skipperkongen

Trying stuff with code

Turning big hard problems into smaller, less hard problems.

November 14, 2010 Algorithms

Here I have captured a thought process I had while reading about algorithms for hard graph problems. The thoughts are inspired by MapReduce, distributed merge sort and the more colorful newspapers of the world.

Summary of thoughts

Given an instance of an problem (think Max Clique, Traveling Salesman or another hard graph problem)…

Thought 1:

> *Compute an instance that is "easier" but has the same optimal solution. This is done by a "reducer algorithm".*

Thought 2:

> *Reducer algorithms may run in parallel.*

Thought 3:

> *Reducer algorithms may be different.*

Thought 4:

> *Reducer algorithms can "gossip" with each other during execution. Gossip helps an algorithm by yielding new information about the problem being solved.*

Thought 5:

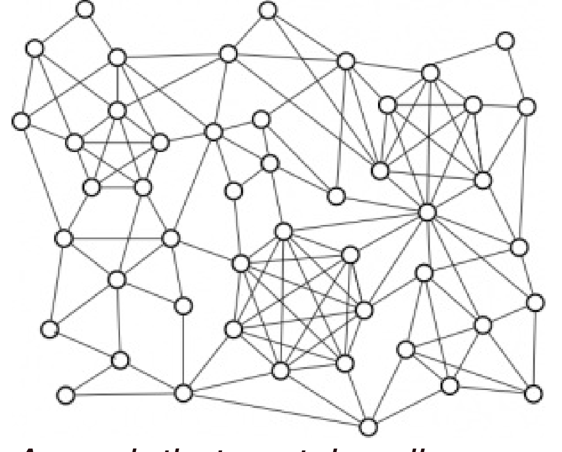
> *Gossip is either a suboptimal solution or a reduced problem instance. This information can be used as a lower bound, or in other ways.*

Thought 6:

> *"Merger algorithms" can combine problem instances from different reducer algorithms into one.*

A full example of reducing and merging: Maximum Clique Problem.

Here is an instance of the Maximum Clique Problem, in this case a non-planar graph. By the way, planar graphs are boring because they can only contain cliques of size 4 or smaller.

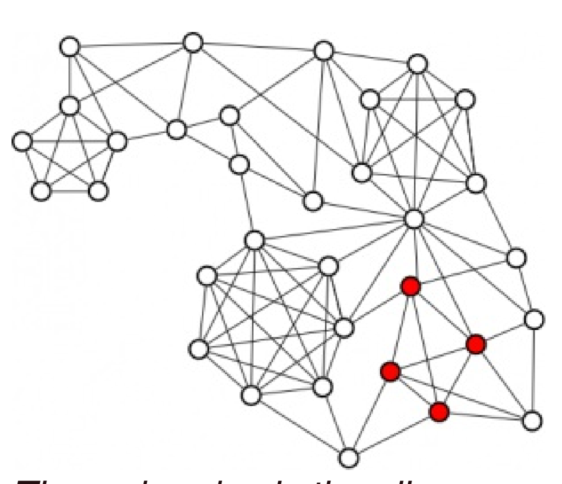


A graph that contains cliques of different sizes.

Let's see what could happen when running two different reducers (reducer 1 and reducer 2) on this problem instance, and then merging the returned instances.

Reducer 1 works by randomly finding a clique in the graph, and repeatedly deleting nodes that have degree less than the size of the clique. The clique found is emitted as a gossip message (reducer 2 will use this as a lower bound).

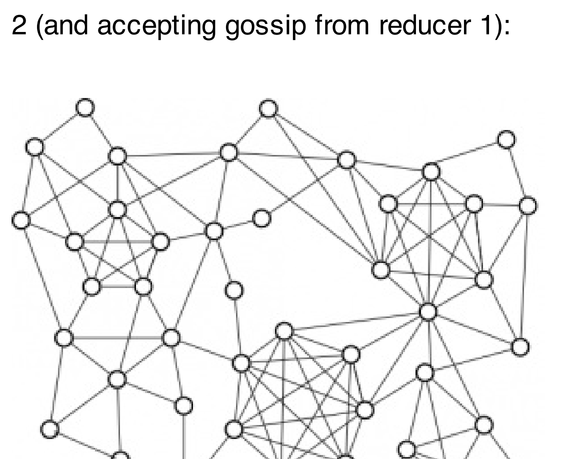
Here is the result of running reducer 1:



The red nodes is the clique found by reducer 1 and gossiped to reducer 2

Let's look at reducer 2. While running reducer 2 could receive a gossip message from reducer 1, that a clique of size 4 has been found. Reducer 2 could use this as a lower bound. Reducer 2 targets nodes of degree around the lower bound. It works (slowly) be examining the targeted node to find out if it is part of a clique. If not it is deleted from the graph.

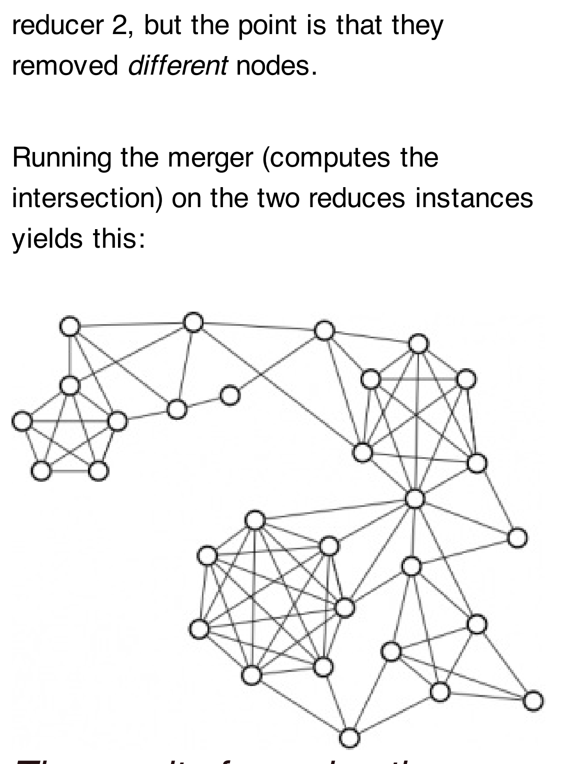
This could be the result of running reducer 2 (and accepting gossip from reducer 1):



After getting a gossip that a 4-clique has been found, reducer 2 targets nodes with degree 4 and removes them if they are not in a clique.

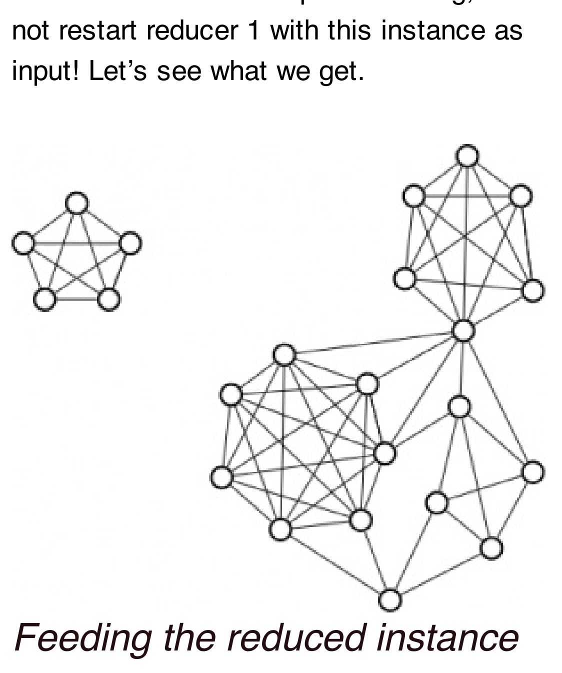
In this madeup example reducer 1 managed to remove more nodes than reducer 2, but the point is that they removed *different* nodes.

Running the merger (computes the intersection) on the two reduces instances yields this:



The result of merging the output of reducer 1 and reducer 2

Yay, an even smaller instance. But while we have the reducers up and running, we not restart reducer 1 with this instance as input! Let's see what we get.



Feeding the reduced instance into reducer 1 for further reduction eliminates even more nodes

This look pretty good. This graph contains only 23 nodes, which is approximately half of the original graph, and that by discovering a relatively small clique of size 4 (compared to the big one of size 7).

Conclusion and a small disclaimer

Most people who deal with such problems call this sort of thing preprocessing. I call it a "reducer network", mainly because it sounds cooler, but also because I think there might be a novel idea here. Namely running a host of algorithms in a distributed environment to perform the preprocessing while emitting and accepting gossip. Of course this is very similar to the ideas behind Google MapReduce and similar services, and might be exactly the same thing. I just felt the need to document my though process, and this post was created 😃

This blog post is based on ideas and thoughts I had while reading "The Algorithm Design Manual" by Skiena (great book). The thougts are just that, thoughts.