Strongly Connected Components
Cpt S 450 Additional Notes
Cheng Li – 10/28/05

Introduction
A digraph $G(V, E)$ is **strongly connected** if every two vertices u and v are reachable from each other, i.e. there is a directed path from u to v and a directed path from v to u. Two vertices u and v are strongly connected if they are reachable from each other. A **Strongly Connected Component (SCC)** of a digraph $G(V, E)$ is a maximal subset of strongly connected vertices (Maximal in the sense that, if you add one more vertex from V to the strongly connected component, the new larger set is not strongly connected anymore). If a digraph is not strongly connected, it can be decomposed into disjoint strongly connected components. The graph of Figure 1(a) has five strongly connected components.

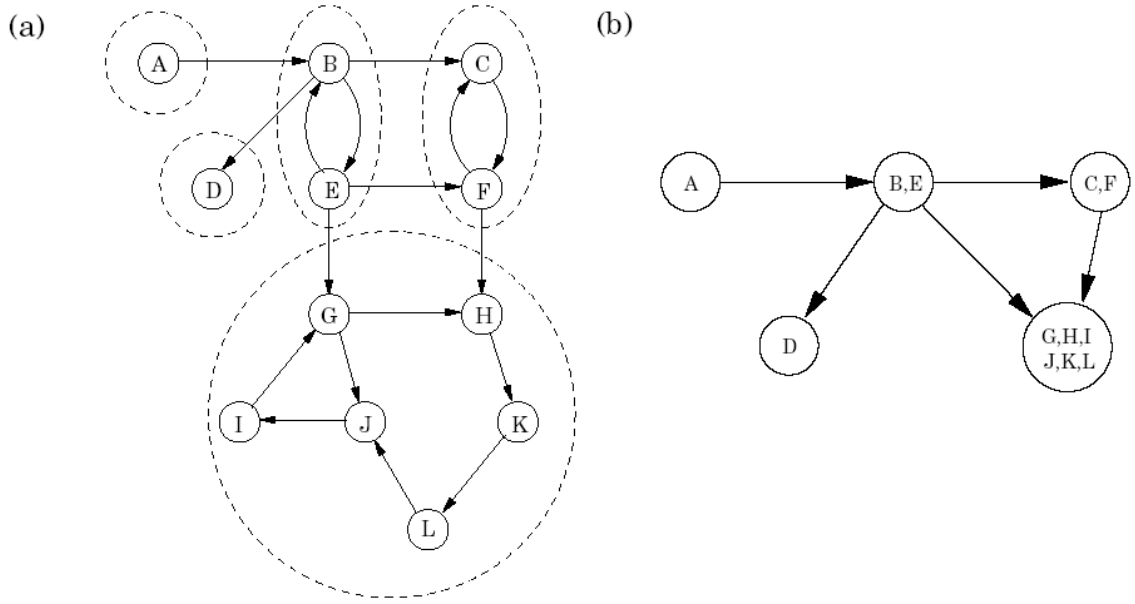

Figure 1. Strongly connected components of a graph

Shrink each strongly connected component down to a single meta-node, and draw an edge form one meta-node to another if there is an edge (in the same direction) between their respective components (Figure 1(b)). The resulting meta-graph must be acyclic. The reason is simple: a cycle containing several strongly connected components would merge them into a single strongly connected component.

Strongly Connected Component Algorithm
The decomposition of a directed graph into its strongly connected components is very informative and useful. Tarjan [1] has presented an elegant algorithm that finds the

strongly connected components in $O(|V|+|E|)$ time in 1972, where $|V|$ is the number of nodes and $|E|$ is the number of edges in the input graph. Besides Tarjan's algorithm, another linear time algorithm --Kosaraju-Sharir algorithm is presented in many textbooks. This algorithm is attributed in [2] to R.Kosaraju and published in [3] by M. Sharir. It requires one depth first traversal of the input graph and another depth first traversal of the graph obtained by reversing the edges of the input graph. Reversing a graph can be done in linear time, but is quite time consuming for large graphs.

The algorithm we present here is the Kosaraju-Sharir algorithm, which is simpler but somewhat less efficient because of the graph reversing.

Kosaraju-Sharir algorithm
Input: A directed graph $G(V,E)$
Output: All strongly connected components of G
Step 1: call $DFS(G)$ to compute finishing times $f[u]$ for each vertex u
--$O(|V|+|E|)$
Step 2: compute G^T, inversing all edges in G using adjacency list
--$O(|V|+|E|)$
Step 3: call $DFS(G^T)$, but in the main loop of DFS, consider the vertices in order of decreasing f[u] as computed in step 1 --$O(|V|+|E|)$
Step 4: output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected component --Constant Time

The algorithm uses, in addition to $G(V,E)$, the reverse graph $G^T(V,E^T)$, with $E^T=\{<u,v>|<v,u>\in E\}$. G^T is easily computed in $O(|V|+|E|)$ time (provide G is given in adjacency list form). The running time of the algorithm is therefore that of a DFS, $O(|V|+|E|)$.

Run this algorithm on the graph of Figure 1, it will output five strongly connected components: {A},{B,E},{D},{C,F},{G,H,I,J,K,L}

Application in Software Verification

Program verification has been an important topic in the computer science research since the development of the first programming language. While programs are written and designed by humans, there will always be a possibility of an error. Testing and debugging are commonly used during the development cycle of a program to locate and fix these errors. The model checking [4] is a technique to allow to automatically analyze behaviors of programs. In the model checking approach to program verification, a finite state transition system is first constructed from the program. This model is then verified against a property (a given temporal formula). Both the property and the model of the program are combined together into a graph representing the program behavior. The model checking procedure is then reduced to finding strongly connected components of this graph.

When considering a Büchi-automaton corresponding to a program, the Büchi-automaton is represented by a digraph $G(V, E)$. One of model-checking problems -- the emptiness problem of a Büchi-automaton becomes: Given a digraph $G(V, E)$, a subset of vertices F, and an initial vertex $s_0 \in V$, find a strongly connected component C of G, such that C is reachable from s_0, and C contains at least one vertex from F. If the strongly connected components is found, the Büchi-automaton corresponding to G in not empty and the property is violated, otherwise, the Büchi-automaton is empty and the property is satisfied.

References

[1] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146-160, June 1972.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Mass., 1983.

[3] M. Sharir. A strong-connectivity algorithm and its application in data flow analysis. *Computers and Mathematics with Applications*, 7:67-72, 1981.

[4] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, Cambridge, Mass., 2000