451: Strongly Connected Components

G. Miller, K. Sutner
Carnegie Mellon University
2020/09/24

Last time we dealt with ugraphs:

- connected components (vertex-disjoint)
- biconnected components (edge-disjoint)

Both can be handled nicely in linear time using variants of DFS.

So how about digraphs?

How about "connected components" in a digraph G? Again let $U \subseteq V$ (though we often think of U as being a vertex-induced subgraph of G).

One way define components for G is to simply ignore the direction of the edges and pretend that G is a ugraph. Then use the connected components from there.

These are called weakly connected components and less useful than the following.

Definition

U is strongly connected if there is a directed path between any two points in U. U is a strongly connected component (SCC) if U is strongly connected but no proper superset of U is strongly connected.

Obviously strongly connected components are contained in weakly connected components, but in general they provide a finer partition.

Note that if C and C' are two SCCs then there may be a path from C to C' or the other way around, but not both.

If C_1, C_2, ..., C_k are the connected components of a ugraph G and G_i the corresponding induced subgraphs, then G is the disjoint sum of the G_i.

But if C_1, C_2, ..., C_k are the strongly connected components of a digraph G and G_i the corresponding induced subgraphs, then G is in general larger than the disjoint sum of the G_i.

This time, the sum misses all edges between strongly connected components.

Also note that it is often interesting to distinguish between trivial and non-trivial SCCs: the latter have at least one edge as a subgraph (could be just one vertex if self-loops are allowed).

Suppose C_1, ..., C_k are the SCCs of digraph $G = \langle V, E \rangle$.

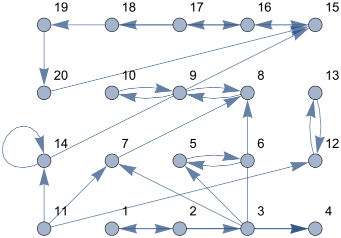
Define the condensation or collapse G^{cond} of G to be the digraph with

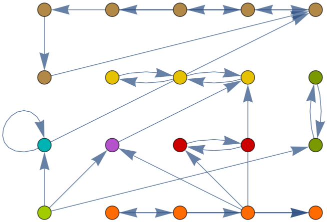
- Vertices: $C_1, \ldots, C_k$
- Edges: $C\,C'$ if $\exists\, x \in C, y \in C'\ (xy \in E)$.

Proposition

The condensation of a digraph is a DAG (directed acyclic graph).

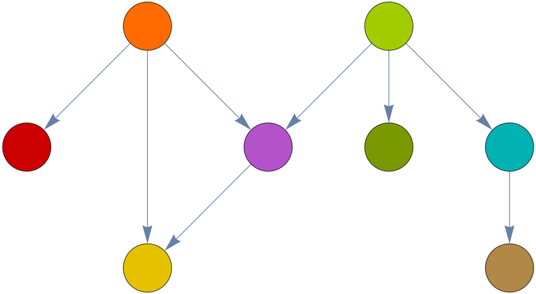
We will show how to compute the condensation in linear time.





SCCs:

$$\{7\}, \{11\}, \{14\}, \{5,6\}, \{12,13\}, \{8,9,10\}, \{1,2,3,4\}, \{15,16,17,18,19,20\}$$



There are several well-known algorithms to compute SCCs:

- Brute force (late 1950s): Boolean matrix multiplication
- Warshall's algorithm (1962): dynamic programming
- Tarjan's algorithm (1972): clever depth-first-search
- Kosaraju's algorithm (1978): two depth-first-searches (in G and G^{op})

Tarjan's algorithm is arguably the most clever and elegant: it just adds a few numerical labels to an ordinary depth-first-search and magically computes the SCCs in time $O(n+m)$.

At first glance, it is hard to believe that the algorithm is correct; it seems there is not enough work going on.

Both the Boolean matrix approach and Warshall's algorithm compute the reflexive transitive closure $E^\star$ of the edge relation E using

$$E^\star = (I + E)^{n-1}$$

or dynamic programming.

In either case, we compute the finest equivalence relation that extends the edge relation, and represent it by a Boolean matrix.

Given that matrix, one can easily compute the equivalence classes in $O(n^2)$ steps. Likewise, we can compute the condensation graph.

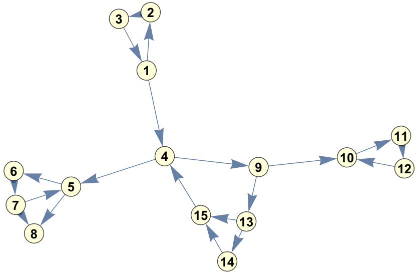
Exercise

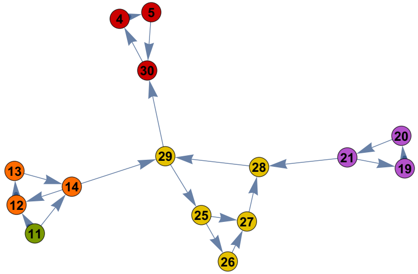
Explain how to do this.

Kosaraju's algorithm runs in two phases:

- Run DFS on G, and generate a list of vertices C that is sorted by completion time.
- Run DFS repeatedly on G^{op}, in order of C. Output the DFS trees generated this way.

Unlike the last two methods (algebra and algorithm design), this approach relies on structural properties of the graph.





On the left, G labeled by discovery times.

On the right, G^{op} labeled by completion times and SCCs.

Let us extend our timestamps to sets of points:

$$\mathsf{dsc}(U) = \min\big(\, \mathsf{dsc}(x) \mid x \in U \,\big)$$
$$\mathsf{cmp}(U) = \max\big(\, \mathsf{cmp}(x) \mid x \in U \,\big)$$

Claim: Suppose C and C' are two SCCs and there is an edge uv from C to C'. Then $\mathsf{cmp}(C) > \mathsf{cmp}(C')$.

$\mathsf{dsc}(C) < \mathsf{dsc}(C')$**:** Consider the moment when DFS first touches the vertex $r \in C$ such that $\mathsf{dsc}(r) = \mathsf{dsc}(C)$ (the so-called root of C). The the subtree T_r contains both C and C'. But then $\mathsf{cmp}(r) = \mathsf{cmp}(C) > \mathsf{cmp}(C')$.

$\mathsf{dsc}(C) > \mathsf{dsc}(C')$**:** This time, let r be the root of C', so C' is contained in T_r At this point, no vertex in C can have been discovered, so again $\mathsf{cmp}(C) > \mathsf{cmp}(C')$.

Theorem

Kosaraju's algorithm correctly determines the SCCs of a digraph.

Proof. Consider the SCC C in G^{cond} that maximizes $\mathsf{cmp}(C)$. By the claim, C must be a indegree 0 vertex in G^{cond}. Pick a witness $x \in C$ such that $\mathsf{cmp}(x) = \mathsf{cmp}(C)$, so in phase 2 we first call DFS on x in G^{op}.

Clearly $C \subseteq T_x$, so suppose $C \subset T_x$. But then there is an edge uv in G^{op} that leads to another SCC C'. This contradicts the claim.

Done by induction. $\square$

We have seen that we can augment vanilla DFS to compute biconnected components in ugraphs. It is tempting to try something similar for strongly connected components in digraphs: the problem seems similar. So we will use extra vertex labels (whatever they may turn out to be) to identify SCCs during a DFS traversal.

Suppose x, y lie in some SCC C. We need to discover a path $x \longrightarrow y$ and a path $y \longrightarrow x$. So we run DFS and get a tree T. In the lucky case, we have $x \xrightarrow{T} y$ and there is a back edge $yx \in E_{\mathrm{b}}$.

Of course, this not going to work in general:

- $x \xrightarrow{T} y$ may not hold (and neither $y \xrightarrow{T} x$)
- instead of a back edge there may be a path of consisting of tree, back and cross edges.

Roots: In keeping with our first-touch principle, we define the root of a SCC C to be the unique vertex $r \in C$ such that $\text{dsc}(r) = \text{dsc}(C)$: the vertex where DFS first touches C. Write $\text{root}(C)$ for the root of C, and $\text{root}(x)$ for the root of the SCC containing x.

Wishful Thinking: We would like to compute something like

$$\lambda(x) = \min\big(\text{dsc}(z) \mid z \in \text{ SCC of } x \big)$$

to identify roots: $\lambda(x) = \text{dsc}(x)$ iff x is a root. Of course, this does not work as written, we don't know what the SCCs are. We will need to find a way around this problem.

Some modified version of λ that still works for root identification, but can be tagged on to DFS.

Proposition

Suppose C is a SCC with root r.

- $r \xrightarrow{T} x$ *for all* $x \in C$
- $\mathsf{cmp}(r) = \mathsf{cmp}(C)$

Proof.

When DFS first touches r all other nodes in C are new. Since all nodes in C are reachable from r the search must construct a tree path to all nodes in C (last lecture).

Well . . .

$\square$

Lemma

Suppose C is a SCC with root r. Then x belongs to C iff $r \xrightarrow{T} x$ and that path does not encounter any other roots.

Proof. First suppose $x \in C$, so that $r \xrightarrow{T} z \xrightarrow{T} x \longrightarrow r$ where z is any intermediate vertex. Then $z \in C$, and thus cannot be another root.

For the opposite direction, suppose $r \xrightarrow{T} x$ and let $r' = \text{root}(x)$, so that $r' \xrightarrow{T} x$. By our assumption, r' fails to lie on the path from r to x, so we must have $r' \xrightarrow{T} r \xrightarrow{T} x \longrightarrow r'$. But then $r = r'$, done.

$\square$

The last lemma suggests an algorithm: generate the SCCs bottom-up with respect to the DFS tree. So the first SCC to be found will be a leaf in the condensation graph of G.

Assume for the moment that we have modified DFS so that, at the end of a call to a vertex x, we can check whether x is a root. If so, we associate x with all the vertices that have already been found, but are not associated with any other root: that produces the SCC of x. This can be handled easily with a stack.

This should sound very familiar to the problem of finding articulation points in the biconnected components case. Unsurprisingly, we will again use low(x) labels to identify roots.

Write $E_{\mathrm{bc}} = E_{\mathrm{b}} \cup E_{\mathrm{c}}$ and, for a set of vertices U, define

$$\lambda(U) = \min\big(\, \lambda(x) \mid x \in U \,\big)$$

Here is a version of λ that we can actually compute:

$$\lambda(x) = \min \begin{cases} \mathsf{dsc}(x) & \\ \lambda(z) & xz \in E_{\mathrm{t}} \\ \mathsf{dsc}(z) & xz \in E_{\mathrm{bc}}, \mathsf{cmp}(x) \leq \mathsf{cmp}(\mathsf{root}(z)) \end{cases}$$

This may look utterly hopeless (what is root(z)?) but we will see pseudocode in a moment that implements λ using a vertex label low.

The first two cases are straightforward.

For the third case, note that if xz is a back edge, then the condition $\mathsf{cmp}(x) \leq \mathsf{cmp}(\mathsf{root}(z))$ is automatically satisfied. Let $r = \mathsf{root}(z)$. We have

$$r \xrightarrow{T} z \xrightarrow{T} x \to z$$

Proposition

Let v, x, z *be vertices such that* $v \xrightarrow{T} x$, $xz \in E_{\mathrm{bc}}$, *but not* $v \xrightarrow{T} z$. *Then* $\mathsf{dsc}(z) < \mathsf{dsc}(v)$.

Proof.

First assume xz is a back edge. Since there is no tree path from v to z we must have

$$z \xrightarrow{T} v \xrightarrow{T} x$$

If xz is a cross edge and we had $\mathsf{dsc}(v) < \mathsf{dsc}(z)$, then there would be a tree path v to z, contradiction.

$\square$

Lemma

$\lambda(v) = \mathsf{dsc}(v)$ *iff* v *is a root.*

Proof. It suffices to establish the following two claims.

Claim 1: $\lambda(v) \geq \mathsf{dsc}(\mathsf{root}(v))$.

Claim 2: If $v \neq \mathsf{root}(v)$ then $\lambda(v) < \mathsf{dsc}(v)$.

If v is a root, by Claim 1, $\mathsf{dsc}(v) \geq \lambda(v) \geq \mathsf{dsc}(v)$.

If $\lambda(v) = \mathsf{dsc}(v)$ but v is not a root we get a contradiction to Claim 2.

Claim 1: $\lambda(v) \geq \mathsf{dsc}(\mathsf{root}(v))$.

Proof is by induction on $\mathsf{cmp}(v)$.
There are three cases depending on the value of $\lambda(v)$.

Case 1: $\lambda(v) = \mathsf{dsc}(v)$.

By the definition of root, $\mathsf{dsc}(v) \geq \mathsf{dsc}(\mathsf{root}(v))$.

Case 3: $\lambda(v) = \mathsf{dsc}(z) < \mathsf{dsc}(v)$ where $vz \in E_{\mathrm{bc}}$, $\mathsf{cmp}(v) \leq \mathsf{cmp}(\mathsf{root}(z))$.

Let $r = \mathsf{root}(z)$, so $r \xrightarrow{T} z$ and thus $\mathsf{dsc}(r) \leq \mathsf{dsc}(z) < \mathsf{dsc}(v)$.
By our assumption about the edge, $\mathsf{cmp}(v) \leq \mathsf{cmp}(r)$. So the call to v is nested inside the call to r, we have tree path from r to v: $r \xrightarrow{T} v \to z \longrightarrow r$.
Therefore, $r = \mathsf{root}(v)$, done.

Case 2: $\lambda(v) = \lambda(z) < \mathsf{dsc}(v)$ where $vz \in E_{\mathrm{t}}$.

The call to z is nested inside the call to v, so $\mathsf{cmp}(v) > \mathsf{cmp}(z)$. But then the IH applies and we have $\lambda(z) \geq \mathsf{dsc}(\mathsf{root}(z))$.

Also, since $vz \in E_{\mathrm{t}}$, we must have $\mathsf{root}(z) = \mathsf{root}(v)$ or $\mathsf{root}(z) = z$.

In the first case we are done.

In the second case, $\lambda(z) = \mathsf{dsc}(z) > \mathsf{dsc}(v)$ since $vz \in E_{\mathrm{t}}$, contradicting our assumption.

$\square$

Claim 2: If v is not a root, then $\lambda(v) < \mathsf{dsc}(v)$.

Let $r = \mathsf{root}(v) \neq v$. Then for some edge $xz \in E_{\mathrm{bc}}$ we must have

$$r \xrightarrow{T} v \xrightarrow{T} x \to z \longrightarrow r$$

where there is no T-path from z to r (the right edge may not be the first one encounters down the tree). By the last proposition we have $\mathsf{dsc}(z) < \mathsf{dsc}(v)$.

Now $\mathsf{cmp}(x) < \mathsf{cmp}(r)$, and r is the root of x, whence $\lambda(x) \leq \mathsf{dsc}(z)$ (3rd case in def of λ). But then

$$\lambda(v) \leq \lambda(x) \leq \mathsf{dsc}(z) < \mathsf{dsc}(v).$$

$\square$

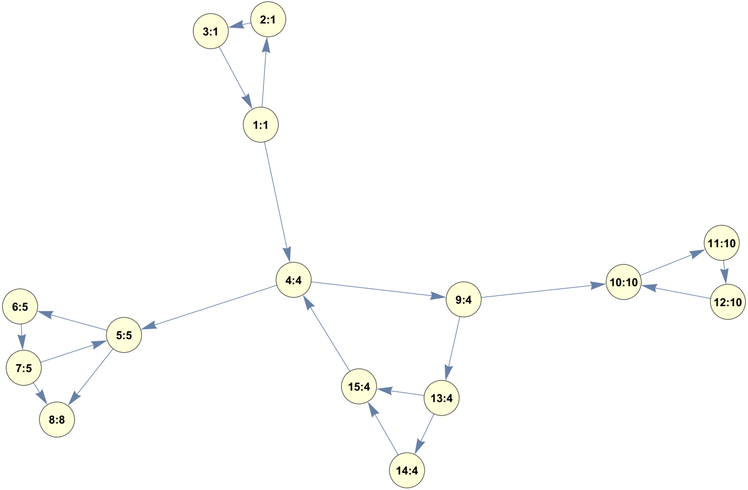
```
defun dfsscc(x : V)

    dsc(x) = low(x) = t++
    push x onto stack

    forall xy ∈ E do
        if dsc(y) == 0                        // xy tree edge
        then
            dfsscc(y)
            low(x) = min(low(x), low(y))
        elseif y in stack
        then low(x) = min(low(x), dsc(y))
    od

    if dsc(x) = low(x)
    then pop the stack down to x              // SCC of x
```

Note that $\mathsf{cmp}(x)$ is important for the proof, but is not actually computed by the algorithm (similar to potentials).

It is a labor of love to verify that the given code makes sure that $\mathsf{low}(x) = \lambda(x)$. Hence we are correctly identifying the roots of SCCs.

Exercise

Do it.



Previous example with discover numbers and low numbers.

In order to compute the transitive closure of a digraph $G = \langle V, E \rangle$ we could use DFS, Boolean matrices or Warshall. Another approach is to

- Compute the condensation G^{cond} of G.
- Compute the transitive closure of the DAG G^c.

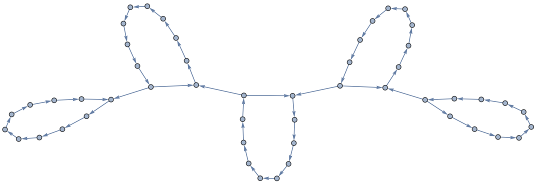
This can be attractive if G^{cond} is much smaller than G.

Satisfiability for Boolean formulae in 2-CNF can be tested in linear time. Convert the formula into the directed graph $G_\varphi = \langle V, E \rangle$ whose vertices are the literals in the formula, and whose edges are defined by

$$\overline{x}y, \overline{y}x \in E \iff \{x, y\} \text{ is a clause.}$$

Then φ is satisfiable iff no SCC of G_φ contains both x and $\overline{x}$.

Here is a digraph whose condensation graph is a path:



The version of this graph with $n = 100\,000$ and loops of length 200 produces 500 components and takes 0.015 seconds.

Random graphs tend to have one giant SCC, and a number of tiny ones, see Knuth.