1). Library Methods

Gelly has a growing library of off-the-shelf algorithms that users can take advantage of by simply calling the run() method on the graph instance.

In the near future, the following algorithms will be added:

- Affinity Propagation (FLINK-1707)
- HITS (FLINK-2044 / FLINK-3879)
- Minimum Spanning Tree (FLINK-1526)
- Strongly Connected Components (FLINK-2926)

2). Graph Partitioning Techniques

Graph Partitioning plays a key role in application parallelization and in scaling data analysis up. Processes need to evenly be assigned to machines while maintaining communication costs to a minimum. We would like to implement several graph partitioning algorithms for Gelly. As a starting point, Flink’s Graph API can support hash partitioning. We chose this particular approach as it is very easy to implement and it maps to Flink. Afterwards, depending on the type of problem, or the type of vertex different algorithms will be proposed.

For low-degree vertices and for small graphs edge-cut is a viable technique. Vertices are divided into clusters of equal size with the aim to minimize the number of edges spanning different clusters.

For high-degree vertices, on the other hand, the preferred partitioning scheme is vertex-cut, as proven by the authors of PowerGraph. In vertex-cut, edges are evenly distributed across machines with the goal of minimizing the number of replicated vertices.

Finally, after having these two solutions in place and after detecting the skewed nodes with the algorithm in 1), we could apply a hybrid-partitioning scheme which will partition high-degree vertices with vertex-cut and low-degree vertices with edge-cut.

Overall, the following concrete tasks should be implemented:

- hash/random partitioning
- vertex-cut
- edge-cut
- hybrid-cut
- a performance comparison between the different techniques

Ongoing work in this direction can be tracked using the following JIRA issue: FLINK-1536.

3). Partition-centric Iterations

Using Flink’s mapPartition() operator, it should be straight-forward to also add support for the partition-centric computation model. This model exposes the partition structure to the user and allows exploiting the local graph structure inside a partition to avoid unnecessary communication. A more thorough description of the model and its advantages can be found in this paper.

A POC for partition-centric iterations has already been implemented and is available in this github repository.

4). Generic Iterations

The implementation of Boruvka’s distributed minimum spanning tree using Gelly raised several issues as described in the Gelly iteration abstractions mailing list discussion. Among the general problems are the following: branching inside an iteration, convergence checks of internal iterations, tedious/difficult to understand approaches towards differentiating between various algorithm phases (e.g. using aggregators), etc.

These issues could be solved by simply using for-loops. Ongoing work in this direction is the usage of loop unrolling to solve algorithms such as K-core decomposition or DMST. Unfortunately, this approach leads to erroneous behavior such as GC limit exceptions in the Optimizer. Hence, for the loops to be efficiently supported, we should cache intermediate results.

5). Performance evaluation

We would like to compare Gelly’s capabilities to other state-of-the-art graph processing systems. Integration with the Graphalytics benchmarks is work in progress in this github repository.

6). Bipartite Graph Support

A bipartite graph is a graph for which the set of vertices can be divided into two disjoint sets such that each edge having a source vertex in the first set, will have a target vertex in the second set. We would like to support efficient operations for this type of graphs along with a set of metrics, in the near future. Coresponding JIRA issue: FLINK-2254.