TESTING GRAPH CONNECTIVITY

by

R. Endre Tarjan

Memorandum No. ERL-M422

November 1973

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Testing Graph Connectivity

by

R. Endre Tarjan
Computer Science Division
U.C. Berkeley
Berkeley, California.

November, 1973.

ABSTRACT:

An algorithm proposed by Dinic for finding maximum flows in networks and by Hopcroft and Karp for finding maximum bipartite matchings is applied to graph connectivity problems. It is shown that the algorithm requires $O(V^{1/2}E)$ time to find a maximum set of node-disjoint paths in a graph, and $O(V^{2/3}E)$ time to find a maximum set of edge-disjoint paths. These bounds are tight. Thus the node connectivity of a graph may be tested in $O(V^{5/2}E)$ time, and the edge connectivity of a graph may be tested in $O(V^{5/3}E)$ time.

Keywords and Phrases: Graph, Connectivity, Network, Flow, Matching, Maximum Flow.

† This research was partially supported by the National Science Foundation under Grant Number NSF-GJ-35604X.

Testing Graph Connectivity

by

R. Endre Tarjan

Introduction: There is a class of problems in graph theory and network analysis which call for the maximization of a certain function defined on a problem graph. Examples include finding a maximum flow in a transportation network and finding a maximum matching in a graph. Many of these problems have efficient algorithms based on choosing an initial solution and incrementally improving it until a maximum is reached. Here we study an efficient algorithm of this type. The algorithm was developed for finding maximum network flows by Dinic [1] and independently developed for finding maximum bipartite matchings by Hopcroft and Karp [2]. We use the algorithm to find the connectivity of a graph, and give bounds on its running time. The results generalize the bounds given by Hopcroft and Karp [2]. We also show that the bounds are tight.

Definitions: A graph $G = (V,E)$ is a collection of vertices V and edges E. V denotes the number of vertices and E denotes the number of edges. The edges may be either unordered pairs of distinct vertices (the graph is undirected) or ordered pairs of distinct vertices (the graph is directed). An edge is denoted by $e = (v,w)$; v and w are adjacent; v and (v,w) are incident. The degree of a vertex v is the number of edges incident to v. The out-degree of a vertex v is the number of edges (v,w); the in-degree of v is the number of edges (u,v). A sequence of edges $p = (v_1,v_2)(v_2,v_3),\ldots,(v_{n-1},v_n)$ in G is called a path from v_1 to v_n.

The length of p is n - 1. The distance from v to w is the length of the shortest path from v to w. A matching in G is a set of edges no two of which have a common vertex. A bipartite graph is a graph whose vertices may be partitioned into two sets A and B such that every edge of the graph has one vertex in A and one vertex in B.

A network $\Gamma = (G,s,t,c)$ is a directed graph G with two distinct distinguished vertices s and t, called the source and the sink, and with a non-negative real-valued capacity $c(v,w)$ associated with each edge of the graph. We assume that if (v,w) is an edge of the network then so is (w,v). A flow f in the network is a real-valued function on the edges of the graph such that $0 \leq f(v,w) \leq c(v,w)$ and

$$F(v) = \sum_{(v,w)\in E} f(v,w) - \sum_{(u,v)\in E} f(u,v) = 0$$

unless v = s or v = t. The value of the flow is $v(f) = F(s)$.

A cut of a network is a set of vertices $S \subseteq V$ such that $s \in S$ and $t \notin S$. The capacity of the cut is $c(S) = \sum_{\substack{v\in S \\ w\notin S}} c(v,w)$.

The Dinic-Hopcroft-Karp Algorithm: A famous theorem due to Ford and Fulkerson [3] states that the maximum value of a flow in a network is equal to the minimum capacity of a cut. The proof of this theorem gives rise to an algorithm for finding maximum flows. This algorithm sometimes runs slowly, however; and it may not terminate or even give a flow converging to a maximum value if the capacities are not integral [3]. Edmonds and Karp [4] have described a variation of the algorithm which runs in $O(VE^2)$ time and always gives a maximum flow whatever the values of the capacities. Independently, Dinic [1] has found an improved

version of the Edmonds-Karp algorithm which runs in $O(V^2E)$ time independent of capacities.

Dinic's algorithm works by finding paths from s to t along which the flow can be increased and then increasing the flow. This process is continued until the flow cannot be increased further. To find an augmenting flow Dinic's method uses breadth-first search. It is by using this particular kind of search that the algorithm achieves an $O(V^2E)$ time bound.

Specifically, let f be a flow in a network $\Gamma = (G,s,t,c)$. For any edge (v,w), let $\delta(v,w) = c(v,w) - f(v,w) + f(w,v)$. This value is the residual capacity of the pair of arcs (v,w), (w,v) in the presence of the flow f. An augmenting path p from s to t is a path such that $\Delta = \min_{(v,w) \text{ on } p} \delta(v,w) > 0$. We can increase the value of the flow from v(f) to $v(f) + \Delta$ by changing the flow in the arcs of p and in the arcs opposite to them so that $f'(v,w) - f(v,w) + f(w,v) - f'(w,v) = \Delta$ for all (v,w) on p. The residual capacities change as follows: $\delta'(v,w) = \delta(v,w) - \Delta$ and $\delta'(w,v) = \delta(w,v) + \Delta$ for (v,w) on p and $\delta'(v,w) = \delta(v,w)$ for other arcs.

Let $\tilde{G}$ be the arcs (v,w) of G which satisfy $\delta(v,w) > 0$. To find augmenting paths, we carry out a breadth-first search of $\tilde{G}$. Let V_i the the set of vertices whose shortest path from s has length i in $\tilde{G}$. ($V_o = \{s\}$, V_1 is the set of all vertices adjacent to s, and so on.) If $t \in V_m$ let $k = m$; if t is not reachable from s in $\tilde{G}$ let $k = V + 1$. Let G_k be the graph with vertices $\bigcup_{i=0}^{k} V_k$ and having for arcs all (v,w) such that (v,w) is an arc of $\tilde{G}$ and $v \in V_i$, $w \in V_{i+1}$ for some $0 \leq i < k$. Clearly if t is reachable from s in $\tilde{G}$, G_k contains all shortest-length paths from s to t in $\tilde{G}$.

We start at t in G_k and follow arcs backward to s to find an augmenting path from s to t. After augmenting the flow along this path, we delete all arcs on the path whose residual capacity is now zero. We must also delete vertices and their incident edges which are no longer reachable from s or t. This may be accomplished by deleting all vertices with no incoming or with no outgoing arcs, and continuing to delete such vertices until every vertex left has at least one incoming and one outgoing arc. After all necessary deletions are carried out, we search for another augmenting path. We continue finding augmenting paths and deleting edges until no more paths from s to t exist. We then construct a new G_k. We continue constructing G_k's and finding augmenting paths until t is not reachable from s in the current $\tilde{G}$ ($k = V + 1$).

The complete algorithm is:

```
procedure MAXFLOW; begin
    initialize flow to 0;
    construct initial G_k;
    while t is a vertex of G_k do begin
        for each vertex v of G_k do begin
            calculate indegree (v);
            calculate outdegree (v);
            if (indegree (v) = 0) or (outdegree (v) = 0) do add v to nullist;
        end;
        while t has an incoming edge do begin
            trace back from t to s to find an augmenting path;
            augment flow along path;
            delete edges along path which now have zero residual capacity,
                updating indegrees, outdegrees, and nullist;
            while some vertex v is on nullist do
                delete v and incident edges from nullist, updating indegrees,
                    outdegrees, and nullist;
        end;
        construct new G_k;
    end;
end;
```

It is an immediate consequence of the max-flow min-cut theorem that this algorithm does not terminate unless the flow is actually maximum. Let us call the processing of one G_k a stage. Dinic proved that after a stage, the length of the shortest augmenting path from s to t must increase. Thus there are at most V + 1 stages before the algorithm terminates. If the graph is represented as a set of adjacency lists, one for each vertex, the various parts of the algorithm have the following time bounds:

initialization of flow: O(E)

construction of one G_k: O(E) using breadth-first search [5]

initialization of indegrees, outdegrees and nullist: O(E)

finding of one augmenting path: O(V)

deletion of edges and updating of indegrees, outdegrees and nullist during one stage: O(E) since any edge is deleted only once and the amount of work per edge is O(1).

Each augmenting path found causes the deletion of at least one edge in G_k; thus there are at most E augmenting paths found per stage. The overall time bound is

$$O(\underset{\text{initialization}}{E} + \underset{\substack{\text{no of}\\ \text{stages}}}{V} \cdot [\underset{\text{path}}{V} + \underset{\text{paths}}{E} + \underset{\substack{\text{initialization}\\ \text{and updating}\\ \text{per stage}}}{E}])$$

$$= O(V^2E)$$

The Edmonds-Karp algorithm differs from Dinic's in that a new G_k is constructed after every augmentation; the total number of augmentations is O(VE) as in Dinic's algorithm but finding one augmenting path requires O(E) time and an $O(VE^2)$ time bound results. Certain graphs actually

achieve this worst-case time bound [6].

Dinic's algorithm works with the same time bound if the network has undirected edges or if the network has vertex capacities in addition to edge capacities. To handle an undirected edge, we convert it into two directed edges, one in each direction, each with the capacity of the initial edge. If the maximum flow has flow in both of these edges it may be modified so that there is only flow in one of the edges, and then it corresponds to a flow in the original graph.

If a vertex $v \notin \{s, t\}$ in a network has non-negative capacity $c(v)$ the flow f is required to satisfy

$$\sum_{(u,v)} f(u,v) \leq c(v).$$

We can change a vertex capacity into an edge capacity by deleting vertex v and replacing it with two vertices v',v'', and adding an edge (v',v'') with capacity c, edges (u,v') for every (u,v) in the original network, and edges (v'',w) for every (v,w) in the original network. Any maximum flow in the new network corresponds to a maximum flow in the old network, and vice-versa. Ford and Fulkerson [3] discovered these constructions.

Dinic's algorithm has many applications, including finding a maximum matching in a bipartite graph. Suppose $G = (V,E)$ is a bipartite graph with edges between vertex sets A and B. Let $\Gamma = (G',s,t,c)$ where $G' = (A \cup B \cup \{s,t\}, \{(v,w) | v \in A, w \in B, \text{ and } (v,w) \in E\} \cup \{(s,v) | v \in A\} \cup \{(w,t) | w \in B\})$ and $c(v) = 1$ for all vertices in A and B. A maximum matching on G corresponds exactly to a maximum flow in Γ. The network Γ is very special, and Dinic's algorithm has a worst-case time bound of $O(V^{1/2}E)$ for finding maximum bipartite matchings as proved by Hopcroft

and Karp [2]. The only important fact used to derive this bound is that all capacities are one. In the next section we generalize Hopcroft and Karp's bipartite matching results to derive better bounds on Dinic's algorithm when all capacities are one. Then we apply these results to graph connectivity problems.

Upper Bounds When All Capacities Are One

We are interested in two special cases of the network flow problem: (1) all edge capacities are one and the vertex capacities are integers; (2) all vertex capacities are one and the edge capacities are integers. In case 2 we may clearly assume without loss of generality that all edge capacities are one; thus problem 2 is a special case of problem 1. In graphs with integer capacities there are integer maximum flows [3], and all flows constructed during execution of Dinic's algorithm are integral. It is possible to immediately tighten the bound on the algorithm as follows: every edge on an augmenting path must be deleted from G_k since its residual capacity must change from one to zero. Thus only O(E) time per stage can be spent finding augmenting paths, the total time per stage is O(E), and the overall time is O(VE). We can reduce the time bound further by showing that only a small number of stages can occur. Crucial is the next theorem:

Theorem 1: Let $\Gamma = (G,s,t,c)$ be a network, and let f be a flow on Γ. Let M be the maximum possible flow on Γ. Then in $\tilde{G}$ there is a flow of value $M - v(f)$ from s to t.

Proof: This theorem is a quantitative version of the max-flow min-cut theorem. If $\tilde{G}$ does not have a flow of value $M - v(f)$, then $\tilde{G}$ has a cut S with capacity $c'(S) < M - v(f)$. But then the capacity of S in G is

$$c(S) = \sum_{\substack{(v,w) \\ v \in S \\ w \notin S}} c(v,w) = c'(S) + \sum_{\substack{(v,w) \\ v \in S \\ w \notin S}} f(v,w) - f(w,v) < M - v(f) + v(f) < M.$$

By the max-flow min-cut theorem this is impossible, so every cut in $\tilde{G}$ has capacity at least $M - v(f)$ and again by the max-flow min-cut theorem $\tilde{G}$ has a flow of value $M - v(f)$.

Corollary 2: Let $\Gamma = (G,s,t,c)$ be a network with unit edge capacities and integer vertex capacities, let Γ' be the corresponding network with vertex capacities converted into edge capacities, let f be an integral flow on Γ', and let M be the value of a maximum flow on Γ. Then in $\tilde{G}'$ there is an augmenting path of length $\leq \frac{2E}{M - v(f)}$.

Proof: By Theorem 1, $\tilde{G}'$ has a flow of value $M - v(f)$, which we may assume is integral [3]. This flow consists of a set of paths from s to t. Since any edge in Γ' corresponding to an edge in Γ can be in at most one of these paths, and since Γ has at most E edges, there must be a path of length $\leq \frac{2E}{M - v(f)}$. (At least half the edges of a path in Γ' correspond to edges in Γ.) This path is augmenting with respect to f.

Corollary 3: Let $\Gamma = (G,s,t,c)$ be a network with unit vertex capacities, let $\Gamma' = (G',s,t,c')$ be the corresponding network with vertex capacities replaced by edge capacities, let f be an integral flow on Γ', and let M be the value of a maximum flow on Γ'. Then in $\tilde{G}'$ there is an augmenting path of length $\leq \frac{2V}{M - v(f)}$.

Proof: By theorem 1, $\tilde{G}'$ has a flow of value $M - v(f)$, which we may assume is integral. This flow consists of a set of edge-disjoint paths from s to t, and every other edge on such a path corresponds to a vertex (other than s or t) of G. Since any edge in Γ' can be in at most one of these paths,

and since G has only V - 2 vertices other than s or t, there must be a path of length $\leq \frac{2V}{M - v(f)}$. This path is augmenting with respect to f.

Corollaries 2 and 3 are generalizations of Hopcroft and Karp's Corollary 2 [2]. These corollaries lead to improvements in the time bound on Dinic's algorithm for problems (1) and (2). Specifically, suppose Dinic's algorithm has completed the first $E^{1/2}$ stages on a problem of type 1. Then all augmenting paths in $\tilde{G}'$ have length $\geq E^{1/2}$. By Corollary 2, $E^{1/2} \leq \frac{2E}{M - v(f)}$; or, $M - v(f) \leq 2E^{1/2}$. Since every stage except the last leads to at least one augmentation, at most $2E^{1/2} + 1$ more stages will occur. Thus Dinic's algorithm has a time bound of $O(E^{3/2})$ for problems of type 1 ($O(E^{1/2})$ stages; $O(E)$ time per stage). Similarly, using Corollary 3 we may show that Dinic's algorithm has a time bound of $O(V^{1/2}E)$ for problems of type 2 ($O(V^{1/2})$ stages; $O(E)$ time per stage). This result is a generalization of Hopcroft and Karp's bipartite matching bound.

We may improve the bound for problem 1 even further by using the fact that the problem graph has no multiple edges.

Theorem 4: Let G be a directed graph with distinct vertices s and t. Suppose there are at least Δ edge-disjoint paths between s and t. Then the distance between s and t in G is at most $2V/\Delta^{1/2}$.

Proof: Let $V_i = \{v \mid v \text{ is at distance } i \text{ from } s\}$. Suppose $t \in V_k$. Every path from s to t must contain at least one edge from V_i to V_{i+1} of all $0 \leq i < k$. If there are Δ edge-disjoint paths, it must be the case that $|V_i|\ |V_{i+1}| \geq \Delta$, since G contains no multiple edges. Thus for all $0 \leq i < k$, either $|V_i| \geq \Delta^{1/2}$ or $|V_{i+1}| \geq \Delta^{1/2}$. Since

$$\sum_{i=0}^{k} |V_i| \leq V, \quad k \leq \frac{2V}{\Delta^{1/2}} .$$

Now suppose Dinic's algorithm has completed the first $V^{2/3}$ stages of a problem of type 1, where Γ is the original network and Γ' is the network with edge capacities in place of vertex capacities. Let the maximum flow in $\tilde{G}'$ equal Δ. Then in $\tilde{G}$ there are Δ edge-disjoint paths from s to t. Since $\tilde{G}$ is a graph, theorem 4 applies, and the distance from s to t in $\tilde{G}$ is $\leq \frac{2V}{\Delta^{1/2}}$. Then the distance between s and t in $\tilde{G}'$ is $\leq \frac{4V}{\Delta^{1/2}}$, which implies that $\Delta \leq 16V^{2/3}$. Thus only $16V^{2/3} + 1$ more stages can occur, and Dinic's algorithm has a time bound of $O(V^{2/3}E)$ for problems of type 1.

Summarizing the results of this section, Dinic's algorithm has worst case time bounds of

$O(\min(V^{2/3}, E^{1/2})E)$	for finding maximum flows in networks with unit edge capacities;
$O(V^{1/2}E)$	for finding maximum flows in networks with unit vertex capacities.

Lower Bounds and Possible Algorithmic Improvements:

The bounds given in the last section are tight. That is, there exist networks with unit vertex capacities on which Dinic's algorithm requires $k_1 V^{1/2}E$ time for some k_1, and networks with unit edge capacities on which Dinic's algorithm requires $k_2 V^{2/3}E$ time for some k_2. One bad example is based on a graph which consists of disjoint paths of lengths one, three, five, ..., $2n + 1$. Suppose we try to find a matching on such a graph using Dinic's algorithm. If we are unlucky enough to choose the even edges on these paths as the first stage matching (Figure 1), then we will have to

carry out n more stages, one for each disjoint path. This example is not really sufficient as a worst-case example, but it can be modified so that it works, as described in the following theorems.

Theorem 5: There is a constant k_1 such that for any n there is a bipartite graph with $3n(n-1)$ vertices on which Dinic's algorithm requires $k_1 n^5$ time to find a maximum matching.

Proof: Let G be the bipartite graph with vertex set

$$V = \{a_i \mid 1 \le i \le \frac{3n(n-1)}{2}\} \cup \{b_i \mid 1 \le i \le \frac{3n(n-1)}{2}\}$$

and edge set

$$E = \{(a_i,b_j) \mid 1 \le i \le n(n-1) \text{ and } 1 \le j \le \frac{n(n-1)}{2}\}$$

$$\cup \{(a_i,b_j) \mid n(n-1) < i \le \frac{3n(n-1)}{2} \text{ and } \frac{n(n-1)}{2} < j \le \frac{3n(n-1)}{2}\}$$

$$\cup \{(a_i,b_i) \mid \frac{n(n-1)}{2} < i \le n(n-1)\}$$

$$\cup \{(a_{i+1},b_i) \mid \frac{n(n-1)}{2} < i \le n(n-1) \text{ and } i \ne \frac{k(k-1)}{2} + \frac{n(n-1)}{2} \text{ for any } k \ge 1\}$$

Suppose Dinic's algorithm is applied to G and that it finds the following edges in a first-stage matching:

$$\{(a_i,b_i) \mid 1 \le i \le \frac{n(n-1)}{2} \text{ and } i \ne \frac{k(k-1)}{2} + 1 \text{ for any } k \ge 1\}$$

$$\cup \{(a_{i+\frac{n(n-1)}{2}},b_i) \mid 1 \le i \le \frac{n(n-1)}{2} \text{ and } i = \frac{k(k-1)}{2} + 1 \text{ for some } k \ge 1\}$$

$$\cup \{(a_{i+1},b_i) \mid \frac{n(n-1)}{2} < i \le n(n-1) \text{ and } i \ne \frac{k(k-1)}{2} + \frac{n(n-1)}{2} \text{ for any } k \ge 1\}$$

$$\cup \{(a_{\frac{n(n-1)}{2}+1}, b_{\frac{n(n-1)}{2}+1})\}$$

$$\cup \{(a_{i+\frac{n(n-1)}{2}}, b_i) \mid \frac{n(n-1)}{2} < i \leq n(n-1) \text{ and } i = \frac{k(k-1)}{2} + \frac{n(n-1)}{2} \text{ for some } k \geq 1\}$$

$$\cup \{(a_i, b_i) \mid n(n-1) < i \leq \frac{3n(n-1)}{2} \text{ and } i \neq \frac{k(k-1)}{2} + n(n-1) \text{ for some } k \geq 2\}.$$

(see Figure 2.) Now n-2 more stages must be carried out, to find augmenting paths of lengths 7, 9, 11, ..., 2n+1. Constructing G_k for each of these stages requires exploration of at least $(\frac{n(n-1)}{2})2$ edges, so the total time spent by Dinic's algorithm is $k_1 n^5$ for some suitable k_1 independent of n.

Theorem 6: There is a constant k_2 such that for any n there is a graph with $3n^3 + n^2 + 2$ vertices on which Dinic's algorithm requires $k_2 n^8$ time to find a maximum set of edge-disjoint paths between vertices s and t.

Proof: Let G be the graph with vertex set

$$V = \{s,t\} \cup \{a_i \mid 1 \leq i \leq n^3\} \cup \{b_i \mid 1 \leq i \leq n^3\} \cup \{c_i \mid 1 \leq i \leq n^2\}$$

$$\cup \{d_{ij} \mid 1 \leq i \leq n^2 \text{ and } 1 < j \leq n\} \cup \{e_i \mid 1 \leq i \leq n^2\}$$

and edge set

$$E = \{(s,a_i) \mid 1 \leq i \leq n^3\} \cup \{(e_i,t) \mid 1 \leq i \leq n^2\}$$

$$\cup \{(a_i,b_j) \mid 1 \leq i,j \leq n^3\} \cup \{(b_i,c_j) \mid 1 \leq i \leq n^3 \text{ and } 1 \leq j \leq n^2\}$$

$$\cup \{(c_i,d_{i1}) \mid 1 \leq i \leq n^2\} \cup \{(d_{n^2i}, e_j) \mid 1 \leq i \leq n \text{ and } 1 \leq j \leq n^2\}$$

$$\cup \{d_{ij}, d_{i+1j} \mid 1 \leq i < n^2 \text{ and } 1 \leq j \leq n\}$$

(Figure 3)

When Dinic's algorithm is applied to this graph, it will find augmenting paths of lengths 6, 7, ..., n^2+5. Each stage will require examination of at least n^6 edges. Thus the total time taken by Dinic's algorithm on this graph

is $k_2 n^8$ for some suitable constant k_2 independent of n.

It is possible to modify Dinic's algorithm somewhat so that the examples in Theorems 5 and 6 are no longer bad, but similar worst-case examples work for all variations of the algorithm tried. However, evidence does exist to support the conjecture that there is a better algorithm.

Theorem 7: Let G be a graph to which Dinic's algorithm is applied to find a set of vertex disjoint paths between distinct vertices s and t. Then the total length of all augmenting paths found is O(V log V).

Proof: Applying Corollary 3, the total length of all augmenting paths is $\leq \sum_{k=1}^{V} \frac{2V}{k} = O(V \log V)$.

Theorem 8: Let G be a graph to which Dinic's algorithm is applied to find a set of edge disjoint paths between distinct vertices s and t. Then the total length of all augmenting paths is $O(\min (V^{3/2}, E \log V))$.

Proof: Applying Corollary 2 as in the proof above gives an $O(E \log V)$ bound on the total length of augmenting paths. Applying Theorem 4, the total length of all augmenting paths is $\leq \sum_{k=1}^{V} \frac{4V}{\sqrt{k}} = O(V^{3/2})$.

On the basis of Theorems 7 and 8, I conjecture that there is an $O(V^2 \log V)$ algorithm for finding maximum sets of node-disjoint paths, and an $O(V^{5/2})$ algorithm for finding maximum sets of edge-disjoint paths.

Applications and Open Problems

The edge connectivity of a graph is the minimum number of edges in a set S such that there exist vertices s and t such that every path from s to t contains an edge of S. (S is called an edge separator of s and t.) The vertex connectivity of a graph is the minimum number of vertices in a set S such that there exist vertices s, $t \notin S$ such that every path from s to

t contains a vertex of S (S is called a vertex separator of s and t). (If a graph G with V vertices contains all possible edges then its vertex connectivity is said to be V - 1.) A special case of the max-flow min-cut theorem, called Menger's theorem, states that the size of a maximum set of edge-disjoint (vertex-disjoint) paths from s to t is equal to the minimum size of an edge separator (vertex separator) of s from t. It follows that we may compute the edge or vertex connectivity of a graph by solving a $O(V^2)$ network flow problems with unit capacities.

Gomory and Hu [7] observed that only V such problems need to be solved to determine the edge connectivity of an undirected graph. This result extends to directed graphs: let S be a minimum edge separator of a directed graph; suppose S separates s and t. Choose any fixed vertex a. If S does not separate s from a then s must separate a from t. Suppose we solve the 2V - 2 possible network flow problems having a as either source or sink. Then one of them must give us a minimum edge separator. Thus, using Dinic's algorithm we can determine the edge connectivity of a directed or undirected graph in $O(VE \min(V^{2/3}, E^{1/2}))$ time.

Dinic's algorithm is useful in two more-complicated problems having to do with edge connectivity on undirected graphs. Gomory and Hu [7] have defined an object called a cut tree which contains much information about the edge separators of a graph. They show that a cut tree may be constructed by solving V - 1 network flow problems. Thus, using Dinic's algorithm we may construct a cut tree in $O(VE \min(V^{2/3}, E^{1/2}))$ time. A related concept is that of a narrow slicing, as studied by Matula [8]. He shows that a narrow slicing may be found by solving $O(V^2)$ network flow problems, so use of Dinic's algorithm gives an $O(V^2E \min(V^{2/3}, E^{1/2}))$ time bound for this problem.

Determining node connectivity seems to be harder than determining edge connectivity. Hopcroft and Tarjan [9, 10, 11] have given $O(V+E)$ algorithms for determing whether the vertex connectivity of an undirected graph is zero, one, two, or greater than two. Tarjan [10] has given an $O(V+E)$ algorithm for testing whether the vertex connectivity of a directed graph is zero (this is the "strong connectivity" problem). Even [12], generalizing on work of Kleitman [13], has given an $O(k^3E + kVE)$ algorithm for testing whether the vertex connectivity of an undirected or directed graph is at least k. If $k \leq \sqrt{V}$, Even's algorithm has an $O(V^{3/2}E)$ time bound; however, for arbitrary k the time bound can be as large as $O(V^3E)$. Using Dinic's algorithm to solve $V(V - 1)$ network flow problems, we can exactly determine the vertex connectivity of a directed or undirected graph in $O(V^{5/2}E)$ time.

A generalization of the bipartite matching problem is the bipartite b-matching problem: given a bipartite graph and integer capacities on the nodes we want to find a maximum set of edges such that the number of edges incident to a vertex does not exceed its capacity. An obvious construction will enable us to use Dinic's algorithm to solve a b-matching problem in $O(\min(V^{2/3}, E^{1/2})E)$ time. This problem is also called the degree-constrained subgraph problem [14]. A construction due to Ford and Fulkerson [3] allows us to apply Dinic's algorithm to find a maximum set of pairwise incomparable elements in a partial ordering in $O(V^{1/2}E)$ time, where V is the total number of elements and E is the number of comparable pairs.

Many questions are still unanswered. Here are a few. Is there a better algorithm than Dinic's? Can the vertex connectivity of a graph be found by solving less than $O(V^2)$ network flow problems? Can the algorithms

for testing small connectivity be improved or generalized? And, perhaps most tantalizingly, can the results here be extended to the problem of finding a maximum matching (or b-matching) in an arbitrary graph? Edmonds has constructed an algorithm for solving this problem [14,15]. If carefully implemented, the algorithm runs in O(VE) time [16,17]. It seems likely, however, that this bound is improvable.

REFERENCES

[1] E.A. Dinic, "Algorithm for solution of a problem of maximum flow in a network with power estimation", Sov. Math. Dokl. Vol. 11, No.5 (1970), pp. 1277-1280.

[2] J.E. Hopcroft, and R.M. Karp, "A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", SIAM J. Comput., Vol. 2, Nc. 4 (December 1973) pp. 225-231.

[3] L.R. Ford, and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, N.J. (1962).

[4] J. Edmonds, and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", JACM Vol. 19, No. 2, (April 1972), pp. 248-264.

[5] C. Berge, The Theory of Graphs and its Applications Alison. Doig, tr., John Wiley & Sons, Inc., New York (1962), pp. 65-68.

[6] N. Zadeh, "Theoretical efficiency of the Edmonds-Karp algorithm for computing maximal flows", JACM Vol. 19, No.1, (January 1972) pp. 184-192.

[7] R.E. Gomory, and T.C. Hu, "Multi-terminal network flows", J. SIAM, Vol. 9, No. 4, (December 1961), pp. 551-570.

[8] D.W. Matula, "k-components, clusters, and slicings in graphs", SIAM J. Appl. Math., Vol. 22, No. 3 (May 1972) pp. 459-480.

[9] J.E. Hopcroft, and R. Tarjan, "Algorithm 447: efficient algorithms for graph manipulation", CACM, Vol. 16, No. 6, (June 1973), pp. 372-378.

[10] R. Tarjan, "Depth-first search and linear graph algorithms", SIAM J. Comput., Vol. 1, No. 2 (June 1972), pp. 146-160.

[11] J.E. Hopcroft, and R. Tarjan, "Dividing a graph into triconnected components", SIAM J. Comput. Vol. 2, No. 3 (September 1973), pp. 135-158.

[12] S. Even, "An algorithm for determining whether the connectivity of a graph is at least k", T.R. 23-184, Department of Computer Science, Cornell University (September 1973).

[13] D.J. Kleitman, "Methods for investigating connectivity of large graphs", IEEE Trans. on Circuit Theory, Vol. CT-16, No. 2, (May 1969), pp. 232-233.

[14] A.J. Goldman, "Optimal matchings and degree-constrained subgraphs", J. Res. Nat. Bur. Stand., Vol. 68B (1964), pp. 27-29.

[15] J. Edmonds, "Paths, Trees, and Flowers", Canadian J. of Math, Vol. 17, (1965), pp. 449-467.

[16] M.L. Balinski, "Labelling to obtain a maximum matching", Combinatorial Mathematics and its Applications, R.C. Rose and T.A. Dowling, ed., University of North Carolina Press, Chapel Hill, North Carolina (1967), pp. 585-602.

[17] H. Gabow, "An efficient implementation of Edmond's maximum matching algorithm", Technical Report No. 31, Digital Systems Laboratory, Stanford University (June 1972).

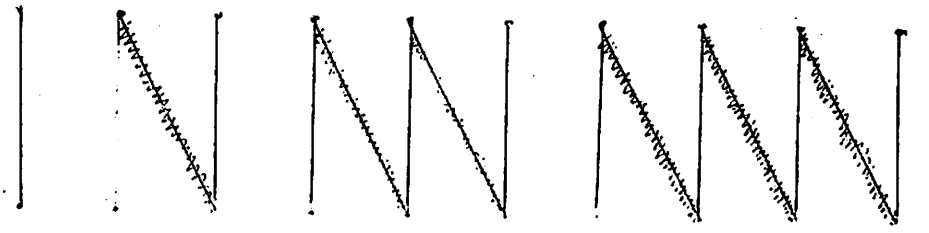

Figure 1: A bad example for bipartite matching.
First-stage matching in wavy lines.

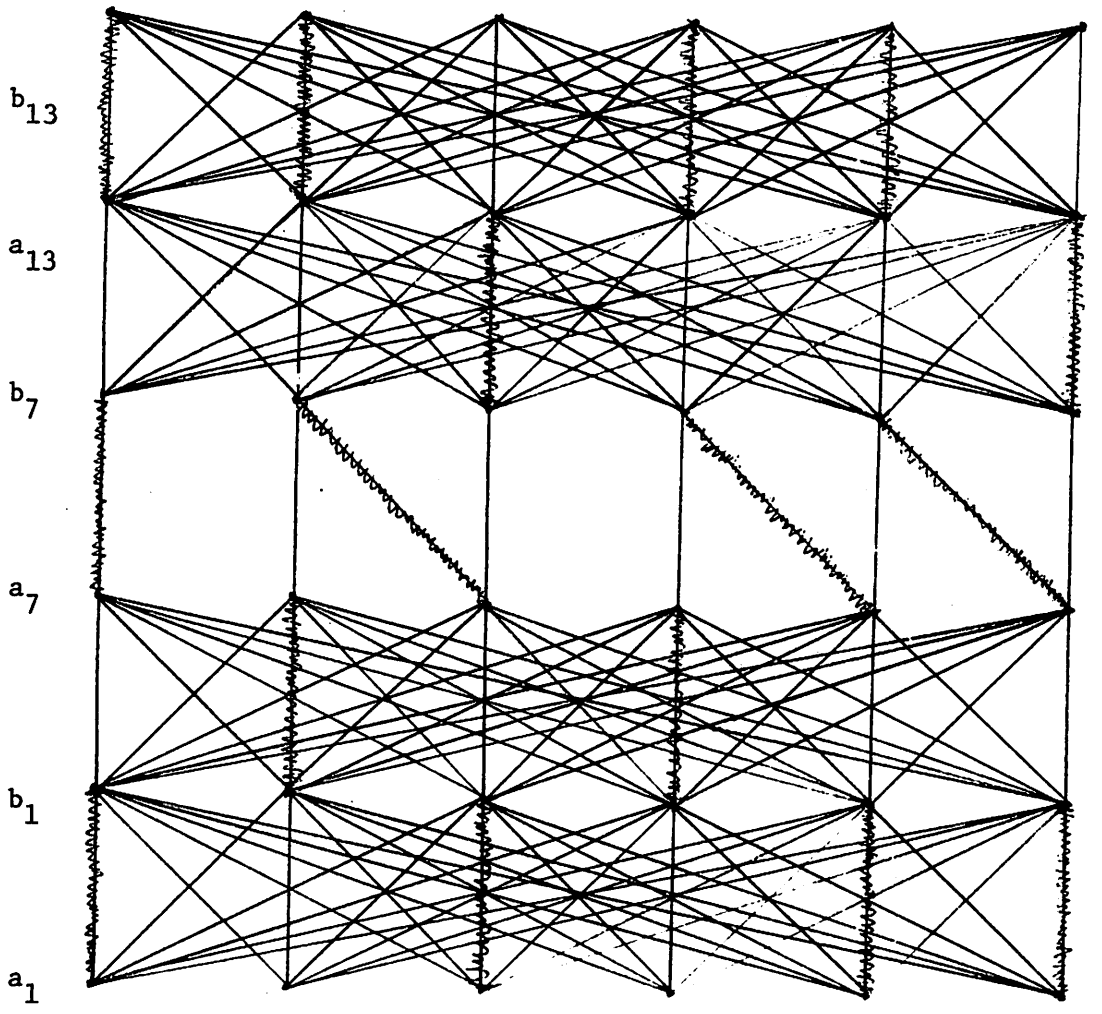

Figure 2: A worst-case example for bipartite matching (n = 4).

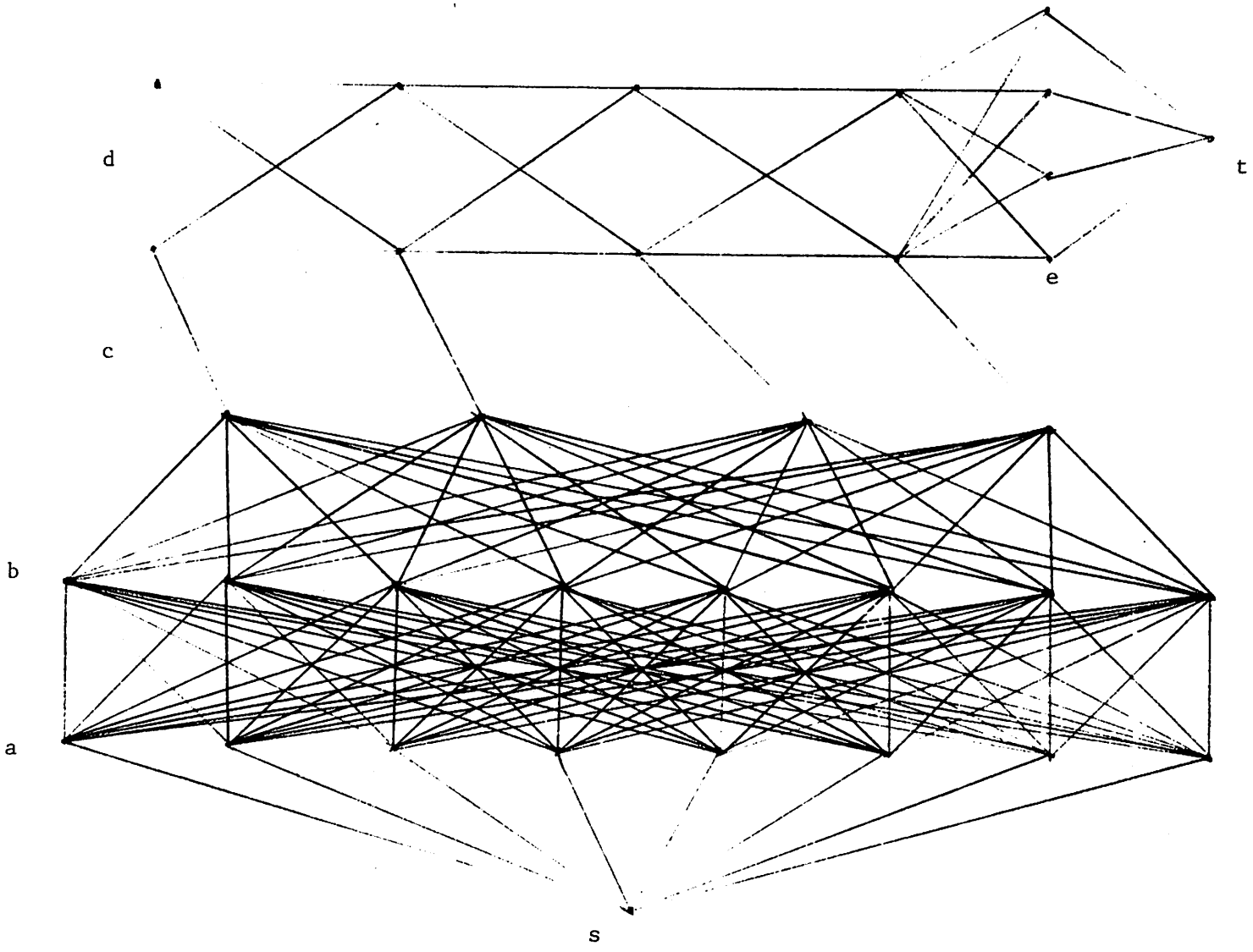

Figure 3: A worst-case example for edge-disjoint paths (n = 2).