Efficient Stepping Algorithms and Implementations for Parallel Shortest Paths

Xiaojun Dong, Yan Gu, Yihan Sun, Yunming Zhang
UC Riverside, MIT

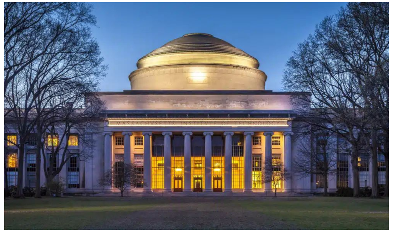
Published July 2021

Authors

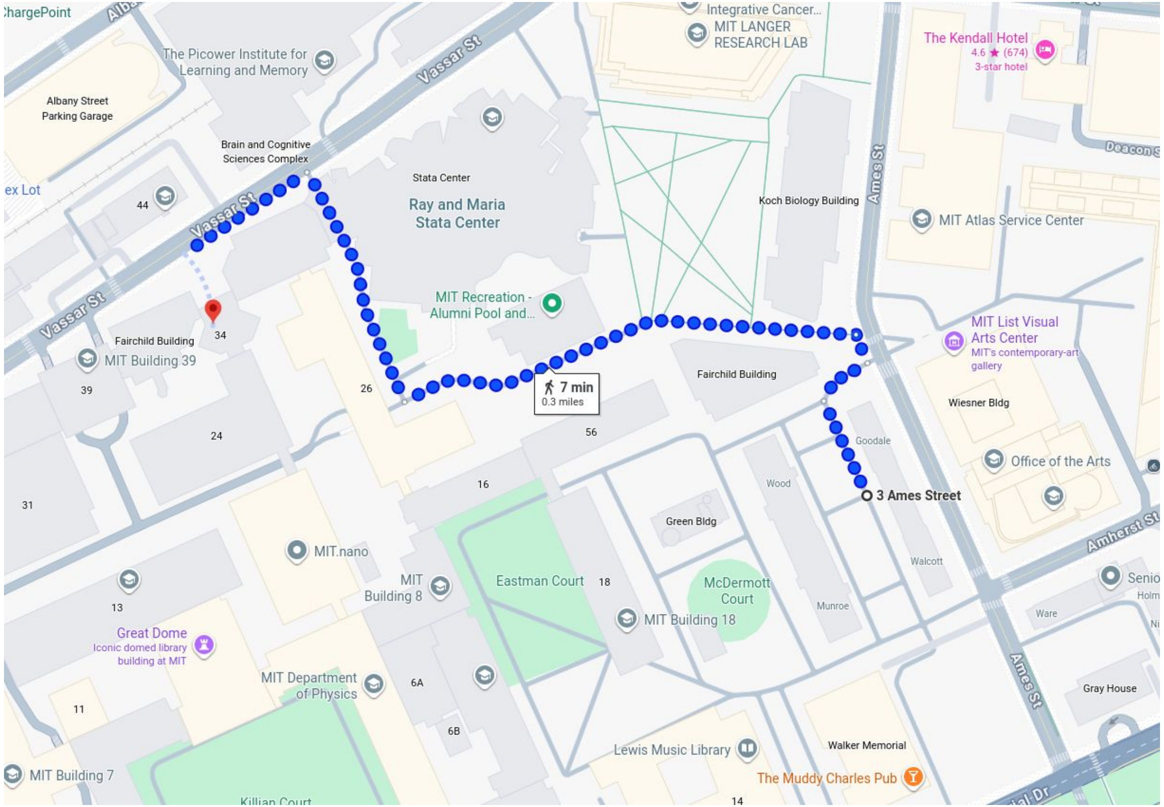
Xiaojun Dong — *Yan Gu* — *Yihan Sun* — *Yunming Zhang*



advises

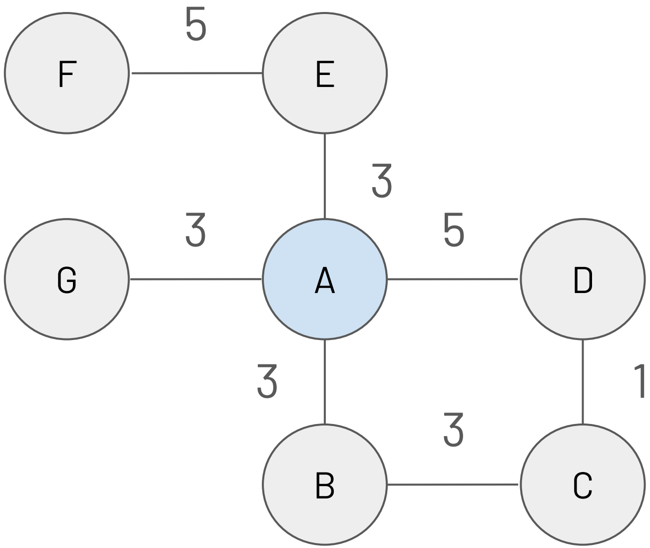


Practical Motivation



Recall: SSSP: Single-Source Shortest Path

A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
0	3	∞	5	3	∞	3
0	3	6	5	3	8	3



Making this Parallel…?

Delta-Stepping Idea: Explore some fraction (delta) of the frontier

Dijkstra's

Delta-Stepping

Bellman-Ford

Sequential → *Parallel*

More Efficient (Less Work) ← *Less Efficient (More Work)*

The Two Contenders...

	Delta Stepping	Radius Stepping
Theory	- No provable bounds (performance highly dependent on delta)	+ Provable bounds
Practice	+ Works well in practice	- No implementation / not as good as delta stepping

How can we get both provable bounds and good practical performance?

Other Related Works

Theoretical Provable Bounds

Radius Stepping

Practical Implementations

Delta Stepping

Parallel Priority Queues

(parallel queue in Dijkstra – "no interesting results" for work/span bounds and not better than delta-stepping in practice)

Previously studied independently

This Paper: Combine these fields of research

This Paper's Contribution

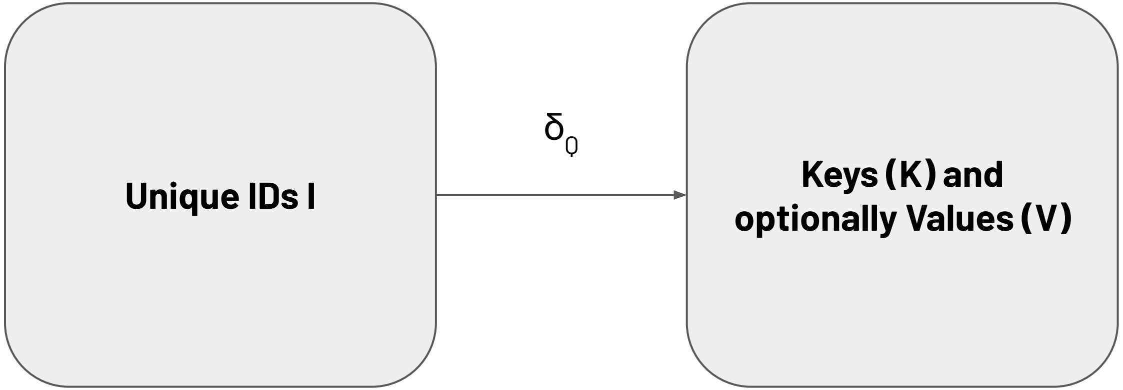
Lazy-Batched Priority Queue (LaB-PQ) → *efficient abstract data type implementation which extracts the semantics of the priority queue needed by stepping algorithms*

Stepping Algorithm Framework → *abstracts general ideas in existing parallel SSSP algorithms*

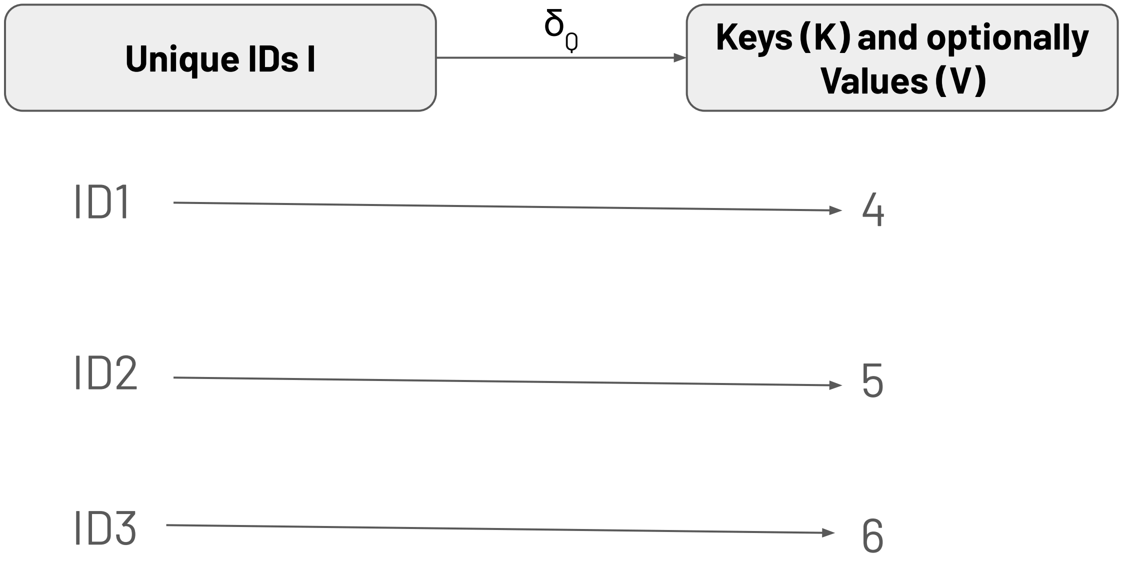
Rho-Stepping and Delta-Star-Stepping → *two new stepping algorithms which are efficient in theory and practice*

Contribution 1/3: Lazy Batched Priority Queue (LaB-PQ)

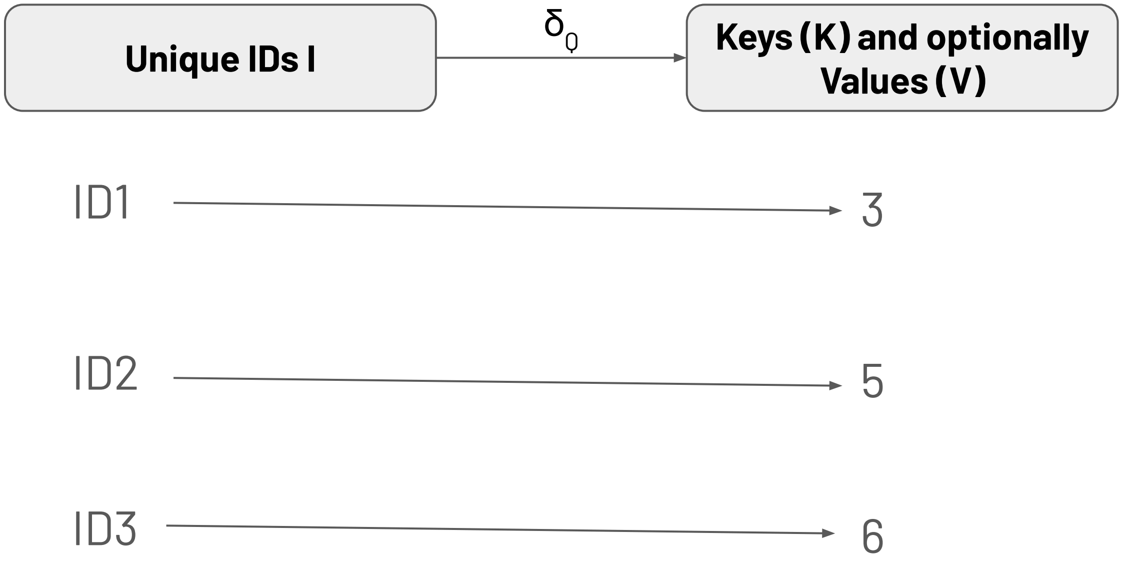
Big Idea: Map with Update Functions and Deletion



Big Idea: Map with Update Functions and Deletion

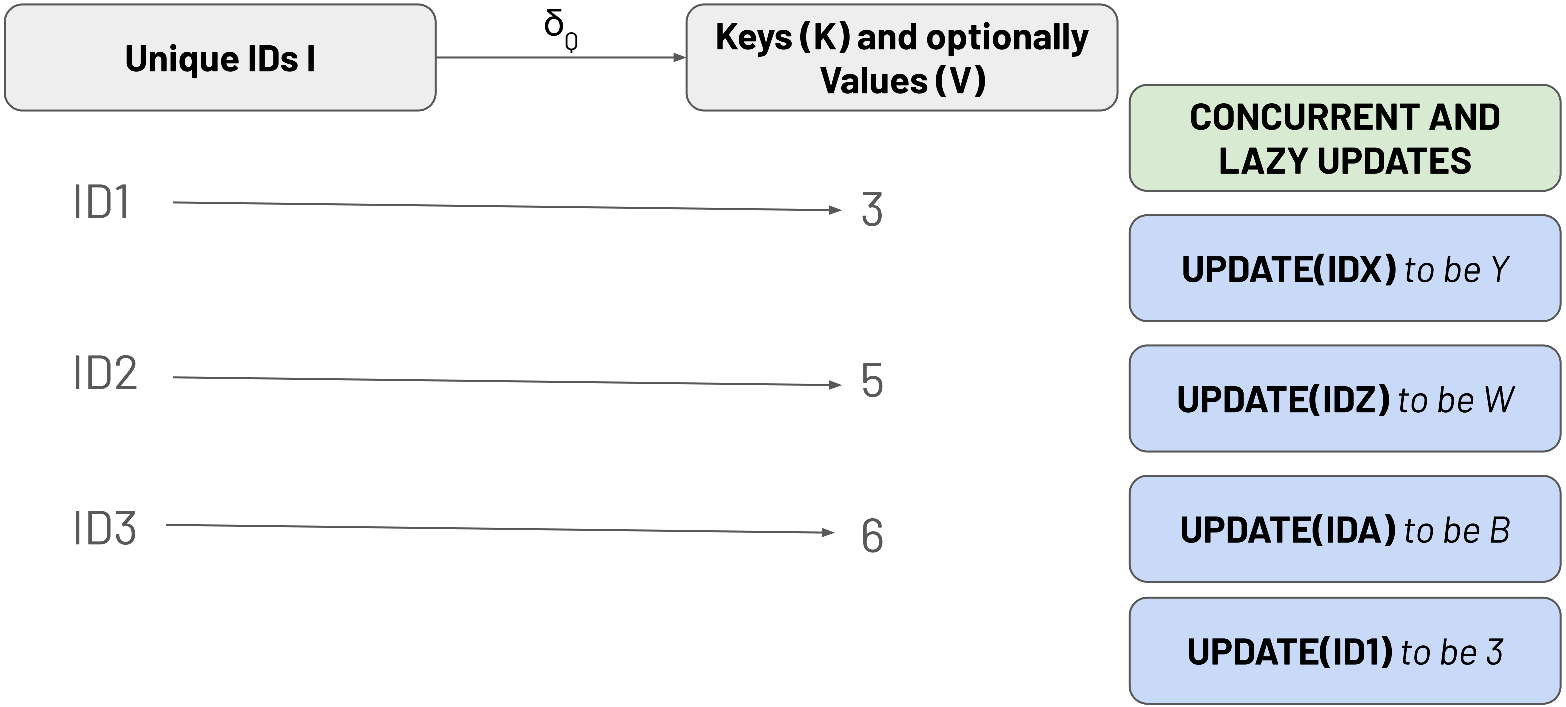


Big Idea: Map with Update Functions and Deletion

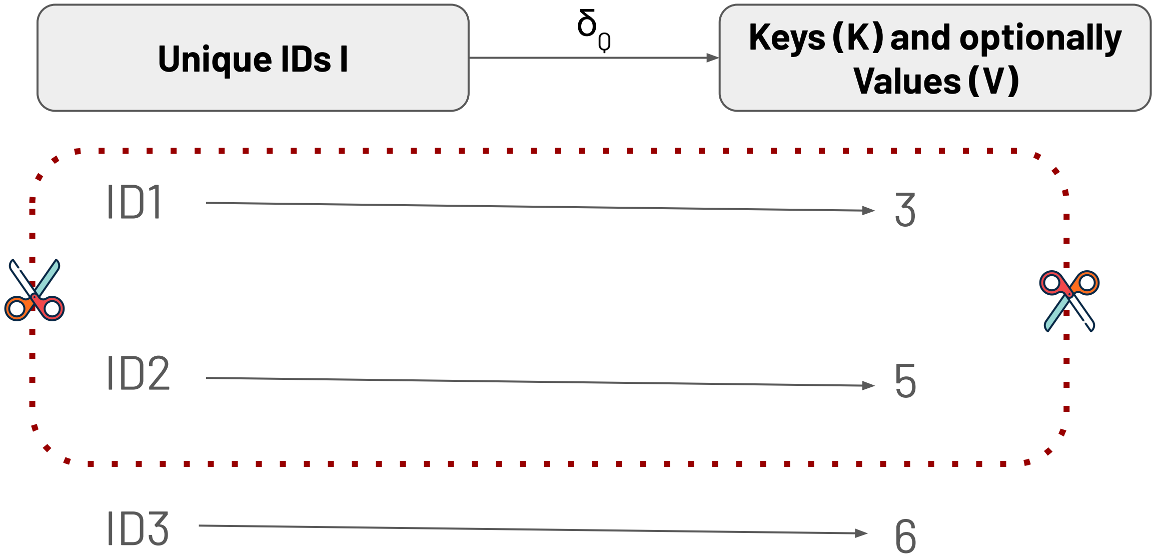


UPDATE(ID1) *to be 3*

Big Idea: Map with Update Functions and Deletion

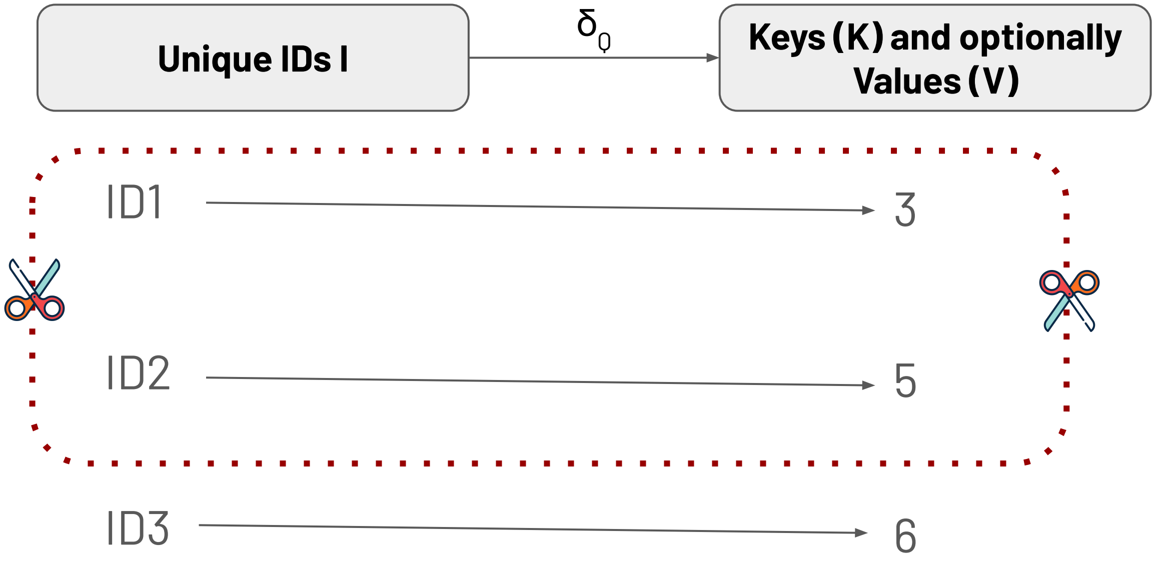


Big Idea: Map with Update Functions and Deletion



EXTRACT(5)

Big Idea: Map with Update Functions and Deletion



CANNOT run concurrently

EXTRACT(5)

LaB-PQ

Assume existing functions (atomic, unit-cost)

WRITE_MIN(p, v)
*if (v < *p) {*p = v;}*

TEST_AND_SET(p)
Reads and attempts to set the boolean value pointed to by p to true. Returns true is successful and false otherwise

id	$\in$	I	unique id type
k	$\in$	K	key type
v	$\in$	V	value type (optional)
$<_Q$	$\in$	$K \times K \rightarrow Bool$	comparison function across K
δ_Q	$\in$	$I \rightarrow K \times V$	mapping from id to key (and optionally value)

- **UPDATE(id)** → commits an update to Q regarding the record with identifier id. Can be updated lazily. CAN be executed concurrently with other UPDATES.
- **EXTRACT(theta)** → returns all IDs in Q with key <= theta and deletes them from the Q. Reflects all previous modifications to Q, including Update functions and deletions from the previous Extract. CANNOT be executed concurrently with other functions
- **Q.REDUCE()** → maps each record in Q to a value of type A, uses a binary commutative and associative operator ⊕ to compute abstract sum of all records in Q using ⊕

LaB-PQ

				Encoding as a graph for SSSP
id	$\in$	I	unique id type	Node id
k	$\in$	K	key type	Shortest Path Weight
v	$\in$	V	value type (optional)	-
$<_Q$	$\in$	$K \times K \rightarrow Bool$	comparison function across K	Weight comparison
δ_Q	$\in$	$I \rightarrow K \times V$	mapping from id to key (and optionally value)	ID to weight

Contribution 2/3: Stepping Algorithm Framework

User-Defined Functions: ***EXT_DIST*** *and* ***FINISH_CHECK***

Stepping Algorithm

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.

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

Initialize all vertices to ∞ and the source to 0

Extract vertices to process

Relax the neighbors

Some post-processing / substep

Contribution 3/3: Rho-Stepping and Delta-Star-Stepping

Defining Algorithms in this Framework

Algorithm	ExtDist	FinishCheck	Work	Span
Dijkstra [46]	$\theta \leftarrow \min_{v\in Q}(\delta[v])$	-	$\tilde{O}(m)$	$\tilde{O}(n)$
Bellman-Ford [11, 50]	$\theta \leftarrow +\infty$	-	$\tilde{O}(k_n m)$	$\tilde{O}(k_n)$
Δ-Stepping [68]	$\theta \leftarrow i\Delta$	if no new $\delta[v] < i\Delta$, $i \leftarrow i+1$	-	-
Δ^*-Stepping (new)	$\theta \leftarrow i\Delta$	-	$\tilde{O}(k_n m)$	$\tilde{O}\left(\frac{k_n(\Delta+L)}{\Delta}\right)$
Radius-Stepping [24]	$\theta \leftarrow \min_{v\in Q}(\delta[v] + r_\rho(v))$	if there exists $\delta[v] < \theta$, do not recompute ExtDist	$\tilde{O}(k_\rho m)$	$\tilde{O}\left(\frac{k_\rho n}{\rho} \cdot \log L\right)$
ρ-Stepping (new)	$\theta \leftarrow \rho$-th smallest $\delta[v]$ in Q	-	$\tilde{O}(k_n m)$	$\tilde{O}\left(\frac{k_\rho n}{\rho}\right)$ (undirected)

Table 2: SSSP Algorithms in the stepping algorithm framework, their ExtDist and FinishCheck, and the work and span bounds based on the LaB-PQ implementation in Sec. 4. Here L is the longest edge in the graph (assuming the shortest has length 1). ρ, k_ρ and k_n are related to (k,ρ)-graph defined in Sec. 2. $\tilde{O}()$ omits $\log n$ and lower-order terms for simplicity, and the full bounds are shown in Tab. 3.

Theory + Implementation Details

Implementations

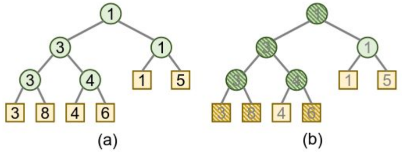


Figure 4: A tournament tree. Square leaf nodes store the records and round interior nodes keep the smallest key in their subtrees. (a) is a tournament tree containing 6 records 3, 8, 4, 6, 1 and 5. (b) shows an update on a batch of 3, 8 and 6. The shaded nodes are marked as *renewed.*

	Tournament Tree	Array
Practice	- Less efficient in practice (large memory footprint, random accesses)	+ Better in practice
Theory	+ Has a tighter work bound	- Less tight work bound
Work B = batch of UPDATE between extract	Modification: O(n log (n/b)) Extraction: O(b log(n/b))	Modification: O(b) Extraction: O(n)
Span	Extract/Update: O(log n)	Extract/Update: O(log n)

Theoretical Guarantees

General Step Results:

Step Bound: In a stepping algorithm, a vertex v will not be extracted more than k_n times

Work Bound: S steps and U updates: for tournament-tree-based implementation, total work is O(U log(n S / U))

Algorithm	Work		Span	Previous Best	
	Tournament-tree-based	Array-based		Work	Span
Dijkstra [28, 46]	$O\left(m \log \frac{n^2}{m}\right)$	$O(m+n^2)$	$O(n \log n)$	$O(m \log n)$	same
Bellman-Ford [11, 50]	$O(k_n m)$	$O(k_n m)$	$O(k_n \log n)$	same	same
Δ^*-Stepping	$O\left(k_n m \log \frac{nL}{m\Delta}\right)$	$O\left(k_n m + \frac{k_n n(\Delta+L)}{\Delta}\right)$	$O\left(\left(\frac{k_n(\Delta+L)}{\Delta}\right) \log n\right)$	-	-
Radius-Stepping[†] [24]	$O\left(k_\rho m \log \frac{n^2 \log \rho L}{m\rho}\right)$ (U)	$O\left(k_\rho m + \frac{k_\rho n^2}{\rho} \cdot \log \rho L\right)$ (U)	$O\left(\frac{k_\rho n}{\rho} \cdot \log \rho L \log n\right)$ (U)	$O(k_\rho m \log n)$ (U)	same
Shi-Spencer[†] [77]	$O\left((m+n\rho) \log \frac{n^2}{m+n\rho}\right)$ (U)	$O\left(m+n\rho+\frac{n^2}{\rho}\right)$ (U)	$O\left(\frac{n \log n}{\rho}\right)$ (U)	$O((m+n\rho) \log n)$ (U)	same
ρ-Stepping	$O\left(k_n m \log \frac{n^2}{m\rho}\right)$	$O\left(k_n m + \frac{n^2 k_\rho}{\rho}\right)$ (U) $O\left(k_n m + \frac{n^2 k_n}{\rho}\right)$	$O\left(\frac{k_\rho n \log n}{\rho}\right)$ (U) $O\left(\frac{k_n n \log n}{\rho}\right)$	-	-

Table 3: New work and span bounds for the stepping algorithms and comparison to previous results. (U) indicates the bound only works for undirected graphs. (-) indicates no non-trivial bound is known to the best of our knowledge. (same) indicates the previous bound matches the tournament-tree-based work or the span. All new work bounds for Δ^*-Stepping, Radius-Stepping, Shi-Spencer, and ρ-Stepping are based on the distribution lemma (Lem. 5.2) and the LAB-PQ bounds. Radius-Stepping and Shi-Spencer (noted with †) require preprocessing.

Tricks

- Sparse-dense optimization
- Queue size estimation and scattering
- Bidirectional relaxation for undirected graphs
- Threshold estimation for Rho-Stepping
- Large neighbor sets

Theoretical Guarantees

- General Step Bound
  - In a stepping algorithm, a vertex v will not be extracted more than k_n times

- Work Bound
  - S steps and U updates: for tournament-tree-based implementation, total work is O(U log(n S / U))
  - Delta-Star-Stepping: $O(k_n m\log(nL/(m\ Delta)))$ work and $O(k_n(Delta+L)/Delta\ logn)$ span

- Step Bound
  - $(k_{rho}$, rho)-graph (directed): rho-stepping algorithm finishes in $O(k_n n$ / rho)
  - Number of steps on undirected graph: $O(k_{rho} n$ / rho)
  - Delta-Star-Stepping uses $O(k_n(delta+L)$ / delta) steps

Results

Experiment Setup

- Quad-socket machine with Intel Xeon Gold 6252 CPUs with 96 cores (192 hyperthreads)
  - 1.5 TB of main memory, 36MB L3 on each socket
  - G++ and used CilkPlus

- Graphs Used:
  - 4 social networks com-orkut, Live-Journal, Twitter, Friendster
  - 1 web graph WebGraph
  - 2 road graphs RoadUSA and Germany

- Tuned for the best Delta

Running Time Results

	Graph	Social												Web			Road					
		OK			LJ (D)			TW (D)			FT			WB (D)			GE			USA		
	#vertices	3M			4M			42M			65M			89M			12M			24M		
	#edges	234M			68M			1.47B			3.61B			2.04B			32M			58M		
	#threads	(1)	(96h)	(SU)	(1)	(96h)	(SU)	(1)	(96h)	(SU)	(1)	(96h)	(SU)	(1)	(96h)	(SU)	(1)	(96h)	(SU)	(1)	(96h)	(SU)
Δ-step.	GAPBS	3.42	.240	14.2	1.14	.103	11.0	58.6	2.42	24.2	84.7	2.95	28.7	50.8	1.92	26.5	2.01	0.22	9.1	1.83	0.33	5.5
	Julienne[1]	4.82	.268	18.0	2.86	.140	20.4	43.1	1.82	23.7	95.4	2.75	34.7	86.1	2.04	42.2	1.54	6.62	0.2	2.04	10.16	0.2
	Galois	3.08	.194	15.9	1.72	.113	15.1	29.7	1.23	24.2	92.2	2.76	33.4	45.0	1.45	31.1	2.80	0.22	12.8	2.72	0.29	9.3
	PQ-Δ	3.45	**.123**	28.1	2.04	.082	25.0	39.3	1.07	36.9	115.4	2.55	45.3	62.8	1.27	49.6	5.54	**0.18**	30.7	4.81	**0.26**	18.8
BF	Ligra	5.07	.248	20.5	2.55	.115	22.1	42.6	1.55	27.5	218.2	5.12	42.6	81.4	2.13	38.2	-	-	-	-	-	-
	**PQ-BF*	3.71	.134	27.7	2.58	.095	27.2	45.7	1.18	38.6	147.7	2.72	54.4	97.6	1.71	57.2	12.97	0.30	42.6	16.28	0.41	39.8
ρ-step.	**PQ-ρ-fix*	3.56	.132	27.0	2.46	.087	28.2	37.6	**0.93**	40.6	112.7	**2.02**	55.8	60.6	1.07	56.7	6.43	0.21	31.1	3.84	0.30	12.7
	**PQ-ρ-best*	3.42	.125	27.5	2.07	**.080**	28.6	37.6	**0.93**	40.6	112.7	**2.02**	55.8	57.5	**1.06**	54.1	6.43	0.21	31.1	3.86	0.30	12.8
		($\rho = 2^{19}$)			($\rho = 2^{19}$)			($\rho = 2^{21}$)			($\rho = 2^{21}$)			($\rho = 2^{22}$)			($\rho = 2^{21}$)			($\rho = 2^{23}$)		

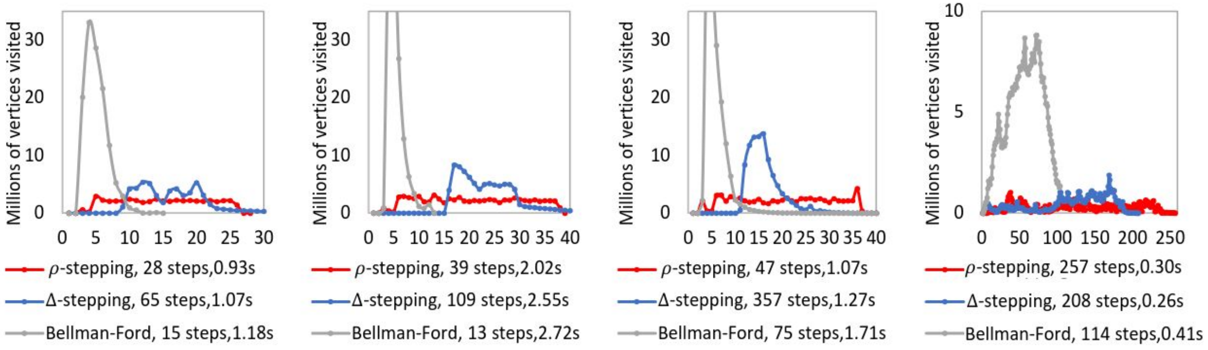
Table 4:
(1): runni
fastest ru
instance.
across all
[1]: Julier
optimized

Notice:

- One of their implementations is always better
- Implementations in their framework is always better
- PQ-rho is better on social and PQ-delta-star is better on road
    - *Roads have a smaller frontier, so PQ-rho loses some performance*

Performance Analysis

Notice: PQ-rho and PQ-delta-star do less work



(a). TW (b). FT (c). WB (d). USA

Figure 5: Number of visited vertices in each step in *PQ-ρ*, *PQ-Δ and *PQ-BF*.** Here we only run on one source vertex, since it has unclear meaning to compute the average of multiple runs on each step. Hence, the runtimes can be different from Table 4 (average on 100 runs from 10 source vertices), and some curves are bumpy. We use 96 cores (192 hyperthreads).

k_{rho} hops to reach rho neighbors for any vertex

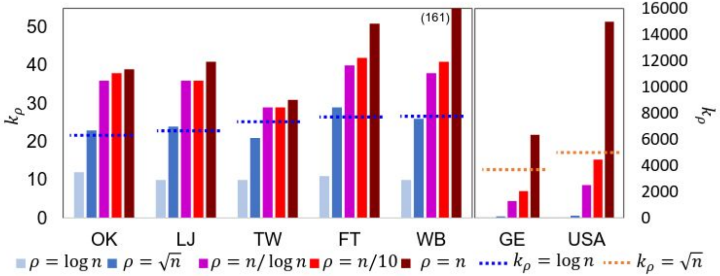


Figure 6: The values of k_ρ with different values of ρ for different graphs.

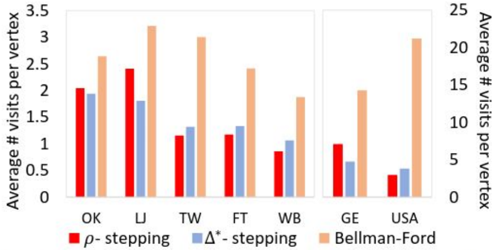


Figure 7: Number of visits per vertex and per edge, respectively, for *PQ-ρ*, *PQ-Δ and *PQ-BF* on all graphs.**

For smaller graphs, rho-stepping is similar to BF to maximize parallelism

Strengths/Weaknesses/Future Directions

- Strengths
  - Developed new ADT with a generalized framework for SSSP that has promising results
  - Proposed a theoretical bound for SSSP algorithms using their framework
- Weakness
  - Delta is still highly tunable and probably still greatly affects the performance
- Future Directions
  - Do certain smaller graphs work better with the tournament tree implementation?

Discussion Questions

- Where might LaB-PQ be used other than SSSP?
  - It essentially is a map that can do concurrent updates and non-concurrent remove filters.
- How important are theoretical bounds vs. practicality?
  - If this was yet another paper that had provable bounds but didn't beat delta-stepping (which, to be honest, their best implementation is basically just delta stepping), would it still be interesting/useful? For what scenarios might having these bounds be useful?
- The total code isn't huge (only a few hundred lines of C++) → Could this get adopted into real-world use cases? If so, where?
  - Semi-Related: Should we update our benchmarks? For real-world evaluation, no one really uses the social networks they used (Live-Journal, Friendster, com-orkut) (I mean, people still use Twitter but it's X now...)