8.3 Paper Review

8.3.1 A functional approach to external graph algorithms

Abello, Buchsbaum, and Westbrook's paper presents, as the title helpfully indicates, a functional approach to external graph algorithms. It is characterized by the concision and power of the functions that are exposed to the user, including primitives (scan, sort, bucket) and transformations (contract, relabel, select). This approach differs from previous methods, which typically operate using in-place modifications of the graph, by completely avoiding mutation of the input data. This property is desirable in external memory algorithms because the bulk of running time is consumed by I/O operations that read and write to disk (in fact, they contribute so much to complexity that the authors approximate operations on RAM as "free" relative to the cost of I/O operations.)

Central to the challenge that the paper addresses is how primitives batch the data so that B items exist in a single disk block, that is, the number of items that are written or read per I/O operation. In the use-cases this method is designed for, value B is generally less than the number of items in main memory M, which in turn is less than the total number of items in an instance N. For example, the simplest exposed primitive `scan(N)` takes $\lceil N/B \rceil$ I/O operations to read all N items, as it reads from disk in batches of size B. In general, these methods allow us to reduce N I/O cost to N/B, and to reduce the number of I/O passes to $\log_{M/B} N/B$ instead of $log_2 N$. [1]

The algorithms presented in this paper work by recursively reducing the size of the problem until it fits in main memory and computations can be efficiently performed on the reduced graph using RAM. Once performed, the graph is expanded out again. This divide-and-conquer approach can be applied to many algorithms in the literature, including connected components, minimum spanning forests, bottleneck minimum spanning forests (first implementation for external memory), maximal matching (first implementation for external memory), and maximal independent sets (first implementation for external memory). All of these algorithms operate on properties that are retained over the course of recursive contraction. The authors give the example of an algorithm that requires information about graph planarity as something that could not be processed by their method, as local differences in planarity would be lost as the graph is contracted.

In addition to being novel, the implementations are competitive with other examples in the literature. Deterministic connected components and maximum spanning forests slightly improve over the asymptotic I/O bounds of earlier work, and the randomized algorithms at least match earlier I/O bounds. The semi-external model, which assumes that (much like social networks and other real-world graphs) the number of edges can sometimes far exceed the number of vertices, so stores vertices in local memory but does not store edges, further improves on these bounds.

I appreciate the intuition of the paper's solution – take existing graph problems and circumvent I/O overhead by doing as little external memory work as possible, and batching the data that you do read/write so that you perform as few I/O operations as possible. The authors also present a compelling suite of applications that demonstrate how rich their implementation is.

The method is tantalizing and introduces many avenues for future work around the implementation of other algorithms (shortest path etc.) in a functional framework. I would also love to see the semi-external applications applied to real-world, large-scale, small-world networks. It seems as though there could be demonstrable and dramatic improvements to previous external-memory solutions to parsing these large networks.

[1] The authors have an interesting discussion about the way parallel algorithms can be simulated on serial hardware using buffering data structures.