

NETWORK FLOW AND TESTING GRAPH CONNECTIVITY*

SHIMON EVEN† AND R. ENDRE TARJAN‡

Abstract. An algorithm of Dinic for finding the maximum flow in a network is described. It is then shown that if the vertex capacities are all equal to one, the algorithm requires at most $O(|V|^{1/2} \cdot |E|)$ time, and if the edge capacities are all equal to one, the algorithm requires at most $O(|V|^{2/3} \cdot |E|)$ time. Also, these bounds are tight for Dinic's algorithm.

These results are used to test the vertex connectivity of a graph in $O(|V|^{1/2} \cdot |E|^2)$ time and the edge connectivity in $O(|V|^{5/3} \cdot |E|)$ time.

Key words. Dinic's algorithm, maximum flow, connectivity, vertex connectivity, edge connectivity

1. Network flow. Let $G(V, E)$ be a finite directed graph, where V is the set of vertices and E is the set of edges. Each edge e is assigned a capacity $c(e) \geq 0$. One of the vertices, s , is called the *source*, and another, t , is called the *sink*. We seek a flow function $f(e)$ on the edges such that for every e , $c(e) \geq f(e) \geq 0$ and such that the total flow which enters a vertex, other than s or t , will equal the total flow which leaves the vertex. Of all such flows, we want one for which the net total flow which emanates from s is maximum.

This well-known network flow problem [1] was recently reexamined. A solution in $O(n^5)$ steps, where n is the number of vertices, was produced by Edmonds and Karp [2] in 1969. A solution in $O(|V|^2 \cdot |E|)$ steps was published in Russian by Dinic [3] in 1970.

In this section we present a solution in $O(|V|^2 \cdot |E|)$, essentially the same as Dinic's. (This version was discovered independently by S. Even and J. Hopcroft.)

The algorithm runs in phases, at most $|V| - 1$ in number. We start with zero flow; that is, $f(e) = 0$ for every $e \in E$. In each phase, the flow is increased. New phases are applied until no increase is possible. At that point, the proof of maximality is the same as that of Ford and Fulkerson [1], and it will not be repeated here. However, the algorithm up to that point is not a restriction of the freedom allowed by the Ford and Fulkerson algorithm—as is the case with the Edmonds and Karp algorithm. The computation within each phase is through a different method of labeling and path finding.

Assume that we have a present flow $f(e)$. An edge is *usable* in the *forward direction* if $f(e) < c(e)$, and it is usable in the *backward direction* if $f(e) > 0$. Clearly, an edge may be usable in both directions.

Each phase starts with a breadth-first search from s . That is, we start by labeling s with 0; i.e., $\lambda(s) = 0$. Next, we label with 1 all unlabeled vertices which are reachable from s via a single usable edge, where the usable direction is from s to

* Received by the editors June 27, 1974, and in revised form November 15, 1974.

† Computer Science Department, Technion-Israel Institute of Technology, Haifa, Israel. On leave of absence from the Department of Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. Parts of this work were completed during the summers of 1972 and 1973 while he visited the Department of Computer Science, Cornell University, Ithaca, New York.

‡ Computer Science Division, University of California at Berkeley, Berkeley, California 94720. The work of this author was supported in part by the National Science Foundation under Grant NSF-GJ-35604X, and by a Miller Research Fellowship.

them. This action is called *scanning* s . We scan the vertices in the order they are labeled; that is, first-labeled, first-scanned. When v is scanned, all unlabeled vertices reachable from v are labeled $\lambda(v) + 1$. Once t is labeled while scanning, say, w , we continue scanning all labeled but unscanned vertices v for which $\lambda(v) = \lambda(w)$, and terminate the breadth-first search once we reach a vertex v , waiting to be scanned, for which $\lambda(v) > \lambda(w)$.

It is easy to see that this is nothing but the well-known algorithm [4] for finding a shortest path from s to t when length is measured by the number of edges on the path. Edges are used only in a usable direction, and every shortest path indicates an augmenting path for increasing the flow.

As we conduct the breadth-first search, we prepare a copy of all vertices and edges traced. For every vertex v , we keep a list of the edges which are usable from v to vertices which are labeled $\lambda(v) + 1$. The total number of steps required for this is $O(|E|)$ if the data structure of the graph is originally by lists of adjacent edges for each vertex. Let us call the newly prepared structure the *auxiliary graph*. All paths from s to t in the auxiliary graph are of length $\lambda(t)$ edges.

Now we use the auxiliary graph to trace flow augmenting paths, from s to t . These paths are found by depth-first search [5], [6]. We start tracing from s , move through a usable edge to a vertex labeled 1, move from there to a vertex labeled 2, etc. If we reach t , we have an augmenting path, and we push through it as much flow as is possible. All edges along the path which cease to be usable (in the direction used) due to this change in the flow are erased from the auxiliary graph, and a new depth-first search is started. Clearly, each time an augmenting path is used, at least one edge is removed from the auxiliary graph. (Such an edge is called a *bottleneck* of the path.) If the depth-first search ends in a dead-end, namely, a vertex v from which no usable edge leads to a vertex whose label is $\lambda(v) + 1$, then we retrace to the vertex preceding v on the path and erase the last edge from the path and from the auxiliary graph. We continue the search from there. If we cannot proceed from s the phase is over.

Finding one successful path takes $O(\lambda(t))$ steps, and in the case of failure (when we retrace), the number of steps cannot exceed $O(\lambda(t))$. In either case, at least one edge is erased. Thus the total number of steps in tracing paths during one phase is bounded by $O(|V| \cdot |E|)$. It follows that each phase cannot take more than $O(|V| \cdot |E|)$ steps. We shall show that the number of phases is bounded by $|V| - 1$, and therefore the whole algorithm does not take more than $O(|V|^2 \cdot |E|)$.

Each auxiliary graph, when first constructed, describes all shortest augmenting paths for the present $f(e)$. It has the property that there is no usable edge which leads, in a usable direction, from a vertex v to a vertex whose label is higher than $\lambda(v) + 1$. The changes in the flow performed by pushing flow through a shortest augmenting path may create a new usable direction for some of its edges, but these directions are from some v to a vertex labeled $\lambda(v) - 1$. Thus the property remains valid through all the changes during the phase. It follows that at the end of the phase, a shortest augmenting path is of length higher than $\lambda(t)$. Thus, from phase to phase, $\lambda(t)$ increases, and therefore the number of phases is bounded by $|V| - 1$.

In the last phase, the labeling does not reach t , and the proof of maximality (which brings up the max-flow min-cut theorem) is identical with that of Ford and Fulkerson [1].

2. Zero-one network flow. Consider now a network flow problem, as above, except that for all $e \in E$, $c(e) = 1$. (One should realize that even if for all $e \in E$, $c(e)$ is integral, not necessarily 1, the algorithm described above will never introduce fractions.¹) It follows that for all flow functions along the way and for every e , $f(e)$ is either zero or one.

When we trace an augmenting path, the increase in the flow through it is exactly 1, and all edges used in it are erased from the auxiliary graph; that is, each edge, when used, is a bottleneck. Also, each time we backtrack one edge, it is erased. It follows that the number of steps per phase is at most $O(|E|)$, and the total number of steps the algorithm requires is bounded by $O(|V| \cdot |E|)$.

Another bound, $O(|E|^{3/2})$, which is better for sparse graphs can be proved, but we have to prepare a few tools first.

Let $G(V, E)$ be a network with integral edge capacities $c(e)$. Assume the maximum total flow from s to t is M . Also, assume that through Dinic's algorithm (or any other algorithm which does not introduce fractions), the flow has been increased from zero to a present flow function f , and the present total flow from s to t is F . Define now the network $\tilde{G}(V, \tilde{E})$ with capacities $\tilde{c}(e)$ as follows:

- (i) If $e \in E$ and $f(e) < c(e)$, then $e \in \tilde{E}$ and $\tilde{c}(e) = c(e) - f(e)$.
- (ii) If $e \in E$ and $f(e) > 0$ then $e' \in \tilde{E}$, where e' connects between the same two vertices as e , but in the reverse direction, and $\tilde{c}(e') = f(e)$.

Clearly, each edge of G generates at least one edge in \tilde{G} , and if $0 < f(e) < c(e)$, e generates two edges. However, in the case that $c(e) = 1$ for every edge e , each edge generates exactly one edge in \tilde{G} , since $f(e)$ can be either 0 or 1.

We shall use the following notation: $(S; \bar{S})_G$ is a cut separating s from t in G ; that is, $S \cup \bar{S} = V$, $S \cap \bar{S} = \emptyset$, $s \in S$, $t \in \bar{S}$ and $(S; \bar{S})$ is the set of edges in G which lead from a vertex in S to a vertex in \bar{S} .

LEMMA 1. *The maximum flow in \tilde{G} is $M - F$.*

Proof. The definition of \tilde{G} implies that

$$\sum_{a \in (S; \bar{S})_{\tilde{G}}} \tilde{c}(a) = \sum_{a \in (S; \bar{S})_G} (c(a) - f(a)) + \sum_{e \in (\bar{S}; S)_G} f(e),$$

However,

$$F = \sum_{a \in (S; \bar{S})_G} f(a) - \sum_{e \in (\bar{S}; S)_G} f(e),$$

Thus

$$\sum_{a \in (S; \bar{S})_{\tilde{G}}} \tilde{c}(a) = \sum_{a \in (S; \bar{S})_G} c(a) - F,$$

This implies that a minimum cut of \tilde{G} corresponds to a minimum cut of G (namely, is defined by the same S). By the max-flow min-cut theorem, the value of the minimum cut of G is M . Thus the value of a minimum cut in \tilde{G} is $M - F$. Again, by the max-flow min-cut theorem, the maximum flow in \tilde{G} is $M - F$. Q.E.D.

LEMMA 2. *Let $G(V, E)$ be a network in which $c(e) = 1$ for every $e \in E$. Assume the maximum flow from s to t is M . The distance from s to t when the flow is zero everywhere is at most $|E|/M$.*

¹ This holds for all "reasonable" algorithms for network flow problems.

Proof. Let $V_i = \{v|v \text{ is at distance } i \text{ from } s\}$. Here the distances are with zero flow and V_i corresponds to the set of vertices on the i th level of the first phase of Dinic's algorithm. Let l be the distance from s to t . The set of edges from V_i to V_{i+1} is a cut, and therefore the number of edges between V_i and V_{i+1} is at least M . Thus

$$l \cdot M \leq |E|. \qquad \text{Q.E.D.}$$

THEOREM 1. *For networks with unit edge capacities, Dinic's algorithm requires at most $O(|E|^{3/2})$ steps.*

Proof. If $M \leq |E|^{1/2}$, then the number of phases is bounded by $|E|^{1/2}$, and the result follows. Otherwise, consider the phase during which the flow reaches the value $M - |E|^{1/2}$. The value of the flow, F , when the auxiliary graph for this phase is constructed is less than $M - |E|^{1/2}$. However, this auxiliary graph is identical with the initial auxiliary graph for the network \tilde{G} . \tilde{G} still has unit edge capacities, and by Lemma 1 its maximum flow is

$$\tilde{M} = M - F > M - (M - |E|^{1/2}) = |E|^{1/2}.$$

Thus, by Lemma 2, the length, l , of a shortest augmenting path satisfies

$$l \leq \frac{|E|}{\tilde{M}} \leq |E|^{1/2}.$$

Thus the number of phases up to this point is at most $|E|^{1/2} - 1$, and since the number of phases to completion is at most $|E|^{1/2}$, the total number of phases is at most $2|E|^{1/2}$. Q.E.D.

A network is of *type 1* if it satisfies the following conditions:

- (i) All (edge) capacities are equal to 1.
- (ii) There are no parallel edges; that is, an edge is identified by its start and end vertices.

LEMMA 3. *Let $G(V, E)$ be a network of type 1, with maximum flow M from s to t . The distance from s to t when the flow is zero everywhere is at most $2|V|/\sqrt{M}$.*

Proof. Let V_i and l be as in the proof of Lemma 2. Since the network is of type 1, we have $|V_i| \cdot |V_{i+1}| \geq M$. Thus, for all $0 \leq i < l$, either

$$|V_i| \geq \sqrt{M} \quad \text{or} \quad |V_{i+1}| \geq \sqrt{M}.$$

Since $\sum_{i=0}^l |V_i| \leq |V|$, we have

$$\left\lfloor \frac{l+1}{1} \right\rfloor \cdot \sqrt{M} \leq |V|,$$

and $l \leq 2|V|/\sqrt{M}$. Q.E.D.

THEOREM 2. *For networks of type 1, Dinic's algorithm requires at most $O(|V|^{2/3} \cdot |E|)$ steps.*

Proof. The proof is similar to that of Theorem 1: if $M \leq |V|^{2/3}$, the result follows immediately. Let F be the flow when the auxiliary graph for the phase during which the flow reaches the value $M - |V|^{2/3}$ is constructed. Again, this auxiliary graph is identical to the initial auxiliary graph for the network \tilde{G} . \tilde{G} may not be of type 1 since it may have parallel edges, but it can have at most two

parallel edges from one vertex to another.² By Lemma 1, $\tilde{M} > |V|^{2/3}$. By a variation of Lemma 3, the length l of a shortest augmenting path satisfies

$$l \leq \frac{2\sqrt{2}|V|}{\sqrt{|V|^{2/3}}} = 2\sqrt{2} \cdot |V|^{2/3}.$$

Thus the number of phases up to this point is at most $O(|V|^{2/3})$ and since the number of phases to completion is at most $|V|^{2/3}$, the total number of phases is at most $O(|V|^{2/3})$. Q.E.D.

In certain applications, we need upper bounds on the flow through vertices. This restriction can be translated to bounds on the flow through edges as follows: each vertex v which has a vertex capacity $c(v)$ is split into two vertices, v' and v'' , which have no explicit upper bound on the flow through them; a new edge e connects from v' to v'' and $c(e) = c(v)$; all edges which formerly led to v now lead to v' , and all edges which emanated from v now emanate from v'' . Clearly, the new edge e and its capacity implicitly specify the upper bound on the flow through v .

A network is of type 2 if all (edge) capacities are equal to 1 and every vertex v other than s or t either has a single edge emanating from it or has a single edge entering it.

One important source of such networks is the case of networks with vertex unit capacities for all vertices other than the source and the sink, which were translated into edge capacities as above. Even if the original network had no edge capacities (∞), all edge flows (assuming no edges go directly from s to t) are implicitly bounded by 1.

LEMMA 4. *Let $G(V, E)$ be a network of type 2 with maximum flow M from s to t . The distance from s to t when the flow is zero everywhere is at most $(|V| - 2)/M + 1$.*

Proof. The structure of G implies that a flow in G can be decomposed into vertex-disjoint directed paths from s to t .³ The number of these paths is equal to the value of the flow. Assume we have a flow function f which achieves M . Let l be the length of a shortest path among the paths implied by f . Thus each path uses at least $l - 1$ intermediate vertices. We have

$$M \cdot (l - 1) \leq |V| - 2. \qquad \text{Q.E.D.}$$

LEMMA 5. *If G is a network of type 2 and the present flow function is f , then \tilde{G} is also of type 2.*

Proof. If there is no flow through v (per f), then v still satisfies the condition that there is a single edge entering it or a single edge emanating from it. If the flow going through v is 1, assume it enters via e_1 , and leaves via e_2 . In \tilde{G} both these edges do not appear, but each gives rise to an edge in the reverse direction. The other edges of G which are incident to v remain intact in \tilde{G} . Thus the number of incoming edges and the number of outgoing edges of v did not change. Q.E.D.

² In G , we may have antiparallel edges; that is e and e' , where both connect between the same two vertices but in opposite directions. One of them may stay in \tilde{G} while the other gives rise to an edge which is parallel to the first.

³ Namely, no two paths share a vertex except s and t . In addition, the flow may imply directed cycles which are of no interest to us.

THEOREM 3. *For a network of type 2, Dinic's algorithm requires at most $O(|V|^{1/2} \cdot |E|)$ steps.*

Proof. If $M \leq |V|^{1/2}$, then the number of phases is bounded by $|V|^{1/2}$, and the result follows. Otherwise, consider the phase during which the flow reaches the value $M - |V|^{1/2}$. Thus the value of the flow, F , when the auxiliary graph for this phase is constructed is less than $M - |V|^{1/2}$. However, this auxiliary graph is identical to the initial auxiliary graph for the network \tilde{G} . By Lemma 5, \tilde{G} is also of type 2. By Lemma 1, the maximum flow in \tilde{G} is greater than $|V|^{1/2}$. By Lemma 4, the length, l , of a shortest augmenting path satisfies

$$l \leq \frac{|V| - 2}{|V|^{1/2}} + 1 = O(|V|^{1/2}).$$

Thus the number of phases up to this point is at most $O(|V|^{1/2})$. Since the number of phases to completion is at most $|V|^{1/2}$, the total number of phases is at most $O(|V|^{1/2})$. Q.E.D.

3. Applications. We want to point out two areas of applications of the results of the previous sections. They are:

- (i) matching in the bipartite graph;
- (ii) connectivity of a graph.

The best known algorithm for finding a maximum matching in a bipartite graph is that of Hopcroft and Karp [7]. Their algorithm takes at most $O(n^{2.5})$ steps; it is a variant of the Hungarian method and is very close to Dinic's algorithm, in spite of the fact that they do not use the network-flow formulation. In fact, we have borrowed the idea for the bounds of the previous section from them. However, their result can be viewed as a special case of Theorem 3. One can use the network-flow approach to solve the maximum matching in the bipartite graph [1], and the network is of type 2.

In the remainder of this section we shall discuss the testing of connectivity in a graph.

Let $G(V, E)$ be a finite undirected graph. We assume that G has no self-loops. A set of vertices, S , is called an (a, b) vertex separator if $\{a, b\} \subset V - S$ and every path connecting a and b passes through at least one vertex of S . Let $N(a, b)$ be the least cardinality of an (a, b) vertex separator, assuming one exists.⁴ It is a theorem that $N(a, b)$ is equal to the maximum number of vertex disjoint paths connecting a with b . This theorem is well known and is one of the variations of Menger's theorem [8]. It is not only reminiscent of the max-flow min-cut theorem, but in fact can be proved by it. Dantzig and Fulkerson [9] pointed out this relationship, and their proof offers an algorithm to determine $N(a, b)$. This is done as follows:

Construct a directed network flow graph $\bar{G}(\bar{V}, \bar{E})$, where $\bar{V} = V$ and \bar{E} is a set of directed edges; for each $e \in E$, we have e' and e'' in \bar{E} , where e' and e'' connect between the two end vertices of e and are directed in opposite directions. Each v , other than a and b , has vertex capacity 1. These vertex capacities can now be translated to edge capacities, as was pointed out in the previous section. The maximum flow in this network is equal to $N(a, b)$. This last network is of type 2, and therefore Dinic's algorithm achieves this result in at most $O(|V|^{1/2} \cdot |E|)$ steps. (See Theorem 3).

⁴ Clearly, if a and b are connected by an edge, then no (a, b) vertex separator exists.

The *vertex-connectivity*, c , of G is defined in the following way:

- (i) If G is completely connected,⁵ then $c = |V| - 1$.
- (ii) If G is not completely connected, then $c = \min_{a,b} N(a, b)$.

(If G is not completely connected, then the minimum value of $P(a, b)$, where $P(a, b)$ is the maximum number of vertex disjoint paths connecting a and b , will be equal to $\min_{a,b} N(a, b)$, in spite of the fact that $N(a, b)$ is only defined for pairs which are not connected by an edge.)

The obvious way, then, to find c if G is not completely connected is to compute $N(a, b)$ for all pairs a, b which are not connected by an edge. This leads to at most $O(|V|^2)$ computations, and each requires at most $O(|V|^{1/2} \cdot |E|)$ steps. Hence at most $O(|V|^{2.5} \cdot |E|)$ steps.

However, a slightly better bound can be proven.

LEMMA 6. *The (edge or vertex) connectivity, c , of an undirected graph $G(V, E)$ with no self-loops and no parallel edges satisfies $c \leq 2|E|/|V|$.*

Proof. The connectivity cannot exceed $\min_v d(v)$, where $d(v)$ is the degree of vertex v .⁶ Also,

$$\sum_v d(v) = 2 \cdot |E|,$$

Thus $c \leq 2|E|/|V|$. Q.E.D.

Now let us conduct the procedure in the following manner: we choose a vertex v_1 and compute $N(v_1, v)$ for each v not connected to v_1 by an edge; there are at most $|V|$ such computations. We repeat the computation for v_2, v_3 , etc. We terminate with v_k once k exceeds the minimum value of $N(a, b)$ observed so far, γ .

THEOREM 4. *The value γ resulting from the procedure above is equal to the connectivity, c .*

Proof. By Menger's theorem, there is a vertex separator S such that $|S| = c$. Thus at least one of the vertices v_1, v_2, \dots, v_{c+1} is not in the separator. Assume it is v_j . There is a vertex v such that $N(v_j, v) = c$. Clearly $\gamma \geq c$. Also $k > \gamma$. Thus $k \geq c + 1$. Therefore $\gamma = c$. Q.E.D.

Lemma 6 and Theorem 4 imply that $k \leq 2|E|/|V| + 1$. Thus the total number of steps of our procedure is at most $O(|V|^{1/2} \cdot |E|^2)$.

In case G is a directed graph, similar definitions and approach lead to the same result, except that for each v_1, v_2, \dots, v_k , we compute both $N(v_i, v)$ and $N(v, v_i)$ (which are now not necessarily the same) for each applicable v .⁷

A natural idea, in relation to these computations is to use the technique of Gomory and Hu [10]. They find the maximum flows between every two vertices in an undirected flow graph by solving only $|V| - 1$ flow problems. However, their technique is not applicable to directed graphs. Observe that the network flow problems we solve, even for the vertex connectivity of undirected graphs, are all directed. Thus this does not suggest an improvement.

Now let us consider the question of edge connectivity. Again, let $G(V, E)$ be a finite undirected graph. A set of edges, T , is called an *(a, b) edge separator* if

⁵ Each pair of vertices is connected by an edge. In this case, there are no vertex separators.

⁶ The degree of a vertex is the number of edges incident to it.

⁷ If there is an edge from v_i to v , $N(v_i, v)$ is not computed, and if there is an edge from v to v_i , $N(v, v_i)$ is not computed.

$\{a, b\} \subset V$ and every path connecting a and b passes through at least one edge of T . Let $M(a, b)$ be the least cardinality of an (a, b) edge separator. It is a theorem that $M(a, b)$ is equal to the maximum number of edge disjoint paths connecting a with b . This is another variation of Menger's theorem, and again, one can use the network flow approach to determine $M(a, b)$. Here, too, we may construct \bar{G} , but one can use the undirected graph, with edge capacities all equal to 1. Since \bar{G} is of type 1, both Theorem 1 and Theorem 2 provide upper bounds on the number of steps Dinic's algorithm will need. Thus

$$O(|E| \cdot \min \{|V|^{2/3}, |E|^{1/2}\})$$

is an upper bound on the number of steps for evaluating $M(a, b)$.

The *edge connectivity*, c' , of G is defined by $c' = \min_{a,b} M(a, b)$.

Let T be a minimum edge separator in G ; that is, $|T| = c'$. Let v be any vertex of G ; then every vertex v' on the other side of T satisfies $M(v, v') = c'$. Thus in order to determine c' , we can use

$$c' = \min_{v' \in V - \{v\}} M(v, v').$$

This takes at most $O(|V| \cdot |E| \cdot \min \{|V|^{2/3}, |E|^{1/2}\})$ steps.

The approach described above can be used to determine the edge connectivity for directed graphs, too, with the modification that for every v' , both $M(v, v')$ and $M(v', v)$ have to be computed. The same bounds follow.

It is interesting to note that using the technique of Gomory and Hu would yield the same bound for edge connectivity in the undirected case, when one uses Dinic's algorithm to solve each of the $|V| - 1$ flow problems.⁸ However, our observation is simpler and works for directed graphs as well.

4. Lower bounds. In this section we shall show that the upper bounds on Dinic's algorithm, discussed in §§ 1 and 2, are tight. Namely, in each case there are graphs for which the number of steps is as high as the upper bound.

N. Zadeh [11] showed a family of flow problems for which Edmonds and Karp's algorithm requires $O(n^3)$ augmenting paths and a total of $O(n^5)$ steps. The same family requires $O(|V|^2 \cdot |E|)$ steps when Dinic's algorithm is used, thus proving that the bound given in § 1 cannot be improved.

Let us now consider the problem of maximum matching for a family of bipartite graphs.

Let

$$\begin{aligned} X_m &= \{a_{ij} | 1 \leq j \leq i \leq m\}, & Y_m &= \{b_{ij} | 1 \leq j \leq i \leq m\}, \\ E'_m &= \{(a_{ij}, b_{ij}) | 1 \leq j \leq i \leq m\}, \\ E''_m &= \{(a_{ij}, b_{i,j+1}) | 1 \leq j < i \leq m\}, \\ E_m &= E'_m \cup E''_m. \end{aligned}$$

The bipartite graph $G_m(X_m, Y_m, E_m)$ is drawn for $m = 4$ in Fig. 1. Clearly, the maximum matching in this case is unique and is given by E'_m . The value of the

⁸ In the case of edge connectivity of undirected graphs their technique is applicable.

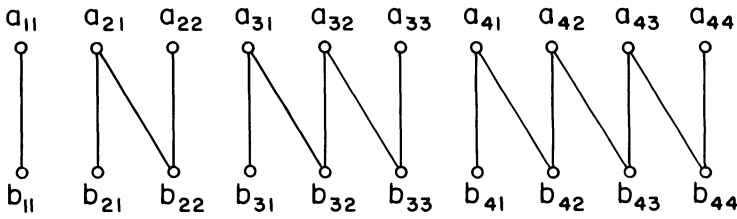


FIG. 1

matching is

$$\frac{m(m + 1)}{2} = O(m^2),$$

and the number of vertices in the graph is

$$|V| = m(m + 1) = O(m^2).$$

Assume now that we are looking for a maximum matching for G_m by using Dinic's algorithm, as suggested in § 3. The network is achieved by adding two new vertices: s , the source, and t , the sink. Also, connect s with each a_{ij} via a directed edge (s, a_{ij}) with capacity 1, direct all the edges of G_m from X_m to Y_m and assign each edge the capacity 1 (one may use here ∞ as well), and, finally, connect each b_{ij} to t via a directed edge (b_{ij}, t) again with capacity 1. The network is shown, for $m = 4$, in Fig. 2.

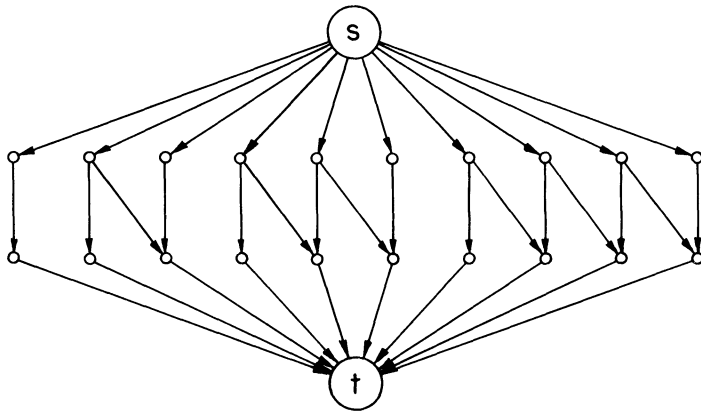


FIG. 2

The first phase of Dinic's algorithm may produce a unit flow in each of the edges of E_m'' . The corresponding matching is shown in Fig. 3, for $m = 4$, where the edges in the matching are represented by wiggly lines. In this case, the second phase will add $\{(a_{21}, b_{21}), (a_{22}, b_{22})\}$ to the matching, and subtract $\{(a_{21}, b_{22})\}$. In general, the i th phase will add to the matching the set

$$\{(a_{i1}, b_{i1}), (a_{i2}, b_{i2}), \dots, (a_{ii}, b_{ii})\}$$

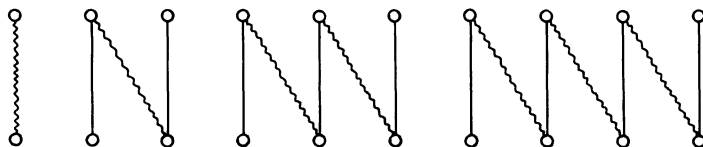


FIG. 3

and subtract the set

$$\{(a_{i1}, b_{i2}), (a_{i2}, b_{i3}), \dots, (a_{i,i-1}, b_{ii})\}.$$

Therefore the number of phases will be m . It is not hard to see that the total number of steps is $O(m^3)$. Since $|E| = O(m^2)$, these examples show that the bounds given by Theorems 1 and 3 are tight. Namely, the bound $O(|E|^{3/2})$ is the best possible, for graphs with unit edge capacities, in terms of $|E|$, and the bound $O(|V|^{1/2} \cdot |E|)$ is the best possible, for graphs of type 2 in terms of $|V|$ and $|E|$.

Clearly, for dense graphs of type 2, the bound $O(|V|^{1/2} \cdot |E|)$ is more informative than $O(|E|^{3/2})$, and one may wonder if $O(|V|^{1/2} \cdot |E|)$ still remains tight there. A family of dense graphs ($O(|E|) = O(|V|^2)$) for which this bound is still tight is achieved by adding to G_m the following set of edges:

$$\{(a_{ij}, b_{kl}) | (1 \leq j \leq i < k) \wedge (j \leq l \leq k \leq m)\}.$$

The steps of Dinic's algorithm can be chosen in such a way that none of these edges ever enters the matching. An examination reveals that

$$|E| = O(m^4), \quad |V| = O(m^2),$$

and the number of steps is $O(m^5)$.

Next, we want to show that the bound given by Theorem 2 cannot be improved either. Let

$$\begin{aligned} V_m = \{s, t\} &\cup \{a_i | 1 \leq i \leq m^3\} \cup \{b_i | 1 \leq i \leq m^3\} \\ &\cup \{c_i | 1 \leq i \leq m^2\} \cup \{d_{ij} | (1 \leq i \leq m^2) \wedge (1 \leq j \leq m)\} \\ &\cup \{e_i | 1 \leq i \leq m^2\}, \end{aligned}$$

and

$$\begin{aligned} E_m = \{(s, a_i) | 1 \leq i \leq m^3\} &\cup \{(e_i, t) | 1 \leq i \leq m^2\} \\ &\cup \{(a_i, b_j) | 1 \leq i, j \leq m^3\} \\ &\cup \{(b_i, c_j) | (1 \leq i \leq m^3) \wedge (1 \leq j \leq m^2)\} \\ &\cup \{(c_i, d_{i1}) | 1 \leq i \leq m^2\} \\ &\cup \{(d_{m^2,i}, e_j) | (1 \leq i \leq m) \wedge (1 \leq j \leq m^2)\} \\ &\cup \{(d_{ij}, d_{i+1,k}) | (1 \leq i < m^2) \wedge (1 \leq j, k \leq m)\}. \end{aligned}$$

The graph $G_m(V_m, E_m)$ is shown for the case $m = 2$ in Fig. 4. Now assume we want to find the maximum number of edge disjoint paths between s and t . If we use Dinic's algorithm for finding a maximum flow from s to t , where all edge capacities

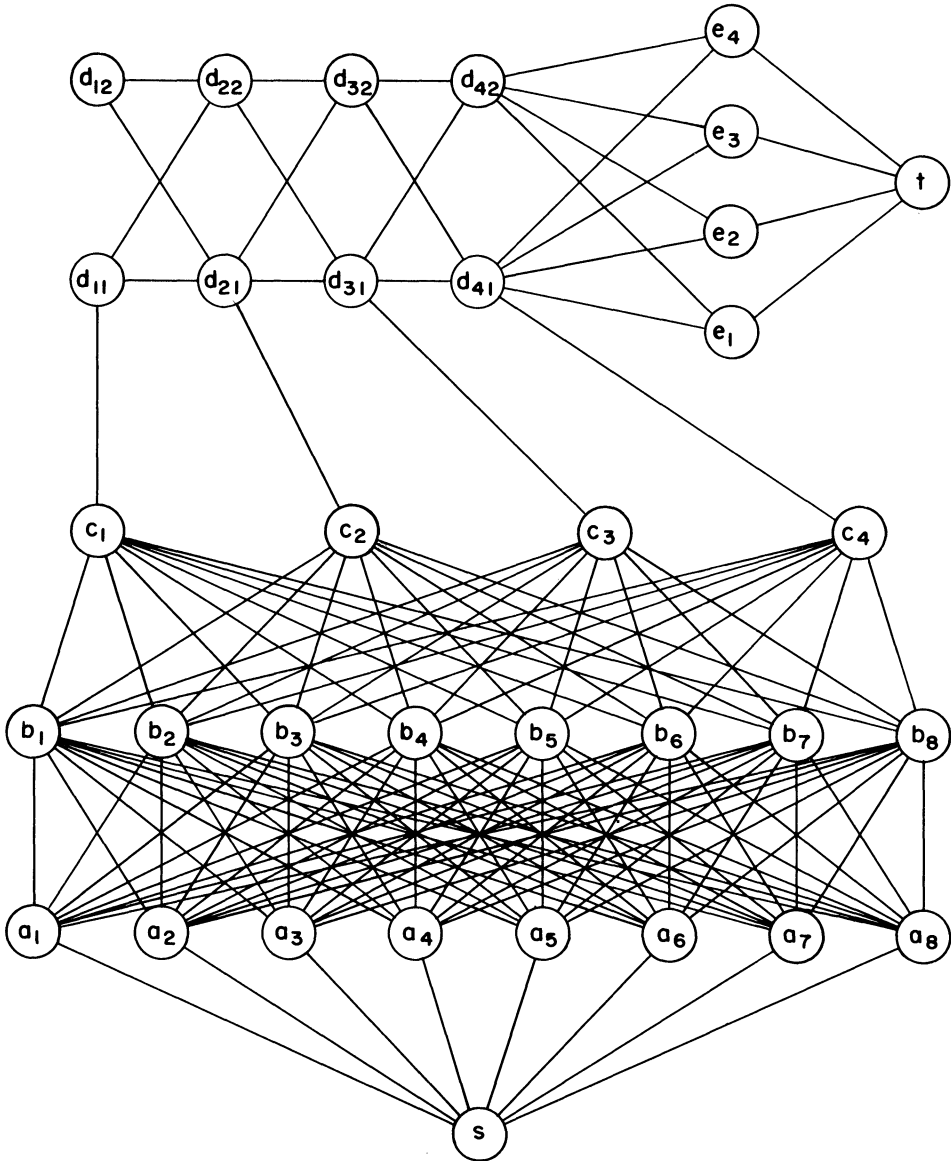


FIG. 4

are 1, we first find a path of length 6 (via $d_{m^2,1}$), then of length of 7, etc., up to a path of length $m^2 + 5$ (via d_{11}). The number of phases is therefore $O(m^2)$. The number of edges is $O(m^6)$, and so is the number of steps per phase. Thus the total number of steps is $O(m^8)$. Since $|E| = O(m^6)$ and $|V| = O(m^3)$, $O(m^8) = O(|V|^{2/3} \cdot |E|)$. This shows that the bound given by Theorem 2 is tight.

5. Remarks. One may make changes in Dinic's algorithm to produce an algorithm for which the lower bounds, as established by the examples of the previous section, are not equal to the upper bounds of § 2. However, for all the changes

we have tried, we could find other examples which showed that the upper bounds of § 2 are tight. Yet, let us show some results which seem to indicate that a better algorithm exists.

THEOREM 5. *For a network of type 2, the total length of all augmenting paths in Dinic's algorithm is at most $O(|V| \cdot \log |V|)$.*

Proof. The last augmenting path, by Lemma 4, is of length at most $(|V| - 2)/1 + 1$, the next to last is at most $(|V| - 2)/2 + 1$ long, etc. Since the number of augmenting paths is at most $|V| - 1$, the total length of the augmenting paths L satisfies

$$L \leq |V| - 1 + (|V| - 2) \cdot \sum_{i=1}^{|V|-1} \frac{1}{i} = O(|V| \cdot \log |V|).$$

THEOREM 6. *For a network of type 1, the total length of all augmenting paths in Dinic's algorithm is at most*

$$O(\min \{|V|^{3/2}, |E| \cdot \log |V|\}).$$

Proof. By a similar argument, following this time Lemma 3, and observing that the number of augmenting paths is again bounded by $|V| - 1$, we get

$$L \leq |V| - 1 + 2|V| \sum_{i=1}^{|V|-1} \frac{1}{\sqrt{i}} = O(|V|^{3/2}).$$

On the other hand, if we use Lemma 2, we get

$$L \leq |E| \cdot \sum_{i=1}^{|V|-1} \frac{1}{i} = O(|E| \cdot \log |V|) \quad \text{Q.E.D.}$$

It remains to be shown that one could trace all these paths without spending more time than their total lengths.

REFERENCES

- [1] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, N.J., 1962.
- [2] J. EDMONDS AND R. M. KARP, *Theoretical improvements in algorithmic efficiency for network flow problems*, J. Assoc. Comput. Mach., 19 (1972), pp. 248–264.
- [3] E. A. DINIC, *Algorithm for solution of a problem of maximum flow in a network with power estimation*, Soviet Math. Dokl., 11 (1970), pp. 1277–1280.
- [4] E. F. MOORE, *The shortest path through a maze*, Proc. of an Internat. Symp. on the Theory of Switching (April 1957), Harvard University Press, Cambridge, Mass., 1959, pp. 285–292.
- [5] J. HOPCROFT AND R. TARJAN, *Algorith 447: Efficient algorithms for graph manipulation*, Comm., ACM, 16 (1973), pp. 372–378.
- [6] R. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 2 (1972), pp. 146–160.
- [7] J. E. HOPCROFT AND R. M. KARP, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, this Journal, pp. 225–231.
- [8] K. MENGER, *Zur allgemeinen Kurventheorie*, Fund. Math., 10 (1927), pp. 96–115.
- [9] G. B. DANTZIG AND D. R. FULKERSON, *On the max-flow min-cut theorem of networks*, Linear Inequalities and Related Systems, Annals of Math. Study 38, Princeton University Press, Princeton, N.J., 1956, pp. 215–221.
- [10] R. E. GOMORY AND T. C. HU, *Multi-terminal network flows*, J. Soc. Indust. and Appl. Math., 9 (1961), pp. 551–570.
- [11] N. ZADEH, *Theoretical efficiency of the Edmonds–Karp algorithm for computing maximal flows*, J. Assoc. Comput. Mach., 19 (1972), pp. 184–192.