Strongly Connected Components

Graph is arguably one of, if not the most important form of data structure in the field of computer science. Graph abstracts problems into the directional relationship(edges) between entities(vertices). In any problem, especially ones that are complex, can have subsets of un-overlapped nodes that are strongly connected to one another. In other words, a set of Strongly Connected Components(SCC) are a maximal subset of vertices in a directed graph where each nodes can reach and can be reached from every other nodes*(McLendon et al., 2005), p. 901)*. SCC analysis is one of the fundamental analyses done in graph theory that gives insight into the structure, and the connectivity of the nature of the problem.

In graph studying, a graph is defined as $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is a set of all possible edges. There are undirected graphs and directed graphs. An undirected graph is a graph where two nodes of an edge, denoted as (v_1, v_2), are unordered; There is no specific direction in the relationship between the twos($(v_1, v_2) \equiv (v_2, v_1)$). That is not the case in directed graphs, where the direction of the edge matters($(v_1, v_2) \not\equiv (v_2, v_1)$ (Tarjan, 1972, p. 146).

Traditionally, SCC analysis is done on the foundation of Depth First Search(DFS). Algorithms such as Tarjan's(Tarjan, 1972, p. 155), and Kosaraju-Sharir(Aho, Hopcroft, & Ullman, 1983, Chapter 6.7) are elegant ways that do the analysis in linear time using DFS. Forward-Backward(FWDBWD) is another approach to perform SCC decomposition that is based on the idea of Kosaraju-Sharir and combines it with symbolic analysis. Chain is a similar algorithm that is recently developed that is truly symbolic(Larsen et al., 2023, p. 353).

Tarjan's algorithm performs the decomposition in one traverse of the graph. It uses the idea of lowlink, which is the node with the lowest discovery time within one SCC that is reachable by all nodes(including itself) in that SCC. The algorithm loops through all nodes calling the DFS that keeps track of the order the node discovered. It then loops through all of its neighbors. If the neighbor has been discovered, it means that it has gone back to one of the local roots, which has the lower lowlink value, which represents one SCC. If not, it performs DFS on that neighbor, and keeps track of the neighbor lowlink values and

perform search on all of its neighbor recursively until all vertices are discovered; After the recursive of each neighbor, it assigns the lower lowlink value between the node and neighbor to the node. The result lowlink contains a list of duplicate values, where nodes that have the same value belong to one SCC(Tarjan, 1972, pp. 156-159). Because DFS on the adjacency list takes $O(V + E)$ to traverse, the algorithm runs in linear time.

Kosaraju-Sharir's algorithm works in two traverses of the graph using DFS. The first traverse collects the time in which the discovery of the node is completely finished, including itself and all of its children. Its purpose of this traverse is to look for the connectivity degree of each node. The nodes that appear later in the stack are more connected in the graph than the ones that appear earlier, and the one that appears last is the most connected nodes in the graph. The second traverse is done on the transposed graph, which is a directed graph where all edges' directions are inverted. It starts with the most connected nodes; Once the DFS meets a visited node, it has gone back to the root of the SCC, It then goes on to look for other unvisited nodes in the decreasing connectivity order until the stack is empty. Using two traverses asymptotically makes the algorithm $O(V + E)$ (Aho et al., 1983, Chapter 6.7).
Forward-backward algorithm also does the SCC analysis by doing two traverses. Unlike Kosaraju-Sharir, this one performs forward reachability and backward reachability(on the transpose graph) at the same time and combines the common nodes in the two lists until all the nodes are in their propers SCC. Chain algorithm is a similar adaptation of the ideas except it calls recursion on the rest of the forward reachability set minus the SCC, and the rest of the vertices(excluding the forward set)(Larsen et al., 2023, p. 357). The two algorithms would work on both DFS and BFS, which gives the running time of $O(V + E)$.

These are all elegant algorithms to perform SCC analysis in linear time. However, they fall short when dealing with massive networks with intricately complicated relationships. Especially due to the nature of BFS and DFS, it is hard to parallelize the decomposition of the graph (Hong, Rodia, & Olukotun, 2013). Multiple attempts have been made to parallelize the SCC analysis, which has yielded certain results. However, they are all greater than linear in running time (McLendon et al., 2005).

Referecens

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing, 1*(2), 146–160. https://doi.org/10.1137/0201010

McLendon, W. III, Hendrickson, B., Plimpton, S. J., & Rauchwerger, L. (2005). Finding strongly connected components in distributed graphs. *Journal of Parallel and Distributed Computing, 65*(9), 901–910. https://doi.org/10.1016/j.jpdc.2005.03.007

Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Data Structures and Algorithms*. Addison-Wesley. Chapter 6: Directed Graphs.

Larsen, C. A., Schmidt, S. M., Steensgaard, J., Jakobsen, A. B., de Pol, J. v., & Pavlogiannis, A. (2023). A truly symbolic linear-time algorithm for SCC decomposition. In S. Sankaranarayanan & N. Sharygina (Eds.), *Tools and algorithms for the construction and analysis of systems*. *Lecture notes in computer science* (Vol. 13994). Springer, Cham. https://doi.org/10.1007/978-3-031-30820-8_22

Hong, S., Rodia, N. C., & Olukotun, K. (2013). On fast parallel detection of strongly connected components (SCC) in small-world graphs. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (SC '13), Article 92, 1–11. https://doi.org/10.1145/2503210.2503246

McLendon, W. III, Hendrickson, B., Plimpton, S. J., & Rauchwerger, L. (2005). Finding strongly connected components in distributed graphs. *Journal of Parallel and Distributed Computing, 65*(8), 901–910. https://doi.org/10.1016/j.jpdc.2005.03.007