Efficient Algorithms for Shortest Paths in Sparse Networks

DONALD B. JOHNSON

The Pennsylvania State University, University Park, Pennsylvania

ABSTRACT Algorithms for finding shortest paths are presented which are faster than algorithms previously known on networks which are relatively sparse in arcs Known results which the results of this paper extend are surveyed briefly and analyzed A new implementation for priority queues is employed, and a class of "arc set partition" algorithms is introduced For the single source problem on networks with nonnegative arcs a running time of $O(\min(n^{1+1/k} + e, (n + e) \log n))$ is achieved, where there are n nodes and e arcs, and k is a fixed integer satisfying $k > 0$ This bound is $O(e)$ on dense networks For the single source and all pairs problem on unrestricted networks the running time is $O(\min(n^{2+1/k} + ne, n^2 \log n + ne \log n)$

KEY WORDS AND PHRASES. shortest paths, analysis of algorithms, worst-case analysis, graph theory, network theory, single source problem, all pairs problem, sparse networks

CR CATEGORIES 5 25, 5 32, 5 42, 8 3

1. *Introduction*

Finding shortest paths in networks is a fundamental problem in combinatorial optimization The best routings of vehicles or messages, for instance, are shortest paths with respect to costs associated with the arcs of the transportation or communications network involved. Finding shortest paths is a subproblem in many optimization problems such as finding optimal flows in networks (treated for example in [11]).

Dreyfus [8] surveys known algorithms for shortest paths. A more recent survey appears in [23]. In these references and in [16] can be found additional references not cited in this paper.

We present algorithms which are faster than those currently in use on networks which are relatively sparse in arcs, a circumstance which commonly occurs in many problems involving shortest paths. The algorithm we give for shortest paths to all nodes from a single source in networks with no negative arc weights is asymptotically optimal over a broad range of arc density. We introduce a class of "arc set partition" algorithms in which are found good algorithms for solving the single source problem on unrestricted networks. The results are extended to yield an algorithm for finding shortest paths between all node pairs, which is of essentially the same complexity as the single source algorithm we present for unrestricted networks.

In the remainder of this section we introduce notation and present known results to be used in our development.

A *directed graph* $G = (V, X)$ consists of a nonempty and finite set V of *nodes* and a set of *arcs* $X \subseteq \{(u, v) \mid u, v \in V \text{ and } u \neq v\}$. Each arc is an ordered pair. There are n nodes and e arcs in G. A *network* $N = (G, w)$ is a directed graph G together with a real valued function $w : X \to \mathcal{R}$. The real number $w(u, v)$ is the *weight* of arc $(u, v) \in X$.

A *path* from u to v in G is a finite sequence $q_{uv} = (u = v_1, v_2, \ldots, v_k = v)$ of nodes of

Copyright © 1977, Association for Computing Machinery, Inc General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery

This work was supported in part by the National Science Foundation under Grant GJ-28176 and a National Science Foundation graduate fellowship

Author's address Computer Science Department, Whitmore Laboratory, The Pennsylvania State University, University Park, PA 16802

G satisfying $(v_i, v_{i+1}) \in X$ for $i = 1, 2, \ldots, k-1$, where $k > 0$. Unless stated otherwise, a numbering of the nodes as in q_{uv} is without loss of generality. It may be that some nodes are repeated in q_{uv}, but if not (except possibly for $u = v$), q_{uv} is *elementary*. The weight function w is extended to paths in the following manner:

$$w(q_{uv}) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}) .$$

A path q_{uv} is a *cycle* if $u = v$. It may be seen that for any nodes u and v in G there is a path (or cycle) q_{uv} if and only if there is an elementary path (or elementary cycle) p_{uv}.

A *shortest path* from u to v in a network N is a path q_{uv} for which the path weight $w(q_{uv})$ is a minimum over the weights of all paths from u to v. A consequence of this definition is that there is a shortest path from u to v exactly when there is a path p_{uv} and, for every path $q_{uv} = (u = v_1, v_2, \ldots, v_k = v)$, $v_i = v_j$ implies $w(v_i, v_{i+1}, \ldots, v_j) \geq 0$ for all i and j, $1 \leq i < j \leq k$. Thus for any pair of nodes u and v, if there is a shortest path q_{uv}, there is an elementary shortest path p_{uv}. In addition it may be seen for a shortest path $q_{uv} = (u = v_1, v_2, \ldots, v_k = v)$ that the subpath $q_{v_i v_j} = (v_i, v_{i+1}, \ldots, v_j)$ is a shortest path from v_i to v_j for all i and j, $1 \leq i < j \leq k$.

A *shortest path problem* is a network N together with a set $\psi \subset V \times V$ of ordered pairs of nodes between which shortest paths are to be found. We consider two problems in this paper: the *single source problem* (N, ψ_s), where ψ_s is defined by a distinguished *source node* s as $\psi_s = \{(s, v) \mid v \in V\}$, and the *all pairs problem* (N, ψ_*), where $\psi_* = \{(u, v) \mid u, v \in V, u \neq v\}$. For simplicity of exposition we confine our discussions to ψ satisfying $(v, v) \notin \psi$ for all v. It is easy to extend the results to the finding of shortest cycles.

While a complete solution to a shortest path problem (N, ψ) will contain a set of all distinct shortest paths for each pair in ψ, we shall be content to find one path for each pair. Such a solution may be represented by a minimum weight function $\mu : \psi \to (\mathcal{R} \cup \{-\infty, \infty\})$ satisfying

$$\mu(u, v) = \begin{cases} \infty & \text{if there is no path } q_{uv}, \\ w(p_{uv}) & \text{if there is a shortest path } p_{uv}, \\ -\infty & \text{if there is a path } q_{uv} \text{ but no shortest path is defined for } (u, v). \end{cases}$$

The symbols $-\infty$ and ∞ satisfy the relations $-\infty < z < \infty$ for all z in $\mathcal{R}$. We shall discuss computing μ for a given problem (N, ψ). For $\mu(u, v) > -\infty$ it will be clear how to implement bookkeeping operations to record some shortest path from u to v. If $\mu(u, v) = -\infty$ for any node pair (u, v) it may be that the algorithms will fail to compute μ. In this connection it should be mentioned that we do not treat the problem of finding shortest elementary paths for node pairs (u, v) in networks where $\mu(u, v) = -\infty$. As was first noticed by Dantzig [5], a procedure to solve such a problem implies a procedure of equal asymptotic complexity for the traveling salesman problem, known to be NP-complete [20].

The following algorithm for the single source problem is due to Ford [11, 3].

Algorithm A

```
1. d(v) := ∞ for all v ≠ s,
2  d(s) = 0;
3  while R(X) do
4     begin
5        Let (u, v) satisfy R(u, v),
6        d(v) = d(u) + w(u, v)
7     end
```

where $R(u, v) \equiv (u, v) \in X$ and $d(v) > d(u) + w(u, v)$, and

$R(Y) \equiv$ there exists $(u, v) \in Y$ satisfying $R(u, v)$

is the extension of R to subsets Y of X.

If $\mu(s, v) > -\infty$ for all $(s, v) \in \psi_s$, Algorithm A terminates with $d(v) = \mu(s, v)$ for all

$v \in V$. Otherwise, if $\mu(s, v) = -\infty$ for some $(s, v) \in \psi_s$, then the algorithm does not terminate. Correctness in the former case follows by application of the observation made above that all subpaths of shortest paths are shortest and the fact that, for all v and at each step of the algorithm, $d(v)$ is either ∞ or the weight $w(p_{sv})$ of some path p_{sv}.

The efficiency of Algorithm A depends on how the search is organized for $(u, v) \in X$ satisfying $R(u, v)$. Moore [24] and Bellman [2] found algorithms which may be described as Algorithm A with the search organized into stages in each of which $\{(u, v) \mid u \neq v\} \supseteq X$ is searched exhaustively and the indicated updates in d made for each arc found to satisfy R. As they note, n stages suffice. If X is stored as an $n \times n$ incidence matrix, where a node pair (v_i, v_j) is associated with its weight $w(v_i, v_j)$ if (v_i, v_j) is an arc or with ∞ otherwise, the running time of such an implementation is $O(n^3)$. If X is stored compactly as, say, a list of arcs, an implementation with a running time of $O(ne)$ can be found. As we observe in Section 3, some exhaustive search strategies can yield exponential running times for Algorithm A.

Search strategies need not be exhaustive in X to be sufficient for correctness of Algorithm A. If some arc (u, v) is found to satisfy $d(v) \leq d(u) + w(u, v)$ it need not be looked at again unless $d(u)$ is subsequently reduced in value. Also, if it becomes known that $d(u) = \mu(s, u)$, no arc (v, u) need be considered again, and every arc (u, w) need be considered at most once more. The first observation is employed in the following refinement of Algorithm A.

Algorithm B

```
1  d(v) = ∞ for all v ≠ s,
2  d(s) = 0,
3  U = {s},
4  while U ≠ ∅ do
5    begin
6      Choose u ∈ U,
7      U = U − {u},
8      for each (u, v) ∈ X do
9        if d(v) > d(u) + w(u, v) then
10         begin
11           d(v) = d(u) + w(u, v),
12           U = U ∪ {v}
13         end
14   end
```

It may be seen that $R(X)$ implies $U \neq \emptyset$ when the test in step 4 is performed Thus this algorithm is a correct implementation of Algorithm A under any rule of choice in step 6.

It may be shown by induction over the execution of the algorithm that, on any problem satisfying $\mu(s, v) > -\infty$ for all nodes v, $U \neq \emptyset$ at step 6 implies the existence of $u_0 \in U$ satisfying $d(u_0) = \mu(s, u_0)$. If such a u_0 is always chosen in step 6 it follows that, once deleted, no node ever returns to the set U. In such a case the block beginning at step 5 is executed at most n times.

For an acyclic graph, such a u_0 may be chosen by the rule

6 *Choose* $u \in U$ *satisfying for all* $(v, u) \in X$ *there exists a path* p_{si} *implies* v *has been removed from* U, (1)

This rule can be implemented so that the time expended by the rule over the entire execution of Algorithm B is $O(n)$ [21, pp. 258–268].

When all arc weights are nonnegative, the rule

6 *Choose* u *satisfying* $d(u) = \min_{v \in U}(d(v))$ (2)

always selects such a u_0. Algorithm B embodying this rule of choice is attributed to Dijkstra [7]. Since the node u minimizing d in U can be found in time $O(\mid U \mid)$,[1] a simple analysis shows that Algorithm B with rule (2) runs in time $O(n^2)$ In Section 2 we show how to reduce this bound significantly on families of networks in which e grows more slowly than n^2.

[1] $\mid Y \mid$ denotes the cardinality of the set Y

The above analysis indicates that Algorithm B with rule (2) is a correct implementation of Algorithm A whether or not all arc weights are nonnegative. The restriction to nonnegative arc weights is important nonetheless. On general networks with $\mu(s, v) > -\infty$ for all v, a worst-case running time of $O(n\,2^n)$ can be realized by Algorithm B with rule (2), as was shown in [17].

2. *A Single Source Algorithm Optimal on Dense Nonnegatively Weighted Networks*

Under a class of selection rules including rules (1) and (2), the set U in Algorithm B is a *priority queue*, a set of elements, labeled with real-valued priorities, together with certain procedures which operate on the set. The procedures of interest are those which insert a new element into the queue, delete a specified element, and find and delete an element which maximizes or minimizes priority in the queue. In Algorithm B, elements are inserted in steps 3 and 12. Elements are deleted in step 7 according to the priority defined by the selection rule in step 6. For selection rules which depend on d, deletion of a node v is also implied by the change in $d(v)$ which occurs in step 11.

When a large number of deletions by priority is anticipated, it is useful to store a priority queue as a *heap*, a binary tree ordered on each path from the root according to element priority. As is well known [22, 1], each of the operations (insert, delete, and delete by priority) may require reordering of some path in the heap containing at most a number of elements proportional to the logarithm of the size of the heap. Since the size of set U in Algorithm B is bounded by n, using a binary heap for U leads to a running time analysis for Algorithm B using a heap for U as follows:

Step	Bound on step execution time
1	$O(n)$
6, 7	$O(\log n)$
11, 12	$O(\log n)$

Logarithms are base 2 except where stated otherwise. All other steps of Algorithm B cost $O(1)$. As we have seen, where selection rule (2) is used and all arc weights are nonnegative, steps 6 and 7 are executed at most n times and steps 9 through 13 are executed at most e times. It follows that there is an implementation of Algorithm B using rule (2) which runs in time $O((n + e) \log n)$ on nonnegatively weighted networks. It can be shown that this bound is realized under an implementation incorporating a binary heap. We would like an implementation with a bound on running time of $O(n + e)$.

A class of priority queue problems is analyzed in [18] which includes the priority queue embedded in Algorithm B with rule (2). An additional operation, *update*, is defined to be an improvement (or decrease) of the priority of a queue element. This is precisely the queue operation which occurs in steps 11 and 12 of Algorithm B. When such a priority decrease occurs, the element with the decreased priority is moved toward the root by repeated interchange with its parent and in the heap until heap order is restored. Updates and insertions have this property in common: They both induce movement of an element over the unique path from some element to the root of the heap. By contrast, deletion either by priority or not induces movement along some path away from the root. Such a path is not uniquely determined by the heap structure, so at each step all offspring in the heap of some element must be examined in order to locate the next element in the path to be reordered. Thus updates and insertions cost $O(\log_\beta \iota)$ and deletions cost $O(\beta \log_\beta \iota)$, where β is the maximum number of offspring of any heap element in a heap of size ι.

In the usual analysis of queues using binary heaps, update costs time proportional to the logarithm of the size of the queue. In [18], heaps of bounded height, in which the length of every path from the root is bounded by a constant $k > 1$, are introduced in order to take advantage of the fact that the path to be reordered in an update operation is uniquely determined by the position of the element updated. Let the height of the heap be bounded by a constant $k > 1$. Such a heap of size n can be constructed in which each

element has at most $\beta = \lceil n^{1/k} \rceil$ offspring succeeding it on paths from the root. It follows that deletion by priority will cost $O(n^{1/k})$, and updates and insertions will be of constant cost. Applying these ideas in an implementation of Algorithm B gives an overall cost of $O(n^{1+1/k})$ for deletion by priority and $O(e)$ for updates and insertions. As already observed, the other operations in Algorithm B are $O(n + e)$, provided the network is nonnegatively weighted. These results are stated in the following theorem.

THEOREM 1. *There is an algorithm for finding shortest paths from a single source in nonnegatively weighted networks which runs in time* $O(\min(n^{1+1/k} + e, (n + e) \log n))$ *for a fixed integer* $k > 0$.

A correct interpretation of this asymptotic formula in the two parameters n and e involves the notion of families of networks defined by arc density. Thus for a fixed function g, all graphs satisfying $e = \lceil g(n) \rceil$ are said to have the same arc density. With respect to the asymptotic formula, as n grows, the number of arcs e grows accordingly, so that the dominant term of the asymptotic formula depends on arc density. It is assumed that in making use of these results there will be some choice of k, and therefore a simple test based on the bounds in Theorem 1 may be made to choose which implementation to apply to a given problem.

Let a *dense network* satisfy $e = \Omega(n^{1+\epsilon})$ for some fixed positive ϵ, where $\Omega(f(n))$ denotes any function which is greater than $cf(n)$ for some positive c and for all but finitely many n. With this definition we note that all dense graphs except those of maximum arc density are normally termed "sparse." As the following corollary emphasizes, Theorem 1 presents a new optimal bound for sparse graphs in this class.

COROLLARY 1. *There is an algorithm which runs in optimal time* $O(e)$ *for finding shortest paths from a single source in dense nonnegatively weighted networks.*

In the preceding analysis, heap order is restored after each update and insertion. Since heap order is not required until a deletion by priority must be performed, it is possible to defer restoring heap order until it is needed. An analysis of deferred reordering in the general case is found in [16]. It is shown there that deferred reordering is never better asymptotically for any network, dense or otherwise. An algorithm and an analysis of the "regular" case, where the network N has an equal number of arcs incident from each node, is analyzed in [19]. Another algorithm which exploits arc sparsity is discussed by Wagner [29]. This algorithm differs from the ones we present in an important respect. Its complexity grows with the path weights themselves. Thus Wagner's algorithm is appropriate for specialized applications where arc weights are known to be small and nonnegative.

3. *Behavior of Algorithm A on Unrestricted Networks*

In Section 1 we observed that Algorithm A fails to terminate if $\mu(s, v) = -\infty$ for some node v. (Algorithm B has the same property.) Once a search strategy is chosen in Algorithm A, it is usually a simple matter to modify the algorithm so that termination is assured on unrestricted networks. If the strategy is analyzed to obtain a worst-case bound on the number of arcs that will be examined for networks in which $\mu(s, v) > -\infty$ for all nodes v, a counter can be incorporated into the algorithm to count the number of arcs examined. If the worst-case bound is ever exceeded the algorithm is made to terminate, reporting that a cycle of negative weight has been discovered In this way the $O(ne)$ bound of Section 1 holds for an algorithm which either finds shortest paths from a single source or detects a cycle of negative weight.

Some search strategies give very bad results in Algorithm A. Let a fixed list of the arcs of a given network be searched from the beginning each time $R(X)$ is evaluated in step 3. This apparently innocuous strategy yields exponential running time in the worst case.

THEOREM 2. *There exists a simple, exhaustive search strategy under which Algorithm A has a running time of* $\Omega(2^n)$ *on certain networks with nonnegative weights.*

PROOF. Let $N_n = ((V, X), w)$ where the nodes of V are represented by their indices, i.e. $V = \{1, 2, \ldots, n\}$. Let the arc set be $X = \{(i, j) \mid 1 \leq j < i \leq n\}$ and define w as

$$w(2, 1) = 1,$$
$$w(i, j) = 2^{i-j}w(i - 1, 1), \quad 1 \leq j < i < n, \ i > 2,$$
$$w(n, n - 1) \geq 2w(n - 1, 1), \quad n > 2,$$
$$w(n, j - 1) \geq 2w(n, j), \quad n > 2, \ 1 < j < n.$$

For example, the weights for N_5 are

w \ i \ j	1	2	3	4	5
1					
2	1				
3	4	2			
4	32	16	8		
5	$\geq 2w(5, 2)$	$\geq 2w(5, 3)$	$\geq 2w(5, 4)$	≥ 64	

Let the single source problem (N_n, ψ_n) be posed, and let X be represented by a list L_X, ordered by columns,

$$L_X = ((2, 1), (3, 1), \ldots, (n, 1), (3, 2), (4, 2), \ldots, (n, 2), \ldots, (n, n - 1)).$$

Let $R(X)$ be evaluated in step 3 of Algorithm A by a search of L_X. We claim that the loop composed of steps 4 through 7 will be executed 2^{n-1} times if step 5 employs the arc found by the evaluation of $R(X)$.

Each evaluation of $R(X)$ except the last to be performed results in an assignment of a value in d. Let a sequence S_n be formed from the values so assigned. For N_5, the sequence begins

$$S_5 = (w(5, 1), w(5, 2), w(5, 2) + 1, w(5, 3), w(5, 3) + 4, w(5, 3) + 2, w(5, 3) + 3, \ldots).$$

For any N_n we claim there are exactly $2^{n-1} - 1$ elements in S_n. This hypothesis can be verified directly for $n = 2$; the sequence $S_2 = (1)$.

Assume the hypothesis for all $N_1, N_2, \ldots, N_n$, and consider the result of running the algorithm on any N_{n+1}. Since arc $(n + 1, n)$ is the only path from s to n, it is clear that $w(n + 1, n)$ will appear somewhere in the sequence S_{n+1}. Consider the sequence of searches before this point. It must be identical to that on some N_n' in which row n of the weight matrix is row $n + 1$ of the weight matrix for N_{n+1}. Thus $2^{n-1} - 1$ entries in S_{n+1} precede $w(n + 1, n)$.

By the nonincreasing property of d we know that the remaining elements in S_{n+1} will not involve any weights on row $n + 1$ of the weight matrix for N_{n+1} except $w(n + 1, n)$. Since $w(n + 1, n)$ plus any $n - 2$ weights from rows above row $n + 1$ is less than any $d(i)$, $i < n$, it is clear that the remainder of S_{n+1} will be a sequence identical to that formed by some N_n'' but with $d(s) = w(n + 1, n)$ at the start.

We conclude that S_{n+1} has $2^{n-1} - 1 + 1 + 2^{n-1} - 1 = 2^n - 1$ elements, and, by induction, that there are 2^{n-1} evaluations of $R(X)$ performed for any N_n. □

4. *Arc Set Partition Algorithms*

In Algorithm B the search for an arc to reduce some path weight in d is restricted to a subset of arcs from which are excluded dynamically certain arcs known to not satisfy R. It is useful to restrict the search further by confining the search to each element in turn of a fixed partition of X, exhausting all candidates in one element before proceeding to the next. The general arc set partition algorithm, incorporating an a priori partition into Algorithm B, is as follows.

Let $X_1, X_2, \ldots, X_k$ be some partition of X.

```
Algorithm C(X_1, X_2, ..., X_k)
    1   d(v) := ∞ for all v ≠ s,
    2   d(s) = 0;
    3   U_1 = U_2 := ... U_k = {s};
    4   while U ≠ ∅ do
    5     begin
    6       while U_1 ≠ ∅ do S_1,
    7       while U_2 ≠ ∅ do S_2,
                .
k + 5       while U_k ≠ ∅ do S_k
k + 6     end
```

where $U = U_1 \cup U_2 \cup \cdots \cup U_k$ and for $1 \leq i \leq k$, S_i is the block

```
    1.  begin
    2     choose u ∈ U_i,
    3     U_i = U_i - {u},
    4     for each (u, v) ∈ X_i do
    5       if d(v) > d(u) + w(u, v) then
    6         begin
    7           d(v) = d(u) + w(u, v),
    8           U_1 = U_1 ∪ {v},
    9           U_2 = U_2 ∪ {v},
                .
k + 7           U_k := U_k ∪ {v}
k + 8         end
k + 9   end
```

It may easily be seen that for any partition X the above algorithm is simply Algorithm B under a rule of choice which depends on both the partition and the rules of choice in the S_i. In fact, Algorithm C reduces to Algorithm B under the trivial partition $X = X$. Thus for any rule of choice in the S_i and for any partition of X, Algorithm C terminates correctly if $\mu(s, v) > -\infty$ for all v and does not terminate otherwise.

For any partition $X = X_1 \cup X_2 \cup \cdots \cup X_k$ a path p_{sv} is partitioned into *uniform subpaths*, maximal subpaths confined to single elements of the partition of X. A shortest path may contain at most $n - 1$ uniform subpaths. Algorithm C extends a shortest path by an entire uniform subpath in any S_i in which the path is extended at all. Two-element partitions, $X = X_1 \cup X_2$, are of particular interest since uniform subpaths will alternate between X_1 and X_2 on any shortest path. Consequently the test in line 4 of Algorithm $C(X_1, X_2)$ can be replaced by a test on U_1 alone: **while** $U_1 \neq \emptyset$ **do.** Each execution of S_1 or S_2 except possibly the first and last will extend some shortest path by a uniform subpath The remainder of this section is devoted to analyzing Algorithm $C(X_1, X_2)$.

Let an execution of

```
j + 5   while U_j ≠ ∅ do S_j
```

for $j = 1, 2$ during execution of Algorithm $C(X_1, X_2)$ be called a *half-iteration*, and designate $U_1 \cup U_2$ by U. We wish to find a bound on the number of half-iterations executed by Algorithm $C(X_1, X_2)$. Half-iterations will be counted in order of execution, beginning with 1.

LEMMA 1. *On any problem satisfying $\mu(s, v) > -\infty$ for all nodes v, if half-iteration i is completed, then every node u for which there is a shortest path with no more than $i - 1$ uniform subpaths satisfies $d(u) = \mu(s, u)$.*

PROOF. The proof is by induction. It suffices to state the proof for networks in which all nodes are reachable from s. Assume immediately following half-iteration $i = h$ for some $h \geq 1$ that, for each node v, there exists a shortest path $p_{sv} = q_{v_0v_1}q_{v_1v_2} \cdots q_{v_{l-1}v_l}$ composed of uniform subpaths $q_{v_{j-1}v_j}$ for $1 \leq j \leq l$ and satisfying the following conditions:

(a) There is no shortest path from s to v with fewer uniform subpaths.

(b) If $q_{v_0v_1}$ is drawn from X_1, then

(i) $d(v) = \mu(s, v)$ if $l \leq i$,
(ii) $d(v_i) = \mu(s, v_i)$,
(iii) $v_i \in U$,
(iv) $d(v_j) > \mu(s, v_j)$ for $i < j \leq l$
} if $l > i$.

(c) If $q_{v_0v_1}$ is drawn from X_2, then
(i) $d(v) = \mu(s, v)$ if $l \leq i - 1$,
(ii) $d(v_{i-1}) = \mu(s, v_{i-1})$,
(iii) $v_{i-1} \in U$,
(iv) $d(v_j) > \mu(s, v_j)$ for $i - 1 < j \leq l$
} if $l > i - 1$.

We know from the discussion of Section 1 that, if there exists a node v satisfying $d(v) > \mu(s, v)$ following half-iteration h, half-iteration $h + 1$ will be executed. The discussion of Section 1 also applies individually to **while** $U_j \neq \emptyset$ **do** S_j for $j = 1, 2$. Thus we observe that the execution of **while** $U_j \neq \emptyset$ **do** S_j in half-iteration $h + 1$ will set $d(u) = \mu(s, u)$ for all nodes u for which there is some shortest path p_{wu}, confined to X_1 or X_2, respectively; w belongs to U following half-iteration h; and at that time $d(w) = \mu(s, w)$.

Given our assumption for $i = h$, let half-iteration $h + 1$ be executed. For each node v, if $d(v) = \mu(s, v)$ then (i) of (b) or (c) holds for $i = h + 1$ and some shortest path p'_{sv}. If $d(v) > \mu(s, v)$, then p_{sv} satisfying (b) or (c) for $i = h$ must satisfy (ii) and (iii) of (b) or (c) for $i = h + 1$ by the path extension property of **while** $U_j \neq \emptyset$ **do** S_j just discussed. Furthermore, (iv) of (b) or (c) must hold for $i = h + 1$. Otherwise (a) would have been violated for $i = h$ since it is clear that **while** $U_1 \neq \emptyset$ **do** S_1 cannot extend a path in X_2 nor can **while** $U_2 \neq \emptyset$ **do** S_2 extend a path in X_1.

Applying the same arguments on **while** $U_1 \neq \emptyset$ **do** S_1 to the given conditions $U = \{s\}$ and $d(v) > \mu(s, v)$ for $v \neq s$, we can conclude our assumption holds for $h = 1$. Thus by induction the lemma is proved. □

COROLLARY 2 *On any problem satisfying* $\mu(s, v) > -\infty$ *for all nodes* v, *if half-iteration* i *is completed, then* $| U | \leq n - i + 1$.

We are now ready to state the arc set partition algorithm applicable to general single source problems

Algorithm SINGLESOURCE(X_1, X_2)

```
1   d(v) = ∞ for all
2   d(s) := 0,
3   U1 = {s},
4   U2 = {s},
5   count = 0,
6   while 0 < | U1 | ≤ n+1 − count do
7     begin
8       innercount := n+1 − count,
9       while 0 < | U1 | ≤ innercount do
10        begin
11          S1 where the rule of choice guarantees that u chosen will not return to U1 if μ(s, v) > −∞ for all v,
12          innercount = innercount−1
13        end,
14      count = count+1,
15      innercount = n+1 − count,
16      while 0 < | U2 | ≤ innercount do
17        begin
18          S2 where the rule of choice guarantees that u chosen will not return to U2 if μ(s, v) > −∞ for all v,
19          innercount = innercount−1
20        end,
21      count = count+1
22    end,
23. if | U1 | = 0 then report "success"
                  else report "failure"
```

THEOREM 3. *Algorithm SINGLESOURCE*(X_1, X_2) *always terminates and, for* X_1 *and* X_2 *satisfying* $X_1 \cup X_2 = X$ *and* $X_1 \cap X_2 = \emptyset$,

(*i*) *when* $\mu(s, v) > -\infty$ *for all* v, *the algorithm reports "success" and* $d(v) = \mu(s, v)$ *for all* v, *and*

(*ii*) *when there exists* v *for which* $\mu(s, v) = -\infty$ *the algorithm reports "failure".*

PROOF Termination is assured by the counters. Correctness follows from the correctness of Algorithm B, Lemma 1, Corollary 2, and the analysis in Section 1 of the complexity of Algorithm B under rules of choice which prevent nodes from returning to the set U. □

We now consider the running time of Algorithm SINGLESOURCE. Let additions and comparisons of path weights be *active operations*, and let c_1 and c_2 be monotonic functions chosen so that $c_1(\mid U_1 \mid)$ and $c_2(\mid U_2 \mid)$ bound the number of active operations required to choose and delete node u in steps 2 and 3 of S_1 and S_2, respectively. S_1 and S_2 also perform active operations when $d(v)$ is tested and possibly changed in steps 5 and 7. The test and change can be implemented to cost 2 active operations in all cases In addition, let b_1 and b_2 be monotonic functions chosen so that $b_1(\mid U_1 \mid)$ and $b_2(\mid U_2 \mid)$ bound the active operations required to restore heap order in U_1 and U_2, respectively, in steps 8 and 9 following a change in $d(v)$ for some node v. Let $\tau(n, e)$ be the number of active operations performed in the worst case by Algorithm SINGLESOURCE on inputs with n nodes and e arcs.

In general, the number of active operations performed in Algorithm SINGLESOURCE will depend on the choice of the start node s. We give below an expression which bounds the number of operations with respect to a given graph with n nodes and e arcs and a particular indexing of the nodes. Maximization of this expression over all graphs and all indexings will then bound $\tau(n, e)$. The first term $c_1(1)$ is the cost of choosing s from U_1 in the very first iteration of S_1. The second term accounts for the nodes admitted to U as a result of expanding s. The third term completes the operation count for the remaining iterations of the first execution of S_1. The fourth term accounts for the remaining executions of S_1. The fifth term accounts for the executions of S_2.

For n even,

$$\tau(n, e) \leq \max_{\pi \in \Pi, G \in G(n,e)} \Bigg[c_1(1) + (2 + b_1(\Theta(\pi, n, X_1)) + b_2(\Theta(\pi, n, X_1)))\Theta(\pi, n, X_1) + \sum_{\iota=1}^{n-1} (c_1(\iota) + (2 + b_1(\iota) + b_2(n))\Theta(\pi, \iota, X_1)) + \sum_{k=1,3,\ldots,n-1} \sum_{\iota=1}^{k} (c_1(\iota) + (2 + b_1(\iota) + b_2(k))\Theta(\pi, \iota, X_1)) + \sum_{k=2,4,\ldots,n} \sum_{\iota=1}^{k} (c_2(i) + (2 + b_2(\iota) + b_1(k))\Theta(\pi, i, X_2)) \Bigg],$$

where Π is the set of all permutations on $(1, 2, \ldots, n)$, $G(n, e)$ is the set of all directed graphs with n nodes and e arcs, and $\Theta(\pi, \iota, Y) = \mid \{(v_{\pi(\iota)}, w) \mid (v_{\pi(\iota)}, w) \in Y\} \mid$ for $Y \subseteq X$ and $v_1, v_2, \ldots, v_n$ an arbitrary fixed indexing of V. From the definition of Θ it may be seen that $\Theta(\pi, \iota, X_1) + \Theta(\pi, \iota, X_2) = \Theta(\pi, \iota, X)$ for all π, ι, and partitions $X_1 \cup X_2 = X$. Also it is clear that, for any maximizing π and $i, j < n$, $\Theta(\pi, \iota, Y) > \Theta(\pi, j, Y)$ implies $i < j$. Thus we may simplify, collecting terms and using the monotonicity of b_1, b_2, c_1, and c_2.

$$\tau(n, e) \leq \max_{\pi \in \Pi, G \in G(n,e)} \Bigg[\sum_{\iota=1}^{n} (c_1(i) + c_2(\iota) + 2\Theta(\pi, \iota, X) + b_1(\iota)\Theta(\pi, i, X) + b_2(\iota)\Theta(\pi, \iota, X)) + \frac{1}{2} \sum_{k=1}^{n} \sum_{\iota=1}^{k} (c_1(\iota) + c_2(i) + 2\Theta(\pi, i, X) + b_1(k)\Theta(\pi, i, X) + b_2(k)\Theta(\pi, \iota, X)) \Bigg]. \tag{3}$$

For any subset of arcs $Y \subseteq X$,

$$\max_{\pi \in \Pi, G \in G(n,e)} \sum_{\iota=1}^{n} \Theta(\pi, \iota, Y) = \mid Y \mid, \tag{4}$$

$$\max_{\pi\in\Pi, G\in G(n,e)} \sum_{k=1}^{n}\sum_{i=1}^{k} \Theta(\pi, i, Y) \le \sum_{j=1}^{[|Y|/(n-1)]} (n-1)j + \sum_{j=1+[|Y|/(n-1)]}^{n} |Y| \le |Y|(n - |Y|/2n + 1). \quad (5)$$

Combining (3), (4), and (5) gives upon further simplification

$$\tau(n, e) \le 2\sum_{i=1}^{n} \max(c_1(i), c_2(i)) + \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{k} (c_1(i) + c_2(i)) + \frac{1}{2}(2 + b_1(n) + b_2(n))\, e\, (n - e/2n + 5). \quad (6)$$

The above result (6) has been stated so that it holds as well for n odd. With this analysis of Algorithm SINGLESOURCE we may state the following theorem.

THEOREM 4. *Any single source problem may be solved within the following bounds, finding $\mu(s, v)$ for all nodes v provided $\mu(s, v) > -\infty$ for all v, or detecting that there exists a node u for which $\mu(s, u) = -\infty$.*

(a) $\frac{1}{2}n^3 + O(n^2)$ active operations,

(b) $O(\min(n^{2+1/k} + ne, n^2 \log n + ne \log n))$ operations, where k is a fixed integer satisfying $k > 0$.

PROOF. We assume the given network is presented in a suitable form as discussed in Section 1. Thus the size of the input is taken as $O(n + e)$ and in time bounded by this quantity the subnetwork, of the given network, reachable from s can be found. Thus we assume every node of the given network is reachable from s.

Define the partition $\vec{X} \cup \overleftarrow{X} = X$ where for an arbitrary indexing $V = v_1, v_2, \ldots, v_n$ the set $\vec{X} = \{(v_i, v_j) \mid (v_i, v_j) \in X \text{ and } i < j\}$ and $\overleftarrow{X} = X - \vec{X}$. Since both $(V, \vec{X})$ and $(V, \overleftarrow{X})$ are acyclic, if we let $X_1 = \vec{X}$ and $X_2 = \overleftarrow{X}$, rules of choice are easily found for S_1 and S_2 which guarantee that, once removed from the appropriate set, U_1 or U_2, a node u will not return to the set during the same execution of S_1 or S_2. For S_1, choose $u = v_j$ where the index j is minimized in U_1. For S_2, choose a node with maximum index in U_2. Since $c_1(i) = 0$, $c_2(i) = 0$, $b_1(i) = 0$, $b_2(i) = 0$, $1 \le i \le n$, it is easily seen from (6) that $\tau(n, n(n-1)) \le \frac{1}{2}n^3 + O(n^2)$ as claimed. A bound of $\tau(n, n(n-1)) \le \frac{2}{3}n^3 + O(n^2)$ is achieved by Yen [31] (see also [32]).

For part (b) of the bound, define the partition $X_p \cup X_m = X$, where $X_p = \{x \mid x \in X \text{ and } w(x) \ge 0\}$ and $X_m = X - X_p$. The sets X_p and X_m can be constructed in $O(n + e)$ time given a suitable input for the network as discussed in Section 1. Furthermore, the graph (V, X_m) can be tested for cycles in $O(n + e)$ time. If a cycle is found, then $\mu(s, u) = -\infty$ for some node u, and the input can be rejected within the bound (b) of the theorem. Thus we may assume (V, X_m) is acyclic. Let $X_1 = X_p$ and $X_2 = X_m$. We again need suitable rules of choice. For S_1 we employ rule (2). For S_2 the nodes V can be ordered to be consistent with the partial order defined by (V, X_m). As above, a rule of choice can then be implemented for which $c_2(i) = 0$ and $b_2(i) = 0$. The bookkeeping costs associated with insertion and deletion on U_2 will be constant for each insertion and deletion provided a cost of $O(n)$ is charged to each half-iteration of S_2 to account for scanning an ordering of V consistent with (V, X_m). A heap implementation as described in Section 2 is employed for U_1, so that $c_1(i) = \beta \log_\beta i$ and $b_1(i) = \log_\beta i$ for some $\beta \ge 2$. In this case, bookkeeping is dominated by c_1 and b_1, and it may be seen that for the algorithm as a whole active operations are not dominated by other operations. Thus we may employ (6), giving

$$\tau(n, e) = O\left[\sum_{k=1}^{n}\sum_{i=1}^{k} (\beta \log_\beta i) + ne + ne(\log_\beta n)\right].$$

As was seen in the analysis for Theorem 1, $\beta \log_\beta i = O(kn^{1/k})$ for suitable β. Thus

$$\tau(n, e) = O(\min(n^{2+1/k} + ne, n^2 \log n + ne \log n))$$

for a fixed integer $k > 0$. □

A comparison of algorithms with respect to worst-case performance is shown in Figure 1. It may be seen from our analysis that the behavior of Algorithm SINGLESOURCE under the partition (X_p, X_m) will tend toward the bound of $O(\min(n^{1+1/k} + e, (n + e) \log n))$ given in Theorem 1 as the number of arcs of negative weight decreases or as the maximum number of uniform subpaths in the best such path for each node declines.

5. *An Application to the All Pairs Problem*

The standard algorithm for solving the all pairs problems (N, ψ_*) is due to Floyd [10] and Warshall [30]. This algorithm typically requires $2n^3 - O(n^2)$ active operations. Hoffman and Winograd [14] present an algorithm in which n^3 comparisons but only $O(n^{5/2})$ addition-subtraction operations are required on the data. Using some of the same ideas, Fredman [13, 12] reduces the number of active operations required to $O(n^{5/2})$. However, bookkeeping and symbol manipulation operations dominate this bound. Fredman attributes to M. Paterson a preprocessing scheme which gives an algorithm running in $O(n^3 \log \log n / \log n)$ time An algorithm is presented in [15] which runs in as little time as $O(n^2)$ on some networks. However, good results depend on finding a nontrivial decomposition of the network. Networks with as few as $2n-2$ arcs exist for which there is no nontrivial decomposition. Also it is not shown in [15] how to find suitable decompositions when they exist (see [16]).

Our result for the all pairs problem improves on the matrix algorithms mentioned above when applied to sparse networks.

LEMMA 2 [9]. *Let h be a function on nodes of a network $N = (G, w)$ which, for all nodes u and v, satisfies $w'(u, v) = w(u, v) + h(u) - h(v) \geq 0$ and a path p_{uv} in N is shortest if and only if p_{uv} is shortest in $N = (G, w')$. Such a function h is defined if and only if N has no negative cycles.*

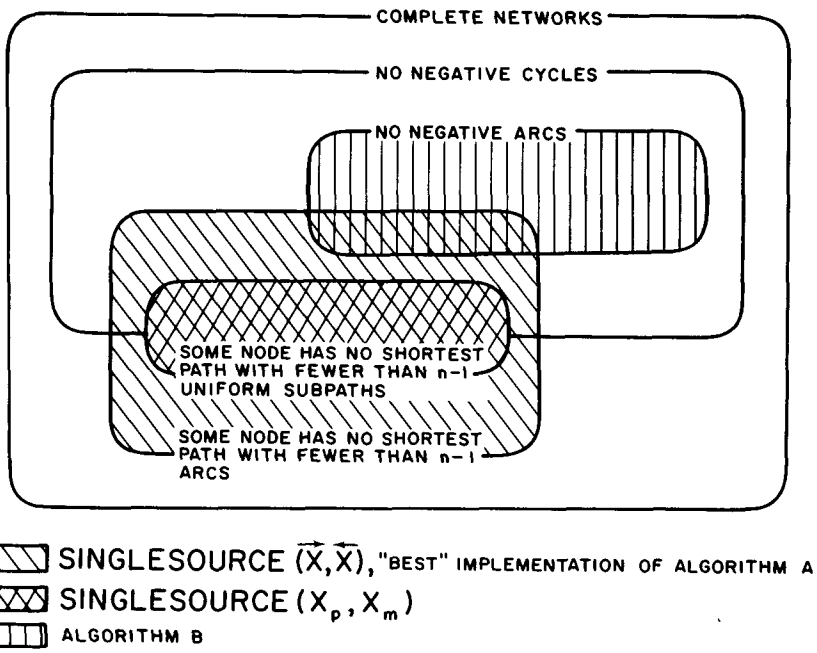

FIG 1 Problems on which worst-case performance is realized by certain algorithms for the single source problem

Such a function is also described in [26] and [27].

In any network with a node s from which all other nodes are reachable, $\mu(s, \cdot)$ satisfies the requirements for h provided $\mu(s, u) > -\infty$ for all nodes u. If a network is *strongly connected* (i.e. for any nodes u, v, there is a path p_{uv}), then h is easily computed by the methods discussed in Section 4.

THEOREM 5. *The all pairs problem can be solved or the existence of a cycle of negative weight detected by an algorithm which runs in time* $O(\min(n^{2+1/k} + ne, n^2 \log n + ne \log n))$ *for fixed* $k > 0$.

PROOF. The algorithm first constructs a strongly connected network by adding at most n arcs of weight sufficiently large so that such an arc participates in a shortest path p_{uv} in the augmented network if and only if there is no path from u to v in the given network. A weight of $w^* = n \cdot \max_{x \in X}(w(x))$ suffices. The augmentation may be done in $O(n + e)$ time. At most n arcs can be added in the augmentation.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a fixed, arbitrary ordering of V. The function h and the solution to the problem on ψ_{v_1} of the augmented network can be found in $O(\min(n^{2+1/k} + n(n + e), n^2 \log n + n(n + e) \log n))$ by the algorithm of Theorem 4. With h available, the problems on $\psi_{v_2}, \psi_{v_3}, \ldots, \psi_{v_n}$ may be solved using the algorithm of Theorem 1, in an overall running time of the same order. To translate the results on the augmented network back to the given network requires $O(n^2)$ time. If $d'_u(v)$ is the weight of a shortest path from u to v in the augmented network, then $d_u(v)$, the desired path weight, may be computed as

$$d_u(v) = \begin{cases} d'_u(v) & \text{if } d'_u(v) < w^*, \\ \infty & \text{otherwise.} \end{cases}$$

□

6. *Concluding Remarks*

Shortest path problems commonly arise on networks which are relatively sparse in arcs. We have extended the usual asymptotic analysis, parameterized by problem size expressed as the number of nodes in the network, to results which depend on arc density as well. We have shown how the work of Ford, Dijkstra, and others can be extended to yield algorithms which are faster on networks which are relatively sparse in arcs than those before known. The algorithms we show take no more time asymptotically to detect negative cycles than to solve the shortest path problem when no negative cycles are detectable. Our analysis assumes that, when negative arcs are allowed, it is not known a priori whether there are negative cycles. In special cases when it is known that negative arcs do not form negative cycles, it is likely that special purpose fast algorithms can be devised.

An algorithm for the single source problem on nonnegative networks is shown which is optimal on dense networks, those with $\Omega(n^{1+\epsilon})$ arcs for some positive ϵ. The major innovation in this algorithm is the exploitation, by a heap of fixed depth, of the embedded priority queue problem which is unsymmetric in the number of updates and insertions relative to the number of deletions by priority. A class of algorithms which we call "arc set partition" algorithms is introduced. In this framework the single source problem on unrestricted networks is solved. Our development generalizes and extends previous results. For the all pairs problem use is made of a labeling function found by Edmonds and Karp, Nemhauser, and Nilsson. An algorithm with the same bound as for the single source problem is presented.

The algorithms we give are simple enough to be useful as well as of theoretical interest.

ACKNOWLEDGMENTS. The author is grateful to David Gries for valuable comments and his critical reading of an early version of the manuscript, and to John Hopcroft for valuable discussions at the inception of this work.

REFERENCES

(Note References [4, 6, 25, 28, 33] are not cited in the text)

1 AHO, A V , HOPCROFT, J E , AND ULLMAN, J D *The Design and Analysis of Computer Algorithms* Addison-Wesley, Reading, Mass , 1974
2 BELLMAN, R E On a routing problem *Quart Appl Math 16*, 1 (1958), 87-90
3. BERGE, C *Théorie des Graphs et Ses Applications* Dunod, Paris, 1958
4 DANTZIG, G B On the shortest route through a network *Manage Sci 6*, 2 (1960), 187-190
5 DANTZIG, G B All shortest routes in a graph Proc Int Symp on Theory of Graphs (Rome, 1966), Gordon and Breach, New York, 1967, pp 91-92
6 DANTZIG, G B , BLATTNER, W O , AND RAO, M R All shortest routes from a fixed origin in a graph Proc Int Symp on Theory of Graphs (Rome, 1966), Gordon and Breach, New York, 1967, pp 85-90
7 DIJKSTRA, E W A note on two problems in connexion with graphs. *Numer Math 1* (1959), 269-271
8 DREYFUS, S E An appraisal of some shortest-path algorithms *Operations Res 17*, 3 (May-June 1969), 395-412
9 EDMONDS, J , AND KARP, R M Theoretical improvements in algorithmic efficiency for network flow problems *J ACM 19*, 2 (April 1972), 248-264
10 FLOYD, F W Algorithm 97· shortest path *Comm ACM 5*, 6 (June 1962), 345
11 FORD, L R JR , AND FULKERSON, D R *Flows in Networks* Princeton U Press, Princteon, N J , 1962
12 FREDMAN, M L On the decision tree complexity of the shortest path problems Conf Rec 16th Annual IEEE Symp on Foundations Comptr. Sci , Oct 1975, 98-99
13 FREDMAN, M L New bounds on the complexity of the shortest path problem *SIAM J Comput 5*, 1 (March 1976), 83-89
14 HOFFMAN, A J , AND WINOGRAD, S Finding all shortest distances in a directed network *IBM J Res Devel. 16*, 4 (July 1972), 412-414
15 Hu, T C , AND TORRES, W.T Shortcut in the decomposition algorithm for shortest paths in a network *IBM J Res Devel 13*, 4 (July 1969), 387-390
16 JOHNSON, D B Algorithms for shortest paths Tech Rep 73-169, Dep Comptr Sci , Cornell U , Ithaca, N Y , May 1973, available as PB-220 872, NTIS, Springfield, Va
17 JOHNSON, D B A note on Dijkstra's shortest path algorithm *J ACM 20*, 3 (July 1973), 385-388
18 JOHNSON, D B Priority queues with update and finding minimum spanning trees *Information Processing Letters 4*, 3 (Dec 1975), 53-57
19 JOHNSON, E L On shortest paths and sorting Proc ACM 25th Annual Conf , Aug 1972, pp 510-517
20 KARP, R M Reducibility among combinatorial problems In *Complexity of Computer Computations*, R E Miller and J W Thatcher, Eds , Plenum Press, New York, 1972, pp 85-103
21 KNUTH, D E *The Art of Computer Programming, Vol 1 Fundamental Algorithms* Addison-Wesley, Reading, Mass , 1968
22 KNUTH, D E *The Art of Computer Programming, Vol 3 Sorting and Searching* Addison-Wesley, Reading, Mass , 1973
23 LAWLER, E L *Combinatorial Optimization Networks and Matroids* Holt, Rinehart, and Winston, New York, 1976
24 MOORE, E F. The shortest path through a maze Proc Int Symp. on Theory of Switching, Part II, April 2-5, 1957; Annals of the Computation Lab of Harvard University 30, Harvard U Press, Cambridge, Mass , 1959, pp 285-292
25 MORÁVEK, J A note upon minimal path problem *J Math Anal Appl 30* (1970), 702-717
26 NEMHAUSER, G L A generalized permanent label setting algorithm for the shortest path between specified nodes *J Math Anal Appl 38*, 2 (May 1972), 328-334
27. NILSSON, N J. *Problem-Solving Methods in Artificial Intelligence* McGraw-Hill, New York, 1971
28 PIERCE, A R Bibliography on algorithms for shortest path, shortest spanning tree and related circuit routing problems (1956-1974) *Networks 5*, 2 (April 1975), 129-149
29 WAGNER, R A A shortest path algorithm for edge-sparse graphs *J ACM 23*, 1 (Jan 1976), 50-57
30. WARSHALL, S A theorem on Boolean matrices *J ACM 9*, 1 (Jan 1962), 11-12
31 YEN, J Y A shortest path algorithm Ph.D Th , U of California, Berkeley, Calif , 1970.
32 YEN, J Y An algorithm for finding shortest routes from all source nodes to a given destination in general networks *Quart Appl Math 27*, 4 (Jan 1970), 526-530
33 YEN, J Y Finding the lengths of all shortest paths in N-node nonnegative-distance complete networks using $\frac{1}{2} N^3$ additions and N^3 comparisons *J ACM 19*, 3 (July 1972), 423-424

RECEIVED MARCH 1976, REVISED MAY 1976