GraphIn: An Online High Performance Incremental Graph Processing Framework

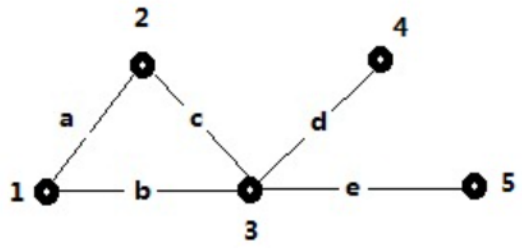
This only supports deletion and insertion of edges

Something to know in advance

Clustering coefficient的定义有两种；全局的和局部的。

全局的算法基于triplet。triplet分为开放的triplet(open triplet)和封闭的triplet(closed triplet)两种（A triplet is three nodes that are connected by either two (open triplet) or three (closed triplet) undirected ties）。
可以用下面结构定义一个triplet

struct triplet { int key; set<int> pair;};
例如下图{1，(2,3)}构成的triplet是封闭的，{3,（4,5）}构成的triplet是开放的



全局的Clustering coefficient比较简单，公式如下：Clustering coefficient(global) = number of closed triplet / number of triplet(closed+open)

- GAS : 1. general to express a broad set of graph algorithms
2. Describe a directed graph
3. 将基于vertex的图计算抽象成一个通用的计算模型GAS模型，分为三个阶段：Gather，Apply和Scatter:

```
        - Gather阶段，用户自定义一个sum操作，用于各个vertex，将vertex的相邻vertex和对应edge收集起来；
        - Apply阶段各个vertex利用上一阶段的sum值进行计算更新原始值；
        - Scatter阶段利用第二阶段的计算结果更新vertex相连的edge值.
- The key assumption:The dynamic graph changes slower than the static processing rate.
- use matrix operations to implement the vertex programs
- reveiw connected component : Connected Components即连通体算法用id标注图中每个连通体，将连通体中序号最小的顶点的id作为连通体的id
```

abstract

- allow incremental graph processing

- GraphIn achieves up to 9.3 million updates/sec and over 400 x speedup when compared to static graph recomputation.
- novel programming model called Incremental-Gather-ApplyScatter (or I-GAS) to incrementally process a continuous stream of updates (i.e., edge/vertex insertions and/or deletions) as a sequence of batches, based on the gather-apply-scatter(GAS)programming paradigm.
- This model allows problem to be reduced to a sub- problem/graph. And this will enable I-GAS to outperform static processing techniques.
- GraphIn may perform worse than static recomputation, such as with incremental BFS, where updates may a↵ect the entire BFS tree. Solution: introduce the notion of “dual-path execution” or property-based switching between incremental and static graph processing based on built-in and user defined properties (e.g., vertex degree information).
- It’s design allows it to run on top of GAS-based static graph analytics framework like X-Stream , GraphMat (used in this work) or Graphchi

this paper uses GraphMat (not Graphchi)

The amazing thing is coming:

1. data struccture Compressed Sparse Row (CSR) formats [5]:provide both space eciency combined with fast traversal (often easily parallelized),,,,,,,disadvantages: updates are expensive because each update requires shifting of the graph data throughout the array to match the compressed format. (Review the CSR:https://www.cnblogs.com/xbinworld/p/4273506.html 数值，列号，以及行偏移。CSR不是三元组，而是整体的编码方式。数值和列号与COO一致，表示一个元素以及其列号，行偏移表示某一行的第一个元素在values里面的起始偏移位置。)

Therefore: - a hybrid data structure involving edge-lists(*faster updates without adversely affecting the performance of incremental computation*) to store incremental updates and compressed matrix format to store a static version of the graph

Five phase

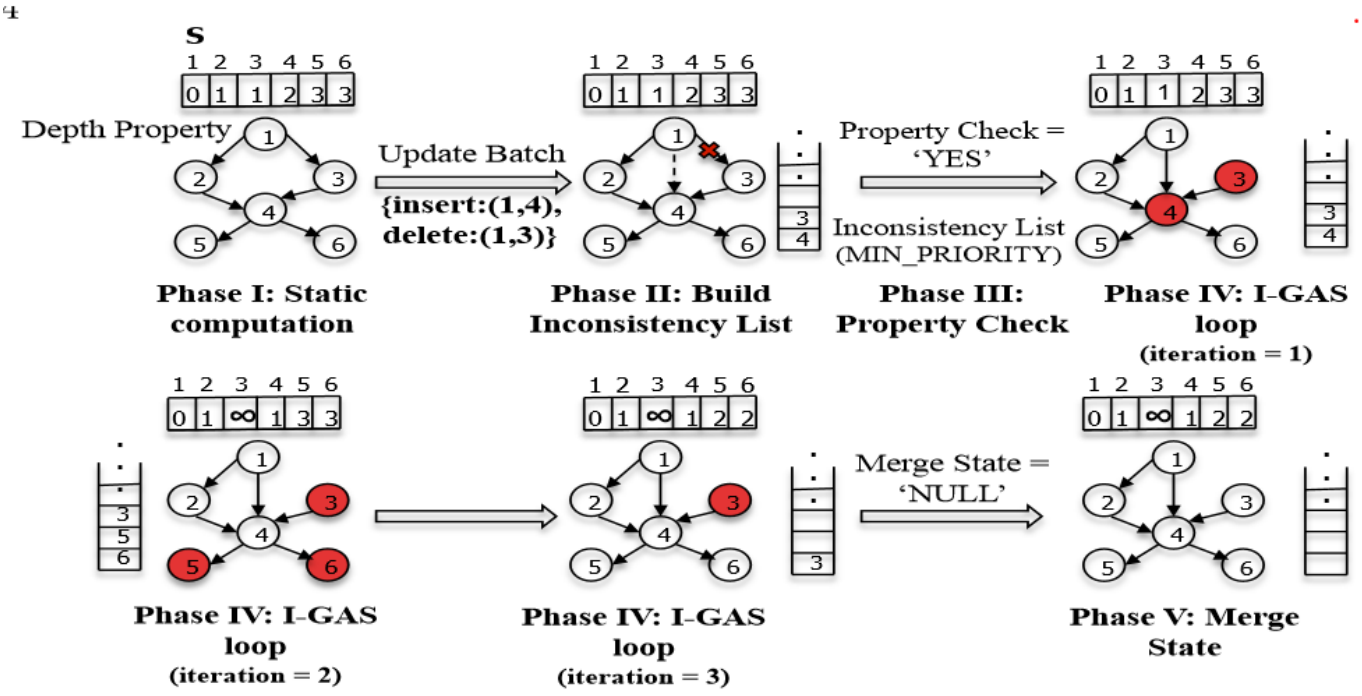


Fig. 2. Incremental BFS Phases.

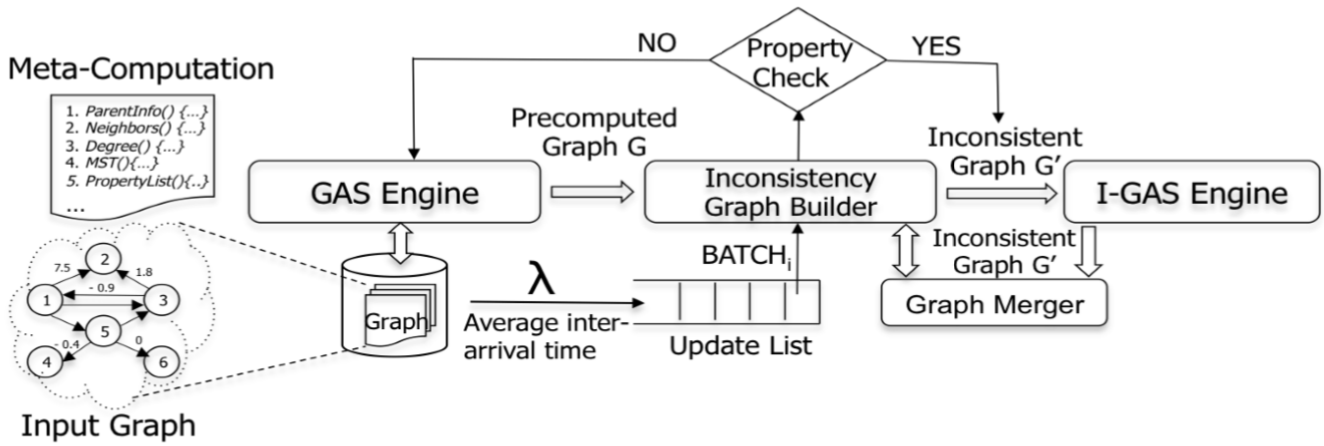


Fig. 1. GraphIn Software Architecture.

The array in this picture represents the level of each vertex in this tree. And this kind of compressed format could save memory

- phase 1 Static Graph Computation

    1. Static graph computation(follow the GAS model) and
    2. Meta-computation to be used later in the incremental logic

- phase 2

```
  1)define an inconsistent vertex to be a vertex for which one or more
properties are a↵ected when the update batch is applied to reduce
unnecessary computation to the vertices whose remain the same over time.*For
example in BFS, addition of edge (vi,v j) can potentially make vj and all
```

```
vertices that are downstream from vj inconsistent.*

  2) responsible for marking the portions of the graph that become
inconsistent

  3) The update batch consists of:
      *edge or vertex insertions and/or deletions from which a list of
inconsistent vertices is built after applying the updates.*
```

- phase 3: Property Check

```
  - Depends both on the algorithm and the particular choice of
updates*Situations that no benefit from this :need to implement the
algorithm that result recomputation over the entire graph*
  - Introduce some function in his implemention that is useless for me
```

- phase 4:Incremental GAS Computation (I-GAS Engine)

```
      - ensures that only the inconsistent part of the graph is recomputed
incrementally, not the entire graph
      -  identifies the overlap between two consecutive versions of the
evolving graph and incrementally processes the graph by opening only the new
computational frontier
```

- phase 5: Merge Graph States

Some incremental algorithms must accommodate both inserts and deletes from the latest update batch in each iteration of the algorithm. These updates must be applied to G before the next update batch is considered
- responsible for both : - merging updated vertex property information - inserting/deleting edges into the most recent version of the static graph G - generating the next version of the graph

```
    - Three kind of algorithm:

        1. All-Merge: Both inserts and deletes are merged with static graph G.
eg.BFS
        2. Partial-Merge: Either deletes or inserts are merged with G. The
framework defers applying the rest of the updates to the original graphs.  eg.
connected component
        3. No-Merge: Neither inserts nor deletes from the update batch are merged
with G. The framework defers applying both inserts and deletes.  eg.deferred
```

Experiment

```
-  Updates are provided in batches of size ranging from 10,000 up to one million
with 1% of all updates being deletions (except for CC where we use only
insertions). The endpoints of the edges used for batch updates are generated
randomly
- Evaluated Algorithms. Three widely used graph algorithms are evaluated,
including Clustering Coecient (CCof), Connected Components (CC) and Breadth First
Search (BFS).
```