SETL Newsletter #204

Tarjan's fast interval finding algorithm

J. T. Schwartz
M. Sharir

3 March 1978

1. The compressed balanced tree representation

Let S be a finite set, $f: S \to S$ a partially defined mapping which admits no cyclic mapping, i.e. $\forall k \geq 1$, $x \in S$, $f^k(x) \neq x$. Then we can define for each $x \in S$, $f^\infty(x) = y$ such that $\exists k > 0$, $f^k(x) = y$ and $f(y)$ is undefined (thus $f^\infty(x) = x$ for all x for which $f(x)$ is undefined). We will write this in infix notation f lim x. We will assume that $f = \emptyset$ initially, and is extended by operations of the form $f(y) = x$, where $f(x)$, $f(y)$ are presently undefined (so that no cyclic mapping will ever occur). f can be interpreted as the father mapping in a forest of trees, starting with a collection of single-node trees, and joining trees together by making a root the father of another root. $f^\infty(x)$ is the root of the tree currently containing x. Then f can be stored in a special form which makes these operations ultra efficient: asymptotically, f lim x can be computed in an almost constant number of cycles.

Method:

Represent f by an auxiliary triple of functions g, n, t such that g has similar properties to those of f, $g^\infty(x) = g(x)$ for all x for which $f^\infty(x)$ has already been computed, $n(x) = \#$ of y such that $g^\infty(y) = x$, and $f^\infty(x) = t(g^\infty(x))$ for all x. g is the father mapping in an auxiliary forest, n(x) is the number of descendants of each root x (i.e. the cardinality of its tree). The forest of g is equal, as a set of sets, to the forest of f, but the roots in each forest need not be the same; t is a "real root" mapping, which maps each g-root to the corresponding f-root. These auxiliary functions are manipulated as follows:

(a) Initially, g := nl; n:= {[x,1]: x ∈ S}; t := {[x,x]: x ∈ S};

(b) Whenever $f^{\infty}(x)$ = f lim x is needed, perform "path compression":

```
s1 := nl;  r := x;
(while g(r) ≠ Ω) s1 with r; r = g(r); end;
(∀z ∈ s1) g(z) := r; end;  $ path compression
return t(r);  $ r is the g-root of x and t(r) is the f-root.
```

(c) $g^{\infty}(x)$ (which is needed in the next operation) is computed in exactly the same way, only returning r instead of t(r).

(d) Whenever f(x) = y is performed, perform also "balancing":

```
r1 := g lim x; r2 := g lim y; $ first find the g roots of x,y
if n(r1) < n(r2) then $ g-assignment is the same direction
                      $ as the f-assignment.
  g(r1) := r2; n(r2) := n(r1) + n(r2); t(r2) := y;
else   $ a reversed g-assignment
  g(r2) := r1;  n(r1) := n(r1) + n(r2); t(r2) := y;
end if;
```

It is well known that using such a representation, a mixed sequence of n assignments and limit calculations will be performed in $O(n\,\alpha(n))$ time, where $\alpha(n)$ is related to the inverse of Ackermann's function, so that it is extremely slowly growing, and for all practical purposes is less than 3.

This technique can be applied to several problems related to program optimization, producing highly efficient algorithms for these problems. Among these problems are: testing a flow graph for reducibility and finding intervals for this graph, and constructing the dominator tree of a flow graph. In this note we will sketch only the first application.

2. Tarjan's fast interval finding algorithm

Let G be a flow graph with a *root*. An interval I in G with a head x is a subset of the nodes of G containing x, with the following properties:

(a) All edges in G which enter I from an outside node, must do so through x (i.e. if $(y,z) \in G$, $y \notin I$ and $z \in I$, then $z = x$).

(b) All nodes in I can be reached from x along a path wholly contained in I.

(c) All cycles wholly contained in I must contain x.

It follows that I can be topologically sorted in such a way that x is its first element, and all edges in G between nodes of I are either forward edges, or else back edges whose target is x.

Definition. Let I be an interval in G with a head x. Let us define a transformation of G, called collapsing I to a single node, as follows:

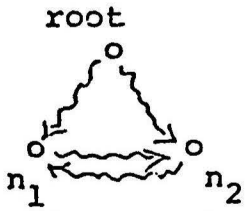
(a) Delete all nodes in $I-\{x\}$ from G.

(b) Replace each edge $(y,z) \in G$, where $y \in I$, $z \notin I$, by the edge (x,z). If $(x,x) \in G$, delete this edge.

The resulting graph is called a derived graph of G.

Definition. A flow-graph G is called reducible if G can be reduced to a single node, by repeated collapsing of intervals.

Theorem 1 (Hecht-Ullman, "Flow-graph reducibility," SIAM J. Computing 1972): G is irreducible if it contains two nodes $n_1, n_2 \neq$ root such that there exist paths $p_1 \in path(root,n_1)$ $p_2 \in path(root,n_2)$, $p_3 \in path(n_1,n_2)$, $p_4 \in path(n_2,n_1)$ such that p_1 does not contain n_2, p_2 does not contain n_1, and $p_3 \cup p_4$ is disjoint from $p_1 \cup p_2$ (except for the end points n_1, n_2). Such a configuration is called a double-entry loop, and is illustrated below:



Our problem is, given a flow graph G, to test whether G is reducible, by finding intervals in G, and collapsing them in an efficient manner. If G is reducible, then we can use the sequence of intervals processed by such an algorithm to solve various data flow problems for G in a linear time (in the length of this sequence and the number of nodes in G).

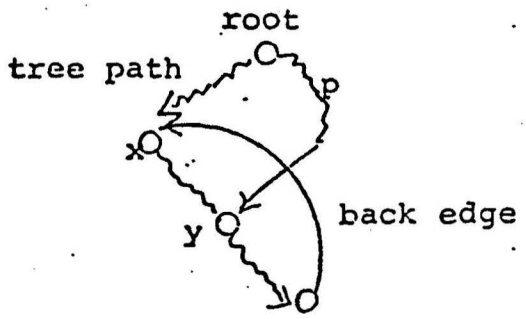
We begin by constructing a depth-first spanning tree (DFST) T of G, numbering its nodes in left-to-right, ancestors first order. Let 'nodeno' denote the node-numbering map, and let 'ndescs' be another map on T, mapping each node a to the number of descendants of x in T (excluding x). All the graph edges can be classified into four categories: tree-edges, forward edges (i.e. edges (x,y) where x is a T-ancestor of y), back edges (edges (x,y) where x is a T-descendant of y) and cross edges (all other edges). It is well known that T is a DFST of G iff all cross edges are right-to-left (i.e. all cross edges (x,y) satisfy nodeno(x) > nodeno(y)).

Definition. For each node x ≠ root, let

reachunder(x) = {all nodes y from which x can be reached along a path not going through x, whose final edge is a 'back edge}.

Theorem 2. G is reducible iff root ∉ reachunder(x), for all x ≠ root.

Proof: If root ∈ reachunder(x), let p be a path from root to x as in the above definition, and let y be the first point on p such that the subpath of p from y to x contains only descendants of x. Then there exists a double-entry loop in G, with x,y as the loop entries, as shown in the following figure.



Thus, by Theorem 1, G is irreducible.
Conversely, if the graph is irreducible, let x and y be entries of a double-entry loop, with nodeno(x) < nodeno(y), such that nodeno(x), nodeno(y) are minimal. Then, by this minimality and the definition of a DFST, it can easily be shown that y is a descendant of x and the path from y to x in the definition of a double-entry loop contains only descendants of y and therefore terminates with a back edge. Since y can be reached from the root via a path which bypasses x, root ∈ reachunder(x).

Q.E.D.

Lemma 3. If root ∉ reachunder(x), then every node in reachunder(x) is a descendant of x, reachunder(x) ∪ {x} is a subtree of T, and all paths into this subtree must go through x.

Proof: If y ∈ reachunder(x) is not a descendant of x, we can go from the root down the tree to y, without going through x, and then from y to x via a path as in the definition of reachunder. Hence root ∈ reachunder(x).
If y ∈ reachunder(x) and z is an ancestor of y and a descendant of x, then we can go from z to x in a similar way as above, obtaining z ∈ reachunder(x). Hence reachunder(x) ∪ {x} is a subtree of T, rooted of x. If y ∈ reachunder(x), z ≠ x and (z,y) ∈ G, then a similar argument shows that z ∈ reachunder(x). Hence, only x can be a target of an edge from outside reachunder(x) ∪ {x}.

Q.E.D.

Corollary 4. Under the same hypothesis, reachunder(x) ∪ {x} is a strongly connected set.

Proof: Any point in this set can be reached from x via a path wholly contained in this set (namely, the tree path from x), and can reach x via a similar path (for all nodes on a path as in the definition of reachunder(x) are also in reachunder(x)).

Q.E.D.

Lemma 5. Let x be the highest numbered node in T (rightmost-bottommost) which is the target of a back edge. Then, if root ∉ reachunder(x) ∪ {x}, then this set is the maximal strongly connected interval whose head is x.

Proof: Lemma 3 and Corollary 4 show that this set is strongly connected and satisfies (a) and (b) in the definition of an interval with head x. To establish (c), let p be a cycle wholly contained in reachunder(x). It is easy to see that p must contain a back edge, and the target of that back edge, being a descendant of x, has a higher node number, which contradicts the choice of x. To show maximality, note first that any interval whose head is x must contain only descendants of x. Let y be such a descendant which belongs to some strongly connected interval with head x. Hence, there exists a path from y to x going only through descendants of x, and so must terminate with a back edge. Thus y ∈ reachunder(x) so that reachunder(x) ∪ {x} is the maximal strongly connected interval with head x.

Q.E.D.

For simplicity, assume with no loss of generality that root is not the target of a back edge.

Tarjan's procedure can now be sketched, as follows: Given a flow graph G,

(1) Compute a DFST T of G, the maps 'nodeno' and 'ndescs' and the set of all back edges in G

(2) Find x as in Lemma 5. If no such x exists, go to step (6).

(3) Compute reachunder(x).

(4) If root ∈ reachunder(x), then the graph is irreducible; halt.

(5) Else, collapse reachunder(x) ∪ {x} to a single node to get a new derived graph G'. Repeat steps (1)-(5) with G'.

(6) At this point, the graph is reducible. The final derived graph is acyclic, and can be topologically sorted in a right-to-left, ancestors first order (of the corresponding DFST).

It is easily seen that this procedure is correct. To see that it can be implemented in a highly efficient manner, using a compressed, balanced tree representation, note the following observations:

Lemma 6. An edge (x,y) ∈ G is a back edge, or, equivalently, x is a descendant of y iff

$$\text{nodeno}(y) \leq \text{nodeno}(x) \leq \text{nodeno}(y) + \text{ndescs}(y)$$

Proof: Trivial.

Next, let 'head' be a map on the nodes of G, that we calculate during execution of the above procedure, in the following way: Initially, head = Ø. After each iteration of our procedure, set head(y) = x for all y ∈ reachunder(x), where x is the node chosen at step (2) of the current iteration. Since all such nodes y are then deleted from the graph, we will never define head(y) more than once. It also follows inductively that, at any moment, head has no cyclic mapping, and that whenever we define head(y) = x, x and y are limit values of the current head map (i.e. head(x) = head(y) = Ω before this definition). Also, x is an ancestor of y in the current DFST.

Lemma 7. At any iteration of the above procedure, if G' denote the current derived graph of G, then

$$G' = \{ (\text{head}^{\infty}(x), y) : (x,y) \in G \mid \text{head}(y) = \Omega \text{ and if } \text{head}^{-1}\{y\} \neq \emptyset \text{ then } \text{head}^{\infty}(x) \neq y \}$$

Proof: By induction. It is certainly true if G' = G, since head = $\emptyset$ at this point. Suppose that it is true for some G', and let G" be the derived graph of G', obtained by collapsing a new interval I' with head x'. Let head_1 , head_2 denote the values of 'head' for G', G" respectively. Then $(u,v) \in G"$ iff (i) $u \notin I'$ and $(u,v) \in G'$, or (ii) $u = x'$, $v \notin I'$ and $\exists w \in I'$ such that $(w,v) \in G'$. Applying the induction hypothesis we get

$$G" = \{ (u,v) \in G' \mid u \notin I' \} + \{ (x',v) : (w,v) \in G' \mid w \in I' \text{ and } v \notin I' \}$$

$$= \{ (\text{head}_1^{\infty}(t), v) : (t,v) \in G \mid \text{head}_1^{\infty}(t) \notin I', \text{head}_1(v) = \Omega \text{ and if } \text{head}_1^{-1}\{v\} \neq \emptyset \text{ then } \text{head}_1^{\infty}(t) \neq v \}$$
$$+ \{ (x',v) : (t,v) \in G \mid w = \text{head}_1^{\infty}(t) \in I', v \notin I', \text{head}_1(v) = \Omega \text{ and if } \text{head}_1^{-1}\{v\} \neq \emptyset \text{ then } \text{head}_1^{\infty}(t) \neq v \}$$
$$= \{ (\text{head}_2^{\infty}(t), v) : (t,v) \in G \mid \text{head}_2^{\infty}(t) \neq x', \text{head}_2(v) = \Omega \text{ and if } \text{head}_2^{-1}\{v\} \neq \emptyset \text{ then } \text{head}_2^{\infty}(t) \neq v \}$$
$$+ \{ (\text{head}_2^{\infty}(t), v) : (t,v) \in G \mid \text{head}_2^{\infty}(t) = x', \text{head}_2(v) = \Omega \text{ and if } \text{head}_2^{-1}\{v\} \neq \emptyset \text{ then } \text{head}_2^{\infty}(t) \neq v \}$$

(This last, set-by-set equality, should be carefully verified by the reader.)

$$= \{ (\text{head}_2^{\infty}(t), v) : (t,v) \in G \mid \text{head}_2(v) = \Omega \text{ and if } \text{head}_2^{-1}\{v\} \neq \emptyset \text{ then } \text{head}_2^{\infty}(t) \neq v \}.$$

Q.E.D.

Lemma 8. Let G' be the derived graph of G at some iteration of the above procedure. Then a DFST T' for G' can be obtained by starting at T and applying to it repeatedly all the collapsing operations of the previous iterations (in which similar DFST's were used).

Proof: By induction. There is nothing to prove if G' = G. Suppose that the assertion is true for some G', with T' the DFST obtained for G' this way. Let G" be the next derived graph, and T" the graph obtained from T' by the collapsing operation. Since a subtree of T' is collapsed, T" is also a tree. By the proof of Lemma 7, any cross edge of G" was obtained from an edge of G' by moving its initial node up T', so that this latter edge must have been a cross edge in G', and by the induction hypothesis went from right-to-left in T'. Thus, the resulting edge in G" must also go from right to left in T", so that T" is a DFST for G". Q.E.D.

Remark. In this case, $head^{\infty}(x) = y$ always implies that y is a T-ancestor of x.

Next, if we carefully examine how T' or G' are used when processed by the procedure, we can show that neither of them has to be formed explicitly.

Let G' be the derived graph during some iteration of the procedure. Define n(G') = minimal node number (in T) which is in the range of the current 'head' map. Obviously, the values of n(G') are nonincreasing during execution of the procedure.

Lemma 9. (a) $(x,y) \in G'$ is a back edge iff nodeno(y) in T is $< n(G')$ and $\exists w \mid (w,y)$ is a back edge in G and $head^{\infty}(w) = x$.

(b) n(G') is the nodeno (in T) of the head of the last interval collapsed to form G'.

Proof: First observe that if $(x,y) \in G'$ is a back edge (in T') then, since T' was obtained by collapsing subtrees of T, y is also a T-ancestor of x.

The proof is by simultaneous induction on (a) and (b). Both are true if G' = G. Assume both to be true for all derived graphs up to and including G', and let G" be the next derived graph. Let $(x,y) \in G"$ be a back edge. By Lemma 7, $\exists (w,y) \in G$, $x = head^{\infty}(w)$. Hence x is a T-ancestor

of w, and by the above observation, (w,y) is a back edge in G. If nodeno(y) $\geq$ n(G'), let G_0 be a previous derived graph such that $n(G_0') \leq$ nodeno(y) $< n(G_0)$. By the induction hypothesis, (x_1,y) is a back edge in G_0, where $x_1 = head^{\infty}(w)$, at the iteration which processes G_0, and it is easily seen that the highest numbered node in T_0 which is the target of a back edge, must be y (here the induction hypothesis on (b) has also been used), so that after that iteration, $head^{-1}\{y\} \neq \emptyset$ and $head^{\infty}(w) = y$. Hence, by Lemma 7, (x,y) cannot be an edge in G". Conversely, if (w,y) is a back edge in G, then y is a T-ancestor of w, and after each collapsing, it still must be a T-ancestor of $head^{\infty}(w)$ (if not, some $head^{\infty}(w)$ will become a T-ancestor of y, so that we will have a lower numbered node than y in the range of 'head', contradicting the fact that nodeno(y) < n(G')). The condition on y implies that $head(y) = \Omega$ and $head^{-1}\{y\} = \emptyset$ so that, by Lemma 7, $(x,y) \in G"$ and is therefore a back edge.

Now, concerning (b), let y be the highest numbered node in T' which is the target of some back edge. By the induction hypothesis, nodeno(y) (in T) < n(G'). This y will be chosen at step (2) of the iteration which processes G', so that at the end of this iteration y will be the only new element in <u>range</u> head, so that n(G") = nodeno(y), which proves (b) for G".

Q.E.D.

To conclude, there is no need to produce explicitly any derived graph or DFST. Instead, the map 'head' should be maintained during the execution of the procedure, using a compressed, balanced tree representation, and the procedure itself should be modified as follows:

- Perform step (1) only once, for the given graph G.
- At step (2), iterate from the last found such x in a decreasing node numbering order (Lemma 9(b)), and use Lemma 9(a) (and Lemma 6) to test for back edges.
- At step (3), one can use the graph $\{(head^{\infty}(x),y) : (x,y) \in G | head(y) = \Omega\}$ insead of G' (cf. Lemma 7). Indeed, this graph contains G', and all extra edges are of the form (y,y). Since we want to construct a set of <u>nodes</u>, these

extra edges will not produce additional nodes in reachunder. (Alternatively, one can use the more complex representation of G' in Lemma 7.) In either case, since these graphs have to be traversed only in a reverse order, this can easily be accomplished using G and 'head' alone. (However, the first graph can be traversed this way more efficiently than the second.)

- Step (4) is unchanged.
- Step (5) amounts now to extending the head map for the nodes in reachunder(x) and branching back to step (2).

Here is a SETL code for the modified procedure:

```
proc intsof(graph,root); $ Tarjan's interval finder
$ Step (1)
nodes := dom graph + range graph; $ the nodes of the graph
inverse := { [y,x]: [x,y] ∈ graph}; $ the inverse graph
[fa,nodeno,ndescs,rleftno] := dfst(graph,root);
                $ depth-first spanning tree
                $ fa is the father mapping of this tree
                $ rleftno is a node-numbering map in a
                $ right-to-left tree walk order, needed in step (6)
nodevect := { [n,x]: [x,n] ∈ nodeno}; $ vector of nodes in order
backedgesinv := { [y,x]∈inverse|nodeno(y)≤nodeno(x)≤nodeno(y)+
                                                +ndescs(y)};
                $ set of all inverse back edges
targbackedges := dom backedgesinv; $ target nodes of back edges
head := nl; initaux; $ initialize auxiliary tree maps
intervals := nult;   $ tuple of all intervals
intno := nl;         $ a map from interval heads to the index of
                     $ their interval in intervals
m := 0;              $ number of intervals encountered
$ step (2)
(∀n := #nodevect ... 2 | x := nodevect(n) in targbackedges)
$ steps (3)-(5)
    m := m + 1;
    intervals with [x]; $ x is the head of the m-th interval
```

```
        intno(x) := m;
        reachunder := {head lim y: y∈backedgesinv{x}};
                                $ all sources of back edges leading
                                $ to x
        (while ∃y∈reachunder-{x}|head(y)=Ω) $ build all reachunder
           head(y) := x; balance(y,x); $ perform 'balancing'
           if root in newreachunder := {head lim z: z∈inverse{y}}then
              return Ω;         $ the graph is irreducible
           else reachunder := reachunder + newreachunder;
           end if;
        end while;
    end ∀;
    $ step (6)
    $ The remaining nodes form the  last interval in the sequence;
    $ its head is root.  Extend 'head' to these nodes.
    (∀y ∈ nodes | head(y) = Ω) head(y) := root; end;
    intervals with nult; intno(root) = m+1;
    rleftvec := { [n,x]: [x,n]∈rleftno}; $ vector of nodes in right-to-
                                         $ left tree walk order.
    (∀y := rleftvec(n)) intervals(intno(head(y))) with y; end ∀;
    return intervals;
    end proc intsof;
```

[x]