Appl. Math. Lett. Vol. 4, No. 5, pp. 57–60, 1991
Printed in Great Britain.

0893-9659/91 $3.00 + 0.00

Worst Case Behavior of the Dinic Algorithm

GARY R. WAISSI

School of Management, University of Michigan at Dearborn

(Received April 1991)

Abstract. Many max-flow phase algorithms use the Dinic algorithm to generate an acyclic network in the first phase, and then solve the maximal flow problem in such a network in the second phase. This process is then repeated until the maximum value flow is found in the original network. In this paper a class of networks is presented where the Dinic algorithm always attains it's worst case bound. The Dinic algorithm requires $(n-1)$ network generations, where n is the number of nodes in the original network for finding the maximum value flow in the original network.

1. INTRODUCTION

Dinic [1] presented in 1970 an algorithm to solve the maximum flow problem by repeating a two phase process. In the first phase, Phase I, the Dinic algorithm generates an acyclic (or layered) network. In the second phase, Phase II, a maximal flow algorithm is applied to such an acyclic layered network. For the maximum value flow these two phases are repeated until the maximum value flow in the original network has been found. Several maximal flow algorithms (e.g., [1–9]) use this two phase approach to solve the maximum value problem.

The referenced maximal flow algorithms, when using the Dinic algorithm, have achieved an overall computational worst case complexity of $O(n^3)$ for the maximum value flow problem in dense networks with $m \approx n^2$. This worst case complexity of $O(n^3)$, is still the best that has been achieved for dense networks.

This paper assumes familiarity with the two phase approach when solving single commodity maximum flow problems in networks. It is also assumed that the reader is familiar with the Dinic algorithm [1]. Several of the references discuss, or refer to, the Dinic algorithm, and a reader is referred to those for review and reference, for example [1–9].

2. THE PROBLEM

Let an original, directed, single commodity network be $G = (N, A, \lambda, c, s, t)$. Here N denotes the set of nodes, A the set of arcs, λ a vector $(\lambda_{i,j} : (i,j) \in A)$ of lower bounds for flows on the arcs, c a vector $(c_{i,j} : (i,j) \in A)$ of capacities for flows on the arcs, and s, t the specified source and sink nodes. Each arc in the network is an ordered pair (i,j), $i \neq j$, with $i, j \in N$. Nodes i, j are called the tail and head nodes of arc (i,j). The arc (i,j) is available for shipping the commodity from node i to node j. The problem is to send as much flow as possible from s to t without violating the flow feasibility and flow conservation constraints.

2.1. Flow Feasibility and Flow Conservation

A flow vector in G is a vector $f = (f_{i,j} : (i,j) \in A)$, where $f_{i,j}$ denotes the units of commodity shipped from node i along the arc $(i,j) \in A$ to node j. Given the flow vector $f = (f_{i,j})$ in G,

define

$$f(\text{in}, i) = \sum (f_{j,i} : \text{over } j \text{ such that } (j,i) \in A),$$
$$f(i, \text{out}) = \sum (f_{i,j} : \text{over } j \text{ such that } (i,j) \in A).$$

The flow vector f is said to be a feasible flow vector if

$$f(s, \text{out}) - f(\text{in}, s) = f(\text{in}, t) - f(t, \text{out}),$$
$$f(\text{in}, i) = f(i, \text{out}), \quad \text{for all } i \in N, \text{ and } i \neq s \text{ or } t,$$
$$\lambda \leq f \leq c.$$

The first two constraints are called the flow conservation constraints and the third the flow bound constraint. When the second constraint holds for some node, say node i, then the flow vector f satisfies flow conservation at node i. If f is a feasible flow vector in G, the common value of the quantities in the first equation is called the value of the flow vector f, and denoted by $v(f)$.

2.2. Flow Augmenting Path

A flow augmenting path (FAP) is a simple path from node s to a node, say i, if the flow can be increased on each forward arc, and decreased on each reverse arc, of the path. A flow augmenting path is defined with respect to a feasible flow vector f.

2.3. Acyclic Network

Let $H = (V, E, 0, \kappa, s, t)$ be an acyclic layered network constructed from the network G using for example the Dinic algorithm. Here V denotes the set of nodes, E the set of arcs, zero lower bound for flow on the arcs, κ the vector of capacities for flows on the arcs, s the source node, and t the sink node.

A directed network H is said to be acyclic if the nodes in V can be numbered in such a way that for every arc $(i,j) \in E$, $i < j$. Such a numbering of nodes is referred to as ***acyclic node numbering.***

In a Dinic network, the nodes in V are partitioned into mutually exclusive sets $L_0, L_1, \ldots, L_{\text{num}}$, with $L_0 = \{s\}$ and $L_{\text{num}} = \{t\}$. Each arc $(i,j) \in E$ is such that, if $i \in L_r$ and $j \in L_p$ for some r, then $p = r + 1$. The sets of nodes $L_0, L_1, \ldots, L_{\text{num}}$ are called the layers of the Dinic network. The number of layers, num, is called the length of the network H. Each arc in E join a node in a layer to a node in the next layer.

The Dinic algorithm constructs an acyclic network by determining FAPs from the source to the sink in the original network G. In the Dinic network the reverse arcs of a FAP are converted to forward arcs to maintain acyclicity of the network. Hence, in the Dinic network all arcs of an FAP are directed from s to t. Because of this property these paths form, actually, chains from s to t. These chains are called flow augmenting chains (FAC).

A feasible flow vector f is said to be a maximal (or blocking) flow if there exists no FAP from s to t with respect to f in the original network G. This FAP in an original network corresponds to a FAC in the Dinic network. It has to be pointed out, that only the shortest FAPs are determined at any network generation. Clearly, f is a maximal flow if and only if there exists at least one saturated arc on every chain from s to t in the Dinic network.

3. A CLASS OF WORST CASE DINIC NETWORKS

Figure 1 gives an example of a class of problems, where the Dinic algorithm attains the worst case bound. This worst case behavior occurs regardless of the maximal flow algorithm applied to Dinic networks. The original network is an acyclic network with n nodes and $2n - 3$ arcs. In the example network each node i is connected by an arc (i,j) to node j, where $j = i + 1$, and to the sink node t with an arc (i,t). The capacity $c_{i,j}$ of each arc (i,j), where $j = i + 1$ and $j \neq t$, is

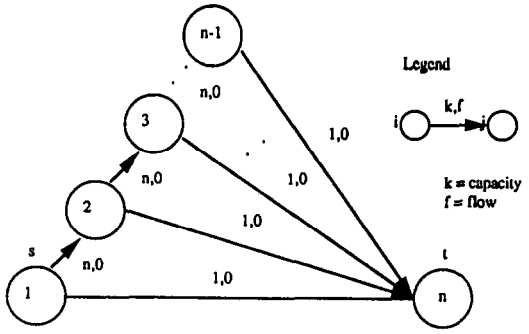

Figure 1. Example network.

set to $c_{i,j} \geq n$. The capacity $c_{i,j}$ of all other arcs (i,j), where $j = t$, is set to $c_{i,t} = 1$. The lower bound of flow in all arcs is set equal to zero.

The initial feasible flow vector is $f = 0$. The Dinic algorithm generates $(n-1)$ layered networks for finding the maximum value flow of $v = n-1$ in this class of networks. There exists only one FAC in each layered network with a flow of one unit on each arc of the FAC, and zero flow on all other arcs. The value of the maximal flow in each layered network is one implying that the value of the flow vector increases by exactly one unit in each iteration.

The Dinic algorithm requires the generation of exactly $(n-1)$ layered networks to solve this class of networks. This computational effort corresponds to the algorithm's worst case bound with respect to the number of auxiliary problems that must be solved, as proved by Dinic. The first, second, third, and $(n-1)^{st}$ Dinic networks are shown in Figure 2.

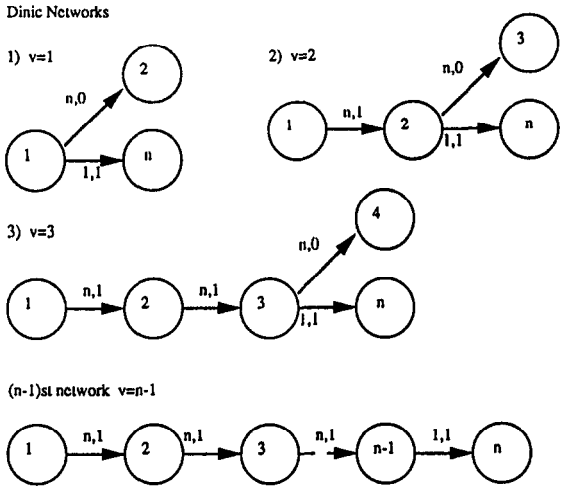

Figure 2. Iterations and layered networks obtained by the Dinic algorithm for the example network.

4. CONCLUSION

One of the main reasons for presenting this worst case network class is an attempt to improve the algorithm with respect to the number of network generations required for maximum value flow. This could be accomplished, for example, by changing the decision rules so that an attempt is made to maximize the number of arcs in each Dinic network. The Dinic algorithm limits the arcs in each network such that they connect only nodes from one layer to the adjacent next layer. This implies that only shortest FAPs are generated in each iteration.

The max-flow algorithms that use the Dinic algorithm do not imply any such limitation to the acyclic layered network. Hence, for example, if simply the *acyclic node numbering* is used

for the presented network to generate an acyclic layered network, only one network generation is required. The *acyclic node numbering* generates the entire network, with all arcs and all n nodes, where the nodes are assigned to n layers one node per layer. The maximum value flow can be found in this single layered network, because it contains all the arcs and nodes.

The author's current research concentrates on studying decision rules of a Dinic-type network generator. The goal is to find a generator, which requires strictly less than n auxiliary networks for the maximum value flow problem without added computational penalty.

REFERENCES

1. E.A. Dinic, Algorithm for solution of a problem of maximum flow in a network with power estimation, *Soviet Math. Dokl.* **11** (5), 1277–1280, (1970). (English translation by R.F. Rinehart).
2. B.V. Cherkasky, Algorithm of construction of maximal flow in networks with complexity of $O(V^2\sqrt{E})$ operations, *Math. Meth. Solution Econ. Prob.*, **7**, 117–125, (1977) (in Russian, English translation in [3] by Z. Galil).
3. Z. Galil, An $O(V(5/3)E(2/3))$ algorithm for maximal flow problem, *Acta Informatica* **14**, Springer-Verlag, 221–242, (1980).
4. Z. Galil and A. Naamad, An $O(EV(\log V)^2)$ algorithm for the maximal flow problem, *Journal of Computer and System Sciences* **21**, 203–217, (1980).
5. V.A. Karzanov, Determining the maximal flow in a network by the method of preflows, *Soviet Math. Doklady* **15** (2), 434–437, (1974). (English translation by R.F. Rinehart.)
6. V.M. Malhotra, M.P. Kumar and S.N. Maheswari, An $O(V^3)$ algorithm for finding maximum flows in networks, *Information Processing Letters* **7** (6), (1978).
7. Y. Shiloach and U. Vishkin, An $O(n^2 \log(n))$ parallel max-flow algorithm, *Journal of Algorithms* **3**, 128–146, (1982).
8. R.E. Tarjan, A simple version of Karzanov's blocking flow algorithm, *Operations Research Letters* **2** (6), 265–268, (1984).
9. G.R. Waissi, Acyclic network generation and maximal flow algorithms for single commodity flow, Ph.D. Thesis, The University of Michigan, Ann Arbor, Michigan, (1985).

School of Management, The University of Michigan at Dearborn, Dearborn, Michigan 48128-1491, U.S.A.