Analytical Results on the BFS vs. DFS Algorithm Selection Problem. Part II: Graph Search[*]

Tom Everitt and Marcus Hutter

Australian National University, Canberra, Australia

October 15, 2015

Abstract

The algorithm selection problem asks to select the best algorithm for a given problem. In the companion paper Everitt and Hutter (2015b), expected runtime was approximated as a function of search depth and probabilistic goal distribution for tree search versions of breadth-first search (BFS) and depth-first search (DFS). Here we provide an analogous analysis of BFS and DFS graph search, deriving expected runtime as a function of graph structure and goal distribution. The applicability of the method is demonstrated through analysis of two different grammar problems. The approximations come surprisingly close to empirical reality.

1 Introduction

Search is a fundamental problem of artificial intelligence (Russell and Norvig, 2010), and a sizeable list of search algorithms with different pros and cons can be found in the literature (Edelkamp and Schrödl, 2012). Examples of search tasks include combinatorial optimisation problems and planning, and core search algorithms include BFS, DFS, A*, simulated annealing, and genetic algorithms. Techniques for selecting the best algorithm for a given problem is of obvious importance (Rice, 1975; Kotthoff, 2014; Hutter et al., 2014).

Tightly related to the algorithm selection problem is the problem of predicting algorithm runtime; in particular expected runtime. In the companion paper (Everitt and Hutter, 2015b) we gave a brief survey of related work and derived (approximations of) expected runtime for breadth-first search (BFS) and depth-first search (DFS) in trees. The results also applied to search in general graphs for the variants of BFS and DFS that do *not* remember visited nodes; so called

[*]Final publication is available on `http://link.springer.com`

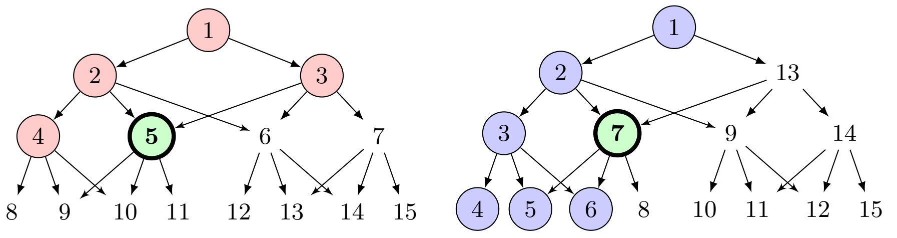

Figure 1: The difference between BFS (left) and DFS (right) in a directed graph where a goal is placed in the second position on level 2 (the third row). The numbers indicate traversal order. Circled nodes are explored before the goal is found. Note how BFS and DFS explore different parts of the tree.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

tree search algorithms. Their name does not stop them from being used also in other types of graphs, but they then run the risk of spending most of the time searching the same nodes many times. This paper analyses expected runtime of *graph search* variants of BFS and DFS that do remember which nodes they have visited. Although graph search variants usually are more efficient in the sense that they search fewer nodes, the extra memory overhead means that they are not always applicable.

Our main contributions are estimates of expected BFS and DFS graph search runtime as a function of graph structure and distributions of goals (Section 3). Note that we focus solely on the time it takes to find a goal, and ignore aspects such as solution quality. We demonstrate the relevance of the results by applying them to two different grammar problems (Section 4). Setup and background are described in Section 2, and experimental verification in Section 5. Finally, conclusions and outlooks come in Section 6. The technical report (Everitt and Hutter, 2015a) offers a greatly extended discussion about the setup.

2 Preliminaries

The graph search variants of BFS and 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. Figure 1 illustrates the different 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 7 respectively in Figure 1). This simplifying assumption relies on node expansion being the dominant operation, requiring similar time throughout the tree. If no goal exists, the search method will explore all nodes before halting. In this case, we define the runtime as the number of nodes in the search problem plus 1. Let Γ be the event that a goal exists, Γ_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* $\mathrm{Geo}(p)$ 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* $X \sim \mathrm{TruncGeo}(p, m)$ if $Y = (X \mid X \leq m)$ for $X \sim \mathrm{Geo}(p)$, which gives

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

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

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

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

Lemma 1. *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 Colliding Branches

The companion paper (Everitt and Hutter, 2015b) explores a model of *tree search*, where path redundancies are not recognized by the search algorithms. In this section we develop a similar model for *graph search* performance. The abstract results of this section are applied to two grammar problems in the next section.

Definition 2. For a given search problem: Let the *level* of a node v, $\mathrm{level}(v)$, be the length of a shortest path from the start node to v. Let $D = \max_v \mathrm{level}(v)$ be the *(generalised) depth* of the search graph. Let δ_n be the first node on level n reached by DFS, $0 \leq n \leq D$. Any node reachable from v is a *descendant of* v.

The *descendant counter* L plays a central role in the analysis. For a given search problem, let

$$L(n, d) = |\{v : \mathrm{level}(v) = d, v \in \mathrm{descendants}(\delta_n)\}|$$

count the number of nodes on level d that are reachable from δ_n.

As in the companion paper, we assume that goals are distributed by level in an iid manner according to a goal probability vector $\mathbf{p}$. We will also assume that the probability of DFS finding a goal before finding δ_D is negligible. We will refer to this kind of problems as *search problems with depth D, goal probabilities* $\mathbf{p}$ *and descendant counter L.* The rest of this section justifies the following proposition.

Proposition 3. *The DFS and BFS runtime of a search problem can be roughly estimated from the descendant counter L, the depth D and the goal probabilities $\mathbf{p} = [p_0, \ldots, p_D]$ when the probability of DFS finding a goal before δ_D is negligible.*

The assumption of DFS not finding a goal before δ_D is not always realistic, but is for example satisfied in the grammar problems considered in Section 4 below.

3.1 DFS Analysis

The nodes $\delta_0, \ldots, \delta_D$ play a central role in the analysis of DFS runtime, since all the descendants of δ_{n+1} will be explored before the descendants of δ_n (excluding the δ_{n+1} descendants). We say that *DFS explores from δ_n* after DFS has explored all descendants of δ_{n+1} and until all descendants of δ_n have been explored. The general idea of the DFS analysis will be to count the number of nodes under each δ_n, and to compute the probability that any of these nodes is a goal.

Some notation for this:

- Let the *δ_n-subgraph* $S_n = \{v : v \in \text{descendants}(\delta_n)\}$ be the set of nodes reachable from δ_n, with cardinality $|S_n| = \sum_{i=0}^{D} L(n,i)$, $0 \leq n \leq D$. Let $S_{D+1} = \emptyset$ and let S_{-1} be a set of cardinality $|S_{-1}| = |S_0| + 1 = \sum_{i=0}^{D} L(0,i) + 1$.
- Let the *δ_n-explorables* $T_n = S_n \setminus S_{n+1}$ be the nodes explored from δ_n.
- Let the *number of level-d δ_n-explorables* $A_{n,d} = L(n,d) - L(n+1,d)$ be the number of level d descendants of δ_n that are not descendants of δ_{n+1} for $0 \leq n, d \leq D$. The relation between T_n and $A_{n,d}$ is the following: $|T_n| = \sum_{i=n}^{D} A_{n,i}$.

Let $q_k = 1 - p_k$ for $0 \leq k \leq D$.

Lemma 4. *Consider a search problem with depth D, goal probabilities* $\mathbf{p}$, *and descendant counter L. The probability that the δ_n-explorables T_n contains a goal is $\tau_n := 1 - \prod_{k=0}^{D} q_k^{A_{n,k}}$, and the probability that T_n contains the first goal is $\phi_n := \tau_n \prod_{i=n+1}^{D}(1 - \tau_i)$.*

Proof. τ_n is 1 minus the probability of *not* hitting a goal at any level d, $n \leq d \leq D$, since at each level d, $A_{n,d}$ probes are made when exploring from δ_n. □

Proposition 5 (Colliding branches expected DFS search time)**.** *The expected DFS search time $t_{\text{CB}}^{\text{DFS}}(D, \mathbf{p}, L)$ in a search problem with depth D, goal probabilities* $\mathbf{p}$, *and descendant counter L is bounded by*

$$t_{\text{CBL}}^{\text{DFS}}(D, \mathbf{p}, L) := \sum_{n=-1}^{D} |S_{n+1}|\phi_n \leq t_{\text{CB}}^{\text{DFS}}(D, \mathbf{p}, L) \leq \sum_{n=-1}^{D} |S_n|\phi_n := t_{\text{CBU}}^{\text{DFS}}(D, \mathbf{p}, l)$$

where $\phi_{-1} = \bar{\Gamma} = 1 - \sum_{n=0}^{D} \phi_n$ is the probability that no goal exists.

The arithmetic mean $\tilde{t}^{\text{DFS}}_{\text{CB}}(D, \mathbf{p}, L) := (t^{\text{DFS}}_{\text{CBL}}(D, \mathbf{p}, L) + t^{\text{DFS}}_{\text{CBU}}(D, \mathbf{p}, L))/2$ of the bounds can be used for a single runtime estimate.

Proof. Let X be the DFS search time in a search problem with the features described above. The expectation of X may be decomposed as

$$\mathbb{E}[X] = P(\bar{\Gamma})\mathbb{E}[X \mid \bar{\Gamma}] + \sum_{n=0}^{D} P(\text{first goal in } T_n) \cdot \mathbb{E}[X \mid \text{first goal in } T_n]. \quad (1)$$

The conditional search time $(X \mid \text{first goal in } T_n)$ is bounded by $|S_{n+1}| \leq (X \mid \text{first goal in } T_n) \leq |S_n|$ for $0 \leq n \leq D$, since to find a goal DFS will search the entire δ_{n+1}-subgraph S_{n+1} before finding it when searching the δ_n-explorables T_n, but will not need to search more than the δ_n-subgraph $S_n = S_{n+1} \cup T_n$ (disregarding the few probes made 'on the way down to' δ_n (i.e. to T_n); these probes were assumed negligible). The same bounds also hold with S_0 and S_{-1} when no goal exists (recall that $|S_{-1}| := |S_0| + 1$). Therefore the conditional expectation satisfies

$$|S_{n+1}| \leq \mathbb{E}[X \mid \text{first goal in } T_n] \leq |S_n| \quad (2)$$

for $-1 \leq n \leq D$. By Lemma 4, the probability that the first goal is among the δ_n-explorables T_n is ϕ_n, and the probability $P(\bar{\Gamma})$ that no goal exists is ϕ_{-1} by definition.

Substituting ϕ_n and (2) into (1) gives the desired bounds for expected DFS search time $\tilde{t}^{\text{DFS}}_{\text{CB}}(D, \mathbf{p}, L) = \mathbb{E}[X]$. □

The informativeness of the bounds of Proposition 5 depends on the dispersion of nodes between the different T_n's. If most nodes belong to one or a few sets T_n, the bounds may be almost completely uninformative. This happens in the special case of complete trees with branching factor b, where a fraction $(b-1)/b$ of the nodes will be in T_0. The companion paper (Everitt and Hutter, 2015b) derives techniques for these cases. The grammar problems investigated in Section 4 below show that the bounds may be relevant in more connected graphs, however.

3.2 BFS Analysis

The analysis of BFS only requires the descendant counter $L(0, \cdot)$ with the first argument set to 0, and follows the same structure as the BFS analysis in (Everitt and Hutter, 2015b). In contrast to the DFS bounds above, this analysis gives a precise expression for the expected runtime. The idea is to count the number of nodes in the upper k levels of the tree (derived from $L(0,0), \dots, L(0,k)$), and to compute the probability that they contain a goal. Let the *upper subgraph* $U_k = \sum_{i=0}^{k-1} L(0,i)$ be the number of nodes above level k. When there is only a single goal level, the following expression for BFS runtime may be readily derived.

Lemma 6 (BFS runtime Single Goal Level)**.** *For a search problem with depth D and descendant counter L, assume that the problem has a single goal level g with goal probability p_g, and that $p_j = 0$ for $j \neq g$. When a goal exists and has position Y on the goal level, the BFS search time is:*

$$\begin{aligned} t_{\mathrm{CB}}^{\mathrm{BFS}}(g, p_g, L, Y) &= U_g + Y, \text{ with expected value} \\ t_{\mathrm{CB}}^{\mathrm{BFS}}(g, p_g, L \mid \Gamma_g) &= U_g + \mathrm{tc}(p_g, L(0,g)) \end{aligned}$$

Proof. When a goal exists, BFS will explore all of the top of the tree until depth $g-1$ (that is, U_g nodes) and Y nodes on level g before finding the first goal. The expected value of Y is $\mathrm{tc}(p_g, L(0,g))$. □

The probability that level k has a goal is $P(\Gamma_k) = 1 - q_k^{L(0,k)}$, and the probability that level k has the first goal is $P(F_k) = P(\Gamma_k)\prod_{i=0}^{k-1} P(\bar{\Gamma}_i)$. To BFS, only the first goal level matter. This allows BFS runtime to be expanded over the F_k events as in Lemma 1. For greater uniformity, a *hypothetical level* $D+1$ only containing goals is introduced to handle the event of no goal in the first D levels.

Proposition 7 (Branch Colliding Expected BFS Performance)**.** *The expected number of nodes that BFS needs to search to find a goal in a search problem with depth D, goal probabilities $\mathbf{p} = [p_0, \dots, p_D]$, $\mathbf{p} \neq \mathbf{0}$, and descendant counter L is*

$$t_{\mathrm{CB}}^{\mathrm{BFS}}(\mathbf{p}, L) = \sum_{k=0}^{D+1} P(F_k) t_{\mathrm{CB}}^{\mathrm{BFS}}(k, p_k, L \mid \Gamma_k)$$

where the goal probabilities have been extended with an extra element $p_{D+1} = 1$, and $F_{D+1} = \bar{\Gamma}$ is the event that no goal exists.

For $p_k = 0$, $t_{\mathrm{CB}}^{\mathrm{BFS}}$ will be undefined, but this only occurs when $P(F_k)$ is also 0. Proposition 5 and 7 give (rough) estimates of average BFS and DFS graph search time given the goal distribution $\mathbf{p}$ and the structure parameter L. The results can be combined to make a decision whether to use BFS or DFS (Figure 3).

4 Grammar Problems

We now show how to apply the general theory of Section 3 to two concrete grammar problems. A *grammar problem* is a search problem where nodes are strings over some finite alphabet B, and the neighbourhood relation is given by a set of production rules. *Production rules* are mappings $x \to y$, $x, y \in B^*$, defining how strings may be transformed. For example, the production rule $S \to Sa$ permits the string aSa to be transformed into $aSaa$. A grammar problem is defined by a set of production rules, together with a *starting string* and a *set of goal strings*. A *solution* is a sequence of production rule applications that transforms the starting string into a goal string. Many search problems can be formulated as grammar problems, with string representations of states

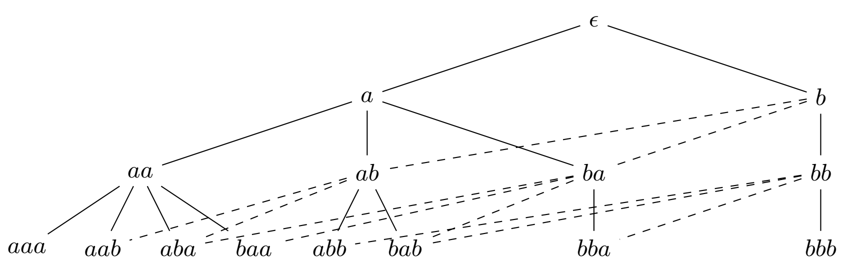

Figure 2: Graph of binary grammar problem with max depth $D = 3$. Contiguous lines indicate first discovery by DFS, and dashed lines indicate rediscoveries.

modified by production rules. Their generality makes it *computably undecidable* whether a given grammar problem has a solution or not. We here consider a simplified version where the search depth is artificially limited, and goals are distributed according to a goal probability vector p.

4.1 Binary Grammar

Let ϵ be the empty string. The *binary grammar* consists of two production rules, $\epsilon \to a$ and $\epsilon \to b$ over the alphabet $B = \{a, b\}$. The starting string is the empty string ϵ. A maximum depth D of the search graph is imposed, and strings on level k are goals with iid probability p_k, $0 \le k \le D$. Since the left hand substring of both production rules is the empty string, both can always be applied at any place to a given string. The resulting graph is shown in Figure 2.

The first node on level n that DFS reaches in the binary grammar problem is $\delta_n = a^n$ for $0 \le n \le D$, assuming that the production rule $\epsilon \to a$ is always used first by DFS. The following lemma derives an expression for the descendant counter L^{BG} required by Proposition 5. Incidentally, the number of level-d δ_n explorables $A_{n,d}$ (Section 3.1) gets an elegant form in the binary grammar problem.

Lemma 8. *For $n < d$, let $L^{\text{BG}}(n,d) = |\{v : \text{level}(v) = d, v \in \text{descendants}(a^n)\}|$ be the number of nodes reachable from a^n, and let $A_{n,d} = L^{\text{BG}}(n,d) - L^{\text{BG}}(n+1,d)$ be the number of descendants of a^n that are not descendants of a^{n+1}. Then $L^{\text{BG}}(n,d) = \sum_{i=0}^{d-n} \binom{d}{i}$, and $A_{n,d} = \binom{d}{d-n}$.*

Proof. The reachable nodes on level d that we wish to count are $d-n$ levels below a^n. To reach this level we must add $i \le d-n$ number of b's and $d-n-i$ number of a's to a^n. The number of length d strings containing exactly i number of b's is $\binom{d}{i}$ (we are choosing positions for the b's non-uniquely with repetition among $d-i+1$ possible positions). Summing over i, we obtain $L^{\text{BG}}(n,d) = \sum_{i=0}^{d-n} \binom{d}{i}$, and $A_{n,d} = L^{\text{BG}}(n,d) - L^{\text{BG}}(n+1,d) = \binom{d}{d-n}$. $\square$

Corollary 9 (Expected Binary Grammar BFS Search Time)**.** *The expected BFS search time $\tilde{t}^{\text{DFS}}_{\text{BG}}(\mathbf{p})$ in a Binary Grammar Problem of depth D with goal*

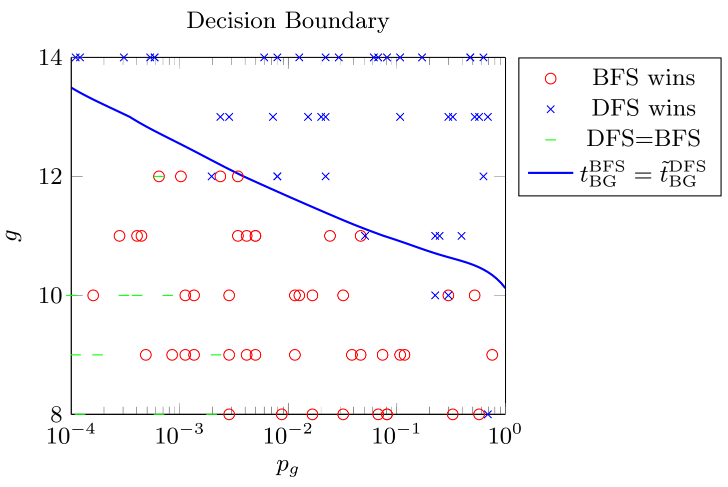

Figure 3: The decision boundary predicted by Corollary 9 and 10, together with empirical outcomes of BFS and DFS search time. The scattered points are based on 100 independently generated binary grammar problems of depth $D = 14$ with uniformly sampled (single) goal level $g \in [8, 14] \cap \mathbb{N}$ and $\log(p_g) \in [-4, 0]$. DFS benefits from a deeper goal level and higher goal probability compared to BFS. The decision boundary gets 87% of the instances correct.

probabilities $\mathbf{p} = [p_0, \dots, p_D]$ *is*

$$t^{\text{BFS}}_{\text{BG}}(\mathbf{p}) = t^{\text{BFS}}_{\text{CB}}(\mathbf{p}, L^{\text{BG}}).$$

Corollary 10 (Expected Binary Grammar DFS Search Time)**.** *The expected DFS search time* $\tilde{t}^{\text{DFS}}_{\text{BG}}(D, \mathbf{p})$ *in a binary grammar problem of depth* D *with goal probabilities* $\mathbf{p} = [p_0, \dots, p_D]$ *is bounded between* $t^{\text{DFS}}_{\text{BGL}}(D, \mathbf{p}) := t^{\text{DFS}}_{\text{CBL}}(D, \mathbf{p}, L^{\text{BG}})$ *and* $t^{\text{DFS}}_{\text{BGU}}(D, \mathbf{p}) := t^{\text{DFS}}_{\text{CBU}}(D, \mathbf{p}, L^{\text{BG}})$*, and is approximately*

$$\tilde{t}^{\text{DFS}}_{\text{BG}}(D, \mathbf{p}) := \tilde{t}^{\text{DFS}}_{\text{CB}}(D, \mathbf{p}, L^{\text{BG}}).$$

Proof of Corollary 9 and 10. Direct application of Lemma 8, and Proposition 7 and 5 respectively. □

The bounds are plotted for a single goal level in Figure 3 and 4.

4.2 Full Grammar

The *full grammar problem* has alphabet $B = \{S, a, b\}$ and start string S. The *production rules* are $S \to \epsilon$ (with ϵ the empty string) plus the *adding rules* $S \to Sa$, $S \to aS$, $S \to Sb$, $S \to bS$, and the *moving rules* $Sa \to aS$, $aS \to Sa$, $Sb \to bS$, and $bS \to Sb$. Only S-less strings can be goal nodes. As usual, a maximum depth D and a goal probability vector $\mathbf{p} = [p_0, \dots, p_D]$ are given.

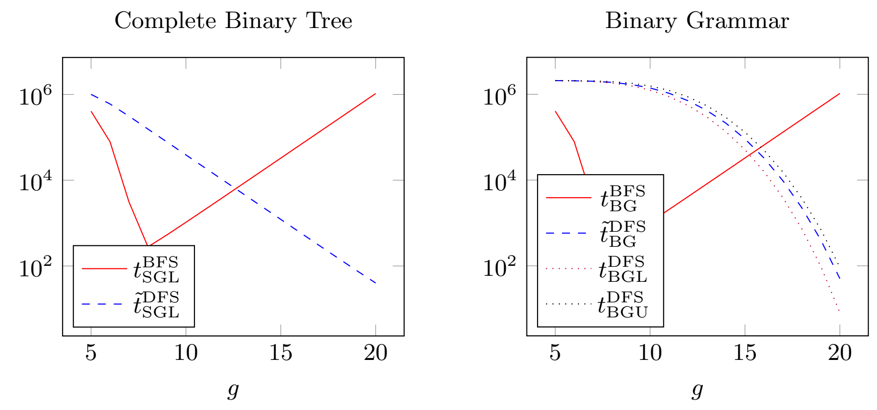

Figure 4: The expected search time of BFS and DFS as a function of a single goal level g with goal probability $p_g = 0.05$ in a tree of depth $D = 20$. BFS has the advantage when the goal is in the higher regions of the graph, although at first the probability that no goal exists heavily influences both BFS and DFS search time. The greater connectivity of the graph in the binary grammar problem permits DFS to spend more time in the lower regions before backtracking, compared to the complete binary tree analysed in the companion paper (Everitt and Hutter, 2015b). This penalises DFS runtime when the goal is not in the very lowest regions of the tree. BFS behaviour is identical in both models.

For simplified analysis, we will abuse notation the following way. We will consider S-less nodes to be one level higher than they actually are. For example, a would normally be on level 2 (e.g. reached by the path $S \to Sa$, $S \to \epsilon$), but we will consider it to be on level 1. A slight modification of BFS and DFS makes them always check the S-less child first (which is always child-less in turn), which means the change will only slightly affect search time. We will still consider $\delta_n = Sa^n$ whenever $S \to Sa$ is among the production rules, however.

The search graph of the full grammar problem is shown in Figure 5.

The problem can be analysed by a reduction to a binary grammar problem with the same parameters D and $\mathbf{p}$. Assign to each string v of the binary grammar problem the set of strings that only differ from v by (at most) an extra S. We call such sets *node clusters*. For example, $\{a, Sa, aS\}$ constitutes the node cluster corresponding to a. Due to the abusing of levels for the S-less strings, all members of a cluster appear on the same level (the level is equal to the number of a's and b's). The level is also the same as the corresponding string in the binary grammar problem.

Lemma 11 (Binary Grammar Reduction)**.** *For every n, d, $n \leq d$, the descendant counter L^{FG} of the full grammar problem is $L^{\mathrm{FG}}(n,d) = (d+2)L^{\mathrm{BG}}(n,d)$.*

Proof. $L^{\mathrm{BG}}(n,d)$ counts the level d descendants of a^n in the binary grammar problem (BGP), and $L^{\mathrm{FG}}(n,d)$ counts the level d descendants of Sa^n in the full

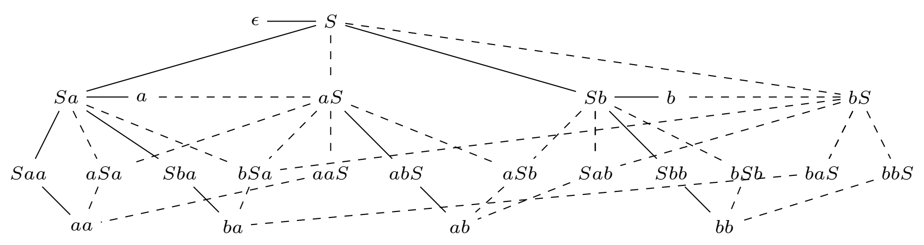

Figure 5: Search graph for the Grammar problem until level 2. Connections induced by moving rules are not displayed. Contiguous lines indicate the first discovery of a child by DFS and dashed lines indicate rediscoveries.

grammar problem (FGP). The node u is a child of v in BGP iff the members of the u node cluster are descendants of Su. Therefore the node clusters on level d descending from Sa^n in FGP correspond to the BGP nodes descending from a^n. At level d, each node cluster contains $d + 2$ nodes. □

Corollary 12 (Expected Full Grammar BFS Search Time)**.** *The expected BFS search time* $\tilde{t}^{\text{DFS}}_{\text{FG}}(\mathbf{p})$ *in a full grammar problem of depth* D *with goal probabilities* $\mathbf{p} = [p_0, \dots, p_D]$ *is*

$$t^{\text{BFS}}_{\text{FG}}(\mathbf{p}) := t^{\text{BFS}}_{\text{CB}}(\mathbf{p}, L^{\text{FG}}).$$

Corollary 13 (Expected Full Grammar DFS Search Time)**.** *The expected DFS search time* $\tilde{t}^{\text{DFS}}_{\text{FG}}(D, p)$ *in a full grammar problem of depth* D *with goal probabilities* $\mathbf{p} = [p_0, \dots, p_D]$ *is bounded between* $t^{\text{DFS}}_{\text{FGL}}(D, \mathbf{p}) := t^{\text{DFS}}_{\text{CBL}}(D, \mathbf{p}, L^{\text{FG}})$ *and* $t^{\text{DFS}}_{\text{FGU}}(D, \mathbf{p}) := t^{\text{DFS}}_{\text{CBU}}(D, \mathbf{p}, L^{\text{FG}})$*, and is approximately*

$$\tilde{t}^{\text{DFS}}_{\text{FG}}(D, \mathbf{p}) := \tilde{t}^{\text{DFS}}_{\text{CB}}(D, \mathbf{p}, L^{\text{FG}}).$$

Proof of Corollary 12 and 13. Direct application of Lemma 11, and Proposition 7 and 5 respectively. □

5 Experimental verification

To verify the analytical results, we have implemented the binary grammar in Python 3 using the `graph-tool` package (Peixoto, 2015).[1] The data reported in Table 1 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 binary grammar of depth 14 has about $2^{15} \approx 3 \cdot 10^5$

[1]Source code for the experiments is available at `http://tomeveritt.se`.

nodes (and a search algorithm must search through all of them in case there is no goal), the rarely occurring event of no goal may 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.*

The binary grammar model of Section 4.1 serves to verify the general estimates of Proposition 5 and 7. The results are shown in Table 1. The estimates for BFS are accurate ($< 3\%$ error). With few exceptions, the lower and the upper bounds $t^{\mathrm{DFS}}_{\mathrm{BGL}}$ and $t^{\mathrm{DFS}}_{\mathrm{BGU}}$ of Corollary 10 for DFS differ by at most 50% on the respective sides from the true (empirical) average. The arithmetic mean $\tilde{t}^{\mathrm{DFS}}_{\mathrm{BG}}$ often gives surprisingly accurate predictions ($< 4\%$) except when $t^{\mathrm{DFS}}_{\mathrm{BGL}}$ and $t^{\mathrm{DFS}}_{\mathrm{BGU}}$ leave wide margins as to the expected search time (when $g = 14$, the margin is up to 84% downwards and 125% upwards). Even then, the $\tilde{t}^{\mathrm{DFS}}_{\mathrm{BG}}$ error remains within 30%.

6 Discussion

Search and optimisation problems appear in different flavors throughout the field of artificial intelligence; in planning, problem solving, games, and learning. Therefore even minor improvements to search performance can potentially lead to gains in many aspects of intelligent systems. It is even possible to equate intelligence with (Bayesian expectimax) optimisation performance (Legg and Hutter, 2007).

Summary. In this paper and Part I (Everitt and Hutter, 2015b) we have derived analytical results for expected runtime performance. Part I focused on BFS and DFS *tree search* where explored nodes were not remembered. A vector $\mathbf{p} = (p_1, \dots, p_D)$ described *a priori* goal probabilities for the different levels of the tree. This concrete but general model of goal distribution allowed us to calculate approximate closed-form expression of both BFS and DFS average runtime. Earlier studies have only addressed *worst-case* runtimes: Knuth (1975) and followers for DFS; Korf et al. (2001) and followers for IDA*, effectively a generalised version of BFS.

This paper generalised the model of Part I to non-tree graphs. In addition to the goal probability vector $\mathbf{p}$, the graph search analysis required additional structural information in the form of a descendant counter L. The graph search estimates for DFS also took the form of less precise bounds. Even so, the arithmetic mean of the lower and the upper bound often came close the empirical average. The analysis of this paper does not supersede the results of Part I, as the bounds become uninformative when the graph is a tree. Overall, the analytical approximations derived in both papers were generally consistent with experimental outcomes.

$g\backslash p_g$	0.001	0.01	0.1
5		46.74	40.53
		46.64	*39.86*
		0.2 %	1.7 %
8	375.7	332.5	265.7
	378.0	*333.9*	*265.0*
	0.6 %	0.4 %	0.3 %
11	2751	2145	2058
	2744	*2147*	*2057*
	0.3 %	0.1 %	0. %
14	17 370	16 480	16 390
	17 380	*16 480*	*16 390*
	0.1 %	0. %	0. %

(a) BFS $t_{\mathrm{BG}}^{\mathrm{BFS}}$

$g\backslash p_g$	0.001	0.01	0.1
5		30 910	27 840
		31 370	*30 190*
		1.5 %	8.4 %
8	28 000	25 160	15 490
	27 410	*24 420*	*15 200*
	2.1 %	2.9 %	1.9 %
11	17 280	5932	1815
	16 790	*5806*	*1788*
	2.9 %	2.1 %	1.5 %
14	1304	122.1	25.60
	1522	*164.6*	*20.06*
	17 %	35 %	22 %

(b) Average DFS $\tilde{t}_{\mathrm{BG}}^{\mathrm{DFS}}$

$g\backslash p_g$	0.001	0.01	0.1
5		30 910	27 840
		30 710	*29 080*
		0.7 %	4.5 %
8	28 000	25 160	15 490
	25 740	*22 150*	*12 070*
	8.1 %	12 %	22 %
11	17 280	5932	1815
	14 160	*3822*	*918.6*
	18 %	36 %	49 %
14	1304	122.1	25.60
	808.8	*54.12*	*3.990*
	38 %	56 %	84 %

(c) Lower DFS $t_{\mathrm{BGL}}^{\mathrm{DFS}}$

$g\backslash p_g$	0.001	0.01	0.1
5		30 910	27 840
		32 020	*31 290*
		3.6 %	12 %
8	28 000	25 160	15 490
	29 080	*26 690*	*18 340*
	3.8 %	6.1 %	18 %
11	17 280	5932	1815
	19 410	*7790*	*2657*
	12 %	31 %	46 %
14	1304	122.1	25.60
	2236	*275.1*	*36.12*
	72 %	125 %	41 %

(d) Upper DFS $t_{\mathrm{BGU}}^{\mathrm{DFS}}$

Table 1: Comparison of analytical estimates with empirical averages for BFS and DFS in binary grammars of depth $D = 14$. Goals are distributed on a single goal level g with goal probability p_g. The BFS estimates $t_{\mathrm{BG}}^{\mathrm{BFS}}$ are highly accurate, and the averaged DFS estimates $\tilde{t}_{\mathrm{BG}}^{\mathrm{DFS}}$ are mostly accurate. Each box contains empirical average/*analytical expectation*/error percentage.

Conclusions and Outlook. The value of the results are at least twofold. They offer a concrete means of deciding between BFS and DFS given some rough idea of the location of the goal (and the graph structure). To make the results more generally usable, automatic inference of model parameters would be necessary; primarily of goal distribution p and graph structure L. (The depth D will often be set by the searcher itself, and perhaps be iteratively increased.) There is good hope that the descendant counter L can be estimated online from the local sample obtained during search, similar to (Knuth, 1975). The goal distribution is likely to prove more challenging, but resembles the automatic creation of heuristic functions, so techniques such as *relaxed problems* could well prove useful (Pearl, 1984). Estimates of goal distribution could possibly also be inferred from a heuristic function.

The results also offer theoretical insight into BFS and DFS performance. As BFS and DFS are in a sense the most fundamental search operations, we have high hopes that our results and techniques will prove useful as building blocks for analysis of more advanced search algorithms. For example, A* and IDA* may be viewed as a generalisations of BFS, and Beam Search and Greedy Best-First as generalisations of DFS.

Acknowledgements

Thanks to David Johnston for proof reading a final draft.

References

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 I: Tree 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.

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.

Legg, S. and Hutter, M. (2007). Universal Intelligence. *Minds & Machines*, 17(4):391–444.

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

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

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

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