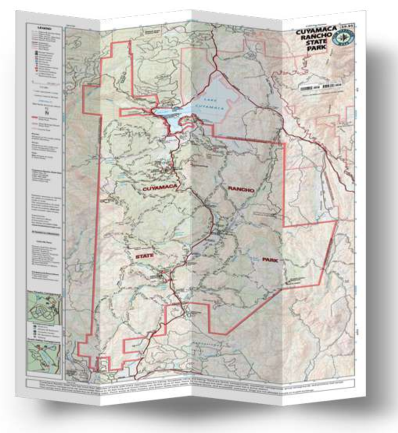
Wave : A Substrate for Distributed Incremental Graph Processing on Commodity Cluster

Swapnil Gandhi, Indian Institute of Science

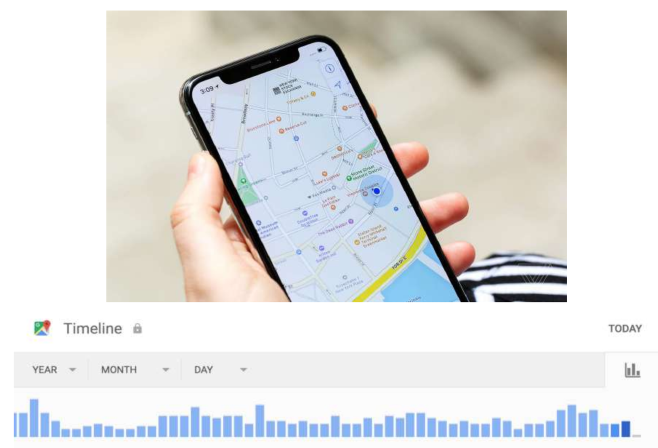
Introduction

- Graphs are widely considered to be natural means of representation for many networks.
- Real-world networks are often evolving with links being added or removed and properties updated over time.

Examples : Social, Citation/Collaboration, Sensor, Financial & Transit Network, Human Connectomes, Internet-of-Things …



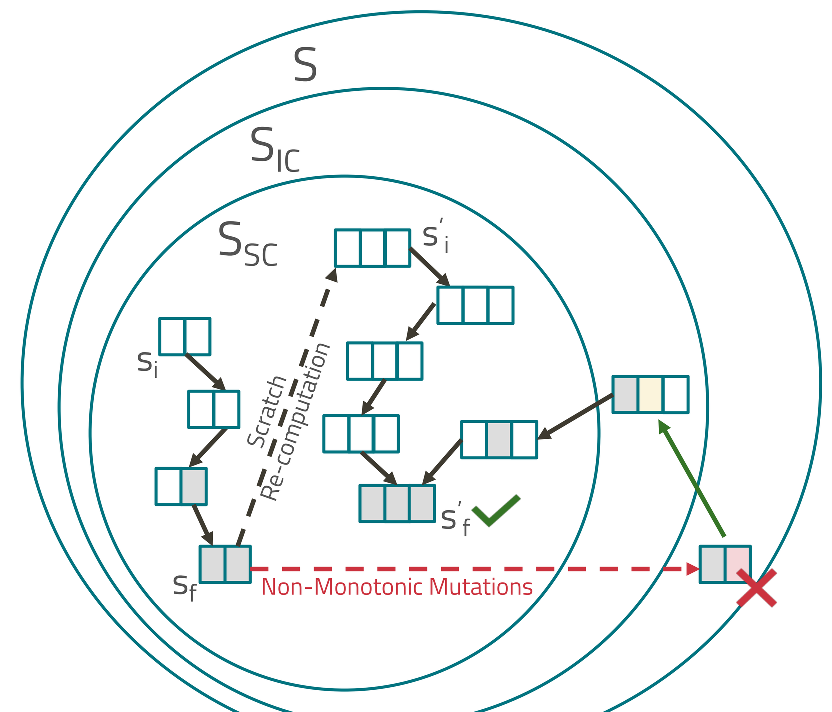
(a) Paper Map



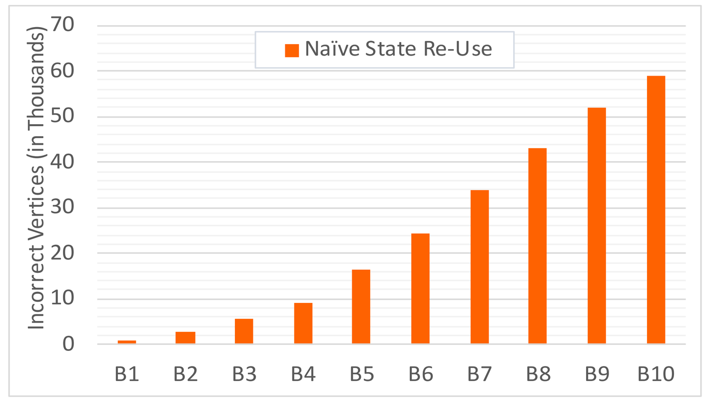
(b) Google Maps

- Incremental computation is used to process such dynamic graphs.

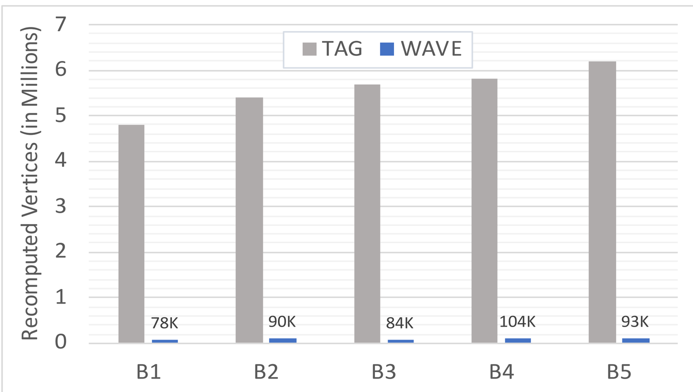
It achieves efficiency by minimizing redundant computation and communication compared to complete, from-scratch computation.



(c) Incremental processing of dynamic graphs



(d) #Vertices with incorrect results (WK)



(e) #Vertices recomputed (WK)

- Challenges:

1. *Safety:* Reusing intermediate state naively leads to incorrect results.

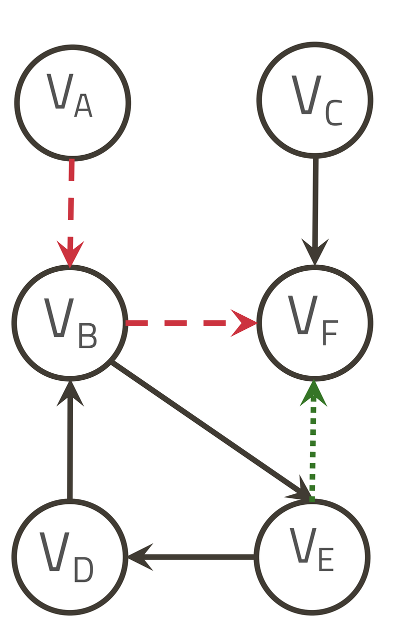
2. *Profitability:* Conservative tag-propagation guarantees safety, however limits state reuse ➲ ends up resetting majority of vertices.

3. Lack of unifying abstraction to operate on dynamic graphs.

Existing abstractions either work for a sub-class of algorithms or lack support for non-monotonic updates. Specialized algorithms are designed for single-threaded shared-memory execution.

This Work : General-purpose distributed programming model to support incremental processing over dynamic graphs while leveraging existing vertex-centric semantics.

Connected Components



(f) Dynamic graph

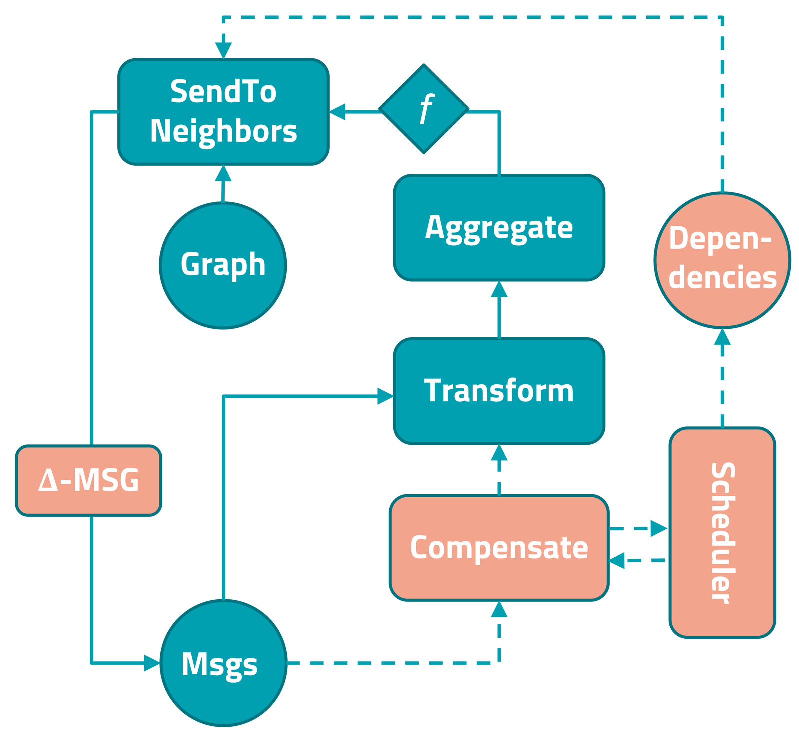
	V_A	V_B	V_C	V_D	V_E	V_F	
Before Deletes	A	A	C	A	A	A	1
	$V_A \rightarrow V_B$ and $V_B \rightarrow V_F$ Deleted						
	A	**B**	C	A	A	**F**	2
	A	B	C	A	**B**	F	3
	A	B	C	**B**	B	**C**	4
After Deletes	A	B	C	B	B	C	5
	$V_E \rightarrow V_F$ Inserted						
After Insert	A	B	C	B	B	**B**	6

Supersteps

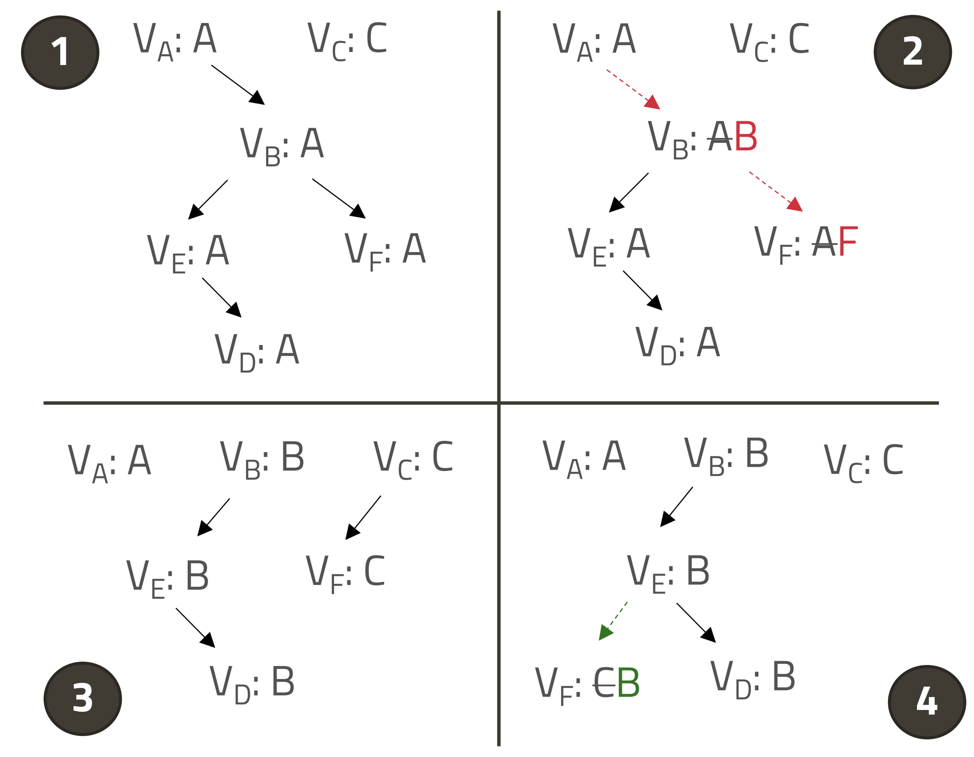
(g) Vertex state updates over supersteps when finding Connected Components. Changes shown in **red**.

Incremental Graph Computation

- *Transparency*: We do not require the vertex program to be made incremental, but instead incrementalize its boundary.
  - Makes the approach applicable to all existing vertex programs
  - Requires minimal additional effort from the programmer to devise an incremental algorithm
- Framework identifies vertices directly and transitively affected by graph mutations.
  - Only parts of the graph affected by input changes are re-computed
  - Graph structure used to actively deduce value dependencies



Incremental Vertex Program in Wave



Dependence Tree before/after mutation

```
void compensate(vertex v, Message[] msgs,
     boolean reScheduledCompute) {
   Message[] cMsgs;
   for(Message msg : msgs) {
         if(msg.getType() ==
              MessageType.RETRACT) {
             retract(vertex v, msg);
         } else {
             cMsgs.append(msg);
         } v.dequeueDeferredMsg(msg.getSRC());
   }
   for(Message deferredMsg :
        v.getDeferredMsgs()) {
      if(deferredMsg.getSuperstepTillDeferred()
           == getCurrentSuperstep()) {
         cMsgs.append(deferedMsg);
         v.dequeueDeferredMsg(deferredMsg);
      }
   }
   if(isVertexStateAffected()) {
      deferMsgs(v, cMsgs);
      if(isInvertible() || reScheduledCompute){
         compute(v, cMsgs);
      } else {
         gatherStateFromInNeighbors(v);
         rescheduleCompute();
      }
   }
}
```

Master Program

Vertex Program

```
void init(vertex v) {
   v.setState(v.getId());
   v.sendToAllNeighbors(v.getState());
}

void compute(vertex v, Message[] msgs) {
   minCC = ∞;
   for(Message msg : msgs) {
      minCC  = min(msg, minCC); }
   if(minCC < v.getState()) {
      v.setState(minCC);
      v.sendToAllNeighbors(v.getState());
   }
}
```

Interaction

```
void retract() {
   if(v.getState().getSRC()==msg.getSRC()) {
      init(); }
}

Message repropogate(vertex v, int dst) {
   reutrn new Message(v.getState());
}
```

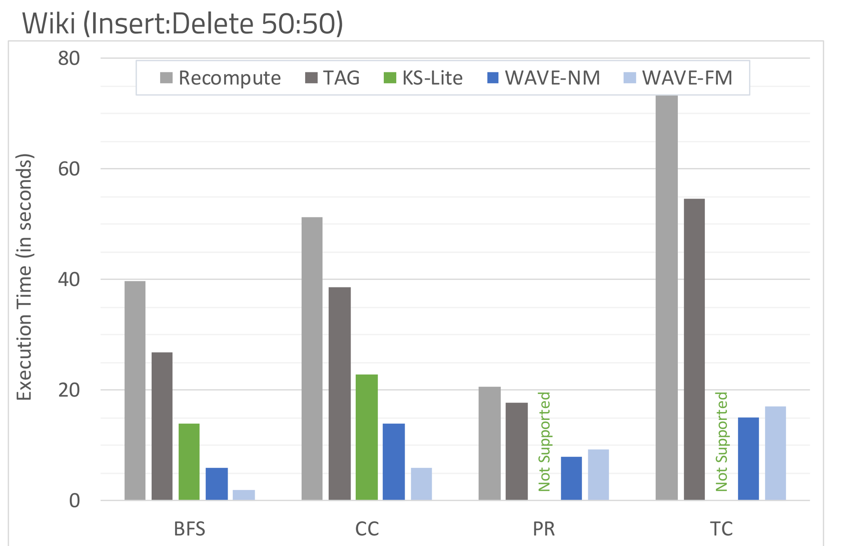
User Program

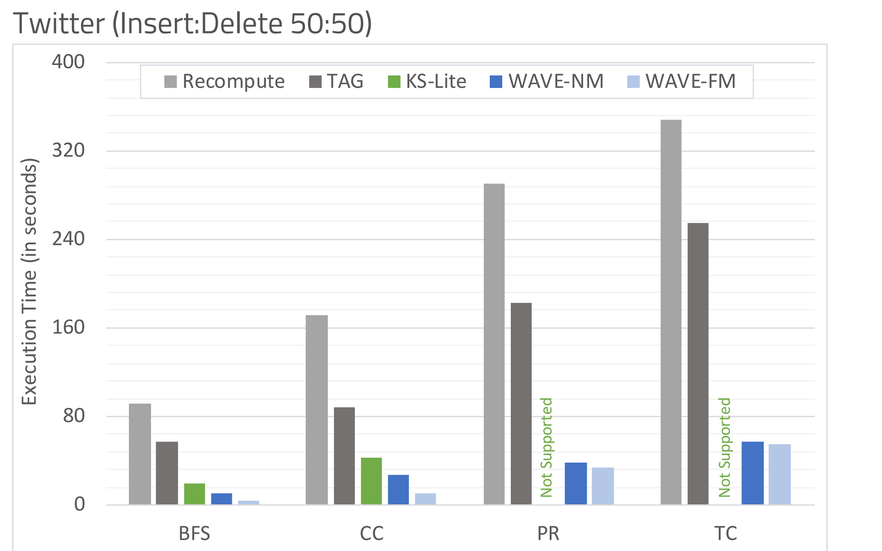
Experimental Evaluation

Algorithm	Aggregation
BFS	$\min_{\forall e=(u,v)\in E}(s(u)+1)$
CC	$\min(v, \min_{\forall e=(u,v)\in E} u)$
PR	$\sum_{\forall e=(u,v)\in E} \frac{s(u)}{out\_degree(u)}$
TC	$\sum_{\forall e=(u,v)\in E} \lvert in\_neighbors(u) \cap out\_neighbors(v) \rvert$

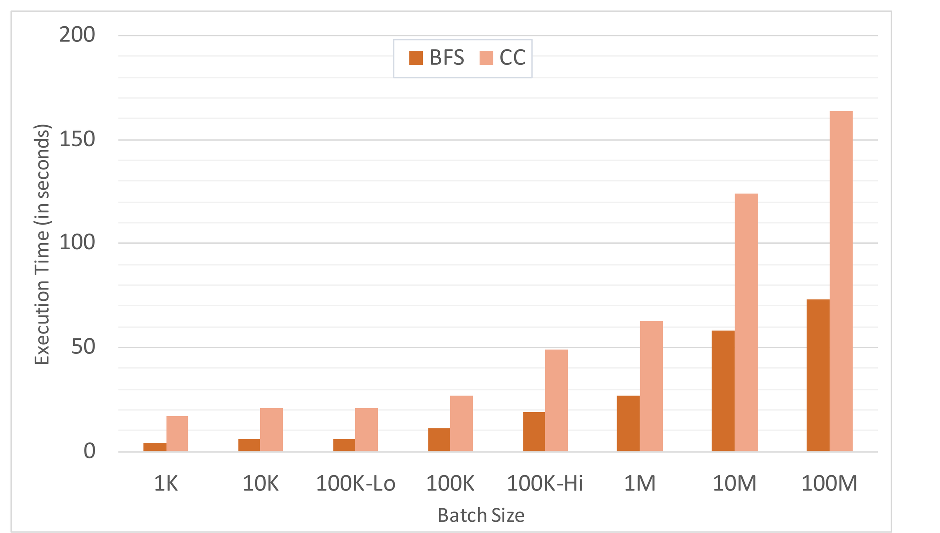
Graph	Vertices	Edges
Wiki (WK)	12M	378M
Twitter (TW)	41M	1.4B

8 Servers (Gigabit Ethernet)
Server : 16 threads / 2.1 GHz / 64 GBs
Apache Giraph 1.3 / Java 8.0

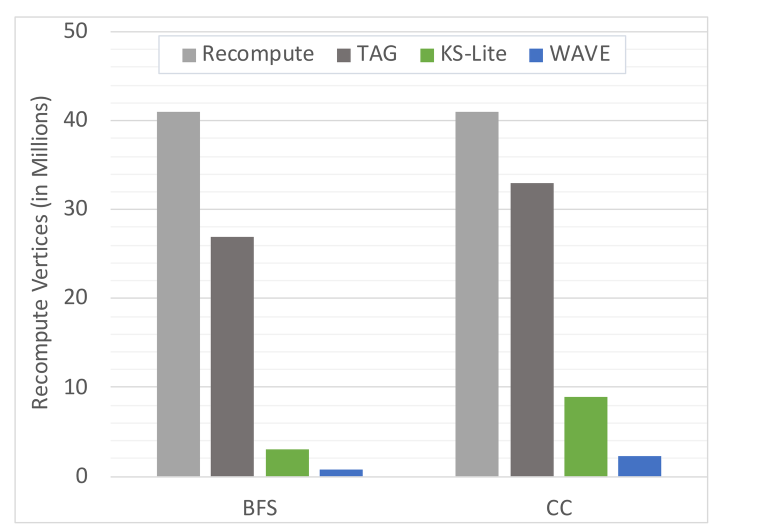




Execution time (in seconds) for Recompute, TAG, Kickstarter-Lite, Wave for 100K mutations
Wave is 6-23x faster than recomputation, 4-14x faster than TAG, and 2-6x faster than KS-Lite



Sensitivity Analysis (TW)



#Vertices recomputed (TW)

Summary

Incremental Graph Processing ➲ Better state re-use ➲ Minimized redundant computation and communication ➲ Faster convergence time

Distributed Research on Emerging Applications & Machines Lab
Department of Computational and Data Sciences
dream-lab.in | Indian Institute of Science, Bangalore
DREAM:Lab