Do NOT Modify Breadth-First Search

Michael Coulombe

Electrical Engineering and Computer Science Department
Massachusetts Institute of Technology
32 Vasser Street, Cambridge, MA
mcoulomb@mit.edu

Abstract

This paper settles the conjecture that Breadth-First Search can solve the single source shortest paths problem on weighted graphs in linear time, by demonstrating counter-example graphs for which BFS and modifications thereof fail.

1. Introduction

Breadth-First Search (BFS) was discovered by Konrad Zuse in 1945 [10]. Algorithm 1 shows modern code for BFS, optimally solving single source shortest paths in an unweighted, directed graph $G = (V, E)$ in $\Theta(|V| + |E|)$ worst-case time.

Algorithm 1 Breadth First Search

```
1:  function BFS(adj, s)
2:      (d, π) ← ({ s : 0 }, { s : null })
3:      frontier ← [s]
4:      while frontier ≠ [] do
5:          next ← []
6:          for u ∈ frontier do
7:              for v ∈ adj[u] do
8:                  if v ∉ d then
9:                      (d[v], π[v]) ← (d[u] + 1, u)
10:                     APPEND(next, v)
11:                 end if
12:             end for
13:         end for
14:         frontier ← next
15:     end while
16:     return (d,π)
17: end function
```

Breadth-First Search has proven useful in a wide variety of applications, from quantum [4] to skeletons [9] to forests [2] to 3SAT in DNA [5] to AI [7], and even to graph isomorphism [6]. A common misconception in the literature is that BFS can solve the problem of finding minimum-weight (or lightest) paths on weighted graphs in linear time. Fighting against the well-known bias against publishing negative results [8][3], this paper settles this controversial issue by providing a counter-example: a family of graphs $\mathfrak{G}$ over which BFS (including its modifications) fails to find lightest paths in linear time.

Copyright © ACM [to be supplied]. . . $15.00
DOI: http://dx.doi.org/10.1145/(to come)

2. The Graph Family $\mathfrak{G}$ and BFS

$\mathfrak{G}$ is an family of simple, positively-weighted, undirected graphs with distinguished source and target vertices. Informally, a graph in $\mathfrak{G}$ is a cycle with a source-target edge, where all vertices may be replaced with another graph in $\mathfrak{G}$. Formally:

$$\mathfrak{G} = \left\{ \mathcal{G}_k^{\mathcal{H}} \middle| \mathcal{H} \in \mathfrak{G} \cup \{\mathcal{V}\}, 3 \leq k \in \mathbb{N} \right\}$$

$$\mathcal{G}_k^{\mathcal{H}} = (\mathcal{W}, \mathfrak{s}_1, \mathfrak{t}_k) \text{ given } \mathcal{H} = (\mathcal{W}_{\mathcal{H}}, \mathfrak{s}, \mathfrak{t})$$

$$\mathcal{W} = \{\{u_i, v_i\} \mapsto w \mid w = \mathcal{W}_{\mathcal{H}}(u, v), i \in [1, k]\}$$
$$\cup \{\{\mathfrak{t}_i, \mathfrak{s}_{i+1}\} \mapsto 1 \mid i \in [1, k)\}$$
$$\cup \{\{\mathfrak{t}_k, \mathfrak{s}_1\} \mapsto k + k \times \delta_{\mathcal{H}}(\mathfrak{s}, \mathfrak{t})\}$$

$$\mathcal{V} = (\emptyset, \mathfrak{v}, \mathfrak{v}), \text{ the singleton vertex } \mathfrak{v}$$

Lemma 1. *For any graph $\mathcal{G}_k^{\mathcal{H}} \in \mathfrak{G}$, $\delta(\mathfrak{s}, \mathfrak{t}) = \mathcal{W}(\mathfrak{s}, \mathfrak{t}) - 1$.*

Proof. A lightest $\mathfrak{s} \to \mathfrak{t}$ path is either just the $\{\mathfrak{s}, \mathfrak{t}\}$ edge or the path that traverses a lightest path of weight $\delta_{\mathcal{H}}(\mathfrak{s}_{\mathcal{H}}, \mathfrak{t}_{\mathcal{H}})$ within each copy of $\mathcal{H}$ as well as each unit-weight edge in-between, for a total weight of $k - 1 + k \times \delta_{\mathcal{H}}(\mathfrak{s}_{\mathcal{H}}, \mathfrak{t}_{\mathcal{H}}) = \mathcal{W}(\mathfrak{s}, \mathfrak{t}) - 1$. □

Corollary 1. *For any graph $\mathcal{G}_{k_1}^{\mathcal{G}_{k_2}^{\mathcal{G}_{k_3}^{\cdot^{\mathcal{G}_{k_r}^{\mathcal{V}}}}}} \in \mathfrak{G}$, $\mathcal{W}(\mathfrak{s}, \mathfrak{t}) = \prod_{i=1}^{r} k_i$.*

Proof. $\mathcal{W}(\mathfrak{s}, \mathfrak{t}) = k + k \times \delta_{\mathcal{H}}(\mathfrak{s}_{\mathcal{H}}, \mathfrak{t}_{\mathcal{H}})$. In the base case, $\mathfrak{s}_{\mathcal{V}} = \mathfrak{t}_{\mathcal{V}}$, so $\mathcal{W}(\mathfrak{s}, \mathfrak{t}) = k$. Inductively, by Lemma 1, we substitute to get: $\mathcal{W}(\mathfrak{s}, \mathfrak{t}) = k + k \times (\mathcal{W}_{\mathcal{H}}(\mathfrak{s}_{\mathcal{H}}, \mathfrak{t}_{\mathcal{H}}) - 1) = k \times \mathcal{W}_{\mathcal{H}}(\mathfrak{s}_{\mathcal{H}}, \mathfrak{t}_{\mathcal{H}})$. □

Lemma 2. *For any graph $\mathcal{G}_k^{\mathcal{H}} \in \mathfrak{G}$, the lightest $\mathfrak{s} \to \mathfrak{t}$ path is the longest simple $\mathfrak{s} \to \mathfrak{t}$ path, traversing every unit-weight edge.*

Proof. By Lemma 1, a lightest path never uses the $\{\mathfrak{s}, \mathfrak{t}\}$ edge of any cycle because the other edges have lower weight. By induction, it can only be the unique longest simple path. □

Theorem 1. *For any graph $\mathcal{G}_k^{\mathcal{H}} \in \mathfrak{G}$, BFS($\mathcal{G}_k^{\mathcal{H}}$, $\mathfrak{s}$) does not return the lightest $\mathfrak{s} \to \mathfrak{t}$ path.*

Proof. BFS outputs the shortest path $[\mathfrak{s}, \mathfrak{t}]$. By Lemma 1, it does not have minimum weight. □

Exercise to the reader: extend proofs to infinite-size $\mathcal{G}_k^{\mathcal{H}} \in \mathfrak{G}$.

Algorithm 2 Modified Breadth First Search (Attempt 1)

```
1:  function MBFS1(adj, s, W)
2:      d ← { s : 0 }
3:      π ← { s : null }
4:      frontier ← [s]
5:      while frontier ≠ [] do
6:          next ← []
7:          for u ∈ frontier do
8:              for v ∈ adj[u] do
9:                  if v ∉ d then
10:                     (d[v], π[v]) ← (d[u] + W(u,v), u)
11:                     APPEND(next, v)
12:                 end if
13:             end for
14:         end for
15:         frontier ← next
16:     end while
17:     return (d,π)
18: end function
```

Algorithm 3 Modified Breadth First Search (Attempt 2)

```
1:  function MBFS2(adj, s, W)
2:      d ← { s : 0 }
3:      π ← { s : null }
4:      frontier ← [s]
5:      while frontier ≠ [] do
6:          next ← []
7:          for u ∈ frontier do
8:              for v ∈ adj[u] do
9:                  if v ∉ d or d[v] > d[u] + W(u,v) then
10:                     if v ∉ d then
11:                         APPEND(next, v)
12:                     end if
13:                     (d[v], π[v]) ← (d[u] + W(u,v), u)
14:                 end if
15:             end for
16:         end for
17:         frontier ← next
18:     end while
19:     return (d,π)
20: end function
```

Algorithm 4 Modified Breadth First Search (Attempt 3)

```
1:  function MBFS3(adj, s, W)
2:      d ← { s : 0 }
3:      π ← { s : null }
4:      frontier ← { s }
5:      while frontier ≠ Ø do
6:          next ← Ø
7:          for u ∈ frontier do
8:              for v ∈ adj[u] do
9:                  if v ∉ d or d[v] > d[u] + W(u,v) then
10:                     (d[v], π[v]) ← (d[u] + W(u,v), u)
11:                     next ← next ∪ {v}
12:                 end if
13:             end for
14:         end for
15:         frontier ← next
16:     end while
17:     return (d,π)
18: end function
```

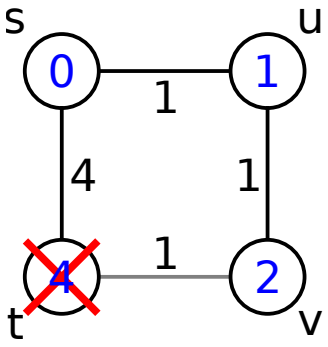

Figure 1. Counter-example Graph $\mathcal{G}_4^{\mathcal{V}}$ for MBFS1

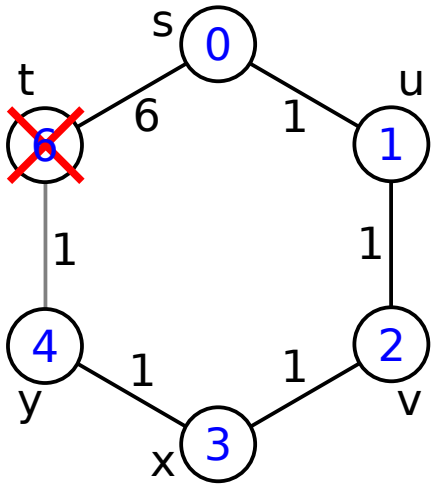

Figure 2. Counter-example Graph $\mathcal{G}_6^{\mathcal{V}}$ for MBFS2

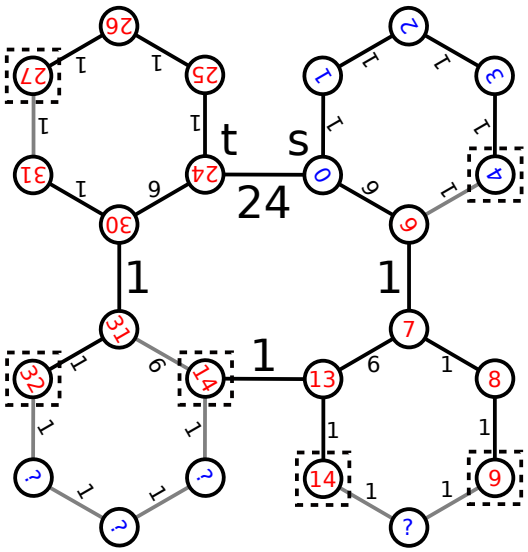

Figure 3. Example Counter-example Graph $\mathcal{G}_4^{\mathcal{G}_6^{\mathcal{V}}}$ for MBFS3, after iteration 4. Next frontier nodes are highlighted by dashed squares.

3. The Absolute Failure of Modified BFS

In previous works [1], skeptics and proponents of the conjecture alike have devised modifications to Breadth-First Search, in hopes that actually using the edge weights given as input would allow for lightest paths to be computed efficiently. This section will survey the three common approaches and demonstrate their failure with counter-example graphs from $\mathfrak{G}$.

3.1 Attempt 1 - Read Edge Weights

The obvious first attempt at a Modified BFS (hereby called MBFS1) is coded up in Algorithm 2. The insight here is that BFS assumes that every edge has unit weight in the formula: d[v] ← d[u] + 1. MBFS1 instead uses the general formula: d[v] ← d[u] + w(u,v).

Theorem 2. *For all $k \geq 4$,* MBFS1*$(\mathcal{G}_k^{\mathcal{V}}, \mathfrak{s})$ will not return the lightest $\mathfrak{s} \to \mathfrak{t}$ path.*

Proof. Consider how MBFS1 fails on the graph $\mathcal{G}_4^{\mathcal{V}}$. Figure 1 shows the computed distances; notably, the $\{\mathfrak{s}, \mathfrak{t}\}$ edge fools it into setting d[t] = 4, but $\delta(\mathfrak{s}, \mathfrak{t}) = 3$. This trivially extends to all $k > 4$. □

3.2 Attempt 2 - Allow Iterative Updates

One reason BFS and MBFS1 fail on many graphs is that once a vertex has been visited once, it is assumed to have been found along the lightest path. This observation naturally leads to MBFS2, written in Algorithm 3. MBFS2 improves upon MBFS1 by allowing using d[v] as an estimate that can be shrunk if better paths are found to neighbors in future iterations.

Theorem 3. *For all $k \geq 6$,* MBFS2*$(\mathcal{G}_k^{\mathcal{V}}, \mathfrak{s})$ will not return the lightest $\mathfrak{s} \to \mathfrak{t}$ path.*

Proof. Consider how MBFS2 fails on $\mathcal{G}_6^{\mathcal{V}}$, as shown in Figure 2. Like the previous attempt, MBFS2 is fooled by the heavy $\{\mathfrak{s}, \mathfrak{t}\}$ edge, and $\mathfrak{t}$'s neighbor y finds its optimal path from x after y was already visited, so it never extends the path back to $\mathfrak{t}$. For $k > 6$, the two sides of the frontier from $\mathfrak{s}$ will collide even further away from $\mathfrak{t}$, thus MBFS2 will still fail. □

3.3 Attempt 3 - Propagate All The Paths

The frontier collision issue is a challenge for all modifications to BFS that do not allow for repeated visiting of vertices. To avoid this problem, a third attempt MBFS3 was made, shown in Algorithm 4. The more powerful MBFS3 permits backtracking by adding updated vertices into the next frontier even if they have been visited before.

Theorem 4. MBFS3*$(\mathcal{G}_k^{\mathcal{G}_6^{\mathcal{V}}}, \mathfrak{s})$ takes $\Omega(k^2) = \omega(|V| + |E|)$ time.*

Proof. Figure 3 shows graph $\mathcal{G}_4^{\mathcal{G}_6^{\mathcal{V}}}$ to illustrate the proof. Consider how the frontier passes through the first copy of $\mathcal{G}_6^{\mathcal{V}}$. Starting at source $\mathfrak{s}_1$, $\mathfrak{t}_1$ is first visited on iteration 2 but **not** on its lightest path. $\mathfrak{t}_1$ will not be visited on its lightest path until iteration 6, when the suboptimal path through $\{\mathfrak{s}_1, \mathfrak{t}_1\}$ has propagated 4 vertices ahead.

By induction, $\mathfrak{t}_i$ will first be visited on iteration $2i$, reached along the top-most cycle's edges, but by Lemma 2 the unique lightest path is the longest path and only uses the unit-weight edges, so it is delayed from reaching $\mathfrak{t}_i$ until iteration $\delta(\mathfrak{s}_1, \mathfrak{t}_i) + 1 = 6i$.

Notice that MBFS3 will visit $\mathfrak{t}_i$ at least once for each previous $\mathfrak{t}_j$ propagating its optimal path: specifically, $\mathfrak{t}_i$ is guaranteed to be in the frontier on distinct iteration $6j + 2(i - j)$ for all $j \leq i$. Since each $\mathfrak{t}_i$ is visited $\Omega(i)$ times and each visit takes $\Theta(1)$ time as $\deg(\mathfrak{t}_i) = 3$, we can lower-bound the running time by $\Omega(k^2)$. Because the graph has $|V| = 6k$ and $|E| \leq 3|V|$, the algorithm takes $\omega(|V| + |E|)$ time, superlinear in the input size. □

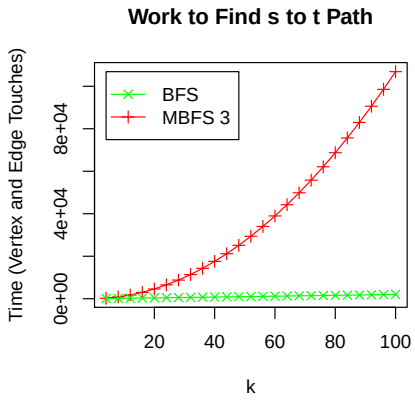

Figure 4. Just look how slow MBFS3$(\mathcal{G}_k^{\mathcal{G}_6^{\mathcal{V}}}, \mathfrak{s})$ is!

4. Future Work

While Theorem 4 shows that MBFS3 is not efficient in general, some may (or may not) ponder about its applicability on other graphs, or if MBFS3 is guaranteed to return lightest paths at all.

Open Problem 1. *Fully characterize the correctness and running time of* MBFS3 *on graphs in $\mathfrak{G}$ and beyond.*

A related algorithm, Depth-First Search, also has a history of shortest and lightest path speculation. Given the success of this paper's approach at destroying all hope of BFS being suitable for these problems, this implication seems prudent:

Conjecture 1. *Depth-First Search cannot be used or modified to find shortest or lightest paths in an arbitrary graph in linear time.*

$\mathfrak{G}$ was formulated for this paper, but the usefulness it had for proving these results about BFS may only be the tip of an iceberg.

Conjecture 2. *$\mathfrak{G}$ is awesome.*

5. Conclusion

Stop trying to modify Breadth-First Search to solve the single source shortest paths problem on weighted graphs in linear time!

References

[1] [redacted], [redacted], et al. Private Communications, 2018.

[2] L. Chaumont. Breadth first search coding of multitype forests with application to lamperti representation. In *In Memoriam Marc Yor-Séminaire de Probabilités XLVII*, pages 561–584. Springer, 2015.

[3] J. Knight. Null and void. *Nature*, 422:554–555, 2003/04/10. URL https://doi.org/10.1038/422554a.

[4] Y. Liang and D. Prendergast. Quantum many-body effects in x-ray spectra efficiently computed using a basic graph algorithm. *Phys. Rev. B*, 97:205127, May 2018. doi: 10.1103/PhysRevB.97.205127.

[5] M. Ogihara and A. Ray. *Breadth first search 3SAT algorithms for DNA computers*. Citeseer, 1996.

[6] R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977.

[7] L. Siklóssy, A. Rich, and V. Marinov. Breadth-first search: some surprising results. *Artificial Intelligence*, 4(1):1–27, 1973.

[8] H. L. L. Y. R. J. Song F, Parekh S. Dissemination and publication of research findings : an updated review of related biases. *Health Technology Assessment*, 14(8), 2010.

[9] A. Tabb and H. Medeiros. Fast and robust curve skeletonization for real-world elongated objects. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1935–1943, March 2018. doi: 10.1109/WACV.2018.00214.

[10] K. Zuse. Der plankalkül. *Konrad Zuse Internet Archive*, pages 96–105, 1972. URL http://zuse.zib.de/item/gHI1cNsUuQweHB6.