THE BELLMAN-FORD ALGORITHM AND "DISTRIBUTED BELLMAN-FORD"

DAVID WALDEN

1. Source of the Name

In the spring of 2003, I began to wonder about the history of the Bellman-Ford algorithm [CLRS01] for finding shortest paths in a graph. In particular, I was interested in understanding when Bellman's name and Ford's name became jointly associated with the algorithm.

To research these questions, I made trips to the BBN Technologies and MIT libraries, did Web searches, and sent emails to university professors who taught courses or wrote books on algorithms, parallel processing, or routing in networks. I didn't find the source of the joint name in the several weeks before I grew tired of searching the massive literature on shortest path algorithms.[1] However, I did learn a good bit, as described below.

Two highly regarded books give some history of the algorithm.

Lawler [Law76] introduces "Bellman's equations," and the Bellman-Ford method, referring to Ford's 1956 RAND note [For56] and Bellman's 1958 paper [Bel58]. He also cites Moore's 1957 [Moo59] paper, saying the Moore's method is more like Dijkstra's [Dij59].

Tarjan [Tar83] introduces Ford's "labeling method" [For56], and then a breadth-first method based on the independent ideas of Bellman [Bel58] and Moore [Moo59]. He doesn't appear to give a name to what he describes.

Deo and Pang's large bibliography [DP84] also points to these three papers, saying, "Algorithms that solve the single-source problem are usually based on the methods proposed by Bellman [Bell 58], Dijkstra [Dijk 59], Ford [Ford 56], and Moore [Moor 59]."

Therefore, I acquired and read the Ford [For56], Moore [Moo59] and Bellman [Bel58] papers. My summary and assessment of what I learned follows (but I am not a professional mathematician, so I may not have correctly understood everything I read).

Ford's 1956 paper. Ford sets up a system of linear inequalities for what he calls Problem B in his paper. He then writes down the dual problem,

$$x_i + l_{ij} \geq x_j \tag{1}$$

$$x_0 = 0 \tag{2}$$

$$\text{maximize } x_n \tag{3}$$

[1]If you know where Bellman's name and Ford's name first became associated jointly with this algorithm, please let me know.

where x_i is the current approximation of the distance from the source node 0 to destination i and l_{ij} is the length from node i to node j. He next turns inequality 1 above into the inequality,

$$x_i - x_j \geq l_{ji} \tag{4}$$

Finally, Ford gives a "computing procedure":

> Assign initially $x_0 = 0$ and $x_i = \infty$ for $i \neq 0$. Scan the network for a pair P_i and P_j with the property that $x_i + l_{ji} \geq x_j$. For this pair replace x_i by $x_i + l_{ji}$. Continue this process. Eventually no such pairs can be found, and x_N is now minimal and represents the minimal distance from P_0 to P_N. Clearly if no such pairs can be found, the system [equations 1-3 above] is satisfied.

So, this "computing procedure" is sort of like Bellman-Ford as it is often described today except that no useful order is given for iterating over the nodes and edge weights.[2] Ford finishes by giving an optimality proof.[3]

Moore's 1957 paper.[4] Moore sketches four methods which he calls algorithms in his paper. Algorithms A and D are relevant to the Bellman-Ford question. His algorithm works as follows (he describes his "algorithms" by giving examples of processing a specific graph, which I paraphrase here):

> Write 0 on the source. Then look at all the neighbors of the source and write 1 on them. Then look at all the neighbors of nodes with 1 on them and write 2 on them. And so on.

Algorithm A counts by one from node to node, and Algorithm D counts by the edge-weights from node to node.[5]

Moore's example for algorithm D includes replacing an earlier higher value with a new lower value when a node is hit for a second time. He doesn't appear to show leaving the earlier value there if it is already lower. Nonetheless, I can intuit a way to implement Moore's algorithm involving queuing (and sometimes requeuing)

---

[2]Of course, "Scan the network for..." does not preclude the order used in the contemporary statement of Bellman-Ford.

[3]Some authors cite Ford and Fulkerson as the source of the Bellman-Ford algorithm. They give the following description of the "algorithm" [LFF62, page 131]:

> (1) Start by assigning all nodes labels of the form $[-, \pi(x)]$, where $\pi(s) = 0, \pi(x) = \infty$ for $x \neq s$.
> (2) Search for an arc (x, y) such that $\pi(x) + a(x, y) < \pi(y)$. (Here $\infty + a = \infty$.) If such an arc is found, change the label on node y to $[x, \pi(x) + a(x, y)]$, and repeat. (That is, the new $\pi(y)$ is $\pi(x + a(x, y))$.) If no such arc is found, terminate.

Compare the description of in Ford and Fulkerson [LFF62, page 131] with the description quoted above from [For56, page 9]; the former is a more formal looking version of the latter and includes saving the actual route as the modern statement of Bellman-Ford does.

[4]According to the cover of the reprint in a Harvard library, Moore presented his paper at a 1957 conference on theory of switching, but the conference proceedings was not published until 1959.

[5]Moore's paper also traces the way back to the source from the destination, e.g., to set up the shortest path circuit in a telephone switching application.

the nodes to be processed in sensible order (i.e., working the way from neighbor to neighbor from the source).[6]

Bellman's 1958 paper. Bellman gives a set of dynamic programming equations, where f_i is the time to travel from source node i to destination node N, and T is a matrix where the elements t_{ij} are the time to travel from node i to node j:

$$f_i = \min_{i \neq j}[t_{ij} + f_j], \;\; i = 1, 2, \ldots, N-1 \tag{5}$$

$$f_N = 0 \tag{6}$$

Then he suggests we can solve these, for suitable initial conditions $f_i^{(0)}$ (the superscript (0) means the 0'th, i.e., initial, approximation of f_i).

For instance, use equations 5 and 6 as an algorithm to be repeated k times (equations 8 and 9 below), for which Bellman suggested the initial conditions indicated by equation 7.

$$f_i^{(0)} = t_{iN}, \;\; \text{for } i = 1, 2, \ldots, N \tag{7}$$

$$f_i^{(k)} = \min_{i \neq j}[t_{ij} + f_j^{(k)}], \;\; \text{for } i = 1 \text{ to } N-1 \tag{8}$$

$$t_N^{(k)} = 0 \tag{9}$$

Execute equations 8 and 9 for $k = 0, 1, 2, \ldots,$ until the solution is converged upon ($f^{(k)}$ means the k'th approximation).

Bellman's method is like Ford's method except Bellman runs over the nodes in the order in which they are numbered and does this indefinitely many times.[7]

Thus, from equations 7–9 we can see an algorithm that can be transliterated into computer code. This is essentially the basic algorithm that many people today describe as Bellman-Ford. Of course, the contemporary statement of the algorithm typically has i looping from 1 to $N-1$, sets $f_s^{(k)} = 0$ for destination s (or source s), and recognizes that k only has to run from 1 to $N-1$. The contemporary statement of the algorithm also includes another loop to discover negative cycles.

The transliterated algorithm (without the extra negative cycles loop) would look something like what is shown in Figure 1, more or less copied from [CLRS01]. The algorithm of Figure 1 also records the actual path to follow. To run the Bellman-Ford algorithm for all sources, the algorithm of Figure 1 is repeated for $s = 1$ to N.

[6]My mental image of an algorithm for Moore's method is shown in the appendix (Figure 4). In thinking about this, I have only considered the case of non-negative edge weights. This intuitively obvious (to me) way of implementing Moore's method with a queue seems (to me) to be sort of a cross between Dijkstra's algorithm and Bellman's algorithm; it processes the nodes in a more sensible order than Bellman's but not quite as perfect an order as Dijkstra's.

[7]This arbitrary order clearly would involve a lot of wasted steps (e.g., comparing infinity with infinity plus an edge-weight). Although the nodes could be renumbered so the processing order was somewhat more sensible, in some sense that would involve solving the problem in order to improve solving the problem.

```
%The data structures for this algorithm are a single vector d, a
%    single vector pi, each with an element for each v in the
%    set of vertices V, and a matrix w(i,j) where each element
%    is the weight (distance) from vertex i to vertex j, i.e.,
%    there are finite entries for edges in the set of edges E.

%Initialize for the source s -- parallels equation 7 of this paper
do for each vertex v∈V
  d[v] ← ∞
  pi[v] ← nil
end do for each
d[s] ← 0

%Compute the distances from s to all other v;
%     parallels equations 7 and 8 of this paper
for k = 1 to the number of vertices minus 1
  do for each edge(i,j)∈E
    if d[j] > d[i] + w(i,j) then
      d[j] ← d[i] + w(i,j)
      pi[j] ← i
    end if
  end do for each
end for
%negative path loop (with true/false return) omitted here
```

Figure 1. Typical example of pseudo code for the contemporary statement of the Bellman-Ford algorithm

2. Bellman-Ford and arpanet Routing

My ulterior motive in researching the question of Part 1 of this note was to understand how the name of the Bellman-Ford algorithm for shortest paths became associated with the routing algorithm implemented in the arpanet in 1969 [HKO$^+$70].[8]

In 1975 Bertsekas and others at MIT began studying network routing algorithms [Woz76], including learning the details of the original arpanet routing algorithm [MW77]. In particular, Bertsekas worked with McQuillan et al. at BBN in the summer of 1978 (looking at stability properties of the algorithm that replaced the original arpanet algorithm), and he published the first convergence and validity analysis of the original arpanet algorithm in [Ber82].

In 1980, Schwartz and Stern published a widely know paper [SS80] on routing techniques in networks that described two basic algorithms. Algorithm A, they said, was "due to Dijkstra [Dij59]" and was well "adapted for centralized computation, while [algorithm] B, a form of Ford and Fulkerson's algorithm, is particularly useful in distributed routing procedures."[9] Later in the paper they cited the arpanet

[8]A sketch of the arpanet algorithm can be found in Figure 3.

[9]Although Schwartz and Stern cite Ford and Fulkerson [For56] for their algorithm B, they describe the algorithm of [Bel58].

from 1969 and said it's routing algorithm "was essentially our algorithm B . . . with information necessary for node updates passed among neighbors at 2/3 s[econd] intervals."[10]

In 1987, Bertsekas and Gallager's popular book [BG87] described the original ARPANET implementation and then defined a very similar algorithm which they named "distributed, asynchronous Bellman-Ford." They defined the algorithm as shown in Figure 2.

At each time t, a node $i \neq 1$ has available:

$D^i_j(t)$**::** The estimate of the shortest distance of each neighbor node $j \in N(i)$ which was latest communicated to node i.

$D^i(t)$**::** The estimate of the shortest distance of node i which was latest computed at node i according to the Bellman-Ford iteration.

The distance estimates for the destination node 1 are defined to be zero, so

$$D_1(t) = 0, \text{ for all } t \geq t_0$$

$$D^i_1(t) = 0, \text{ for all } t \geq t_0, \text{ and } i \text{ with } 1 \in N(0)$$

Each node i also has available the link lengths d_{ij}, for all $j \in N(i)$, which are assumed positive and constant after the initial time t_0. We assume the distance estimates do not change except at some times t_0, t_1, t_2, . . ., with $t_{m+1} > t_m$, for all m, and $t_m \to \infty$ as $m \to \infty$, when at each processor $i \neq 1$, one of three events happens:

(1) Node i updates $D_i(t)$ according to

$$D_i(t) := \min_{j \in N(t)} [d_{ij} + D^i_j t]$$

and leaves the estimate $D^i_j(t)$, $j \in N(i)$ unchanged.

(2) Node i receives from one or more neighbors $j \in N(i)$ the value of D_j which was computed at node j at some earlier time, updates the estimate D^i_j, and leaves all other estimates unchanged.

(3) Node i is idle in which case all estimates available at i are unchanged.

FIGURE 2. Bertsekas-Gallager definition of distributed asynchronous Bellman-Ford algorithm (from pages 327–328 of their book)

These days—a time of much interest in network routing and in parallel processing—many books and courses describe versions of the original ARPANET routing algorithm, often just describing it as a distributed version Bellman-Ford. This is sort of accurate—given the Bertsekas-Gallager 1987 definition which was close to the 1969 ARPANET implementation and given how general Ford's original specification [For56, LFF62] of the algorithm is. However, some authors and teachers describe the original ARPANET routing algorithm as "derived from Bellman-Ford"; this is

[10]This was the earliest attribution I found of the original ARPANET routing algorithm being an implementation of the Bellman-Ford algorithm. Please let me know if you know of earlier papers.

inaccurate. It would be more precise to say that the original ARPANET routing algorithm was "an original distributed development of a shortest path algorithm of the Ford class."

Some books (e.g., [WA98] on parallel processing) call the distributed version of the algorithm Moore's algorithm.[11]

3. Assessments and Assertions

I noted at the end of the previous section that there is a logical argument why the original ARPANET distance vector[12] routing algorithm is a distributed version of Bellman-Ford:

- Bertsekas and Gallager [BG87] define an algorithm they call distributed Bellman-Ford, and their algorithm is very similar to the ARPANET algorithm.
- Ford [For56, LFF62] specified hunting for edges which minimize the distance to some destination until there are no more. No order of edge processing or number of iterations is suggested. Thus, the ARPANET algorithm is an implementation of an algorithm of the Ford class.

In this section I argue why I think that argument is a stretch and the ARPANET algorithm is really quite unique.

As mentioned in the previous section, the ARPANET DV algorithm was implemented in 1969. It was sketched in [HKO+70], presented in many public but unpublished presentations in 1969 and the early 1970s (e.g., [Wal72, Wal74]), and described in detail in [McQ74] and [MW77, pages 232–233].[13] A sketch of the implementation is shown in Figure 3.[14]

As Bertsekas has said [Ber03], "The distributed asynchronous Bellman-Ford algorithm that was originally implemented in the ARPANET in 1969 was quite an original, and the theory for it came much later." It is particularly inaccurate to say the ARPANET DV algorithm was "derived from" Bellman-Ford because:

[11]To me, Moore's algorithm (Figure 4) does seem closer to the original ARPANET algorithm than Bellman-Ford's algorithm (Figure 1) does. Moore's 1957 paper also mentions the possibility of asynchronous update at a particular node, which is perhaps why some authors call the distributed algorithm Moore's algorithm.

[12]In this section, I will use Perlman's nomenclature for the original ARPANET routing algorithm and call it the ARPANET Distance Vector (DV) algorithm [Per92].

[13]Will Crowther was the primary implementer of the original ARPANET routing algorithm, and the solution as implemented is substantially his. Bob Kahn no doubt understood the routing problem before the rest of us on the ARPANET development team. Bernie Cosell played an important role in debugging the whole packet-switching software system of which the routing algorithm was a part. I helped Will, too, as he noted in [CO, page 23]: "...the original [Request for Quotation] specified fixed routing. I looked at the and said, 'That's going to be terrible.' So Dave and I worked out how to do variable routing."

[14]I sketched this example from memory and didn't look up one of the written descriptions, e.g., [Wal72]. Also, I have not checked it carefully enough to be sure there are no errors; and, since it is only a sketch, I have not worried about many details. I know of no copy of the 1969 assembly language listing of the algorithm implementation; I do know of a 1973 copy of the listing as the algorithm was implemented then. The actual implementation included code to deal with declaring links dead or alive, waiting long enough before beginning to transmit routing information after a node starts to make sure the rest of the nodes had declared it inaccessible, gathering good route information before starting to route, and so forth.

```
%Built-in constants
this-node  %number of this node
N = maximum number of nodes in the network  %63 in 1969 ARPANET

%Data structures
vector dv[i]  %distance vector has an element for each possible node i
vectors dvn[k,i] %there is a distance vector from each neighbor k with
                 %an element for each possible node i
vector w[k]     %there is an element for the distance this node adds
                %via link k; in ARPANET two distance measures were
                %maintained: a) adding one for each node (used to
                %discover connectivity); b) adding the queue length
                %for the link (used for routing)
vector route[i] %link for shortest path to destination i

%Initialization at node start-up
for i = 1 to N, dv[i] ← ∞

%Asynchronous arrival of a distance vector from a neighboring node
%over link k; this must not be interrupted by the following
%periodic routine and vice versa
for i = 1 to N
  dvn[k,i] ← element i of distance vector that arrived via link k
end for
%end asynchronous routine

%Periodically executed routine
%approximately every .625 seconds in 1969 ARPANET
for each active link k to a neighboring node
  for i = 1 to N
    dvn[k,i] ← dvn[k,i] + w[k]  %replace w[k] by 1 for hops only
  end for
end for each
for i = 1 to N
  dv[i] ← min{dvn[k,i]} over each active link k
  route[i] ← value of k which minimized prior calculation
end for
dv[this-node] ← 0
for each active link k, send vector dv out the link (to a neighbor)
%end periodically executed routine

%Routing routine
if destination = this-node, this is the destination; done
if d[destination] in hops < N+1, route via link route[destination]
                                 else there is no route
```

FIGURE 3. Sketch of the ARPANET distance vector algorithm, ca. 1969

- It is quite different than traditional (non-distributed) Bellman-Ford. Compare equations 7-9 with Figures 2 and 3. The algorithms have only the minimization term from Bellman-Ford in common, and this inequality is so basic that, in my view, it follows from first principles.[15]
- As I remember what happened, the ARPANET development team made up its algorithm without doing much of a literature search (because of the time pressure the team was under and how intuitive the algorithm is once you are in a parallel processing frame of mind).[16]
- To the extent that the distributed asynchronous ARPANET DV algorithm (what Bertsekas and Gallager called distributed asynchronous Bellman-Ford) has similarity to an earlier algorithm, it seems closer to Moore's algorithm than to Bellman's because it processes the nodes and edges in in order moving from neighbor to neighbor away from the destination.[17]

From my (admittedly biased) viewpoint, the original ARPANET DV algorithm was innovative in its own right:[18]

[15]For me, the significant distinctions between the four shortest path algorithms noted in Deo and Pang's bibliography (see page 1 of this paper) are the order in which each processes and reprocesses the nodes and edges—not the specific arithmetical comparison each algorithm uses.

[16]The Request for Quotation (RFQ) to develop the ARPANET packet-switch [ARP68] asked the contractor to design the ARPANET routing algorithm. The RPQ did include an example algorithm which we read as not very dynamic, based on complete knowledge of the network configuration at a central control facility, and updates from the central facility to the individual packet switches. Of course, a through search of the literature is always a good idea; but even if we had done one, I'm not sure it would have made much difference in what we originally implemented in the time available.

[17]I am not fond of the argument that, because Ford's algorithm says "search for an arc such that $\pi(x) + a(x, y) < \pi(y)$" until there are no more such arcs to be found (and says nothing about the order of processing or even whether it is serial or parallel), it is fair to call the ARPANET DV algorithm a version of Ford's algorithm. By this reasoning, Bellman's algorithm, Moore's algorithm, and perhaps even Dijkstra's algorithm are but special cases of Ford's algorithm which came first in the archival literature. I gather that it is acceptable in the linear or dynamic programming world to leave such "details" to the computer programmer. However, such thinking seems a little of bizarre to me. To me (a computer programmer), the order of processing, how comparisons are made, the way things are summed, the manner of detecting completion, etc., are the essence of what makes an algorithm.

[18]While the ARPANET DV algorithm is widely disparaged these days as a method of routing in a rapidly changing network, in fact it worked well over the first several years of ARPANET history when network loads were low. Furthermore, similar algorithms are still in widespread use today, e.g., in the RIP class of routing algorithms, EGP, etc. And many, if not most, courses or books on algorithms, network routing, or parallel processing highlight this method, often without mentioning its creation as part of the ARPANET implementation. The follow-on distributed routing algorithm used in ARPANET, developed primarily by McQuillan [MRR79] (incorporating Dijkstra's algorithm [Dij59]) is typically highlighted next in the books and courses. Called the ARPANET link state LS algorithm by Perlman [Per92], this algorithm was the prototype for OSPF. (While is was no longer for ARPANET, the ARPANET routing code (McQuillan's version) was changed in one more significant way in the late 1980s [KZ89].

In general, I think there is too much focus on "named algorithms" from the archival literature. For instance, many books and courses describe the ARPANET LS routing algorithm as an implementation of Dijkstra's algorithm. However, as McQuillan has noted, that Dijkstra's algorithm was selected for new ARPANET was not the key issue (although Dijkstra is a good shortest path algorithm). The key to having the ARPANET LS algorithm be a successful routing algorithm was to build a routing data base of topology and traffic for the whole net at every node and to build a complete routing tree at every node. It took a lot of careful work to make the distributed routing

- Each node only needs to know its own number and be able to derive the number of neighbors it has. The rest of the network topology is implicit in the actual configuration of nodes and inter-node links.
- The calculation of the shortest path to a given destination is done in the sensible efficient order, spreading neighbor by neighbor from each destination to all sources.
- The shortest paths to all destinations are calculated in parallel (the elements d_i are the distances to the different destinations—the elements d_i as used in Bellman's algorithm are spread across all the nodes).
- If the calculation is run continuously, there is no need to synchronize calculations of the nodes and the algorithm automatically adapts to changes in node and edges as long as these changes don't happen faster than the routing information can propagate.[19]

It is too bad that we didn't publish a paper in an appropriate theoretical journal in 1969 or 1970.

In a parallel universe where parallel processing was always the way computing was done, I'm sure Bellman, Dijsktra, et al., would have come up with, back in the 1950s and early 1960s, something like the ARPANET distance vector algorithm. However, if the parallel universe then moved to uniprocessing, I doubt the Bellman-Ford algorithm would have been what the parallel processing version evolved into. I'm guessing it would have been something more like a cross between the algorithm of Figure 3 and the algorithm of Figure 4.

data bases reasonably accurate and coherent including better means for measuring network delay and flooding for disseminating the information in a reliable and efficient manner. Other shortest path algorithms could have been substituted for Dijkstra's in the ARPANET LS routing system, e.g., Ford's, Bellman's, or Moore's algorithm.

[19]If the network changes configuration or load and then no change happens again until the shortest path calculation has had time to settle, it is reasonably obvious that this algorithm calculates the shortest path. For instance, suppose a given node goes down; it will no longer send its neighbors distance vectors with a zero as the distance to itself. Thus in time, as each node adds its own contribution to the distance to the down node and passes it on, all up nodes will detect that the down node is farther away than the maximum path across the network and will stop routing to it. If the down node then comes alive, it will again put zero in its own position in the distance vectors it sends, to which neighboring nodes will add their distance contribution and pass it on. In time, every node will know its distance to the now alive node. Of course, how the algorithm reacted in the face of rapid changes was the question that led to its total replacement in the ARPANET and to many improved variations that a variety of authors have written about.

That the ARPANET DV algorithm had no check for termination is cited by some as a problem with the algorithm. I don't see why, in a static network like non-distributed Bellman-Ford is assumed to operate on, the ARPANET DV algorithm couldn't have a similar termination criterion. In a dynamic routing situation, maintaining the network topology on each node provides termination data unavailable to the ARPANET DV algorithm.

Appendix—Sketch of Moore's algorithm

I haven't taken the trouble to try to code up this algorithm in a real programming language to try it out, so there may be errors in the algorithm shown in the figure.

```
%The data structures for this algorithm are a single vector d,
%    a vector ne of lists of neighbors, and a queue q used to
%    order the processing of the neighbors from the source s.

%Initialize for the source s
for each node i from i = 1 to n %n is number of nodes
  do
    d[i] ← ∞
  end do
d[s] ← 0

%Initialize queue
current-node-ptr ← 1
q[current-node-ptr] ← s     %start with source node
end-of-queue ← 2

%Compute the distances from s to all other nodes
do while current-node-ptr < end-of-queue
  node ← q[current-node-ptr]
  do for each neighbor in ne[node]
    if d[node] + w[node,neighbor] < d[neighbor] then
      d[neighbor] ← d[node] + w[node,neighbor]
      q[end-of-queue] ← neighbor
      end-of-queue ← end-of-queue + 1
    end if
  end do for each
  current-node-ptr ← current-node-ptr + 1
end do while
```

Figure 4. Sketch of an implementation of Moore's algorithm

Acknowledgments

Jennie Connolly of the BBN Technologies Library, Marian Bremer of the MIT Lincoln Laboratories, and various staff members at the MIT main campus libraries, Loeb Library at Harvard, and RAND Library helped me find documents.

University professors Dimitri Bertsekas, Narsingh Deo, Jeff Erickson, Richard Han, Charles Leiserson, Bruce MacDonald, Mischa Schwartz, and Anthony Wilkinson (I hope I haven't forgotten anyone) answered emails I sent inquiring for pointers into the literature or their takes on the history of the algorithm. I exchanged multiple emails with Internet pioneer Steve Crocker, routing expert Radia Perlman, and Professor Jose Garcia-Luna-Aceves, who answered my original query and read and commented on a version of this paper. I don't mean to imply anything about whether or not these people endorse my analysis or conclusions.

References

[ARP68] Request for quotation, July 29 1968. 55 page document distributed on behalf of the Advance Research Projects Agency.

[Bel58] Richard Bellman. On a routing problem. *Quarterly Applied Mathematics*, XVI(1):87–90, 1958.

[Ber82] Dimitri Bertsekas. Distributed dynamic programming. *IEEE Transactions on Automatic Control*, AC-27:610–616, 1982.

[Ber03] Dimitri Bertsekas, April 29 2003. e-mail.

[BG87] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. MIT Press, Cambridge, MA, 2001.

[CO] William Crowther and Judy E. O'Neill. Oral history interview with William Crowther. Charles Babbage Institute, University of Minnesota, Minneapolis, call number OH 184, interviewed by Judy E. O'Neill, in Cambridge, MA, March 12, 1990.

[Dij59] E. W. Dijkstra. A note on two problems in connections with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DP84] Narsingh Deo and Chi-yin Pang. Shortest-path algorithms: Taxonomy and annotation. In *Networks*, volume 14, pages 275–323, 1984.

[For56] L. R. Ford, Jr. Network flow theory. Technical Report P-923, RAND, Santa Monica, CA, August, 14 1956.

[HKO$^+$70] F.E. Heart, R.E. Kahn, S.M. Ornstein, W.R. Crowther, and D.C. Walden. The Interface Message Processor for the ARPA Computer Network. In *AFIPS Conference Proceedings*, volume 36, pages 551–567. AFIPS Press, June 1970.

[KZ89] A. Khanna and J. A. Zinky. The revised ARPANET routing metric. In *Proceedings of SIGCOMM '89: Symposium on Communications Architecturs and Protocols*, Austin, TX, September 1989.

[Law76] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976. chapter 3.

[LFF62] Jr. L.R. Ford and D. R. Fulkerson. *Flows In Networks*. Princeton University Press, Princeton, NJ, 1962.

[McQ74] John M. McQuillan. Adaptive routing algorithms for distributed computer networks. Technical Report 2831, BBN, May 1974.

[Moo59] E. F. Moore. The shortest path through a maze. In *Proceedings of an International Symposium on the Theory of Switching, Part II*, pages 285–292, 1959. Presented at the symposium at Harvard University in April 1957; a copy of the paper with a Bell Telephone System Monograph 3523 cover page is in the archives of Harvard's Loeb Library with call number VF NAC 2055 M.

[MRR79] John M. McQuillan, Ira Richer, and Eric C. Rosen. An overview of the new routing algorithm for ARPANET. In *Proceedings Sixth Data Compunications Symposium*, 1979.

[MW77] John M. McQuillan and David C. Walden. The ARPA Network design decisions. *Computer Networks*, 1(5):243–289, August 1977.

[Per92] Radia Perlman. *Interconnections—Bridges and Routers*. Addison Wesley, Reading, Massachusetts, 1992.

[SS80] Mischa Schwartz and Thomas E. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communications*, COM-28(4):539–552, April 1980.

[Tar83] Robert Endre Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983. chapter 7.

[WA98] Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, New York, 1998.

[Wal72] David Walden. The Interface Message Processor, its algorithms, and their implementation. invited lecture, Journees D'Etude Reseaux de Calculateurs, Association Francaise pour la Cybernetique Economique et Technique, Paris, May 25–26 1972.

[Wal74] David Walden. Routing (a memorandum). In *Proceedgins of 1974 International Seminar on Performance Evaluation of Data Processing Systems*, pages 429–433, Rehovot, Israel, 1974. The Weizmann Institute of Science.

[Woz76] John M. Wozencraft. First annual report for the project data network reliability (1 July 1975 – 30 June 1976. Technical Report ESL-IR-677, Electronic Systems Laboratory, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, July 1976. Wozencraft was principle investigator for the ARPA contract; he undoubtedly had co-authors for the report.

Drafted in April and May, 2003
E. Sandwich, MA
Please send comments or questions to `dave@walden-family.com`.