Reimplementation and Analysis of p-Stepping Algorithms for Parallel Shortest Paths

Tian Xia
School of Electrical Engineering and Computer Science (EECS)
University of Ottawa
Ottawa, Canada K1S 5M2
txia102@uottawa.ca

December 20, 2021

Abstract

SSSP is well-known graph algorithm. As for the parallel version of SSSP algorithms, the *Delta*-stepping algorithm[11] and the Radius-stepping [3] algorithm are two existing solutions for this problem, but they all have drawbacks.To solve these drawbacks author of [5] described a new stepping algorithm framework(ρ-Stepping). In this project, we will re-implement and test on ρ-Stepping, to check its efficiency and scalability.

1 Introduction

SSSP(single-source shortest path problem) is well-known graph algorithms, which has been proved can be implemented with time complexity $\mathcal{O}(n \log n + m)$ on a graph with n nodes and m edges using the well-known Dijkstra algorithm. As for the Parallel version of SSSP algorithms, the Δ-stepping algorithm[11] and the Radius-stepping[3] algorithm are two existing solutions for this problem. However, both of these 2 algorithms have their problem. The Δ-stepping algorithm[11] can not find the worst-case bound on general graphs, whereas, the Radius-stepping[3] doesn't have any efficient implementation available. So this paper[5] described a new stepping algorithm framework, and a new data structure called lazy-batched priority queue (LaB-PQ) to implement a better version of parallel SSSP algorithm with non-trivial worst-case bounds.

2 Literature Review

2.1 Sequential Algorithms

Let a directed graph with $|V| = n$ nodes and $|E| = m$ edges $G = (V, E)$, let s become a vertex ("source") of the graph. The object of SSSP is get the shortest path between any nodes v and our source node s. We denote path between node v and s as $dist(v, s)$, or abbreviated as $dist(v)$. If v is unreachable from u, we'll set $dist(v, u) = \infty$. To solve SSSP problem, plenty of methods have been created, most famous and most widely used sequential algorithms are Djkstra[4] and Bellman-Ford[2][6][12]

2.1.1 Dijkstra[4]

To find SSSP of an arbitrary directed graph with n nodes, m edges and non-negative edge weights sequentially, the algorithm maintains a priority queue(min-heap) minQ to save the unvisited nodes and their estimating shortest path $tent(v)$ as key values. It then repeatedly extracts the vertex u with the minimum $tent(u)$ from minQ and relaxes all incident edges from u to any vertex in minQ, We.e., for an $edge(v,u) \in E$ the algorithm sets $tent(v) := min\{tent(v), tent(u) + c(v,u)\}$. When a vertex is extracted from minQ and all relaxations through it are complete, the algorithm treats this vertex as visited and will not visit it again. Dijkstra's[4] algorithm either stops when minQ is empty or when every vertex has been visited. The time complexity of priority queue-based Dijkstra algorithm is $\mathcal{O}(n + n \log m)$.

2.1.2 Bellman-Ford[2][6][12]

Unlike Dijkstra's algorithm that greedily selects node u with minimum $tent(u)$ that has not yet been processed and does this for all its outgoing edges, the Bellman-Ford[2][6][12] algorithm focuses on the edges rather than the nodes, more specifically Bellman-Ford[2][6][12] just relaxes all the edges and does this $|V| - 1$ times, where $|V|$ is the number of vertices in the graph. To be more specific, it first calculates the shortest distances which have at-most one edge in the path. Then, it calculates the shortest paths with at-most 2 edges, and so on. After the ith iteration of loop, the shortest paths with at most We edges are calculated, the loop will stop after $|V|$–1 time (There are at most $|V|$–1 edges in the graph). The time complexity of Bellman-Ford[2][6][12] algorithm is relative high, it can achieve $\mathcal{O}(mn)$ time complexity, in case $m = n^2$, $\mathcal{O}(n^3)$.

2.2 Parallel Algorithms To Solve SSSP Problem

With the development of multi-core CPUs and other parallel hardware, parallel algorithms can now transfer from theory to practice. The state of art algorithms solving SSSP problem nowadays is Δ-stepping algorithm. Before it, most parallel SSSP Algorithms are based on randomized parallel breadth-first search(BFS) algorithm.

2.2.1 Algorithms Before Δ-stepping Algorithm

Plenty of parallel SSSP Algorithms are implemented using Ullman and Yannakaki's randomized parallel breadth-first search(BFS) algorithm[14]. The native implementation version of BFS algorithm performs $\mathcal{O}(\sqrt{n} \cdot log(n))$-limited searches from $\mathcal{O}(\sqrt{n})$ randomly and parallely chosen distinguished nodes.Then it creates an auxiliary graph of nodes with edge weights found from the random search. This native BFS algorithm takes $\mathcal{O}(\sqrt{n} \cdot polylog(n))$ time using $\mathcal{O}(\sqrt{n} \cdot m \cdot polylog(n))$ work with high probability. A more advanced implementation can get $\mathcal{O}(t \cdot polylog(n))$ time using $\mathcal{O}((\sqrt{n} \cdot m + n \cdot m/t + n^3/t^4) polylog(n))$ work for any $t \leq \sqrt{n}$ WHP(with high probability).

2.2.2 Δ-stepping Algorithm[11]

The Δ-stepping algorithm[11] is the state of art parallel algorithms to solve SSSP problem, which combines Dijkstra's algorithm[4] and the Bellman-Ford[2][6][12] algorithm. It calculate the shortest distances in increments of Δ, in step We all the vertices with distances in $[i\Delta, (i+1)\Delta]$ are settled down. And it runs Bellman-Ford[2][6][12] as substeps in its each step. Sequential Δ-stepping[11] can be implemented to run in time

$\mathcal{O}(n + m + L/\Delta + n\Delta + m\Delta)$. For random edge weights, A parallel version can then take $\mathcal{O}(d \cdot L \cdot logn + log^2 n)$ time and $O(n + m + d \cdot L \cdot logn)$ work on average.

2.2.3 Better Implementation of Δ-stepping Algorithm[11]

Theoretically, Δ-stepping algorithm[11] has been analyzed on random graphs, but there is no bounds on the general graphs(can only be analysed on random edge weights graphs). So in the paper[3], author provided a solution to this problem. He provided a advanced version of Δ-stepping algorithm[11] called Radius-stepping algorithm[3]. This algorithms can only run on weighted, undirected graphs. The input includes: a source vertex s, and a target radius parameter $r(\cdot)$, and a weighted, undirected graph g. The output of this algorithm is same as previous SSSP algorithms:the graph distance $\delta(\cdot)$ from s. Radius-stepping[3] is very similar to Δ-stepping algorithm[11], the only different is instead of increasing the search range by a fixed amount($[We\Delta, (We+1)\Delta]$), the radius-stepping taking $\delta(v) + r(v)$ as its new searching range . The the radius stepping algorithm[3] can solve the SSSP problem in $\mathcal{O}(m \log(n))$ work and $\mathcal{O}(\frac{n}{p} \log n \log pL)$ depth, for $p \leq \sqrt{n}$. The preprocessing phase uses $\mathcal{O}(m \log n + np^2)$ work and $\mathcal{O}(p^2)$ depth.

Both of these stepping algorithms have their own advantages, however, the Δ-stepping algorithm can not find the worst-case bound on general graphs, whereas, the Radius-stepping doesn't have any efficient implementation available. So in paper *Efficient Stepping Algorithms and Implementations for Parallel Shortest Paths*[5] the author described a new stepping algorithm framework, and a new data structure called lazy-batched priority queue (LaB-PQ) to implement a better version of parallel SSSP algorithm with non-trivial worst-case bounds. This paper have implemented and tested on several web and social network dataset including: com-orkut(OK)[15], Live-Journal(LJ)[1], Twitter(TW)[7] and Friendster(FT)[15], one web graph WebGraph(WB)[10], and two road graph[13] Road-USA(USA) and Germany(GE).

3 Problem Statement

In the paper [5], author proposed a new stepping algorithm called ρ-stepping algorithm[5]. The author has implemented and tested on several datasets as show in the Section 2. The datasets author tests all have large size of nodes (several millions), and have better performance than Δ-stepping[11] and other parallel algorithms. But, how this algorithm perform on small size graph is unknown. So, in this project, We'll figure out the performance of ρ-stepping algorithm[5] on general size graphs, and compare with other SSSP algorithms.
RQ1: Can ρ-stepping work on general graphs?
RQ2: Is ρ-stepping outperform than other algorithm?

4 Proposed Solution: Reimplementation of ρ-stepping algorithm[5]

4.1 Implementation Overview

In this project, We implemented ρ-Stepping[5], Δ-Stepping[11] and Bellman-Ford[2][6][12] algorithm. The other two algorithms (Δ-Stepping[11] and Bellman-Ford[2][6][12]) are used as a control group to judge whether my algorithm implementation has performed well. It is particularly worth noting that my Bellman-Ford[2][6][12] implementation is paralleled. In addition to these three parallel algorithms, We also implemented a sequential Dijkstra

Algorithm 1: The Stepping Algorithm Framework.

Input: A graph $G = (V, E, w)$ and a source node s.
Output: The graph distances $d(\cdot)$ from s.

```
1 δ[·] ← +∞, associate δ to a LaB-PQ Q
2 δ[s] ← 0, Q.Update(s)
3 while |Q| > 0 do
4 |   ParallelForEach u ∈ Q.Extract(ExtDist) do
5 |   |   ParallelForEach v ∈ N(u) do
6 |   |   |   if WriteMin(δ[v], δ[u] + w(u,v)) then
7 |   |   |   |   Q.Update(v)
8 |   Execute FinishCheck
9 return δ[·]
```

Figure 1: Stepping Algorithm framework

algorithm that is used as a baseline to evaluate the operational efficiency as well as the accuracy of my implemented parallel algorithm.

As for the two stepping algorithm: ρ-Stepping[5], Δ-Stepping[11]. The implementation is based on the Stepping algorithm framework. Algorithm 1 starts with associate the distance array to a queue. It then runs in steps. In each step, vertices are processed with distances within a threshold θ which is computed by ExtDist and used as the parameter of Extract. The extracted vertices will relax their neighbors using WriteMin (Line 6). If successful, we call Update on the corresponding neighbor. As shown in the Figure 1 there are 2 sub-algorithm: EXTDIST and FINISHCHECK. The main difference between ρ-Stepping[5] and Δ-Stepping[11] is they have different ExtDist and FINISHCHECK. However, FinishCheck is not necessary part of stepping algorithm, many other previous works relaxed FINISHCHECK in different ways. The author of ρ-Stepping[5] decided to remove FINISHCHECK in his implementation. So, We'll apply his method, removing FINISHCHECK in my implementation. So the main algorithm need to be coded is ExtDist.

Δ-Stepping[11] as a combination of Dijkstra and Bellman-ford[2][6][12] algorithm,Δ-Stepping[11] visits and settles all the vertices with shortest-path distances between $i\Delta$ and $(i+1)\Delta$ in step i Within each step, the algorithm runs Bellman-Ford[2][6][12] as substeps. Hence we can set θ to $i\Delta$ in substep ExtDist.

ρ-Stepping[5] ρ-Stepping[5] extracts the nearest rho-vertex in the boundary and relax their neighbors. The only step for rho-Stepping in addition to the stepping algorithm framework as shown in the Figure 1 is finding the p-th closest distance among all vertices in the frontier (the ExtDist). In original implementation, the author simply use a sampling scheme that randomly pick $\mathcal{O}(\frac{n}{\rho} + \log n)$ element, sort them and pick the ρth one.

Bellman-Ford[2] Bellman-Ford[2][6][12] is not a Stepping algorithm, but we can also use the Stepping Algorithm frameword to implement it. Since if we set $\Delta \leftarrow +\infty$ in **Δ-Stepping** algorithm, it will degenerate to a Bellman-Ford[2][6][12] Algorithm. Since now, the **Δ-Stepping** algorithm will try to relax all edges, which is same as Bellman-Ford[2][6][12]. In this case we can set θ to $+\infty$. In the real implementation, We used a large Integer(INT_MAX / 2) to represent $+\infty$.

Algorithm	EXTDIST	FINISHCHECK
BF	$\theta \leftarrow +\infty$	-
Δ-Stepping	$\theta \leftarrow We\Delta$	-
ρ-Stepping	$\theta \leftarrow \rho\, th\ smallest\ \delta[v]\ in\ Q$	-

Table 1: **SSSP Algorithms in the stepping algorithm framework**, their ExtDist and FinishCheck

4.2 Implementation Details

As shown in the Figure 's line 4 and line 5, there are two ParallelForEach keywords, which We used parallel_for in Cilk Plus to implement it. As for the customize methods: EXTRACT, WRITEMIN and EXDIST are implemented using plain c++. To be more specific, We used *sequence* from *pbbs* to store the graph data and parallel them between different threads.

5 Experimental Evaluation

5.1 Experimental setup

We run all experiments on a machine with Intel Core i7 11800H CPU with a total of 8 cores (16 hyperthreads). The system has 16GB of main memory and 24MB L3 cache on each socket. Our codes were compiled with g++ 7.5.0 using CilkPlus with -O3 flag. The testing experiment is running on Ubuntu 18.04.6 LTS (Bionic Beaver).

5.2 Datasets

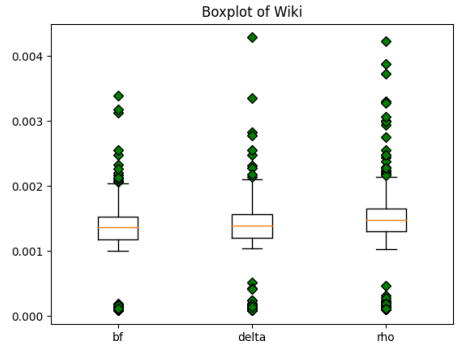
We test four graphs, including one large size networks com-orkut (OK)[15], Two median size netword Stanford[9], com-amazon[15], and One small size database WiKi-Vote[8]. The graph information is provided in Table 4. The largest one - com-orkut[15] graph has 3 millions nodes, which is used to check the efficiency of $\rho - Stepping$ Algorithm[5]. The other three smaller graph are used to check the scalability of the Algorithm.

Name	Nodes	Edges
Wiki-Vote(**WK**)[8]	7115	103689
com-amazon(**AMZ**[15])	334,863	925,872
web-stanford(**SF**[9])	281,903	2,312,497
com-orkut(**OK**[15])	3,072,441	117,185,083

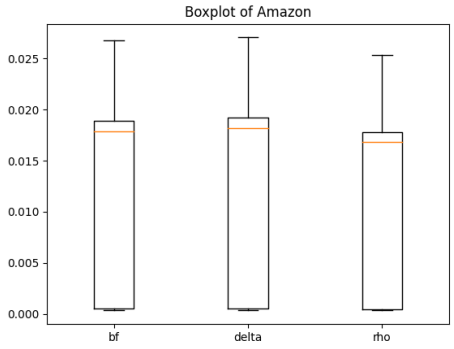
Table 2: Details of dataset

5.3 Experiment Process

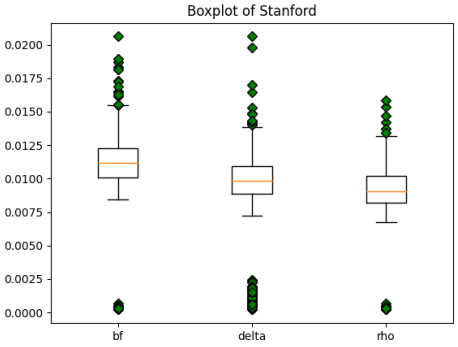
1. Testing on stepping algorithms(Δ-Stepping and ρ-Stepping[5]), and Bellman-Ford[2][6][12]
2. Use each vertex as source node to find SSSP
3. Run 10 round for each source node, and get average running time



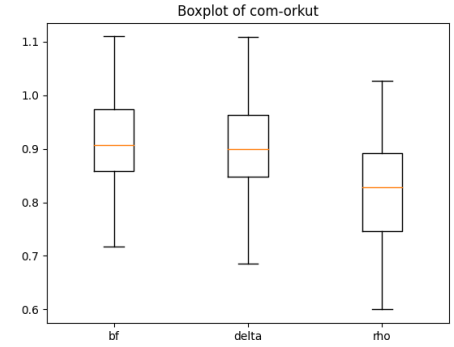
(a) **Wiki**[8]



(b) **Amazon**[15]



(c) **Stanford**[9]



(d) **com-orkut**[15]

Figure 2: Boxplot of Running time(second) of different algorithms on different dataset

4. Verified with running time of Dijkstra Algorithm

5.4 Result

We present the running time of all implementations in Tab 3. In 3 out of 4 cases, the ρ-Stepping Algorithm get best performance. We also show a boxplot of running time in Fig 2, and a barplot(Figure 3) of relative parallel running time to our baseline(Dijkstra Algorithm).

	com-orkut(OK)[15]	Wiki-Vote[8]	web-stanford[9]	Amazon[15]
$\Delta - Stepping$	0.90333033	0.00126664	0.0094191	0.012360241
$\rho - Stepping$	0.82097256	0.00135531	0.0086923	0.011378397
Bellman-Ford[2][6][12]	0.91280997	0.00124240	0.0105609	0.012110078
Dijkstra	5.220662	0.00137284	0.0687417	0.054652069

Table 3: Running time(second) of different algorithms on different dataset, red is the slowest, green is the fastest.

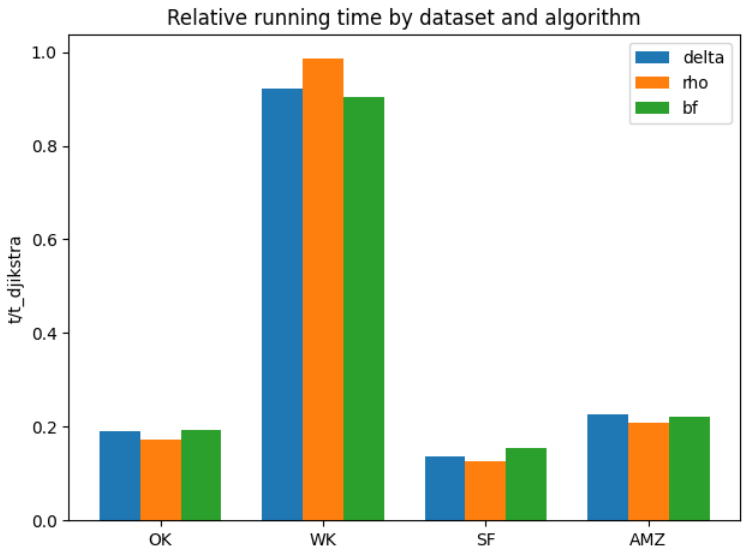


Figure 3: Relative running time of different algorithms: The value is calculate by equation t/t_{djk}.t is running time of algorithms we implemented, t_{djk} is running time of our baseline algorithm Dijkstra. ρ-Stepping algorithm[5] get the best performance except on WK dataset.

5.5 Evaluation

According to the Figure 2 and 3, we can find that ρ-Stepping[5] have the best perform on most dataset except Wiki[8]. All parallel algorithm have dramatically outperform than the baseline(Dijkstra Algorithm). The larger amount of data, the more significant the improvement. In the largest data sets(OK), parallel algorithms can be up to 5 times faster than baseline.ρ-Stepping[5] is the fastest of these parallel algorithms in most cases.

The only exception, Wiki-Vote[8], it was found that the poor performance was indeed not due to unsuitable parameter(ρ). As shown in the Figure 4 , $\rho = 2e7$ achieves the best performance among different parameters, but in this case ρ-Stepping[5] is still the worst-performing parallel algorithm. One possible explanation is that because the amount of data in Wiki[8] is too small, the extra running time generated by the parallel algorithm is greater than the time saved by the it.

6 Conclusions

In this project, we re-implemented and evaluated the ρ-Stepping Algorithm on datasets with different size. Our experiment proves that it does perform well on a large number of data sets. We also gives our own guess about ρ-Stepping Algorithm's poor performance on smaller datasets.

For the future reasearch, ρ-Stepping Algorithm[5] using plain array to store frontiers. As parallel version array may further improve the performance of ρ-Stepping Algorithm[5]. Another direction that can be investigated is that the author of ρ-Stepping Algorithm[5] proposed a framework in order to implement the algorithm, with which it has implemented only a few algorithms, and there are many stepping algorithms that have not been tested,

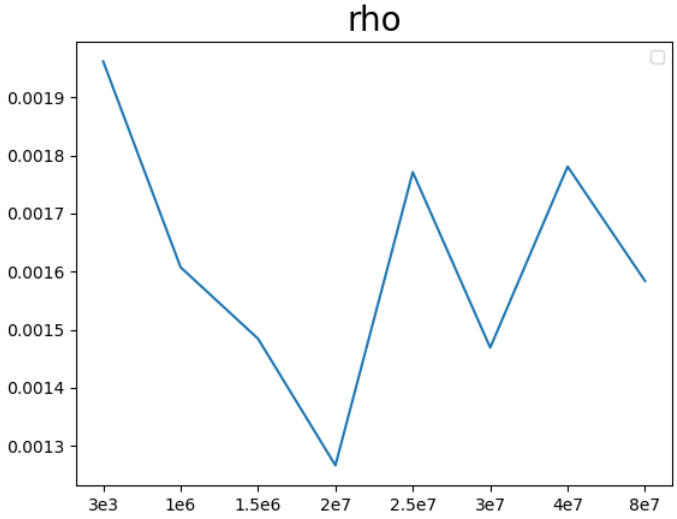


Figure 4: Running time of ρ-Stepping Algorithm[5] on WK dataset with different ρ

such as the radius-stepping algorithm[3]. Another research direction could be to use the framework to implement other stepping algorithms to further verify the correctness of the framework.

References

[1] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 44–54, 2006.

[2] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[3] Guy E Blelloch, Yan Gu, Yihan Sun, and Kanat Tangwongsan. Parallel shortest paths using radius stepping. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 443–454, 2016.

[4] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[5] Xiaojun Dong, Yan Gu, Yihan Sun, and Yunming Zhang. Efficient stepping algorithms and implementations for parallel shortest paths. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 184–197, New York, NY, USA, 2021. Association for Computing Machinery.

[6] Lester R Ford Jr. Network flow theory. Technical report, Rand Corp Santa Monica Ca, 1956.

[7] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600, 2010.

[8] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1361–1370, 2010.

[9] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[10] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. Web data commons-hyperlink graphs. *URL http://webdatacommons. org/hyperlinkgraph/index. html*, 2012.

[11] Ulrich Meyer and Peter Sanders. δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003.

[12] E.F. Moore. *The Shortest Path Through a Maze*. Bell Telephone System. Technical publications. monograph. Bell Telephone System., 1959.

[13] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . URL https://www.openstreetmap.org , 2017.

[14] Jeffrey D Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, 1991.

[15] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.