# Distributed Label Propagation Algorithm

August 2014

Ian Tsybulkin, Cloudozer LLP

## Introduction

This document describes a distributed label propagation algorithm for graph partitioning and its implementation using Erlang. The software emulates a “Connection Machine” where thousands of computing cells find a solution of the graph partitioning problem using label propagation.

## Original algorithm

The original algorithm can be described as:

1. In the beginning every graph vertex gets its own unique label.
2. In the next iteration each vertex changes its label to the label that most of its neighbors have. If there is a tie-break, we chose the label randomly.
3. After a few iterations most of the vertices won’t change their labels. The process is converging. If we limit the number of iterations by N and set a threshold, say to 95%, as a terminating condition, our algorithm stops either after N iterations or after reaching the threshold.

To overcome label oscillations, the authors of the LPA proposed to modify step 2 doing label updates sequentially changing its order in each iteration randomly.

## Distributed algorithm

**Phase 1 - Setup**

To emulate a Connection Machine, which does distributed LPA, first we need to set it up. We represent each graph vertex that changes its label by a computing cell - Erlang process. This process will know nothing, but its neighbor vertices, central process to report its final label, and terminating condition.
Reading the file containing graph edges we check whether two vertices of the edge already spawned. If one vertex does not, we spawn a new process for that vertex.
Then we send to both vertices the information about its new neighbor.

**Phase 2 - Label propagation**

After all edges are read and Network Machine consisting of as many of computing cells as we have vertices in our graph is built we may start label propagation. The central node send ‘Start” message to all processes.
Each process repeat the next steps:

1. It sends its label to all its neighbors and receives similar messages back.
2. When it got N messages, as many as it has neighbors, the computing cell modifies its label and repeat the same two steps again.
3. After K iterations it sends its label to the central process and terminates.

# Algorithm modifications

The algorithm can be modified slightly. For example:

1. Each process can send a message to the central process after each iteration indicating whether it changed its label or not. The central process can monitor the overall convergence of the LPA.
2. If some process converged earlier it may inform all of its neighbors that it terminates earlier, so they won't expect messages from it further and won't send their label updates too.