Technical Perspective: Sortledton: a Universal Graph Data Structure

Angela Bonifati
Liris CNRS Research Laboratory
Lyon 1 University, France
angela.bonifati@univ-lyon1.fr

Graph processing is becoming ubiquitous due to the proliferation of interconnected data in several domains, including life sciences, social networks, cybersecurity, finance and logistics, to name a few. In parallel with the growth of the underlying graph data sources, a plethora of graph workloads have appeared, ranging from graph analytics to graph traversals and graph pattern matching. Graph systems executing both complex and simple graph workloads need to leverage adequate data structures for efficiently processing heterogeneous graph data. While the underlying graph data structures have been extensively studied for the static case, they are less understood for the dynamic case, with the data undergoing several updates per second. Moreover, the existing solutions suffer lack of generality, as they focus on one specific requirement and workload type at a time. Designing a universal data structure that adapts to several kinds of graph workloads in a dynamic setting and achieves significant efficiency on all of them is far from being trivial.

The basic operations necessary to execute diverse graph workloads in dynamic cases belong to the three categories: scans, insertions and intersections. While scans have linear complexity in the size of the input data, and inserts have logarithmic complexity, intersections between participating neighbors in a graph might be quadratic. Intersections are needed for graph pattern matching operations (GPM), thus being critical for supporting dynamic workloads. Scans and insertions on the other side are relevant for all dynamic workloads, encompassing graph traversals, graph analytics and GPM.

Graph workloads are typically memory-bound. When conceiving a new data structure, it is important to optimize for memory access patterns. These include sequential access to all vertices, sequential access to edges in a neighborhood, and random access to algorithm-specific properties, such as PageRank scores or distances for graph traversals. Existing data structures, such as Column Sparse Row (CSR), a fast read-only data structure improving adjacency matrices, optimize for both sequential vertex access and sequential neighborhood access, while Adjacency Lists (AL) optimize only for the latter. Striking a balance between these two designs in a dynamic case is a real challenge and the data structure design needs to consider the pros and cons of each design for all the patterns in a dynamic scenario. An empirical evaluation on representative dynamic datasets can actually guide this choice. The highlighted paper starts with this empirical analysis before proposing an adjacency list-like data structure that optimizes for all the above patterns (random vertex access, sequential neighborhood access and intersections) but the sequential vertex access. The latter is indeed less critical for optimization purposes compared to the other access patterns. It also discusses the choice of adjacency lists compared to adjacency matrices (e.g. CSR-like) in terms of opportunities for parallelization, as required by vertex-centric graph computations, as well as the less expensive maintenance of the index. The design of the Sortledton data structure is elegant and effective at the same time and relies on reusing existing data structures. It combines an adjacency index and adjacency sets, containing an adjacency list for each neighborhood. The index allows to map vertex identifiers to vertex records, containing a pointer to the neighborhood, its size and a read-write latch for parallelization. The adjacency sets store the neighborhood of each vertex, accommodating intersections and sequential neighborhood access. Sorted sets are used here and they are implemented as unrolled skip lists and organized as blocks. The rebalancing is much easier to handle compared to classical database indexes, such as B+- trees. Unrolled skipped lists are used for nodes with higher degrees, whereas for smaller degrees vectors are used.

The experimental assessment conducted in the paper is impressive and offers a comprehensive comparison with relevant baselines, both CSR-like and AL-like. The obtained results clearly show the utility and efficiency of the new graph data structure to handle diverse graph workloads under millions of transactional updates per second. It achieves good performances that are only slightly higher than those of static CSR-like data structures while using only doubled space.

Most state-of-the-art systems do not support a variety of workloads (including graph traversals, analytics and GPM) with dynamic graphs. The new data structure proposed in the highlighted paper unifies sequential access to all vertices, sequential access to edges in a neighborhood and random access to algorithm-specific properties under frequent updates. It paves the way to future adoption within commercial and open-source graph processing systems. For instance, it will certainly benefit existing graph databases especially with transactions and concurrent updates and is extensible to other graph workloads and benchmarks, with hybrid OLTP and OLAP operations.