ICECA

International Conference
Enumerative Combinatorics and Applications
University of Haifa – Virtual – September 4-6, 2023

BFS versus DFS for random targets in ordered trees

Stoyan Dimitrov[1]
Joint work with Luz E. Grisales Gomez, and Yan Zhuang

[1]Department of Mathematics, Rutgers University

EmailToStoyan@gmail.com

Abstract: Assume that we are at the root of an unknown ordered tree T with n edges and that we want to find a specific target node x located at a given level l in T. We compare the performance of the breadth-first search (BFS) and depth-first search (DFS) algorithms in this scenario, when the target x is chosen uniformly at random. Intuition suggests that BFS should have faster average performance when l is small, while DFS must have an advantage when l is large. But where exactly is the threshold, and is it unique?

We obtain explicit formulas for the expected number of steps when using BFS and when using DFS for given n and l. This allows us to show that there exists a constant $C > 0$ for which $l = C\sqrt{n}$ is a threshold where, asymptotically, DFS becomes faster than BFS in expectation. We also introduce the *truncated DFS* algorithm, which performs better than both BFS and DFS when l is known in advance, and we find a formula evaluating the average time complexity of this algorithm. We conclude by presenting several questions for future study.

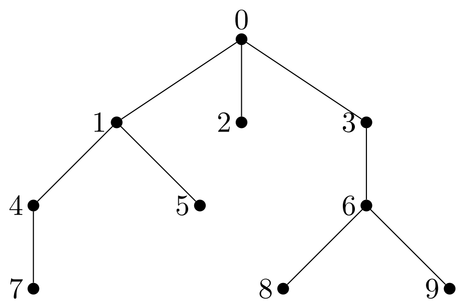
1. Introduction

Several important problems in computer science and artificial intelligence can be formulated as search problems [6, Chapter 3]. A few examples are the traveling salesman problem, scheduling problems, and the problem of finding optimal moves in board games such as chess and backgammon. Different search problems have different optimal algorithms depending on the instance, and choosing the right method can lead to significant gains. Some machine learning approaches exist for algorithm selection [4], but they usually do not provide insight as to why one algorithm is preferred over another. At the same time, most existing analytical results focus on worst-case analysis, which is often less useful than average-case analysis when it comes to algorithm selection.

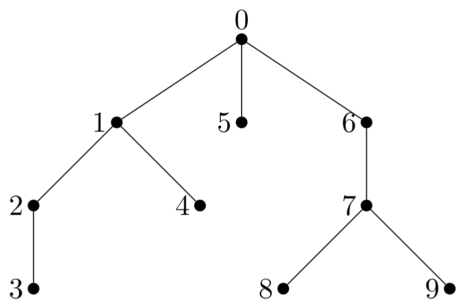
In this paper, we compare the average time complexity of two popular tree search algorithms—breadth-first search (BFS) and depth-first search (DFS)—when searching for a random node at a given level in the set of ordered trees with a fixed number of edges.

1.1. **Problem and notation.** Consider the set $\mathcal{T}$ of all unlabeled rooted ordered trees, which we simply refer to as *ordered trees.* In each of these trees, we have a fixed node called a *root*, and an ordering is specified for the children of each node. Let $\mathcal{T}_n$ denote the subset of trees in $\mathcal{T}$ having exactly n edges (and $n+1$ nodes). The *level* of a node v is the length of the unique path from v to the root.

Breadth-first search (BFS) and depth-first search (DFS) are two popular algorithms for graph and tree traversal. BFS explores the entire neighborhood of the current node, while DFS follows one path for as long as possible and backtracks when stuck. Figure 1 illustrates the orders in which BFS and DFS will explore the nodes of a particular tree.



(A) The order of exploration for the BFS algorithm.



(B) The order of exploration for the DFS algorithm.

FIGURE 1. The bfsScore and dfsScore of each node in a tree with 9 edges.

Let us call these orders the *bfsScore* and the *dfsScore* of a node v, defined as follows:

$$\text{bfsScore}(v) := \text{the number of nodes visited before } v \text{ when using BFS.}$$

$$\text{dfsScore}(v) := \text{the number of nodes visited before } v \text{ when using DFS.}$$

Importantly, bfsScore(v) and dfsScore(v) also give the number of steps when searching for v via BFS and DFS, respectively.

Question 1.1. *For given l and n, does BFS or DFS have a smaller expected number of steps when one performs a search for a random node x at level l in a tree with n edges?*

In the question above, we assume that x is selected uniformly at random from all nodes at level l among all trees in $\mathcal{T}_n$. For instance, when $n=3$ and $l=2$, we have 5 such nodes as seen in Figure 2.

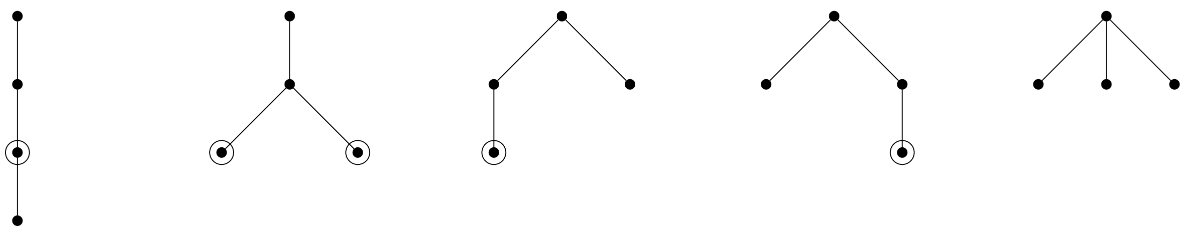


FIGURE 2. The target node is selected uniformly at random among the five circled nodes at level $l=2$, when we have $n=3$ edges.

Intuition suggests that BFS is a better choice if l is small compared to n, while DFS is preferred when l is relatively big. But where exactly is the threshold, and is the threshold unique?

The total number of nodes at level l among all trees in $\mathcal{T}_n$ is known [3, Corollary 5.3]. Thus to compare the average-case time complexity of the two algorithms, it suffices to calculate the sum of all bfsScores and dfsScores over all nodes at level l among trees in $\mathcal{T}_n$. Thus if $\operatorname{lev}(T,l)$ is the set of nodes at level l in the tree T, we have

$$\operatorname{totalB}(n,l) \coloneqq \sum_{T\in\mathcal{T}_n}\sum_{v\in\operatorname{lev}(T,l)} \operatorname{bfsScore}(v),$$

and

$$\operatorname{totalD}(n,l) \coloneqq \sum_{T\in\mathcal{T}_n}\sum_{v\in\operatorname{lev}(T,l)} \operatorname{dfsScore}(v).$$

2. Exact formula for totalD

First, we establish a surprisingly simple formula for $\operatorname{totalD}(n,l)$ given by Theorem 2.1 below.

Theorem 2.1. *For every $n \geq 0$ and $0 \leq l \leq n$, we have*

$$\operatorname{totalD}(n,l) = l\binom{2n}{n-l}.$$

Sketch of Proof. Let us call *diagonal paths*, the lattice paths with steps $(0,1)$ and $(1,0)$ called U and D steps, respectively. In the proof, we use several times a well-known fact from lattice path enumeration [5, Corollary 10.3.2], which gives the number of diagonal paths, $[(0,0)\to(i,j)]$, from $(0,0)$ to (i,j) that stay weakly above $y=x$.

First, we use a folklore bijection between ordered trees and Dyck paths. In it, we perform a pre-order traversal of the input tree—first visiting the root and then recursively the subtrees from left to right—writing a U when we make a step further away from the root and a D when we make a step moving us closer to the root. As a result, a node at level l in a tree with n edges is matched to a point on the corresponding Dyck path. Given a node v in an ordered tree, it is readily seen that $\operatorname{dfsScore}(v)$ is equal to the number of U steps in the prefix of the corresponding Dyck path that ends at the point corresponding to v. Using the latter fact, we obtain the summation formula below.

$$\operatorname{totalD}(n,l) = \sum_{j=l}^{n} j\cdot[(0,0)\to(i,j-1)]\cdot[(i,j)\to(n,n)].$$

After we simplify and invoke the mentioned lattice path enumeration fact, we obtain that it remains to prove the following identity.

$$\sum_{j=l}^{n}\frac{l+1}{2n-2j+l+1}\binom{2j-l-1}{j-1}\binom{2n-2j+l+1}{n-j} = \binom{2n}{n-l}. \tag{1}$$

We show that both sides of (1) count horizontal paths from $(0,0)$ to $(2n,2)$—i.e., those with $n+1$ Us and $n-1$ Ds—that cross the horizontal line $y=l$. □

The last Theorem 2.1 help us to obtain the expected dfsScore over all nodes at a given level l among the trees in $\mathcal{T}_n$:

$$\mathop{\mathbb{E}}_{\substack{x\in\operatorname{lev}(T,l)\\ T\in\mathcal{T}_n}}(\operatorname{dfsScore}(x)) = \frac{l}{2l+1}(n+l+1).$$

For fixed l and large n, this expectation is approximately $n/2$. In private communication, Svante Janson showed us an argument suggesting the same answer. However, we showed that his argument cannot be used directly to obtain Theorem 2.1.

3. Exact formula for totalB

In contrast to the combinatorial methods used previously, our approach here uses generating functions. We begin with the generating function $F_l(x, y, z)$ defined by

$$F_l(x, y, z) := \sum_{n=0}^{\infty} \sum_{T \in \mathcal{T}_n} x^{k(T)} y^{m(T)} z^n,$$

where $k(T)$ is the number of non-root nodes in T at levels smaller than l and $m(T)$ is the number of nodes in T at level l.

We show that if

$$B_l = B_l(z) := \sum_{n=0}^{\infty} \mathrm{totalB}(n, l) z^n,$$

then we can express B_l in terms of F_l in the following way.

$$B_l = \left[\left(\frac{\partial^2}{\partial x \partial y} + \frac{1}{2} \cdot \frac{\partial^2}{\partial y^2} + \frac{\partial}{\partial y} \right) F_l \right]_{x=1,\, y=1}. \tag{2}$$

If C denotes the ordinary generating function for Catalan numbers, then we obtain the formulas below by utilizing several facts related to the so-called Fibonacci polynomials $f_n(z) := \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n-k}{k} z^k$.

Theorem 3.1. *For all $l \geq 1$, we have*

(a)
$$F_l(x, y, z) = \frac{f_{l-1}(-xz) - f_{l-2}(-xz) yzC}{f_l(-xz) - f_{l-1}(-xz) yzC}$$

and

(b)
$$B_l(z) = z^l C^{3l+1} \left(zC \frac{d}{dz} f_l(-z) - \frac{d}{dz} f_{l+1}(-z) \right).$$

By using Lagrange inversion, we extract coefficients from the generating function B_l, in order to find $\mathrm{totalB}(n, l)$. We get the following explicit formula.

Theorem 3.2. *For all $n \geq 0$ and $0 \leq l \leq n$, we have*

$$\begin{aligned}\mathrm{totalB}(n, l) = & \sum_{k=1}^{\lfloor (l+1)/2 \rfloor} (-1)^{k-1} k \binom{l-k+1}{k} \frac{3l+1}{n-k+2l+2} \binom{2n-2k+l+2}{n-l-k+1} \\ & - \sum_{k=1}^{\lfloor l/2 \rfloor} (-1)^{k-1} k \binom{l-k}{k} \frac{3l+2}{n-k+2l+2} \binom{2n-2k+l+1}{n-l-k}.\end{aligned}$$

4. Comparing the average time complexity of BFS and DFS

Next, we compare the average time complexity of BFS and DFS. First, we obtain an alternative way to write the formula for $\mathrm{totalB}(n, l)$ in Theorem 3.2. Let $n^{(l)}$ denotes the rising factorial $n(n+1)(n+2)\cdots(n+l-1)$ and let $n_{(l)}$ denotes the falling factorial $n(n-1)(n-2)\cdots(n-l+1)$. We prove that $\mathrm{totalB}(n, l)$ can be written in the following form.

Theorem 4.1. *For all $n \geq l \geq 1$, we have that*

$$\mathrm{totalB}(n, l) = \frac{4 p_l(n)}{(n+l+2)^{(l)}} \binom{2n+1}{n-l} \tag{3}$$

where $p_l(n)$ is a polynomial of degree $l-1$ with leading coefficient $l(l+1)(2l+1)/6$ (see [7, A000330]*).*

To show that the leading coefficient of $p_l(n)$ is $l(l+1)(2l+1)/6$, we gave a combinatorial prove to the identity below. In particular, we showed that $\binom{n+1}{3}$ counts certain tilings of an $n \times 1$ strip with dominoes and squares. For the remaining tilings, we define a sign-reversing involution.

Theorem 4.2. *For $n \geq 0$,*

$$\sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^{k-1} k \binom{n-k}{k} 2^{n-2k} = \binom{n+1}{3}.$$

Next, we give the main result of this paper: a threshold function for l where DFS becomes asymptotically faster than BFS. We will write $f(n) \prec g(n)$ if f is of smaller asymptotic order than g, and $f(n) \asymp g(n)$ if they are of the same asymptotic order.

Theorem 4.3. *Let l be a function of n. We have the following:*

(a) *If $l = \mathcal{O}(\sqrt{n})$, then there exist positive constants C_1 and C_2 for which*

$$\mathrm{totalB}(n,l) \asymp \frac{C_1 n l^3 + C_2 l^5}{n^2} \binom{2n+1}{n-l}.$$

(b) *When $l \prec \sqrt{n}$ and $n \to \infty$, we have* $\mathrm{totalB}(n,l) < \mathrm{totalD}(n,l)$.

(c) *There exists a constant $C > 0$ such that $l = C\sqrt{n}$ is a threshold for where* $\mathrm{totalB}(n,l)$ *becomes larger than* $\mathrm{totalD}(n,l)$ *when $n \to \infty$. In other words, when $l \asymp \sqrt{n}$ and $n \to \infty$,*

$$C\sqrt{n} - l \text{ is asymptotically positive and unbounded} \implies \mathrm{totalB}(n,l) < \mathrm{totalD}(n,l) \quad \text{and}$$

$$l - C\sqrt{n} \text{ is asymptotically positive and unbounded} \implies \mathrm{totalB}(n,l) > \mathrm{totalD}(n,l).$$

Computational evidence suggests that this is a unique threshold and thus if $\sqrt{n} \prec l$, then DFS is faster in expectation than BFS. In order to establish this, one could try to obtain more accurate bounds for the asymptotics of $\mathrm{totalB}(n,l)$ by using results of Aldous [1]. We also note the following result on the average height (maximum level) of an ordered tree.

Theorem 4.4 (de Bruijn–Knuth–Rice [2])**.** *The average height of a tree in $\mathcal{T}_n$, selected uniformly at random, is $\sqrt{\pi n} + \mathcal{O}(1)$.*

Interestingly, our threshold occurs around the average height of trees in $\mathcal{T}_n$, which leads us to wonder if the constant C in Theorem 4.3 is in fact $\sqrt{\pi}$. We ask a few additional questions in Section 6.

5. Search in case of known target level

If we know l in advance, it is reasonable to use a simple modification of the DFS algorithm that we will call *truncated DFS*, which does not visit nodes at levels above l. In that case, let us write

$$\mathrm{dfsTruncScore}(v) \coloneqq \text{the number of nodes visited before } v \text{ when using truncated DFS.}$$

Figure 3 gives an example of tree traversal when using truncated DFS where $l = 2$; each node is labeled with its dfsTruncScore. Observe that nodes at levels above $l = 2$ are not visited.

Define

$$\mathrm{totalDTrunc}(n,l) \coloneqq \sum_{T \in \mathcal{T}_n} \sum_{v \in \mathrm{lev}(T,l)} \mathrm{dfsTruncScore}(v).$$

It is not difficult to see that truncated DFS has a smaller expected number of steps compared to both BFS and DFS. We suspect that truncated DFS is not only faster than BFS and DFS, but that it is the fastest in expectation among all algorithms using the *deque* data structure. If this is indeed the case, it will be important to compute the corresponding generating function. Using some symmetry observations, generating functions and once again Lagrange inversion, we obtain an explicit, but quite long formula for $\mathrm{totalDTrunc}(n,l)$.

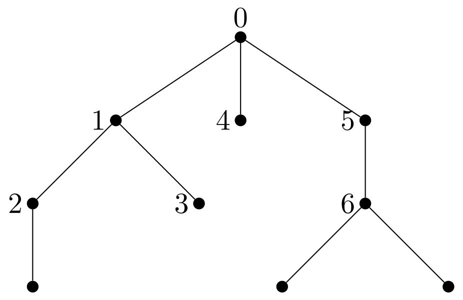


Figure 3. The order of exploration for the truncated DFS algorithm.

6. Further questions

A major future goal will be to determine the value of the constant in our threshold function and to prove that there is no other transition. We present the following additional questions.

Question 6.1. *Suppose that we know the values of n and l in advance and that we perform a search using a deque, i.e, we can switch between BFS and DFS at every step. Under these conditions, what is the optimal algorithm giving the smallest expected number of steps before reaching the target?*

As mentioned earlier, we believe the answer to Question 6.1 is our truncated DFS algorithm.

Question 6.2. *Suppose that we do not know the exact level l where our target node is located, only that l is in an interval $[u, v]$. What is the optimal algorithm, using the deque data structure, in this setting?*

Question 6.3. *Suppose that we have multiple target nodes chosen uniformly at random among those at a given level l. Compare the average case time complexity of BFS and DFS in this scenario; where are the transitions?*

References

[1] David Aldous. The continuum random tree. II. An overview. In *Stochastic Analysis (Durham, 1990)*, volume 167 of *London Math. Soc. Lecture Note Ser.*, pages 23–70. Cambridge Univ. Press, Cambridge, 1991.
[2] N. G. de Bruijn, D. E. Knuth, and S. O. Rice. The average height of planted plane trees. In *Graph Theory and Computing*, pages 15–22. Academic Press, New York-London, 1972.
[3] Nachum Dershowitz and Shmuel Zaks. Enumerations of ordered trees. *Discrete Math.*, 31(1):9–28, 1980.
[4] Lars Kotthoff. Algorithm selection for combinatorial search problems: a survey. In *Data Mining and Constraint Programming*, volume 10101 of *Lecture Notes in Comput. Sci.*, pages 149–190. Springer, Cham, 2016.
[5] Christian Krattenthaler. Lattice path enumeration. In *Handbook of Enumerative Combinatorics*, Discrete Math. Appl. (Boca Raton), pages 589–678. CRC Press, Boca Raton, FL, 2015.
[6] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, fourth edition, 2021.
[7] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. Published electronically at `http://oeis.org`.