Distributed k-Core Decomposition of Dynamic Graphs

Paul Jakma
University of Glasgow
paul@jakma.org

Marcin Orczyk[*]
University of Glasgow
orczykm@google.com

Colin Perkins
University of Glasgow
csp@csperkins.org

Marwan Fayed
University of Stirling
mmf@cs.stir.ac.uk

ABSTRACT

The k-core decomposition can be used to reveal structure in a graph. It is straight-forward to implement using a centralised algorithm with complete knowledge of the graph, but no distributed k-core decomposition algorithm has been published. We present a continuous, distributed, k-core decomposition algorithm for dynamic graphs, outline a proof of correctness, and give initial performance results. We briefly describe an application of this distributed k-core algorithm to landmark selection for compact routing.

Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: Graph Algorithms

Keywords

Graph decomposition, distributed algorithm

1. INTRODUCTION

Graph decomposition is widely used for analysing the structure of social networks, for network visualisation, in developing new routing algorithms, and in other applications. Many of the networks to be analysed are large and/or geographically distributed. This has led to interest in distributed algorithms for graph decomposition.

The k-core decomposition algorithm [1] can reveal a central, well-connected, core of nodes in a graph; the $k_{\max}$-core or *nucleus*. Previous work [4] has shown that the nucleus of the Internet Autonomous System (AS) graph is appropriate for use as a landmark set for Thorup-Zwick (TZ) compact routing [5]. However, known algorithms for k-core graph decomposition are centralised and unsuitable for use in a distributed routing protocol (indeed, it is only recently that attention has been given to the k-core decomposition of dynamic graphs [2]). In the following, we outline a distributed algorithm for k-core decomposition of a dynamic graph. We sketch proofs of correctness and convergence, and present initial performance results. The availability of a distributed algorithm for k-core decomposition of the AS graph brings us one step closer to a practical TZ compact routing protocol.

[*]Current affiliation: Google Inc

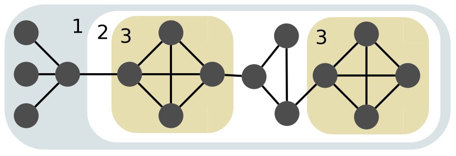


Figure 1: k-core decomposition of a simple graph

2. BACKGROUND

The k-core decomposition of a graph produces a sequence of nested subgraphs of gradually increasing cohesion. Each node in the graph is assigned to a sequence of cores, so that in a k-core each node must have at least k other neighbours. The number k in this context is then an indicator of the centrality in the network: nodes within a higher k valued k-core are more central to the network than the nodes outside of that k-core. An example is shown in Figure 1 with the k-cores labelled: all nodes are in the 1-core; a subset of those are in the 2-core; and two disjoint subgraphs form the 3-core.

Algorithm 1 Graph-centric k-core algorithm

```
1: $k \leftarrow 0$
2: $G_k \leftarrow G$
3: while $|G_k| > 0$ do
4:     while $\{n \in G_k : k > |\deg(n)|\} \neq \emptyset$ do
5:         $G_k \leftarrow G_k \setminus \{n \in G_k : k > |\deg(n)|\}$
6:     end while
7:     $G_{k+1} \leftarrow G_k$
8:     $k \leftarrow k + 1$
9: end while
```

A k-core of a graph, G, is formally defined as follows: "If H is a subgraph of G, $\delta(H)$ will denote the minimum degree of H; each point of H is thus adjacent to at least $\delta(H)$ other points of H. If H is a maximal connected (induced) subgraph of G with $\delta(H) \geq k$, we say that H is a k-core of G." [3]

The k-core decomposition of a graph is the production of all non-empty k-core sub-graphs. That is, the process of finding all k-cores within the graph and determining the k-core membership of each node. A k-core decomposition algorithm for static graphs is shown in Algorithm 1. It starts by assigning all nodes, including disconnected nodes, to the 0th-core. Then, it continuously removes all 0-degree vertices (and their edges) from the graph until no more such vertices remain; the remaining vertices form the $k = 1$ core. Then, all 1-degree nodes are continuously removed, and so on, until

Algorithm 2 Distributed, relaxation-based k-core process

```
1: for all $n \in N$ do
2:     $S_n \leftarrow$ deg // initialise k-value for each neighbour
3: end for
4: $k_t \leftarrow \text{kbound}(S)$
5: SEND_TO_NEIGHBOURS($\langle k_t \rangle$)
6: loop
7:     $t \leftarrow t + 1$
8:     for any $n \in N$ do // wait for message
9:         $S_n = \text{RECEIVE}(n)$
10:        $k_t \leftarrow \text{kbound}(S)$
11:        if $k_t \neq k_{t-1}$ then
12:            SEND_TO_NEIGBOURS($\langle k_t \rangle$)
13:        end if
14:    end for
15: end loop
```

there no nodes left to assign to further cores, and all k-cores are known. This algorithm is centralised, acts on the full state of the graph, and requires various intermediate graphs to be calculated. A distributed implementation is difficult.

3. DISTRIBUTED K-CORE ALGORITHM

We present a distributed algorithm for calculating the k-cores of a static graph in Algorithm 2. This is run in parallel on each node, and relies on messages exchanged with the direct neighbours of the node, N, for each n of which state S_n is maintained. This state is initialised to the node's degree, and updated as messages are received from the neighbour.

When a message is received, a node then determines an upper-bound on its maximal k-core, $\text{kbound}(S)$, such that $\text{kbound}(S)$ is the maximal i such that at least i neighbours have an S_n of at least i or greater. As information comes in from the nodes' neighbours, this kbound is progressively tightened, converging on the correct value for the maximal k-core of the node. In essence, it is probing for the highest set of values that satisfy the k-core condition across the network.

Algorithm 2 can be shown to convergence on the correct maximal k-core values for node, as follows:

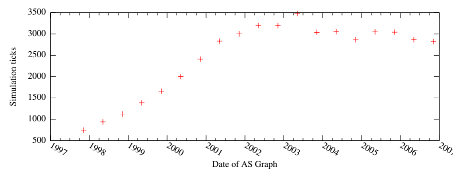
1. Each node starts with its kbound greater than or equal to its maximal k-core
2. If the kbound at any node is greater than the maximal k-core of that node there must be a node whose kbound must change
3. The kbound calculated at each node can only decrease
4. No node can calculate a kbound lower than its maximal k-core, if at all other nodes the kbound is greater than or equal to the respective maximal k-core for that node

This shows the algorithm must stabilise on the correct result (space constraints prevent us including the full proof).

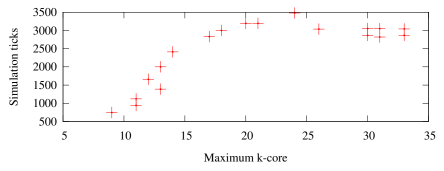
We ran Algorithm 2 on annual snapshots of the Internet AS graph. Figures 2(a) and 2(b) shows preliminary results of how convergence time varies with the date of the AS graph, and the maximum k-core present in the graph. Figure 2(c) shows how the aggregate number of messages sent varies with the size of the AS graph.

4. APPLICATION TO DYNAMIC GRAPHS

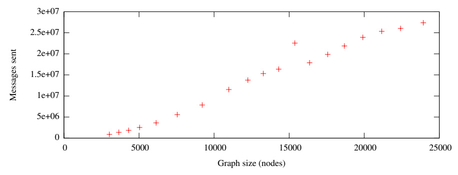
The distributed k-core algorithm described in Algorithm 2 supports edge removal while running. Edge removal can only lower the maximal k-core value of nodes, and the algorithm simply continues with its decreasing search for those values.



(a) Convergence time



(b) Convergence time by kmax



(c) Number of messages

Figure 2: Performance on static graphs

To support the addition of edges, the algorithm must reset its downward search. We extend the algorithm with a generation counter, maintained across each node, included with each message, and stored as part each node's neighbour state. The kbound function is modified to consider only values from neighbours whose generation matches the local generation, and to use the local degree as the value for neighbours whose generation does not match. The generation is increased if a new link is added or if a neighbour increases its generation. When the generation is increased, the neighbour state is reset back to the degree. Thus, the distributed algorithm effectively restarts itself in a distributed manner when links are added.

5. DISCUSSION AND CONCLUSIONS

We presented a distributed k-core decomposition algorithm. Preliminary results to show reasonable scaling behaviour on AS graph snapshots. The k-core decomposition has been suggested as a means to select landmark nodes for compact routing schemes [4] for the Internet. The distributed algorithm presented here could be used to implement such a scheme, and the results given suggest it would scale and be practical.

6. REFERENCES

[1] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of Internet topology using k-shell decomposition. *Proc. Nat. Academy of Sciences*, 104(27):11150–11154, July 2007.

[2] D. Miorandi and F. De Pellegrini. K-shell decomposition for dynamic complex networks. In *Proc. IEEE WiOpt*, June 2010.

[3] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269, 1983.

[4] S. D. Strowes and C. S. Perkins. Harnessing Internet topological stability in Thorup-Zwick compact routing. In *Proc. IEEE Infocom*, March 2012.

[5] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. Symp. Parallel Algorithms and Architectures*. ACM, July 2001.