mid X \leq m)$ for $X \sim \text{Geo}(p)$, which gives

$$P(Y = k) = \begin{cases} \frac{(1-p)^k p}{1-(1-p)^m} & \text{for } k \in \{1, \dots, m\} \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{tc}(p, m) := \mathbb{E}[Y] = \mathbb{E}[X \mid X \leq m] = \frac{1-(1-p)^m(pm+1)}{p(1-(1-p)^m)}.$$

When $p \gg \frac{1}{m}$, $Y$ is approximately $\text{Geo}(p)$, and $\text{tc}(p, m) \approx \frac{1}{p}$. When $p \ll \frac{1}{m}$, $Y$ becomes approximately uniform on $\{1, \dots, m\}$ and $\text{tc}(p, m) \approx \frac{m}{2}$.

A random variable $Z$ is *exponentially distributed* $\text{Exp}(\lambda)$ if $P(Z \leq z) = 1 - e^{-\lambda z}$ for $z \geq 0$. The expected value of $Z$ is $\frac{1}{\lambda}$, and the probability density function of $Z$ is $\lambda e^{-\lambda z}$. An exponential distribution with parameter $\lambda = -\ln(1-p)$ might be viewed as the continuous counterpart of a $\text{Geo}(p)$ distribution. We will use this approximation in Section 4.

**Lemma 1 (Exponential approximation).** *Let $Z \sim \text{Exp}(-\ln(1-p))$ and $X \sim \text{Geo}(p)$. Then the CDFs for $X$ and $Z$ agree for integers $k$, $P(Z \leq k) = P(X \leq k)$. The expectations of $Z$ and $X$ are also similar in the sense that $0 \leq \mathbb{E}[X] - \mathbb{E}[Z] \leq 1$.*

We will occasionally make use of the convention $0 \cdot \textit{undefined} = 0$, and often expand expectations by conditioning on disjoint events:

**Lemma 2.** *Let $X$ be a random variable and let the sample space $\Omega = \dot{\bigcup}_{i \in I} C_i$ be partitioned by mutually disjoint events $C_i$. Then $\mathbb{E}[X] = \sum_{i \in I} P(C_i) E[X \mid C_i]$.*

## 3 Complete Binary Tree with a Single Goal Level

Consider a binary tree of depth $D$, where solutions are distributed on a single *goal level* $g \in \{0, \dots, D\}$. At the goal level, any node is a goal with iid probability $p_g \in [0, 1]$. We will refer to this kind of problems as *(single goal level) complete binary trees with depth $D$, goal level $g$ and goal probability $p_g$* (Section 4 generalises the setup to multiple goal levels).

As a concrete example, consider the search problem of solving a Rubik's cube. There is an upper bound $D = 20$ to how many moves it can take to reach the goal, and we may suspect that most goals are located around level 17 ($\pm 2$ levels) (Rokicki and Kociemba, 2013). If we consider search algorithms that do not remember where they have been, the search space becomes a complete tree with fixed branching factor $3^6$. What would be the expected BFS and DFS search time for this problem? Which algorithm would be faster?

The probability that a goal exists is $P(\varGamma) = P(\varGamma_g) = 1 - (1 - p_g)^{2^g}$. If a goal exists, let $Y$ be the position of the first goal at level $g$. Conditioned on a goal existing, $Y$ is a truncated geometric variable $Y \sim \mathrm{TruncGeo}(p_g, 2^g)$. When $p_g \gg 2^{-g}$ the goal position $Y$ is approximately $\mathrm{Geo}(p_g)$, which makes most expressions slightly more elegant. This is often a realistic assumption, since if $p \not\gg 2^{-g}$, then often no goal would exist.

**Proposition 1 (BFS runtime Single Goal Level).** *Let the problem be a complete binary tree with depth $D$, goal level $g$ and goal probability $p_g$. When a goal exists and has position $Y$ on the goal level, the BFS search time is $t_{\mathrm{SGL}}^{\mathrm{BFS}}(g, p_g, Y) = 2^g - 1 + Y$, with expectation, $t_{\mathrm{SGL}}^{\mathrm{BFS}}(g, p_g \mid \varGamma_g) = 2^g - 1 + \mathrm{tc}(p_g, 2^g) \approx 2^g - 1 + \frac{1}{p_g}$. In general, when a goal does not necessarily exist, the expected BFS search time is $t_{\mathrm{SGL}}^{\mathrm{BFS}}(g, p_g) = P(\varGamma) \cdot (2^g - 1 + \mathrm{tc}(p_g, 2^g)) + P(\bar{\varGamma}) \cdot 2^{D+1} \approx 2^g - 1 + \frac{1}{p_g}$. The approximations are close when $p_g \gg 2^{-g}$.*

**Proposition 2.** *Consider a complete binary tree with depth $D$, goal level $g$ and goal probability $p_g$. When a goal exists and has position $Y$ on the goal level, the DFS search time is approximately $\tilde{t}_{\mathrm{SGL}}^{\mathrm{DFS}}(D, g, p_g, Y) := (Y - 1)2^{D-g+1} + 2$, with expectation $\tilde{t}_{\mathrm{SGL}}^{\mathrm{DFS}}(D, g, p_g \mid \varGamma_g) := (1/p_g - 1)\, 2^{D-g+1} + 2$. When $p_g \gg 2^{-g}$, the expected DFS search time when a goal does not necessarily exist is approximately*

$$\tilde{t}_{\mathrm{SGL}}^{\mathrm{DFS}}(D, g, p_g) := P(\varGamma)((\mathrm{tc}(p_g, 2^g) - 1)2^{D-g+1} + 2) + P(\bar{\varGamma})2^{D+1} \approx \left(\frac{1}{p_g} - 1\right) 2^{D-g+1}.$$

The proofs only use basic counting arguments and probability theory. A less precise version of Proposition 1 can be obtained from (Korf et al., 2001, Thm. 1). Full proofs and further details are provided in (Everitt and Hutter, 2015a). Figure 2 shows the runtime estimates as a function of goal level. The runtime estimates can be used to predict whether BFS or DFS will be faster, given the parameters $D$, $g$, and $p_g$, as stated in the next Proposition.

**Proposition 3.** *Let* $\gamma_{p_g} = \log_2 \left(\mathrm{tc}(p_g, 2^g) - 1\right)/2 \approx \log_2\left(\frac{1-p_g}{p_g}\right)/2$*. Given the approximation of DFS runtime of Proposition 2, BFS wins in expectation in a complete binary tree with depth* $D$*, goal level* $g$ *and goal probability* $p_g$ *when* $g < \frac{D}{2} + \gamma_{p_g}$ *and DFS wins in expectation when* $g > \frac{D}{2} + \gamma_{p_g} + \frac{1}{2}$*.*

The term $\gamma_{p_g}$ is in the range $[-1, 1]$ when $p_g \in [0.2, 0.75]$, $g \geq 2$, in which case Proposition 3 roughly says that BFS wins (in expectation) when the goal level $g$ comes before the middle of the tree. BFS benefits from a smaller $p_g$, with the boundary level being shifted $\gamma_{p_g} \approx k/2$ levels from the middle when $p_g \approx 2^{-k} \gg 2^{-g}$. Figure 2 illustrates the prediction as a function of goal depth and tree depth for a fixed probability $p_g = 0.07$. The technical report (Everitt and Hutter, 2015a) gives the full proof, which follows from the runtime estimates Proposition 1 and 2.

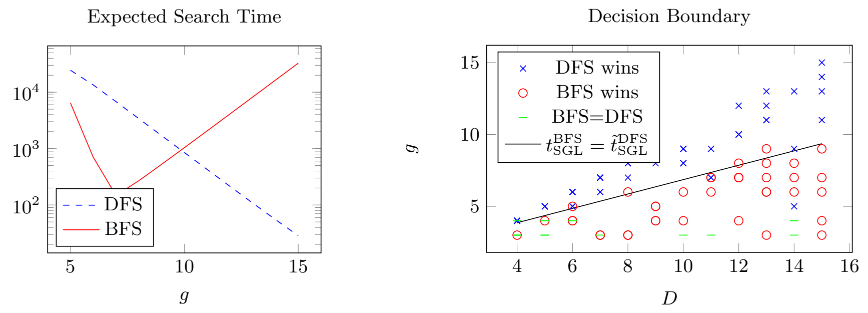

Fig. 2: Two plots of how expected BFS and DFS search time varies in a complete binary tree with a single goal level $g$ and goal probability $p_g = 0.07$. The left depicts search time as a function of goal level in a tree of depth 15. BFS has the advantage when the goal is in the higher regions of the graph, although at first the probability that no goal exists heavily influences both BFS and DFS search time. DFS search time improves as the goal moves downwards since the goal probability is held constant. The right graph shows the decision boundary of Proposition 3, together with 100 empirical outcomes of BFS and DFS search time according to the varied parameters $g \in [3, D] \cap \mathbb{N}$ and $D \in [4, 15] \cap \mathbb{N}$. The decision boundary gets 79% of the winners correct.

It is straightforward to generalise the calculations to arbitrary branching factor $b$ by substituting the 2 in the base of $t^{\text{BFS}}_{\text{SGL}}$ and $\tilde{t}^{\text{DFS}}_{\text{SGL}}$ for $b$. In Proposition 3, the change only affects the base of the logarithm in $\gamma_{p_g}$. See (Everitt and Hutter, 2015a) for further details.

## 4 Complete Binary Tree with Multiple Goal Levels

We now generalise the model developed in the previous section to problems that can have goals on any number of levels. For each level $k \in \{0, \ldots, D\}$, let $p_k$ be the associated *goal probability.* Not every $p_k$ should be equal to 0. Nodes on level $k$ have iid probability $p_k$ of being a goal. We will refer to this kind of problems as *(multi goal level) complete binary trees with depth $D$ and goal probabilities* $\mathbf{p}$.

*DFS Analysis* To find an approximation of goal DFS performance in trees with multiple goal levels, we approximate the geometric distribution used in Proposition 2 with an exponential distribution (its continuous approximation by Lemma 1).

**Proposition 4 (Expected Multi Goal Level DFS Performance).** *Consider a complete binary tree of depth $D$ with goal probabilities $\mathbf{p} = [p_0, \dots, p_D] \in [0,1)^{D+1}$. If for at least one $j$, $p_j \gg 2^{-j}$, and for all $k$, $p_k \ll 1$, then the expected number of nodes DFS will search is approximately $\tilde{t}^{\mathrm{DFS}}_{\mathrm{MGL}}(D, \mathbf{p}) := 1/\sum_{k=0}^{D} \ln(1-p_k)^{-1} 2^{-(D-k+1)}$*

The proof (available in Everitt and Hutter 2015a) constructs for each level $k$ an exponential random variable $X_k$ that approximates the search time before a goal is found on level $k$ (disregarding goals on other levels). The minimum of all $X_k$ then becomes an approximation of the search time to find a goal on any level. The approximations use exponential variables for easy minimisation.

In the special case of a single goal level, the approximation of Proposition 4 is similar to the one given by Proposition 2. When $\mathbf{p}$ only has a single element $p_j \neq 0$, the expression $\tilde{t}^{\mathrm{DFS}}_{\mathrm{MGL}}$ simplifies to $\tilde{t}^{\mathrm{DFS}}_{\mathrm{MGL}}(D, \mathbf{p}) = -2^{D-j+1}/\ln(1-p_j)$. For $p_j$ not close to 1, the factor $-1/\ln(1-p_j)$ is approximately the same as the corresponding factor $1/p_j - 1$ in Proposition 2 (the *Laurent expansion* is $-1/\ln(1-p_j) = 1/p_j - 1/2 + O(p_j)$).

*BFS Analysis* The corresponding expected search time $t^{\mathrm{BFS}}_{\mathrm{MGL}}(D, \mathbf{p})$ for BFS requires less insight and can be calculated exactly by conditioning on which level the first goal is. The resulting formula is less elegant, however. The same technique cannot be used for DFS, since DFS does not exhaust levels one by one.

The probability that level $k$ has the first goal is $P(F_k) = P(\Gamma_k)\prod_{j=0}^{k-1} P(\bar{\Gamma}_j)$, where $P(\Gamma_i) = (1 - (1-p_i)^{2^i})$. The expected BFS search time gets a more uniform expression by the introduction of an extra *hypothetical level* $D+1$ where all nodes are goals. That is, level $D+1$ has goal probability $p_{D+1} = 1$ and $P(F_{D+1}) = P(\bar{\Gamma}) = 1 - \sum_{k=0}^{D} P(F_k)$.

**Proposition 5 (Expected Multi Goal Level BFS Performance).** *The expected number of nodes $t^{\mathrm{BFS}}_{\mathrm{MGL}}(p)$ that BFS needs to search to find a goal in a complete binary tree of depth $D$ with goal probabilities $\mathbf{p} = [p_0, \dots, p_D]$, $\mathbf{p} \neq \mathbf{0}$, is $t^{\mathrm{BFS}}_{\mathrm{MGL}}(\mathbf{p}) = \sum_{k=0}^{D+1} P(F_k) t^{\mathrm{BFS}}_{\mathrm{SGL}}(k, p_k \mid \Gamma_k) \approx \sum_{k=0}^{D+1} P(F_k)\left(2^k + \frac{1}{p_k}\right)$*

See (Everitt and Hutter, 2015a) for a proof. For $p_k = 0$, the expression $t^{\mathrm{BFS}}_{\mathrm{CB}}(k, p_k)$ and $1/p_k$ will be undefined, but this only occurs when $P(F_k)$ is also 0. The approximation tends to be within a factor 2 of the correct expression, even when $p_k < 2^{-k}$ for some or all $p_k \in \mathbf{p}$. The reason is that the corresponding

$P(F_k)$'s are small when the geometric approximation is inaccurate. Both Proposition 4 and 5 naturally generalise to arbitrary branching factor $b$. Although their combination does not yield a similarly elegant expression as Proposition 3, they can still be naively combined to predict the BFS vs. DFS winner (Figure 3).

## 5 Experimental verification

To verify the analytical results, we have implemented the models in Python 3 using the `graph-tool` package (Peixoto, 2015). The data reported in Table 1 and 2 is based on an average over 1000 independently generated search problems with depth $D = 14$. The first number in each box is the empirical average, the second number is the analytical estimate, and the third number is the percentage error of the analytical estimate.

For certain parameter settings, there is only a small chance ($< 10^{-3}$) that there are no goals. In such circumstances, all 1000 generated search graphs typically inhabit a goal, and so the empirical search times will be comparatively small. However, since a tree of depth 14 has about $2^{15} \approx 3 \cdot 10^5$ nodes (and a search algorithm must search through all of them in case there is no goal), the rarely occurring event of no goal can still influence the *expected* search time substantially. To avoid this sampling problem, we have ubiquitously discarded all instances where no goal is present, and compared the resulting averages to the analytical expectations *conditioned on at least one goal being present.*

To develop a concrete instance of the multi goal level model we consider the special case of *Gaussian goal probability vectors*, with two parameters $\mu$ and $\sigma^2$. For a given depth $D$, the goal probabilities are given by $p_i = \min\left\{\frac{1}{20\sqrt{\sigma^2}} e^{(i-\mu)^2/\sigma^2},\ \frac{1}{2}\right\}$. The parameter $\mu \in [0, D] \cap \mathbb{N}$ is the *goal peak*, and the parameter $\sigma^2 \in \mathbb{R}^+$ is the *goal spread.* The factor 1/20 is arbitrary, and chosen to give an interesting dynamics between searching depth-first and breadth-first. No $p_i$ should be greater than 1/2, in order to (roughly) satisfy the assumption of Proposition 5. We call this model the *Gaussian binary tree.*

The accuracy of the predictions of Proposition 1 and 2 are shown in Table 1, and the accuracy of Proposition 4 and 5 in Table 2. The relative error is always small for BFS ($< 10\%$). For DFS the error is generally within 20%, except when the search time is small ($< 35$ probes), in which case the absolute error is always small. The decision boundary of Proposition 3 is shown in Figure 2, and the decision boundary of Proposition 4 vs. 5 is shown in Figure 3. These boundary plots show that the analysis generally predict the correct BFS vs. DFS winner (79% and 74% correct in the investigated models).

## 6 Conclusions and Outlook

Part II of this paper (Everitt and Hutter, 2015b) generalises the setup in this paper, analytically investigating search performance in general graphs. Part II also provides a more general discussion and outlook on future directions.

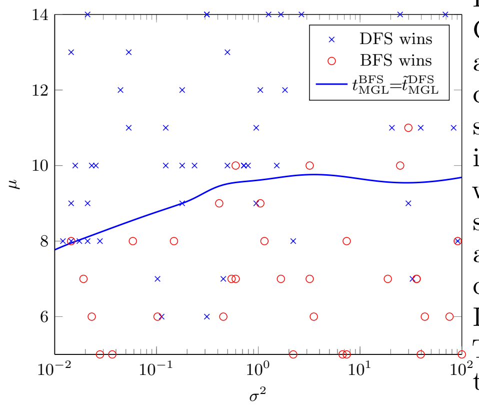

Fig. 3: The decision boundary for the Gaussian tree given by Proposition 4 and 5, together with empirical outcomes of BFS vs. DFS winner. The scattered points are based on 100 independently generated problems with depth $D = 14$ and uniformly sampled parameters $\mu \in [5, 14] \cap \mathbb{N}$ and $\log(\sigma^2) \in [-2, 2]$. The most deciding feature is the goal peak $\mu$, but DFS also benefits from a smaller $\sigma^2$. The decision boundary gets 74% of the winners correct.

| $g \backslash p_g$ | 0.001 | 0.01 | 0.1 |
|---|---|---|---|
| 5 | | 46.33 | 40.01 |
| | | *46.64* | *39.86* |
| | | 0.7 % | 0.4 % |
| 8 | 369.5 | 332.8 | 264.6 |
| | *378.0* | *333.9* | *265.0* |
| | 2.3 % | 0.3 % | 0.2 % |
| 11 | 2748 | 2143 | 2057 |
| | *2744* | *2147* | *2057* |
| | 0.1 % | 0.2 % | 0. % |
| 14 | 17 360 | 16 480 | 16 390 |
| | *17 380* | *16 480* | *16 390* |
| | 0.1 % | 0. % | 0. % |

(a) BFS single goal level

| $g \backslash p_g$ | 0.001 | 0.01 | 0.1 |
|---|---|---|---|
| 5 | | 14 680 | 8206 |
| | | *15 000* | *8053* |
| | | 2.2 % | 1.9 % |
| 8 | 14 530 | 9833 | 1105 |
| | *15 620* | *9967* | *1154* |
| | 7.5 % | 1.4 % | 4.5 % |
| 11 | 11 200 | 1535 | 152.3 |
| | *11 140* | *1586* | *146.0* |
| | 0.5 % | 3.4 % | 4.1 % |
| 14 | 1971 | 208.8 | 30.57 |
| | *2000* | *200.0* | *20.00* |
| | 1.4 % | 4.2 % | 35 % |

(b) DFS single goal level

Table 1: BFS and DFS performance in the single goal level model with depth $D = 14$, where $g$ is the goal level and $p_g$ the goal probability. Each box contains empirical average/*analytical expectation*/error percentage.

| $\mu \backslash \sigma$ | 0.1 | 1 | 10 | 100 |
|---|---|---|---|---|
| 5 | 37.24 | 43.75 | 90.87 | 225.1 |
| | *37.04* | *41.55* | *83.72* | *210.8* |
| | 0.5 % | 5.0 % | 7.9 % | 6.4 % |
| 8 | 261.2 | 171.9 | 119.6 | 212.0 |
| | *261.3* | *173.4* | *119.8* | *211.0* |
| | 0. % | 0.9 % | 0.2 % | 0.5 % |
| 11 | 2049 | 953.0 | 303.9 | 249.5 |
| | *2050* | *953.0* | *305.0* | *247.5* |
| | 0. % | 0. % | 0.3 % | 0.8 % |
| 14 | 16 210 | 5159 | 968.5 | 332.9 |
| | *16 150* | *5136* | *960.6* | *329.7* |
| | 0.4 % | 0.4 % | 0.8 % | 0.9 % |

(a) BFS multi goal level

| $\mu \backslash \sigma$ | 0.1 | 1 | 10 | 100 |
|---|---|---|---|---|
| 5 | 5374 | 8572 | 3405 | 385.8 |
| | *5949* | *10 070* | *3477* | *379.1* |
| | 11 % | 18 % | 2.1 % | 1.7 % |
| 8 | 677.3 | 1234 | 454.6 | 252.6 |
| | *743.6* | *1259* | *473.6* | *260.0* |
| | 9.8 % | 2.1 % | 4.2 % | 2.9 % |
| 11 | 97.38 | 168.1 | 117.4 | 210.0 |
| | *92.95* | *157.4* | *106.7* | *211.7* |
| | 4.5 % | 6.4 % | 9.1 % | 0.8 % |
| 14 | 24.00 | 43.38 | 81.75 | 213.6 |
| | *11.62* | *32.89* | *74.46* | *205.0* |
| | 52 % | 24 % | 8.9 % | 4.0 % |

(b) DFS multi goal level

Table 2: BFS and DFS performance in Gaussian binary trees with depth $D = 14$. Each box contains empirical average/*analytical expectation*/error percentage.

# Bibliography

Chen, P. C. (1992). Heuristic Sampling: A Method for Predicting the Performance of Tree Searching Programs. *SIAM Journal on Computing*, 21(2):295–315.

Edelkamp, S. and Schrödl, S. (2012). *Heuristic Search.* Morgan Kaufmann Publishers Inc.

Everitt, T. and Hutter, M. (2015a). A Topological Approach to Meta-heuristics: Analytical Results on the BFS vs. DFS Algorithm Selection Problem. Technical report, Australian National University, `arXiv:1509.02709[cs.AI]`.

Everitt, T. and Hutter, M. (2015b). Analytical Results on the BFS vs. DFS Algorithm Selection Problem. Part II: Graph Search. In *28th Australian Joint Conference on Artificial Intelligence.*

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206(1):79–111.

Kilby, P., Slaney, J., Thiébaux, S., and Walsh, T. (2006). Estimating Search Tree Size. In *Proc. of the 21st National Conf. of Artificial Intelligence, AAAI, Menlo Park.*

Knuth, D. E. (1975). Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29(129):122–122.

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

Kotthoff, L. (2014). Algorithm Selection for Combinatorial Search Problems: A Survey. *AI Magazine*, pages 1–17.

Lelis, L. H. S., Otten, L., and Dechter, R. (2013). Predicting the size of Depth-first Branch and Bound search trees. *IJCAI International Joint Conference on Artificial Intelligence*, pages 594–600.

Peixoto, T. P. (2015). The graph-tool python library. *figshare.*

Purdom, P. W. (1978). Tree Size by Partial Backtracking. *SIAM Journal on Computing*, 7(4):481–491.

Rice, J. R. (1975). The algorithm selection problem. *Advances in Computers*, 15:65–117.

Rokicki, T. and Kociemba, H. (2013). The diameter of the rubiks cube group is twenty. *SIAM Journal on Discrete Mathematics*, 27(2):1082–1105.

Russell, S. J. and Norvig, P. (2010). *Artificial intelligence: a modern approach.* Prentice Hall, third edition.

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