Empirical Analysis of Distributed Depth-First Search Algorithms

S. A. M. Makki and George Havas*
Department of Computer Science
The University of Queensland
Queensland 4072, Australia
sam@cs.uq.oz.au havas@cs.uq.oz.au

Abstract

We evaluate the practical performance of two improved distributed depth-first search algorithms applied to general and commonly used network structures. In particular we study the behavior of the algorithms in terms of a parameter which depends on the network topology and routing chosen. We deduce that performance is significantly enhanced for smaller networks of sizes which arise in practice. With increasing network size the extended version of the algorithm performs appreciably better than the basic algorithm. We present some unresolved questions about these algorithms.

Keywords: Distributed systems, depth-first search, distributed algorithms, communication graph.

1 Introduction

Depth-first search (dfs) traces a special kind of spanning tree in a connected graph: the nodes are visited in a particular manner. Depth-first search algorithms can produce any one of many different dfs trees from the same graph and starting point because at each stage of the search any unvisited neighbor of a node is a possible candidate for the selection as the next node. This selection is generally done by a process which may be quasi-random. Especially in distributed systems, it is reasonable to assume the choice is randomly made.

In [5] we present distributed algorithms for constructing a depth-first search tree of a communication network which are more efficient than previous methods. Our algorithms require $2|V| - 2$ messages and units of time in the worst case (where $|V|$ is the number of sites in the network) and as little as $|V|$ messages and time units in the best case. The actual number of messages and time units depends on the network topology and possibly on the routing chosen. We can interpret this to mean that the number of messages and time units is $|V|(1 + r)$ in the worst case, where $0 \leq r < 1$ and the value of r depends on the topology and the routing. The improvement over the best of previous algorithms is achieved by dynamic backtracking, with a minor increase in message length.

Our algorithms use FORWARD and RETURN messages to explore the tree. The number of FORWARD messages is always $|V| - 1$; the number of RETURN messages varies between 1 and $|V| - 1$ depending on the topology and routing. Our key improvement is to reduce the number of RETURN messages by using dynamic backtracking. We study the algorithms in terms of both number of RETURN messages and a ratio related to the parameter r.

In the development of distributed algorithms often particular graph structures including Rings, Trees, Star graphs, Complete graphs, Grids and Hypercubes are used as a platform. We investigate some of these structures in detail and also consider general graph behavior. Because of the very large number of dfs trees in even moderate size graphs it is difficult to make complete theoretical analyses so we study the algorithms experimentally. This provides useful insight onto their behavior and raises some interesting questions.

2 Examples

Practical methods for calculating the number of spanning trees rooted at a vertex in an undirected graph [2] are available. We do not have such a convenient calculation method for the number of dfs trees. However, enumerations show that the number of spanning trees and the number of dfs trees increase rapidly with increasing number of nodes. The number of spanning trees for a hypercube of degree three with 8 nodes is 384 while the number of dfs trees is thirty; for a degree four hypercube with 16 nodes the number of spanning trees is $42,467,328$ and the number of

*Supported in part by the Australian Research Council

dfs trees exceeds ten thousand; for a hypercube of degree 5 with 32 nodes the number of spanning trees is 20, 776, 019, 874, 734, 407, 680.

For illustrative purposes we start by considering the degree 3 hypercube (Figure 1) in detail.

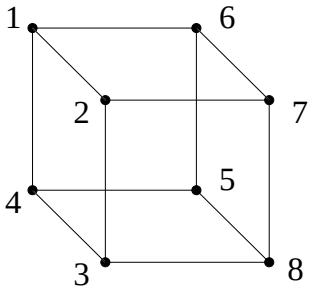

Figure 1: *A degree 3 hypercube*

In Table 1 we show all dfs tree paths for this hypercube from node 1 as the *root* node and with node 2 selected as its first neighbor to visit and node 3 as the next neighbor. (By symmetry we have no loss of generality starting this way.) We do this in the context of our basic Ddfs algorithm [5]. Nodes are shown in normal font as recipients of FORWARD messages and underlined in bold font as recipients of RETURN messages.

Even if we assume that nodes are selected with equal probability whenever a choice is to be made, these five paths do not have equal likelihood of arising. We can calculate the actual probability of each path with the natural assumption that at each vertex a random selection is made for continuing the search. Thus, if the search starts from node 1 the probability of selecting any one of its three neighbors (2, 4, 6) is $\frac{1}{3}$. After node 2 has been selected then either node 3 or 7 is next, with probability of $\frac{1}{2}$ each, and the analysis is readily completed. In Table 1 we also show the probability of selecting each of the listed paths (under the assumption of equal likelihood node selection) and the number of RETURN messages. (These paths represent one sixth of all possible dfs trees rooted at node 1.)

From Table 1 (and similar calculations for all possible paths from node 1) we can see the actual number of return messages for all possible routings and their probabilities of being selected. It follows that the average number of RETURN messages under the natural assumption is given by:

$$\left\{3 \times (5 \times \frac{1}{24}) + (5 \times \frac{1}{48}) + (7 \times \frac{1}{48})\right\} \times 6 = 5.25 \ .$$

Because of the symmetry of the hypercube the same result applies for any other node as the start (*root*) node.

Path	Probability	$\#R$
2 3 4 5 6 7 8 **5 4 3 2 1**	1/24	5
2 3 4 5 8 7 6 **5 4 3 2 1**	1/24	5
2 3 8 5 4 **5** 6 7 **8 3 2 1**	1/48	5
2 3 8 5 6 7 **6 5** 4 **5 8 3 2 1**	1/48	7
2 3 8 7 6 5 4 **5 8 3 2 1**	1/24	5

Table 1: *Paths, probabilities and return messages*

3 Outline of the experimental method

Even though the number of dfs trees does not grow with increasing number of nodes as quickly as the number of spanning trees, it grows too quickly for us to completely analyze even moderate size graphs. Taking symmetries into account can help, but not enough to get around this problem. For this reason we study behavior by undertaking experiments which simulate our natural assumption.

Thus to show the behavior of the algorithm on different network structures we have carried out a reasonable number of tests. In order to improve the quality of our results we have taken advantage of symmetry when appropriate.

We consider a number of the standard graph structures of various sizes as used in distributed systems. In our experiments we counted the actual number of RETURN messages for each of 1200 randomly constructed dfs trees. Since the number of RETURN messages (which we denote by $\#R$) can range from 1 to $|V| - 1$, we also use a ratio (related to r) which ranges from 0 to 1 as a more uniform metric of performance. Thus we define $\rho = (\#R - 1)/(|V| - 2)$. Best possible performance has $\rho = 0$, worst possible performance has $\rho = 1$. In our tables and plots we show the average value of $\#R$ derived from all 1200 experiments in each case and also the value of ρ corresponding to that average.

4 The basic Ddfs algorithm

4.1 Hypercube type networks

An n-dimensional hypercube is a regular graph of degree n which has 2^n nodes and $n2^{n-1}$ edges, with a diameter of n [4]. Hypercubes are symmetric structures for depth-first search in which it is immaterial for our analysis which node is used as

the *root*. Table 2 shows algorithm behavior on this structure.

Graph	$\|V\|$	$\#R$	ρ
2D	4	1	0
3D	8	5.40	0.73
4D	16	13.15	0.87
5D	32	28.45	0.92
6D	64	59.71	0.95
7D	128	122.59	0.96
8D	256	248.40	0.97
9D	512	500.80	0.98

Table 2: *Basic Ddfs for hypercubes*

Because of the exponential relationship between the degree of the nodes and the number of edges some other modified structures based on this configuration are used as alternatives [1, 4]. Cube-connected cycles are hypercubes in which each vertex is replaced by a cycle of n vertices to give better balance between the number of edges and the number of vertices. An n-dimensional shuffle-exchange network has the same number of the nodes as an n-dimensional hypercube and its diameter is $O(\log|V|)$. An n-dimensional butterfly network has $V = (n+1)2^n$ nodes and $n2^{n+1}$ edges, with diameter also $O(\log|V|)$. We tabulate results for butterfly networks in Table 3 and compare all of these graphically for small networks in Figure 2.

Graph	$\|V\|$	$\#R$	ρ
1D	4	1	0
2D	12	8.38	0.74
3D	32	23.53	0.75
4D	80	61.68	0.78
5D	192	154.71	0.81
6D	448	371.18	0.83

Table 3: *Basic Ddfs for butterfly networks*

4.2 Meshes and Toruses

A two-dimensional mesh consists of $m \times n$ processors organized in a rectangular grid of m rows and n columns. Each pair of adjacent processors along the same row or column are connected by a bidirectional communication link. Table 4 shows the behavior of the algorithm on square meshes. (Similar performance is displayed on double-layer meshes.)

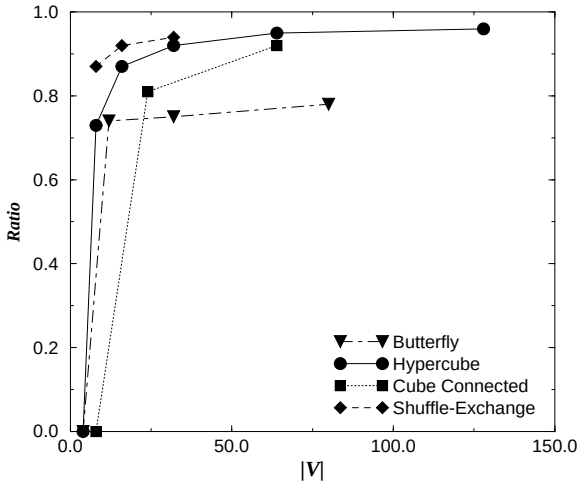

Figure 2: *Basic Ddfs: cube type networks*

Torus structures are obtained from meshes by adding toroidal edges. Addition of these edges not only makes all the nodes have equal degree 4, but also reduces the diameter of $n \times n$ structures to $(n-1)$, which is half the diameter of the corresponding mesh.

		Meshes		Toruses	
Graph	$\|V\|$	$\#R$	ρ	$\#R$	ρ
3×3	9	5.88	0.70	6.61	0.80
4×4	16	13.01	0.86	12.91	0.85
5×5	25	20.96	0.87	21.56	0.89
6×6	36	30.95	0.88	31.87	0.91
7×7	49	42.69	0.89	43.82	0.91
8×8	64	56.24	0.89	57.42	0.91
9×9	81	71.54	0.89	72.81	0.91
10×10	100	88.61	0.89	89.98	0.91
11×11	121	107.51	0.90	108.81	0.91
12×12	144	128.23	0.90	129.51	0.91

Table 4: *Basic Ddfs for meshes and toruses*

These tables and Figure 3 indicate that the basic algorithm generally performs better on mesh and torus structures than hypercube structures

5 An extended Ddfs algorithm

There are various extensions to the basic algorithm suitable for practical networks. Here we consider the extension described in [5] which relies on using longer messages including a variable *root* and a set

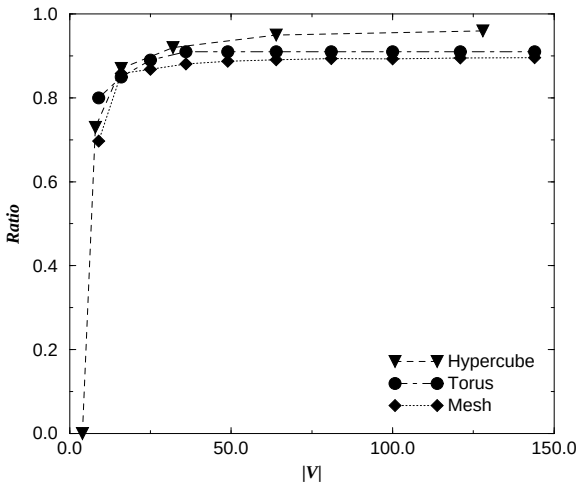

Figure 3: *Basic Ddfs: comparisons*

knownunvisited. This extension improves the average ratio ρ for complete graphs which is near 1 for the basic algorithm (i.e. almost worst possible) to 0 for the extended algorithm (i.e. best possible). For this extended distributed dfs algorithm we present corresponding tables and plots for the same experiments as for the basic algorithm.

First we show in Table 5 how return paths for the 3-dimensional hypercube for the extended algorithm are dramatically shortened relative to the same dfs tree constructed by the basic algorithm. (Theoretical calculations show that the average number of RETURN messages is reduced from 5.25 to 2.25 .) Then we tabulate (Tables 5, 6 and 7) and plot (Figures 4, 5 and 6) the other results. Figure 4 highlights the difference between the basic and extended algorithms for hypercubes.

Path	Probability	$\#R$
2 3 4 5 6 7 8 **5** **4** **1**	1/24	3
2 3 4 5 8 7 6 **1**	1/24	1
2 3 8 5 4 **5** 6 7 **8** **3** **2** **1**	1/48	5
2 3 8 5 6 7 **6** **5** 4 **1**	1/48	3
2 3 8 7 6 5 4 **1**	1/24	1

Table 5: *Paths for the extended algorithm*

Graph	$\|V\|$	$\#R$	ρ
2D	4	1	0
3D	8	2.30	0.22
4D	16	4.70	0.26
5D	32	10.57	0.32
6D	64	24.94	0.39
7D	128	57.46	0.45
8D	256	128.76	0.50
9D	512	264.16	0.52

Table 6: *Extended Ddfs for hypercubes*

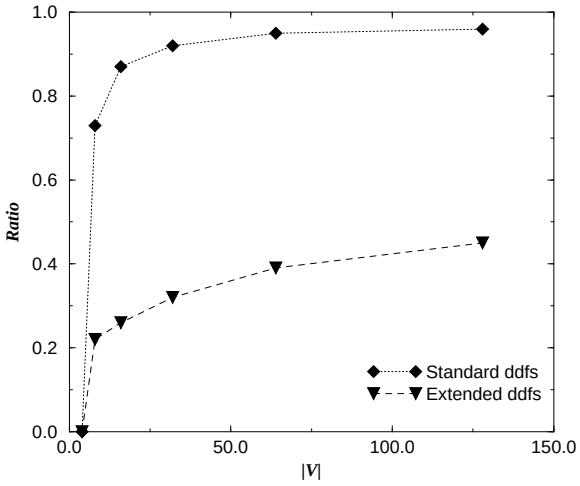

Figure 4: *Basic & Extended Ddfs on Hypercubes*

6 Conclusions

Our experiments show that the performance of the basic Ddfs algorithm gets worse for many standard configurations as the number of vertices goes up. The extended algorithm shows better performance as a result of a modest increase in message size which enables more information about the network topology to be transmitted.

Only a few structures have been fully theoretically analyzed. Our experimental results suggest many questions of interest. We list some of these.

1. Can more configurations be completely analyzed for variants of these algorithms?

2. Does average ρ tend to 1 for the basic algorithm with increasing degree for hypercubes?

3. Does average ρ tend to a constant less than 1 for

Graph	$\|V\|$	Meshes $\#R$	Meshes ρ	Toruses $\#R$	Toruses ρ
3×3	9	4.15	0.45	2.51	0.22
4×4	16	9.06	0.58	4.72	0.27
5×5	25	15.26	0.62	9.99	0.39
6×6	36	24.12	0.68	17.97	0.50
7×7	49	35.22	0.73	28.68	0.59
8×8	64	48.31	0.76	41.67	0.66
9×9	81	63.41	0.79	56.93	0.71
10×10	100	80.48	0.81	74.11	0.75
11×11	121	99.41	0.83	93.34	0.78
12×12	144	120.14	0.84	114.46	0.80

Table 7: *Extended Ddfs for meshes and toruses*

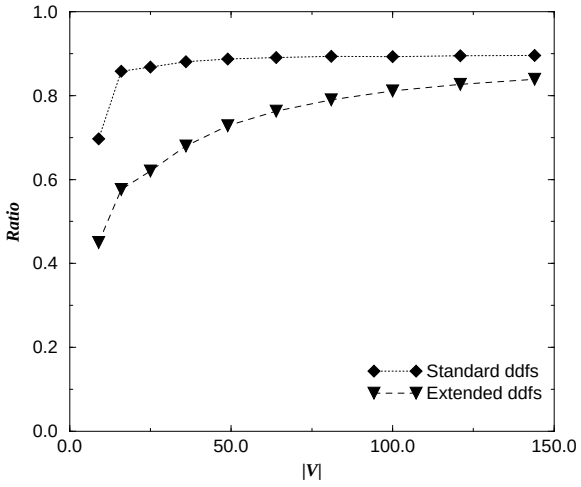

Figure 5: *Basic & Extended Ddfs on Meshes*

the extended algorithm with increasing degree for hypercubes?

4. What is the behavior of the basic and extended algorithms for meshes and toruses of increasing size? Does average ρ tend to a constant less than 1 with increasing size?

5. How do graph degrees and symmetries affect average ρ?

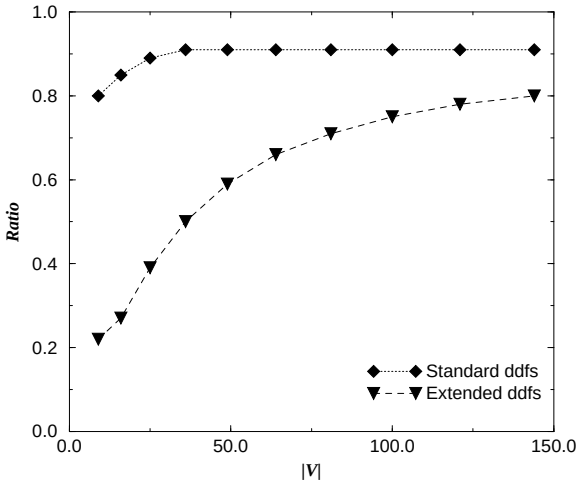

Figure 6: *Basic & Extended Ddfs on Toruses*

Acknowledgements

We are grateful to Julie Lawrence who first simulated these algorithms and presented her results in [3].

References

[1] S. G. Akl, *The Design and Analysis of Parallel Algorithms* (Prentice Hall, 1989).

[2] R. A. Bari, "A Combinatorial Approach to Graphical Polynomials and Spanning Subgraphs", *Annals of the New York Academy of Sciences*, vol. 328 (1979) pp. 21–29.

[3] J. Lawrence, *Aspects of graph theory and algorithms* (Honours Report, The University of Queensland, 1994).

[4] F. T. Leighton, *Introduction to Parallel Algorithms and Architecture: Arrays, Trees and Hypercubes* (Morgan Kaufmann Publishers, 1992).

[5] S. A. M. Makki, G. Havas, "Distributed algorithms for a depth-first search", *Information Processing Letters*, to appear.