FINDING OPTIMUM BRANCHINGS

by

R. Endre Tarjan

Memorandum No. ERL-M506

March 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

FINDING OPTIMUM BRANCHINGS[†]

R. Endre Tarjan

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California at Berkeley

March 1975

Abstract

Edmonds has devised an efficient algorithm to find an optimum branching in a directed graph. We give an implementation of this algorithm which runs in $O(e \log n)$ time if the problem graph has n vertices and e edges. A modification for dense graphs gives a running time of $O(n^2)$. We also show that the unmodified algorithm runs in $O(n(\log n)^2 + e)$ time on an average graph, assuming a reasonable probability distribution.

Keywords: graph algorithm, equivalence algorithm, disjoint branching, minimum spanning tree, priority queue, average time analysis

[†]Research sponsored by National Science Foundation Grant GJ-35604X1 and by a Miller Research Fellowship.

1. Introduction

A directed graph $G = (V,E)$ consists of a set of vertices V of size $|V| = n$ and a set of edges E of size $|E| = e$. Each edge is an ordered pair (v,w) of distinct vertices v, w, called the endpoints of the edge. A semipath joining x_1 and x_k in G is a sequence of vertices $x_1,x_2,\ldots,x_k$ such that $(x_i,x_{i+1}) \in E$ or $(x_{i+1},x_i) \in E$ for each i, $1 \leq i < k$. A set of vertices $W \subseteq V$ is weakly connected if there is a semipath of vertices in W joining any pair of vertices in W. A maximal weakly connected set of vertices is a weakly connected component of G.

A path from x_1 to x_k in G is a sequence of edges $(x_1,x_2)(x_2,x_3)\cdots(x_{k-1},x_k)$. The path is simple if $x_1,\ldots,x_{k-1}$ are distinct. The path is a cycle if $x_1 = x_k$. A set of vertices $S \subseteq V$ is strongly connected if there is a path whose edges have endpoints in S from any vertex in S to any other vertex in S. A maximal strongly connected set is a strongly connected component of G.

A branching B of G is a set of edges such that

(i) if (x_1,y_1), (x_2,y_2) are distinct edges of B then $y_1 \neq y_2$;

(ii) B does not contain a cycle.

Given a real value $c(v,w)$ defined for each edge of G, we desire a branching B such that $\sum_{(v,w)\in B} c(v,w)$ is maximum. Such a set B is called an optimum branching.

Edmonds [2] has given an efficient algorithm for finding an optimum branching in a graph G. Karp [5] has provided a combinatorial

proof that the algorithm is correct (Edmonds' original proof is based on linear programming theory). This paper gives an efficient implementation of the algorithm. If G has n vertices and e edges the algorithm runs in $O(e \log n)$ time. A modification for dense graphs gives an $O(n^2)$ running time. Also, assuming a reasonable probability measure, the algorithm runs in $O(n(\log n)^2 + e)$ time on the average. These time bounds assume $e \geq kn$ for some constant k. (This is not a serious restriction; any weakly connected graph has $e \geq n-1$.)

2. Edmonds' Algorithm for Optimum Branchings

Let $G = (V,E)$ be a weakly connected directed graph, having edge values $c(v,w)$. Edmonds' algorithm finds an optimum branching in G. The algorithm constructs a set of edges H defining a subgraph $G(H) = (V,H)$. At all times during the construction, $G(H)$ is such that for each strongly connected component S, there is at most one edge $(v,w) \in H$ such that $w \in S$, $v \in V-S$. If S is such that no edge $(v,w) \in H$ such that $w \in S$, $v \in V-S$ we call S a root component of $G(H)$.

Initially $H = \emptyset$; thus initially each vertex in V defines both a weakly connected and a strongly connected component of $G(H)$. Here is Edmonds' algorithm for constructing the set H.

Algorithm BRANCH:

First Step:

F1: Choose any vertex v with an edge (x,v) of value $c(x,v) > 0$.

F2: Select the edge (u,v) of largest value.

F3: Add (u,v) to H.

General Step:

G1: Choose any root component S of G(H) having an unexamined edge (x,v) with $v \in S$ and $c(x,v) > 0$.

G2: Find the largest unexamined edge (u,v) such that $v \in S$.

G3: If $u \in S$, discard the edge and stop. Otherwise go to G4.

G4: $u \notin S$. Let W be the weakly connected component of G(H) containing v. If $u \notin W$ add (u,v) to H and stop. Otherwise go to G5.

G5: $u \notin S$, $u \in W$. Find the sequence $S_1,(x_1,y_1),S_2,(x_2,y_2),\ldots,S_k,(x_k,y_k)$ such that each S_i is a strongly connected component of G(H), $(x_i,y_i) \in H$, $y_i \in S_i$, and $x_i \in S_{i+1}$ for all i, $S_k = S$, $(x_k,y_k) = (u,v)$, and $x_k \in S_1$.

G6: Find the edge (x_j,y_j) with minimum value among the (x_i,y_i).

G7: For each unexamined edge (x,y) with $y \in S_i$, modify the value of (x,y) as follows:

$$c(x,y) := c(x,y) - c(x_i,y_i) + c(x_j,y_j) \quad .$$

G8: Add (u,v) to H. (This combines $S_1,\ldots,S_k$ into a single strongly connected component which is a root component of G(H).)

Repeat the general step until there is no root component S of G(H) having an unexamined edge (u,v) with $v \in S$ and $c(u,v) > 0$.

Suppose this algorithm is applied to a graph G. The following results are implicit in the work of Edmonds and Karp [2,5]. Let $G(H) = (V,H)$.

Lemma 1: Each strongly connected component S of G(H) has at most one edge $(v,w) \in H$ with $v \in S$, $u \in V-S$. Each weakly connected component W of G(H) contains exactly one root component.

Lemma 2: Let S be any root component of G(H), let W be the weakly connected component containing S, and let $v \in S$. Then for any $w \in W$ there is a unique simple path in G(H) from v to w.

These lemmas follow easily by induction on the number of edges added to H. When the algorithm is completed, H contains an optimum branching, which we can extract with the help of Lemma 2. All we need to do is determine a unique vertex v in the root component of each weakly connected component of G(H); these vertices determine a branching consisting of the simple paths given by Lemma 2. If we determine these vertices using the following algorithm, the resultant branching is optimum [2,5].

Algorithm ROOT:

R1: Find a root component R in G(H) containing more than one vertex.

R2: Find the sequence $S_1,(x_1,y_1)\cdots S_k,(x_k,y_k)$ determined in BRANCH, step G5 such that edge (x_k,y_k) was added to H to form R.

R3: Find the edge (x_j,y_j) of minimum value among the (x_i,y_i).

R4: Delete (x_j,y_j) from H (This step makes S_j a root component).

Repeat steps R1-R4 until every root component consists of a single vertex.

If BRANCH keeps track of each minimum edge (x_j,y_j) found in step G6 and of each strongly connected component formed in step G8, then ROOT is very easy to implement.

3. Efficient Implementation

We need several bookkeeping mechanisms to implement BRANCH efficiently. We must keep track of (a) the weakly connected components of G(H), (b) the strongly connected components of G(H), and (c) the unexamined edges entering each strongly connected component. For (a) and (b) we use a disjoint set union algorithm described in [4,10]. Given a collection of disjoint sets, the set union algorithm implements two operations:

(i) FIND(x) returns the name of the set containing element x;

(ii) UNION(A,B) adds the elements of set A to set B, destroying set B.

The time required for O(e) FIND's and O(n) UNION's is $O(n \log^* n + e)$ [10], where $\log^* n = \min \{i \mid \underbrace{\log \log \cdots \log}_{i \text{ times}} n \leq 1\}$. We use one collection of sets, manipulated by WFIND and WUNION, to represent the weakly connected components, and another collection of sets, manipulated by SFIND and SUNION, to represent the strongly connected components.

To keep track of the edges entering each strongly connected component, we use a priority queue mechanism described in [6,11]. Given a collection of elements, each with a value, and a collection of disjoint sets (called queues) of elements, the mechanism implements four operations:

(iii) QUNION(C,D) adds the elements in queue D to queue C.
Time required: $O(\log|C| + \log|D|)$

(iv) MAX(C) returns the largest element in queue C, deleting this element from the queue. (If C is empty, MAX(C) returns a dummy element with value $-\infty$.)
Time required: $O(\log|C|)$

(v) INIT(C,L) initializes a queue C to contain all elements in the list L.

Time required: $O(|L|)$

(vi) ADD(a,C) adds a constant a to the value of all elements in queue C.

Time required: $O(1)$

Here is an implementation of the optimum branching algorithm, expressed in Algol-like notation. In the program, the set roots gives the root components of G(H) which may have entering edges of positive value. The set rset gives the root components of G(H) with no entering edges of positive value. For each i, enter(i) gives the unique edge in H entering strongly connected component i. (If there is no such entering edge, enter(i) = (0,0).)

The array min(i) gives, for each i, the root vertex which algorithm ROOT will select if applied to root component i. The variables val and vertex are used to select and record the minimum edge found in step G6. The algorithm assumes that the graph $G = (V,E)$ has vertex set $V = \{1,2,\ldots,n\}$, and that, for each j, $I(j) = \{(i,j) \in E\}$ is an incidence list for vertex j.

```
algorithm BRANCH;
   begin
      roots := ∅;
      for i := 1 until n do
         begin
            INIT(i,I(i));
            initialize an S-set named i containing i as its only element;
            initialize a W-set named i containing i as its only element;
            enter(i) := (0,0);
            roots := roots∪{i};
            min(i) := i;
         end;
      H := ∅;
      rset := ∅;
      while roots ≠ ∅ do
         begin
            delete some entry k from roots;
            (i,j) := MAX(k);
            if v(i,j) ≤ 0 then rset := rset∪{k}
            else if SFIND(i) ≠ k then
               begin
                  H := H∪{(i,j)};
                  if WFIND(i) ≠ WFIND(j) then
                     begin
                        WUNION(WFIND(i),WFIND(j));
                        enter(k) := (i,j);
                     end
                  else
                     begin
                        val := ∞;
                        (x,y) := (i,j);
                        while (x,y) ≠ (0,0) do
                           begin
                              if c(x,y) < val then
                                 begin
                                    val := c(x,y);
                                    vertex := FIND(y);
                                 end;
```

```
                        (x,y) := enter(SFIND(x));
                     end;
                  ADD(val-c(i,j),k);
                  min(k) := min(vertex);
                  (x,y) := enter(SFIND(i));
                  while (x,y) ≠ (0,0) do
                     begin
                        ADD(val-c(x,y),SFIND(y));
                        QUNION(k,SFIND(y));
                        SUNION(k,SFIND(y));
                     end;
                  roots := roots∪{k};
end end end end BRANCH;
```

After BRANCH is executed, the set of vertices $R = \{\underline{\min}(i) \mid i \in \underline{rset}\}$ defines an optimum branching, which may be found in $O(n+e)$ time by using a search [9] to find the set of simple paths from the vertices in R to all other vertices.

The operations in algorithm BRANCH are of three types:

(a) priority queue operations;

(b) disjoint set operations;

(c) other operations.

The priority queue operations required are: $O(n)$ INIT operations, one for each vertex; $O(e)$ MAX operations, and $O(n)$ QUNION operations. Thus the total time for the priority queue operations is $O(e \log n)$. The disjoint set operations required are: $O(e)$ WFIND's, $O(e)$ SFIND's, $O(n)$ WUNION's, and $O(n)$ SUNION's (the set H contains at most $2n-2$ edges). Thus the total time for the set operations is $O(n \log^* n + e)$. The total time for other operations is clearly $O(n+e)$. Thus the total time required to find an optimum branching is $O(e \log n)$ (assuming $e \geq kn$ for some constant k). The space required is $O(e)$.

4. A Modification for Dense Graphs

The time bound given in Section 2 for the optimum branching algorithm is $O(n^2 \log n)$ if the graph G is dense; i.e. e is $\Omega(n^2)$. However, by changing the priority queue representation, we can get the algorithm to run in $O(n^2)$ time. This modification is analogous to that used in a well-known $O(n^2)$ minimum spanning tree algorithm [1,7].

We use a different mechanism to represent the priority queues. For each strongly connected component S, we have a list of edges (i,j), at most one for each i, such that $i \notin S$, $j \in S$, and for all edges (i,k) such that $k \in S$, $c(i,k) \geq c(i,j)$. This list is ordered on i, and takes the place of the priority queue for component S. Associated with each edge (i,j) in the list is its current value $c(i,j)$.

Since such a list has length at most n, a MAX operation requires $O(n)$ time. Furthermore, since every edge (i,j) on such a list has $i \notin S$, algorithm BRANCH executes only $O(n)$ MAX operations, and $O(n^2)$ time total is required for the MAX operations.

A QUNION(C,D) operation requires $O(n)$ time plus time for $O(n)$ SFIND's. Thus the total time for the QUNION operations is $O(n^2)$. ($O(n^2)$ SFIND's plus $O(n)$ SUNION's require $O(n^2)$ time [10].) An INIT($|L|$) operation requires $O(n)$ time since $|L| \leq n$ (the edges $(i,j) \in L$ may be sorted on the value of i in $O(|L|+n)$ time by using a radix sort [6]). Thus the total time for the INIT operations is $O(n^2)$, and the running time of the modified version of BRANCH is $O(n^2)$. The space required is still $O(e)$.

5. Average Time Analysis

In this section we show that the average running time of the unmodified version of BRANCH is $O(n(\log n)^2 + e)$ assuming a reasonable probability measure. The analysis is analogous to that used in [8] to show the existence of an all-pairs shortest path algorithm with an average running time of $O((n \log n)^2)$.

Let G be selected according to some probability distribution from among the $\binom{n\cdot(n-1)}{e}$ labelled graphs having vertex set $\{1,2,\ldots,n\}$ and e edges. Assume that, for all j, the probability of the existence of an edge (i,j) in G is the same for each i. For each j, $1 \leq j \leq n$, let p_j be a real-valued probability distribution, and let the values of the edges (i,j) in G be independent random variables with probability distribution p_j.

With this definition of a random graph, the probability that there exists an edge $(i,j) \in E$ and that (i,j) has minimum value among edges in the set $\{(k,j) \mid (k,j) \in E\}$ is the same for each i. Furthermore, at any time during the execution of BRANCH, the probability that there exists an edge $(i,j) \in E$ and that (i,j) has minimum value among unexamined edges in the set $\{(k,j) \mid (k,j) \in E\}$ is the same for each i such that (i,j) has not yet been returned by a call on MAX. This follows from the fact that the only information known about (i,j) is that if (i,j) is an edge in G then its value must be greater than that of all edges returned by previous MAX calls on queues corresponding to root components containing vertex j; and this information is independent of i.

We now derive an upper bound on the average number of MAX calls used by BRANCH and hence on the average running time of BRANCH. Suppose

MAX(k) is executed for some root component k containing m vertices. Then the probability of MAX(k) returning an edge (i,j) with i outside component k is at least $\frac{n-m}{n}$. Thus an upper bound on $\bar{M}_k$, the average number of MAX(k) operations before a new edge is added to H, is

$$\bar{M}_k \leq \left(\frac{n-m}{n}\right) \sum_{i=1}^{\infty} i\left(\frac{m}{n}\right)^{i-1} = \frac{n}{n-m}$$

The maximum number of edges which can be added to H after some root component reaches size m is 2(n-m). It follows that an upper bound on $\bar{M}$, the average total number of executions of MAX, is

$$\bar{M} = \Sigma \bar{M}_k \leq \sum_{m=1}^{n-1} \frac{2n}{n-m} = O(n \log n)$$

and the average time required by all the MAX operations is $O(n(\log n)^2)$. Combining this with the worst case time bounds on the other parts of the algorithm gives an $O(n(\log n)^2 + e)$ bound on the average running time of the algorithm.

We also have

$$\text{var}(M_k) \leq \overline{2M_k^2} \leq \frac{n-m}{n} \sum_{i=1}^{\infty} i^2\left(\frac{n}{m}\right)^{i-1} \leq \frac{2n^2}{(n-m)^2},$$

and

$$\text{var}(M) \leq \sum_{m=1}^{n-1} \frac{4n^2}{(n-m)^2} = O(n^2) .$$

Thus the standard deviation of M is O(n), and the probability

that the number of executions M of MAX is greater than $\bar{M}+kn$ is $0(1/k^2)$. Thus the probability that the running time of BRANCH is not $0(n(\log n)^2+e)$ is $0(\frac{1}{(\log n)^2})$.

6. Remarks

We have presented an implementation of Edmonds' optimum branching algorithm which has a worst-case running time of $O(e \log n)$, an average running time of $O(n(\log n)^2 + e)$, and can be modified to run in $O(n^2)$ time, which is an improvement if the graph is dense. One naturally wonders if these results are best possible. Recently discovered algorithms for finding minimum spanning trees in undirected graphs achieve a time bound of $O(e \log \log n)$ [11,12]; they can be used to find minimum spanning trees in $O(n \log n \log \log n + e)$ time on the average, using the result that a random graph with more than $\frac{n \log n}{2}$ edges is probably connected [3]. These results suggest the following research questions:

Can optimum branchings be found in $o(e \log n)$ time?

If not, how can one prove that <u>any</u> optimum branching algorithm requires $\Omega(e \log n)$ time in the worst case?

References

[1] E.W. Dijkstra, "A note on two problems in connection with graphs," Numerische Mathematik 1 (1959), 269-271.

[2] J. Edmonds, "Optimum branchings," Journal of Research of the National Bureau of Standards 71B (1967), 233-240.

[3] P. Erdös and A. Rényi, "On the evolution of random graphs," Pub. of the Math. Inst. of the Hung. Acad. of Sciences 5 (1960), 17-60.

[4] J. Hopcroft and J.D. Ullman, "Set-merging algorithms," SIAM J. Comput. 2 (1973), 294-303.

[5] R.M. Karp, "A simple derivation of Edmonds' algorithm for optimum branchings," Networks 1 (1971), 265-272.

[6] D. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass. (1973).

[7] R.C. Prim, "Shortest connection networks and some generalizations," Bell System Tech. J. (1957), 1389-1401.

[8] P. Spira, "A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$," SIAM J. Comput. 2 (1973), 28-32.

[9] R. Tarjan, "Depth-first search and linear graph algorithms," SIAM J. Comput. 1 (1972), 146-160.

[10] R. Tarjan, "Efficiency of a good but not linear set union algorithm," J. ACM, to appear.

[11] R. Tarjan, "Finding minimum spanning trees," SIAM J. Comput., submitted.

[12] A. Yao, "An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees," Info. Proc. Letters, to appear.