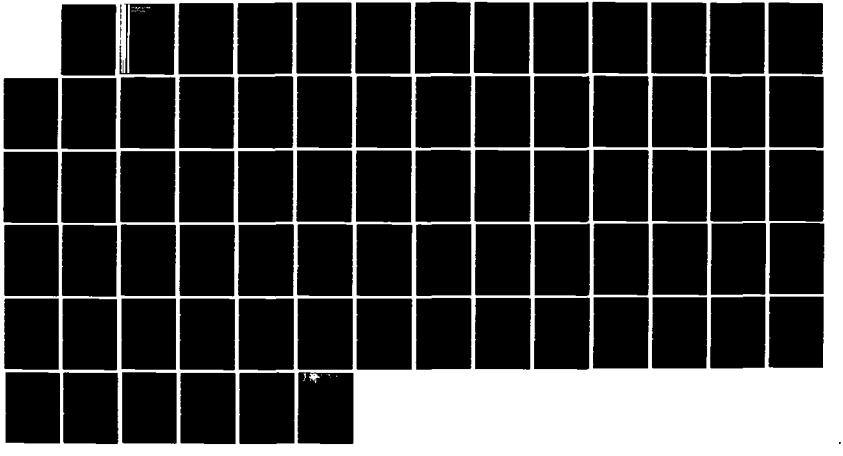
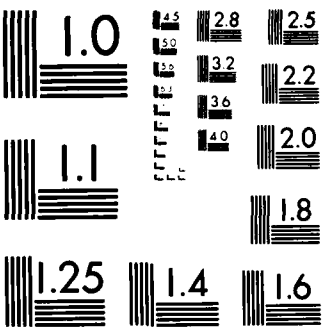
AD-A131 913 RESEARCH IN NETWORK MANAGEMENT TECHNIQUES FOR TACTICAL DATA COMMUNICATION NETWORKS(U) POLYTECHNIC INST OF NEW YORK BROOKLYN R BOORSTYN ET AL. 01 SEP 81 1/1

UNCLASSIFIED DAAK80-80-K-0579 F/G 17/2.1 NL





Polytechnic Institute of New York

(1)

SEMIANNUAL TECHNICAL REPORT

MARCH 1, 1981 TO AUGUST 31, 1981

"RESEARCH IN NETWORK MANAGEMENT TECHNIQUES FOR TACTICAL DATA COMMUNICATION NETWORKS"

FUNDED BY U.S. ARMY CORADCOM

CONTRACT NO. DAAK 80-80-K-0579

PRINCIPAL INVESTIGATORS:

Robert Boorstyn
Aaron Kershenbaum
Polytechnic Institute of New York
333 Jay Street
Brooklyn, NY 11201

SEMIANNUAL TECHNICAL REPORT

MARCH 1, 1981 TO AUGUST 31, 1981

"RESEARCH IN NETWORK MANAGEMENT TECHNIQUES FOR TACTICAL DATA COMMUNICATION NETWORKS"

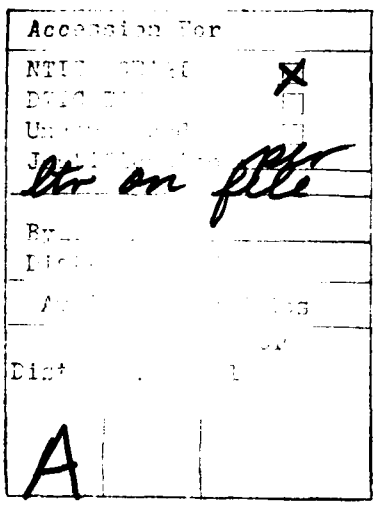
FUNDED BY U.S. ARMY CORADCOM

CONTRACT NO. DAAK 80-80-K-0579

PRINCIPAL INVESTIGATORS:

Robert Boorstyn
Aaron Kershenbaum
Polytechnic Institute of New York
333 Jay Street
Brooklyn, NY 11201

TABLE OF CONTENTS



INTRODUCTION

In this second semiannual report we describe work in three areas. Part A describes our work on packet radio networks. This section extends our work to include the effect of imperfectly heard acknowledgments. It describes and analyzes a new protocol that reduces this effect. The effect of the increase in overhead is also presented.

In Part B we describe our work on new switching techniques. Here we prove and develop an optimum packetization strategy for a particular class of users.

In Part C we discuss our work on probabilistic analysis of algorithms. In this work we analyze a particular algorithm, and discuss its probabilistic performance. Simulation results are also presented.

The Appendix contains the manuscript of a paper on an algorithm for the knapsack problem. It is being submitted for publication.

RESEARCH SUMMARIES

A. PACKET RADIO NETWORKS

Our activities in this area focused on developing and analyzing a new protocol for hop-by-hop acknowledgments in multihop packet radio networks. Our objective was to overcome the major disadvantages of the passive acknowledgment scheme previously in use. In particular, we sought to improve throughout and reduce delay relative to that scheme.

We developed and analyzed a new acknowledgment protocol which we refer to as the virtual acknowledgment protocol. In this protocol, if node j hears a packet transmitted to it by node i, node j includes the identifier of this packet in the next m packets it transmits to any node (in addition, of course, to later transmitting the packet itself). We refer to this as m virtual transmissions (and one actual transmission) of the packet. Node i then has m opportunities, rather than one, to hear that node j received its packet. Thus the probability of an acknowledgment being heard is greatly increased over what it was with passive acknowledgments. This increases throughput significantly, roughly 50% for the topologies we have studied.

In addition to this, node i can make the determination as to whether or not to transmit as soon as it hears any packet transmitted by node j. Thus, node i does not have to wait for its packet to be scheduled for retransmission and retransmitted by node j. This reduces delay significantly.

We present the analysis of throughput and overhead due to the increased length of the header and give a general procedure for computing these quantities. We also present analyses of several specific topologies of interest and compare the performance of this protocol with that of passive acknowledgments for these topologies.

A.1. A NEW PROTOCOL FOR PASSIVE ACKNOWLEDMENTS

In the previous report we discussed the effect of imperfect acknowledgment on the maximum obtainable throughput for various topologies. We saw that the maximum obtainable throughput was reduced more than 30 percent for a chain, and more than 44 percent for the star network. The protocol that we used has the following disadvantages:

1. Each node has only one chance to hear acknowledgments from its neighbors.

2. Since the probability of a node hearing an acknowledgment from its neighbors is significantly less than one, some of the successful transmissions are duplicate transmissions.

3. The next node must recognize the duplicate transmission, otherwise the actual throughput will go to zero when the number of hops between source and destination nodes is large.

4. Since each node must wait until transmission of the packet by its neighbor to hear acknowledgments for that packet, the timeout period is large. Thus delay will be large too.

The protocol that we have developed to overcome these problems is that a node recognizes a passive acknowledgment for its packet by the virtual transmission of that packet. A packet is virtually transmitted if its identifier is transmitted with another packet. Each node when transmitting a packet includes not only the packet's identifier but also the identifiers of previously received packets. A packet's identifier is contained in the next m packets. A node would then have m opportunities to hear if its transmission was received (we do not count the actual transmission). If a packet's identifier is transmitted m times then the acknowledgment for that packet is almost always heard for a

sufficiently large value of m. If a packet's transmission is not received, then after a timeout period that packet is rescheduled for transmission. A node can recognize the success of its transmissions without waiting very long if each node uses a FIFO (a first in first out) discipline for the virtual transmissions. Suppose that a node hears an acknowledgment for packet B, which is transmitted after packet A, but still has not heard a virtual transmission of A, then the node recognizes that packet A was not received, and it reschedules packet A for retransmission.

A.2. The Effects of the New Protocol on Throughput

Inserting n identifiers into a packet will increase a packet length by a factor (see Fig. A.1)

$$\frac{h + d + (n+1)\, b}{h + d + b} \qquad \text{(A-1)}$$

The value of n is related to m as will be shown later. Here n is the number of different identifiers added to the packet. For the moment, we will assume that the average value of n equals m.

The value of m is chosen such that the probability of hearing an acknowledgment, q_{ij}, is close to one. Since every node hears acknowledgments from its neighbors with probability close to one there are almost no duplicate transmissions. Thus nodes do not have to recognize duplicate transmissions for the sake of throughput.

If each node virtually transmits a packet m times then the probability of successfully hearing an acknowledgment can be recomputed as follows:

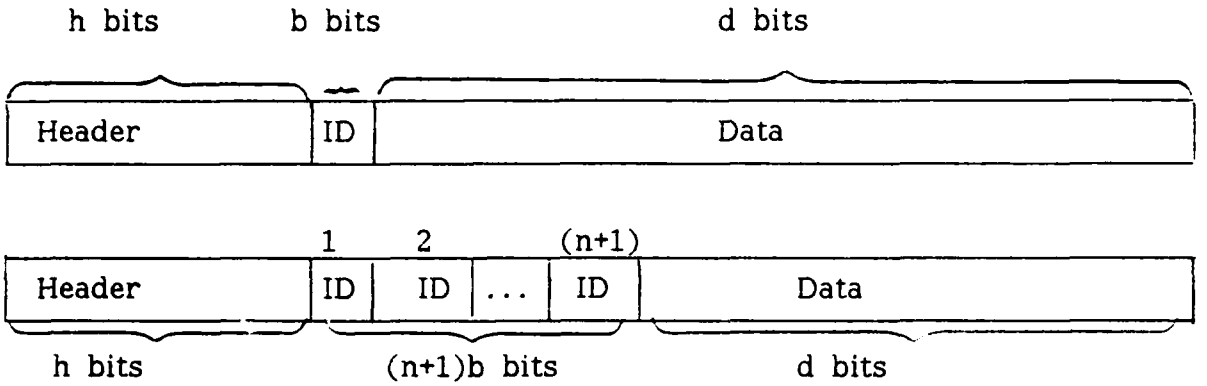


FIGURE A.1. PACKET FORMATS FOR THE OLD AND NEW PROTOCOLS

q_{ij} is the probability of hearing an acknowledgment. Then the probability of not hearing an acknowledgment after m virtual transmissions is

$$q_{ij}^{*} = 1 - (1-q_{ij})^{m} \qquad \text{(A-2)}$$

For large values of m, q_{ij}^{*} goes to 1. When $q_{ij}^{*} = 1$ the throughput (in bits/sec) will increase to the results for perfect capture.

q_{ij}^{*} is substituted for q_{ij} in all throughput equations for chain and store networks which we presented previously. The maximum obtainable throughput for different topologies are given in Tables A.1 through A.3. The probability of hearing acknowledgments is also given in Tables A.4 and A.5.

We see from Tables A.4 and A.5 that the probability of hearing acknowledgments approaches 1 when m = 5 for zero connectivity and m = 4 for full connectivity. The throughput almost increases to the values of perfect acknowledgment for m = 5.

Table A.1. Effect of imperfect acknowledgments on the throughput as a function of M for the chain network ($S_{ij} = S_{ji} = S$).

Length of Chain (N)	Throughput of a chain number of virtual transmissions, m								Perfect Acknowledgment
	1	2	3	4	5	6	7	8	
4	.106	.119	.124	.127	.127	.1275	.1276	.128	.128
5	.083	.1	.106	.108	.11	.1105	.1108	.1109	.111
6	.072	.09	.096	.1	.101	.102	.102	.102	.102
7	.066	.085	.091	.095	.096	.097	.097	.097	.097
8	.063	.081	.088	.091	.093	.094	.094	.094	.094
9	.061								
10	.06	.078	.085	.088	.09	.09	.0903	.0905	.091

Table A.2. Effect of imperfect acknowledgments on the throughput as a function of m for five legs in a star network with zero connectivity ($S_{ij} = S_{ji} = S$).

Length of Legs (K)	Throughput of a star with five legs number of virtual transmissions, m						Perfect Acknowledgment
	1	2	3	4	5	6	
1	.058	.058				.058	.058
2	.033	.039	.042	.043	.044	.0445	.045
3	.029	.037	.04	.042	.043	.0433	.044
$\geq$ 4	.029	.037	.04	.0415	.042	.043	.044

Table A.3. Effect of imperfect acknowledgment on the throughput as a function of m for six legs in a star network with full connectivity ($S_{ij} = S_{ji} = S$).

Length of Legs (K)	Throughput of a star with six legs number of virtual transmissions, m						Perfect Acknowledgment
	1	2	3	4	5	6	
1	same as perfect acknowledgment						.083
2	same as perfect acknowledgment						.042
3	.023	.029	.033	.035	.036	.037	.04
4	.023	.029	.032	.034	.035	.036	.039

Table A.4. Probability of hearing acknowledgment (q_{ij}) as a function of m for five-legs of length 5 in a star network with zero connectivity.

number of virtual transmissions, m

q_{ij}	1	2	3	4	5	6	7	8	9	10
q_{01}	.530	.700	.810	.86	.91	.94	.95	.97		.99
q_{12}	.670	.890	.960	.990	.997	.999	1			
q_{23}	.890	.980	.990	.999	1					
q_{34}	.910	.990	.994	.999	1					
q_{45}	1	1	1	1						
q_{10}	.910	.990	.999	1						
q_{21}	.930	.990	.999	1						
q_{32}	.940	.990	.999	1						
q_{43}	.970	.999	1							
q_{54}	1									

Table A.5. Probability of hearing acknowledgments (q_{ij}) as a function of m for six legs of length 5 in a star network with full connectivity.

number of virtual transmissions, m

q_{ij}	1	2	3	4	5	6	7	8	9	10
q_{01}	1									
q_{12}	.40	.58	.67	.73	.75	.81	.85	.86	.89	.91
q_{23}	.912	.990	.999	1						
q_{34}	.91	.986	.997	1						
q_{45}	1									
q_{10}	1									
q_{21}	.94	.99	.999	1						
q_{32}	.95	.99	.999	1						
q_{43}	.96	.99	.999	1						
q_{54}	1									

We compared the two protocols for acknowledgments and tabulate the results in Tables A.6 and A.7 for chain and star networks. The throughput of the center node increased by 50% for a ten node chain, by 48% for zero connectivity, and by more than 50% for full connectivity. We see from Table A.8 that for $m \geq 4$ full connectivity retains its superiority with respect to zero connectivity.

Table A.6. Comparison between old and new protocols for variable lengths of chains.

Length of Chain (N)	Throughput, Imperfect Acknowledgment		% Increment
	old protocol	new protocol (m=5)	
4	.106	.127	20
5	.083	.11	33
6	.072	.101	40
7	.066	.096	45
8	.063	.093	48
10	.06	.09	50

Table A.7. Comparison of old and new protocols for star network. L= number of legs; K= length of legs.

Connectivity	Throughput of the Center Node. Imperfect Acknowledgments		% Increment
	old protocol	new protocol (m=5)	
Zero L = 5 K $\geq$ 4	.145	.210	48
Full L = 6 K $\geq$ 4	.138	.210	52
Full L = 9 K $\geq$ 4	.139	.216	56

Table A.8. Comparison of connectivities for new protocol.

Connectivity	Throughput of the Center Node number of virtual transmission, m							
	1	2	3	4	5	6	7	8
Zero L = 5 K $\geq$ 4	.145	.185	.20	.206	.210	.215	.2155	.216
Full L = 6 k $\geq$ 4	.128	.174	.192	.204	.210	.216	.222	.224
Full L = 9 K $\geq$ 4	.139	.176	.198	.207	.216	.225	.230	.234

As we mentioned before the length of a packet is increased by a factor

$$f = \frac{h + d + (m+1) \cdot b}{h + d + b}$$

because of the virtual transmissions. We inserted m · b extra bits into a packet. Thus, the throughput is given by

$$S_{ij} = G_{ij} \cdot P(N_i, N_j) \cdot q^*_{ij} \cdot \frac{h + d + b}{h + d + (m+1) \cdot b} \qquad \text{(A-3)}$$

If we choose b = 12, h = 84, m = 5, and d = 960 bits then

$$\frac{h + d + b}{h + d + (m+1) \cdot b} = .946 \qquad \text{or} \qquad \frac{m \cdot b}{h + d + b} \cong .057$$

This means that the throughput decreased 6% when compared with perfect acknowledgments. But we know from Tables A.6 and A.7 that the throughput increased much more than 6% because of the new protocol. The results in Tables A.1, A.2 and A.3 were obtained by using

$$S_{ij} = G_{ij} \cdot P(N_i, N_j) \cdot q^*_{ij} \qquad \text{(A-4)}$$

In order to incorporate Eq. (A-3), the results in Tables A.1, A.2 and A.3 are divided by Eq. (A-1).

A.3. Optimum m for the New Protocol

In the previous section we have studied the effect of the new protocol on throughput. We found that for $m \geq 4$ the star network with full connectivity retains its superiority with respect to the other connectivities. In this section we will more closely study the L-leg star network with full connectivity.

We know that increasing m will increase q^*_{ij} and hence throughput. There is, however, a tradeoff between the throughput and overhead (due to the increase in the packet's length) due to virtual transmissions. For several cases we will find the optimum m. First we assume that at all nodes, if a packet's identifier is transmitted virtually m times, then

we also insert m identifiers of received packets into the header of a packet which is ready for transmission. The next m transmissions, after receipt of the packet in question will include the identifier of that packet.

For this case the throughput is given by

$$S_{ij} = G_{ij} \cdot P(N_i, N_j) \cdot q^*_{ij} \cdot \frac{h + d + b}{h + d + (m+1) \cdot b} \qquad \text{(A-5)}$$

where j is the one of i's neighbors.

The solution of Eq. (5) for the 9-leg-star network is given in Table A-9 when $k \geq 4$. The optimum m is 10 for this case.

Table A.9 Total throughput of a hotspot for a star network (L = 9, K = 4) when increment in header of node i, $I_i = m \cdot b$, i = 0,1,2,..., K.

m	S_T
1	.1323
2	.1682
3	.1886
4	.1961
5	.2025
6	.2091
7	.2130
8	.2145
10	.2146
11	.2142
12	.2138
15	.2110
20	.2045

In Eq. (A-1) we assume that the increment in a packet's length is m · b bits. But we know that the hotspot (node 0) and nodes at the end of the legs do not have to send passive acknowledgments because the transmissions into the hotspot and nodes at the end of the legs will

never have collisions. Thus there is no increment in the packet's length for these nodes. Equation (A-5) can be modified to obtain exact results as follows:

Again we assume that node 0 sends S units of traffic in each direction. Then the nodes that have $m \cdot b$ bits increment in their packet's length must send $I \cdot S$ units of traffic to their neighbors where

$$I = \frac{h + d + (m+1)b}{h + d + b}$$

For left to right transmissions the throughput equations are:

$$S_{01} = S = G_{01} \cdot P(N_0, N_1)$$

$$S_{i,i+1} = I \cdot S = G_{i,i+1} \cdot P(N_i, N_{i+1}) \cdot q^*_{i,i+1} \quad \text{for } i = 1,2,\ldots, K-1$$

For right to left transmissions: (A-6)

$$S_{k,k-1} = S = G_{k,k-1} \cdot P(N_k, N_{k-1})$$

$$S_{i+1,i} = I \cdot S = G_{i+1,i} \cdot P(N_i, N_{i+1}) \cdot q^*_{i+1,i} \quad \text{for } i = 0,1,\ldots, k-2$$

We expect that the results from Eq. (A-6) are greater than the results from Eq. (A-5) because Eq. (A-5) is solved assuming that the packet's length is incremented by $m \cdot b$ bits at all nodes.

The results for Eq. (A-6) are given in Table A-10. We see from Table A-10 that the optimum m is 11 and the total throughput of the hotspot, S_T, is .2222. When $m > 11$ the throughput decreases because of overhead due to virtual transmissions.

We know from Table A-5 that $q^*_{ij} \cong 1$ when $m = 4$ for all i and j except q^*_{12}. This means that nodes 1,3,4,..., k-1 need transmit a packet's identifier only four times. Let us assume that node 2 transmits m' times and all the other nodes transmit m times. Then the packet's length is incremented by

Table A.10. Total throughput of the hot spot in a star network (L = 9, K = 4) when increment in header $I_0 = I_k = 0$; $I_i = m \cdot b$, $i = 1,2,3,\ldots, k-1$

m	S_T
1	.1324
2	.1699
3	.1893
5	.2082
6	.2134
7	.2165
8	.2191
10	.2215
11	.2222
12	.2222
15	.2215

$$\left.\begin{aligned} l_i &= m \cdot b \text{ bits} \qquad \text{for } i = 1,3,4,\ldots, k-1 \\ l_2 &= m \cdot b \text{ bits} \\ l_0 &= l_k = l_{2k} = \ldots = l_{LK} = 0 \end{aligned}\right\} \qquad \text{(A-7)}$$

With these assumptions Eqs. (A-6) become

for left to right transmissions

$$\left.\begin{aligned} S &= S_{01} = G_{01} \cdot P(N_0, N_1) \\ I \cdot S &= G_{i,i+1} \cdot P(N_i, N_{i+1}) \cdot q^*_{i,i+1}, \quad i = 1,3,4,\ldots, k-1 \\ I' \cdot S &= G_{23} \cdot P(N_2, N_3) \cdot q^*_{23} \end{aligned}\right.$$

for right to left transmissions we have:

$$\left.\begin{aligned} I \cdot S &= G_{i+1,i} \cdot P(N_i, N_{i+1}) \cdot q^*_{i+1,i}, \quad i = 0,1,2,3,\ldots, k-1 \\ I' \cdot S &= G_{21} \cdot P(N_1, N_2) \cdot q^*_{21} \\ S &= S_{k,k-1} = G_{k,k-1} \cdot (N_k, N_{k-1}) \end{aligned}\right\} \qquad \text{(A-8)}$$

where

$$I = \frac{h + d + (m+1)b}{h + d + b} \qquad \text{(A-9)}$$

and

$$I' = \frac{h + d + (m'+1)\cdot b}{h + d + b} \qquad \text{(A-10)}$$

For h = 84, b = 12, d = 960, and m = 4 the results for Eq. (A-8) are tabulated in Table A.11. The optimum m´ is 19, and the total throughput of the hotspot, S_T, is .2390. With m = 4 and m´ = 19 all $q^*_{ij} \cong 1$, therefore, the nodes will not send any significant number of duplicate transmissions to the next hop. Since for node 2, m´ = 19, the increment in a packet's length there is 22% (228 bits). For nodes 1 and 3, the increment in the packet's length is 4.5% (48 bits).

Table A.11. Total throughput of the hotspot in a star network (m = 4, L = 9, K = 4) when increment in header length $l_0 = l_k = 0$; $l_2 = m' \cdot b$; $l_i = m \cdot b$, i = 1,3,4,..., k-1

m´	S_T
10	.2306
11	.2327
12	.2340
15	.2370
18	.2383
19	.2390
20	.2390
21	.2390
30	.2383

We can further increase the throughput by reducing overhead as follows:

A node, i, receives $S_{R(i)}$, units of the successful transmission (throughput) from its neighbors where

$$S_{R(i)} = \sum_{j \varepsilon N_i^*} S_{ji} \qquad \text{(A-11)}$$

For each unit of successful transmission node i sends m_i passive acknowledgments so that the total number of acknowledgments at node i is

$m_i \cdot S_{R(i)}$ per unit of time. But node i transmits with a rate of $G_i \cdot P(N_i)$ and we know that $G_i \cdot P(N_i) \geq S_{R(i)}$. Therefore each transmission must carry r_i passive acknowledgments where

$$r_i = \frac{m_i \cdot S_{R(i)}}{G_i \cdot P(N_i)} \qquad \text{(A-12)}$$

The overhead in a packet's length at node i is given by

$l_i = r_i \cdot b$, for i = 0,1,2,..., k.

If we assume that the hotspot (node 0) and nodes at the end of each path (leg) want to send S packets/unit time then the throughput is given by

$$S_{ij} = \frac{h + d + r_i \cdot b}{h + d + b} \cdot S = G_{ij} \cdot P(N_i, N_j) \cdot q^*_{ij} \qquad \text{(A-13)}$$

for i = 0,1,..., k, and where j is a neighbor of node i.

For instance for the transmissions from left to right we have:

when i = 0

$S_{01} = S = G_{01} \cdot P(N_0, N_1) \cdot q^*_{01}$

i = 1

$$S_{12} = \frac{h + d + m \cdot (\frac{S_{01} + S_{21}}{G_1 \cdot P(N_1)}) \cdot b}{h + d + b} \cdot S = G_{12} \cdot P(N_1, N_2) \cdot q^*_{12}$$

i = 2

$$S_{23} = \frac{h + d + m' \cdot (\frac{S_{12} + S_{32}}{G_2 \cdot P(N_2)}) \cdot b}{h + d + b} \cdot S = G_{23} \cdot P(N_2, N_3) \cdot q_{23}$$

i = k-1

$$S_{k-1,k} = \frac{h + d + m \cdot \left(\frac{S_{k-2,k-1} + S_{k,k-1}}{G_{k-1} \cdot P(N_{k-1})}\right) \cdot b}{h + d + b} \cdot S$$

$$= G_{k-1} \cdot P(N_{k-1}, N_k) \cdot q^{*}_{k-1,k}$$

In order to solve Eq. (A-13) we have to know the right hand side of Eq. (A-13) for a given S. We use the flowchart given in Fig. 2 to solve Eq. (A-13). It is important that the initial S must be chosen such that Eq. (A-8) has a solution.

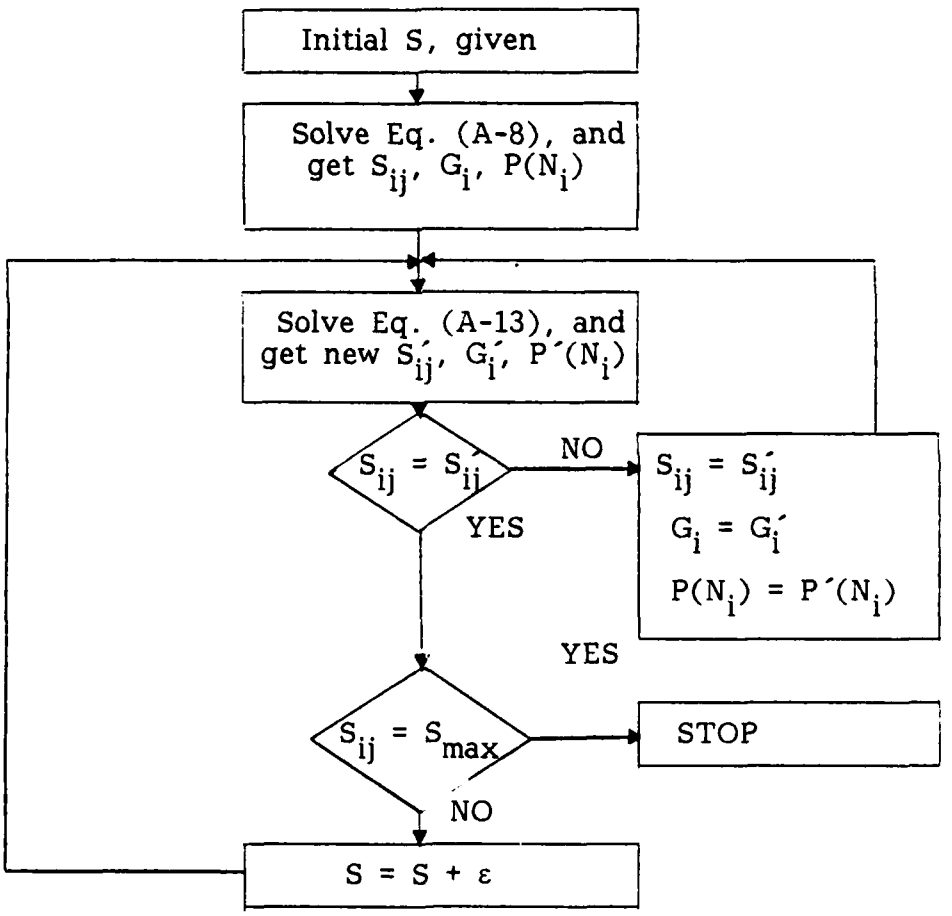


FIGURE 2. FLOWCHART TO SOLVE EQ. (A-13)

The results for Eq. (A-13) are given in Table A.12. We see that the total throughput of the hotspot, S_T, is .2454 and the optimum m´ is 27. Even though node 2 transmits a packet's identifier 27 times the increment in the packet's length is only 78 bits which is a 7% increment. This means that at node 2 each transmission carries 6.5 identifiers ($r_2 = 6.5$). We also found the $r_1 = 3.79$ and $r_3 = 3.82$ which are 4% increments. In the previous report we saw that there was a 46% reduction in throughput due to imperfect acknowledgments when we used the old protocol (no virtual transmissions). Here we see that with the new protocol there is only a 2.6% reduction in throughput with respect to the result for perfect acknowledgments (.252). Up to this point we assumed that the original length of a packet is 1056 bits (header = 96, data = 960).

Table A.12 Total throughput of the hotspot in a star network ($m = 4$, $L = 9$, $k = 4$) when $l_o = l_k = 0$; $l_i = r_i \cdot b$, $i = 1,2,3,\ldots, k-1$

m´	S_T
10	.2327
11	.2348
12	.2364
15	.2405
20	.2433
25	.2447
26	.2447
27	.2454
28	.2454
29	.2454

The effect of the overhead depends also on the length of the data portion of a packet. If the data portion of a packet, d, is small the overhead will be large and the throughput will be reduced. We have solved Eq. (A-13) for different values of d. The results are given in Table A.13. In Table A.13 we also tabulated the reduction from the result for perfect acknowledgments. We see from Table A.13 that when d increases the throughput also increases and the overhead gets smaller. The overhead and r_i are given in Table A.14. When d = 50 bits the overhead at node two is 36% but when d = 2000 the overhead is only 4%. The reduction in throughput from the result of perfect acknowledgment is 18% when d = 50 but it is only .9% when d = 2000.

Table A.13. Total throughput of the hotspot in a star network (m = 4, L = 9, K = 4) for different values of d.

d	m´	S_T	% Reduction from perfect acknowledgments
50	16	.2074	18
100	20	.2173	14
200	15	.2243	11
400	19	.2355	7
600	22	.2405	5
800	27	.2440	3
960	27	.2454	2.6
1200	30	.2468	2
1600	35	.2489	1
2000	35	.2496	.9

Table A.14. The percentage of overhead as a function of d.

d	r_1	over-head %	r_2	over-head %	r_3	over-head %
50	3.49	29	4.35	36	3.5	29
100	3.55	22	5	30	3.6	22
200	3.56	14	4.36	18	3.65	15
400	5.16	13	7	17	5.25	13
600	3.74	6	5.57	10	3.8	7
800	3.79	5	6.5	9	3.82	5
960	3.79	4	6.5	7	3.82	4
1200	3.83	3.5	7.8	7	3.84	3.5
1600	3.83	3	9	6	3.84	3
2000	3.84	2	8	4	3.85	2

where:

$$\text{overhead} = \frac{r_i \cdot b}{h + d + b} \cdot 100 \; ; \; h = 84, \; b = 12$$

B. NEW SWITCHING TECHNIQUES

Our research in this area focused on developing an optimal packetization scheme for situations where conventional packetization techniques are unsuitable. In particular we seek to maximize the number of users which could share a channel of given capacity subject to a constraint on the average per character delay, where delay includes packetization delay as well as queuing delay. We are particularly interested in cases where the packetization delay is significant and the delay constraint is significant. An example of such a situation is the case where many interactive users access a packet switched network with low speed access links and require an echo from a remote host of the characters they enter.

We have obtained on optimal scheme and, in this section give a proof of its optimality. The scheme is surprisingly simple. One obtains via an elementary search procedure the optimal value of $\bar{y}$, the average packet length; i.e., the value of $\bar{y}$ which maximizes M, the number of users in the system. In general, $\bar{y}$ is not an integer. We then packetize with probability 1 - q if n characters have arrived and with probability q if n + 1 characters have arrived, where $\bar{y} = n + q$. The procedure is easy to implement in practice and is significantly better than either packetizing on a fixed number of characters or after a fixed amount of time.

B.1 An Optimal Packetization Strategy

We consider the problem of determining the optimal strategy for packetization in a multi-user packet switched network. The objective is to maximize the number of users, M, in the system subject to a restriction on the average per character delay. This problem and the input model we use are motivated by interactive data entry applications where remote echoing of each input character is required. It is common practice in such situations to packetize on every character or on the basis of a very short timer. The result of this is that very small packets are sent. This is inefficient as the packet header than represents a significant overhead, often in excess of 1000%. This has often led such users to abandon remote echoing (or, alternatively, to abandon packet switching). We here offer a third choice.

We assume that each user produces a character with probability p every t seconds. Characters input by user i are stored in buffer, b_i, until it is determined on the basis of a packetization strategy that a

packet should be formed. At this point, a header of length h is added to the data characters and the resulting packet is placed on a queue for transmission on a channel with capacity c characters per second.

The average per character delay, $\bar{D}$ is given by

$$\bar{D} = \bar{D}_p + \bar{D}_s + \bar{D}_w \qquad \text{(B-1)}$$

where $\bar{D}_p$, $\bar{D}_s$, and $\bar{D}_w$ are the average per character packetization, service, and queueing delay, respectively. All times are measured in slots of length t.

B.2 Packetization Delay

We assume that characters arrive independently. Each character in a packet waits for all the remaining characters to arrive. If the packetization scheme results in a packet length distribution of

f_i = Prob {Packet length is i characters}

then $\bar{D}_p$, the average per character packetization delay, is given by

$$\bar{D}_p = \sum_i f_i \bar{d}_i / \bar{y}$$

where $\bar{d}_i$ is the total average packetization delay suffered by all characters in a packet of length i and $\bar{y}$ is the average packet length, $\bar{y} = \Sigma f_i i$. Since characters arrive independently at a rate of p characters per slot, the average interarrival time between characters is 1/p slots and $\bar{d}_i$ obeys the recurrence relation

$$\bar{d}_i = \bar{d}_{i-1} + \frac{i-1}{p}$$

This together with the initial condition $\bar{d}_i = 0$, yields

$$\bar{d}_i = \frac{i(i-1)}{2p}$$

Thus, for example, if all packets were n characters long

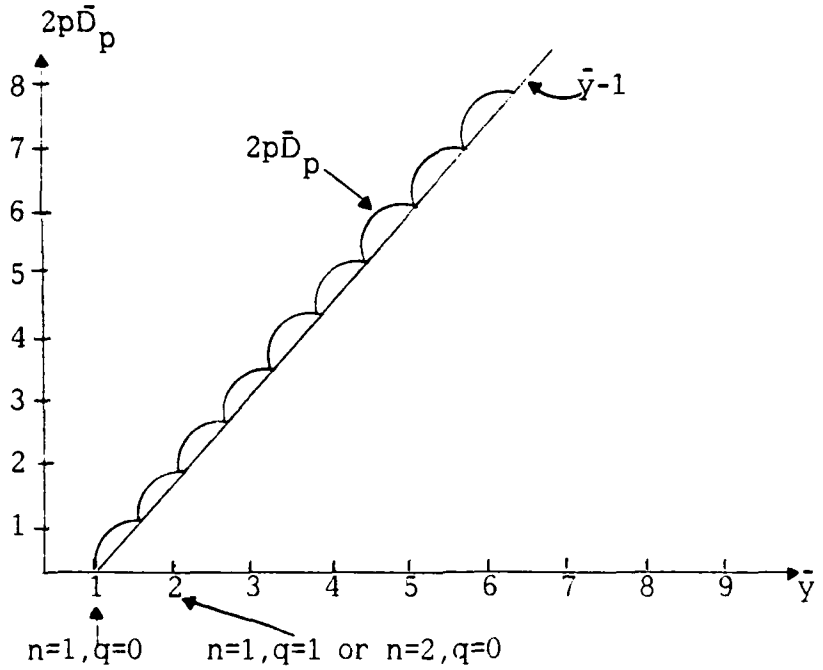
$$\bar{D}_p = \frac{n+1}{2p} \qquad \text{(B-2)}$$

and if packets were n characters long with probability 1 - q and n + 1 characters long with probability q, so that the average packet length $\bar{y}$ is n + q, then

$$\bar{D}_p = \frac{n+q-1}{2p} + \frac{q(1-q)}{2p(n+q)} = \frac{\bar{y}-1}{2p} + \frac{q(q-1)}{2p}$$

Thus we see in this case that $\bar{D}_p$ is linear with $\bar{y}$ for integer values of $\bar{y}$ and is the continuous piece wise-differentiable function given in Figure B.1. Note that the function is indeed continuous and matches the function $\frac{y-1}{2p}$ for integer values of $\bar{y}$ since q(1-q) = 0 for q = 0 or 1. Note that the maximum difference between $\bar{D}_p$ and $\frac{\bar{y}-1}{2p}$ occurs at q=½ and equals $\frac{1}{8p\bar{y}}$ and that this value decreases as $\bar{y}$ increases.

Fig B.1 Average Packetization Delay for 2-point Packet Length Distribution, $\bar{y}$ = n+q



B.4 Service Time

The average service time per character $\bar{D}_s$ given by

$$\bar{D}_s = [(\sum_i f_i \, i^2/\bar{y}) + h]/c$$

where c is the high speed line speed in characters/slot. Thus for constant length packets of length n

$$\bar{D}_s = (n + h)/C$$

and for the 2-point distribution given above with parameters n and q

$$\bar{D}_s = \frac{n + q + h}{c} + \frac{q(1-q)}{c(n + q)}$$

Thus we see that $\bar{D}_s$ behaves exactly the same way $\bar{D}_p$ does. It is continuous, piecewise-differentiable, and follows $\frac{\bar{y}+h}{c}$ closely, the error getting smaller as $\bar{y}$ increases.

B.5 Waiting Time

The delay $\bar{D}_w$ due to waiting time per character is a function of the distribution of the interarrival time between packets. We have carried out an exact analysis of the waiting time for fixed length packets and obtained an expression relating it to the packet length (see the previous semiannual report). Unfortunately, we have been unable to obtain a closed from for the relationship except for y = 1 and y = 2. The results in these cases support the approximation of $\bar{D}_w$ by the M!G!1 queuing delay formula. We will thus make the assumption that $\bar{D}_w$ can be approximated by

$$\frac{M \, \lambda \overline{X^2}}{2c^2(1-\rho)}$$

where M is the total number of active users

λ is the average arrival rate per user in packets/slot; thus $\lambda = 1/\bar{y}$

$\overline{X^2}$ is the second moment of the service time

and ρ is the average channel utilization

$\overline{X^2}$ can be written as

$$\overline{X^2} = \sum_i f_i (i + h)^2 / \bar{y}$$

For constant length packets this is simply $(n+h)^2/c^2$ and for the 2-point distribution above is

$$\overline{X^2} = \frac{(n + q + h)^2}{c^2} + \frac{q(1-q)}{c^2}$$

The channel utilization, ρ, is given by

$$\rho = \frac{M\, p\, (\bar{y} + h)}{cy}$$

We thus have an expression for the total delay per character, $\bar{D}$,

$$\bar{D} = \sum_i f_i \left[\frac{i(i-1)}{2p\bar{y}} + \frac{i^2/\bar{y} + h}{c} + \frac{M\lambda(i+h)^2}{2c^2(1-p)\bar{y}} \right] \qquad \text{(B-3)}$$

B.6 Derivation of the Optimum Packetization Strategy

We now consider the problem of maximizing M, the number of users, subject to a constraint, D*, on the average per character total delay. We thus seek the optimal value of the packet length, $\bar{y}$, and the optimal packetization scheme which will yield this optimal length.

Given M and $\bar{y}$, we see from equation (B-3) that the packetization scheme which minimizes $\bar{D}$ is one which minimizes the second moment of the packet length, $\Sigma i^2 f_i$. We are constrained, of course, to packet

length distributions where the packet length only takes integer values. It is easy to see that the distribution which minimizes the second moment subject to these constraints is the 2-point distribution which we have been discussing; i.e.,

$$f_i = \begin{cases} 1-q & i = n \\ q & i = n+1 \\ 0 & \text{otherwise} \end{cases}$$

where $\bar{y} = n+q$. To see this we use the technique of Lagrange multipliers to find the minimum of

$$\Sigma i^2 f_i + \alpha \Sigma i f_i + \Sigma f_i \text{ where}$$

α and β are the Lagrange multipliers. The last two terms are used to satisfy the constraints $\Sigma i f_i = \bar{y}$ and $\Sigma f_i = 1$. In addition all $f_i \geq 0$. Differentiation with respect to the f_i we get

$$i^2 + \alpha i + \beta \begin{cases} = 0 & \text{for all i such that } f_i > 0 \\ \geq 0 & \text{for all i such that } f_i = 0 \end{cases}$$

Thus there are at most two non-zero values of f_i that can satisfy the equality condition above. To satisfy the inequality condition these two values of i must differ by one.

Thus we see that for a given value of $\bar{y}$ and M, the total delay is minimized by any packetization scheme which gives rise to a 2-point packet length distribution.

We approach this problem in a different way by considering packetization delay and service time together. Any packetization scheme can be thought of as a decision rule which, given the state of the buffers, decides whether to packetize or not. Figure B.2 shows the state space for this decision process. The root of the binary tree shown in Figure B.2 represents the state where the buffer is empty and no time has

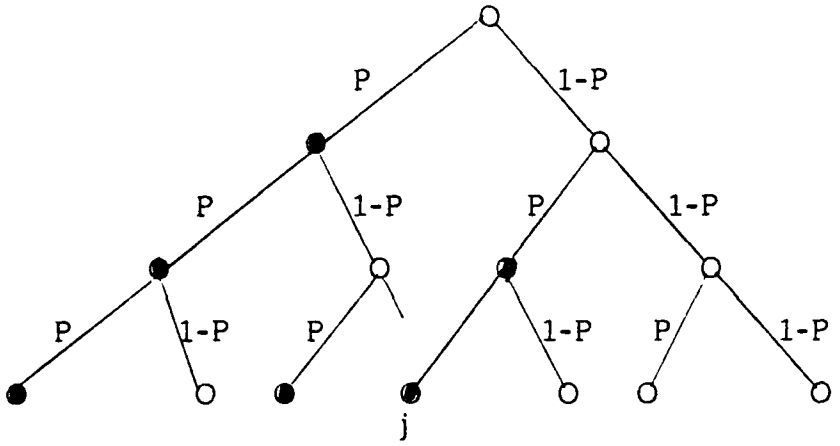


Fig. B.2 The State Space for the Packetization Decision Process

elapsed. Note that the packetization rule considers each user separately and hence the buffer is that for a single user. Each link represents a transition in a single slot time. A transition toward the left (to a shaded node) occurs with probability P and corresponds to the arrival of a character. A transition to the right corresponds to a slot in which no character arrives. Tracing a path from the root to any node, j, one can determine the probability, r_j, of reaching that state by multiplying all of the link probabilities. One can also find the number of characters, n_j, in the buffer by counting the number of shaded nodes in the path from the root to node j. Finally, one can determine d_j, the total delay due to packetization and service time suffered by all characters in the buffer, if a packet is formed at node j. This is done by adding together the delays of all the characters in the packet.

Let x_j be a decision variable taking the value 1 if we packetize at node j and zero if we do not. Any packetization scheme can be defined solely in terms of the x_j. Furthermore,

$$\bar{y} = \sum_j x_j r_j n_j$$

and

$$\bar{D}_1 = \sum_j x_j r_j d_j$$

where $\bar{D}_1$ is the average packetization and service delay for all characters. We seek to maximize $\bar{y}$ subject to the constraint $\bar{D}_1/\bar{y} \leq D^*$. The constraint satisfies the time delay. Making $\bar{y}$ larger increases the channel efficiency. There is also an additional constraint that the packetization scheme be consistent. In particular, $x_j = 1$ implies that $x_k = 0$ for all successors k of node j since once we packetize at node j we will never reach node k.

It is somewhat easier to see what the optimal policy is if we consider the decision process in a slightly different way. We define a new set of decision variables, w_j, associated with the nodes j. The w_j are related to the x_j above by the rule that if $x_j = 1$ then $w_j = 1$ for all predecessors of j and $w_j = 0$ otherwise. For all consistent values of x_j, the w_j are well defined. Setting $w = 1$ can be thought of as deciding not to packetize at node j.

We now define g_j as the increment in packet length due to the decision not to packetize at node j. Similarly we define h_j as the increment in the average total delay, $\bar{D}_1$.

The expressions for $\bar{y}$ and $\bar{D}_1$ can then be rewritten in terms of the new decision variables, w_j, as follows:

$$\bar{y} = \sum_j w_j g_j$$

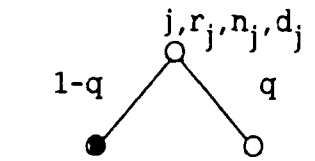
$$\bar{D}_1 = \sum_j w_j h_j$$

The incremental effect of not packetizing at node j is shown in Figure B.3. As can be seen, two nodes, k and l, will be counted in the sums for $\bar{y}$ and $\bar{D}_1$ instead of node j.

$$g_j = (1-q)r_j$$

$$h_j = (n_j + \frac{2n_j + h + 1}{c})r_j$$

In order to maximize $\bar{y}$ for a given D we would prefer to choose nodes j with maximal g_j/h_j, i.e. nodes which maximize the incremental



$r_k=(1-q)r_j$ $\quad r_l = qr_j$

$n_k=n_j+1$ $\quad n_l = n_j$

$d_k=d_j+n_j$ $\quad d_l =d_j + n_j$

$+\frac{2n_j + h + 1}{c}$

Fig. B.3 Incremental Effect of Not Packetizing at Node j

gain in $\bar{y}$ per unit $\bar{D}_1$. Note that in reality it is $\bar{D}_1/\bar{y}$ not $\bar{D}_1$ which is the constraint but since we seek k to mimize $\bar{y}$ we are simultaneously loosening the constraint as much as possible.

In order to maximize g_j/h_j we choose nodes j with the smallest possible n_j and set $w_j = 1$ for those nodes. We continue to pick nodes until we find that setting the next $w_j = 1$ would cause $\bar{D}_1/\bar{y}$ to exceed

D*. Until this point, we have assumed for the sake of simplicity that w_j would only take the values 0 or 1. In fact we are free to assign any value in the range 0 to 1 to each w_j. At the point where setting $w_j = 1$ would violate the constraint on $\bar{D}_1$, there exists a value, a, of w_i which would cause the constraint to be satisfied exactly, since $D^*\bar{y}$ is a monotore function w_j. Thus, we set $w_j = a$ for this last j and satisfy the constraint with equality.

It is clear that this procedure will indeed find an optimal set of values for the w_j in the sense that $\bar{y}$ is maximized. To see this we need only note that in order to increase the value of any w_j above the value set by this procedure, we would have to correspondingly decrease the value of some other w_k with a resulting decrease (or at best no change) in $\bar{y}$. This is because we are selecting the nodes j order of g_j/h_j, largest first.

Note that at any stage there is an infinite number of nodes j with the same value of n_j to choose from. Any selection will yield exactly the same value of $\bar{y}$. Indeed, one need only ask if w_j can be set to 1 for all nodes with a particular n_j. If so, we do this before asking the question for the next larger value of n_j. Thus we avoid any conceptual difficulty with the finiteness of the decision process.

We are left then only with the question the consistency of the packetization policy. This too, however, is no problem. Note that if we set $w_j = 1$ if $n_j < n^*$ and also for a collection of nodes with $n_j = n^*$ and with aggregate probability 1-q then we have a consistent packetization policy which adheres to the optimization criteria. This policy can be simply stated, and equivalently implemented, as waiting until the n^* $\underline{\text{th}}$ character arrives, flipping a coin which comes up heads with

probability q, packetizing at that point if the coin comes up tails, and packetizing after the next character arrives otherwise.

We have proven that given $\bar{y}$, the above packetization scheme is optimal. We now seek the value of $\bar{y}$ which maximizes M, the number of users. For $\bar{y}$ between any two integers, n and n+1, $\bar{D}$ is unimodel and any simple search procedure, e.g. binary search, suffices to minimize $\bar{D}$ or alternatively maximize M. The small "bumps" introduce by the q(1-q) terms in $\bar{D}_p$ and $\bar{D}_s$ necessitate that, strictly, we search each unit interval separately. In practice only a small number of intervals will need to be examined for most values of c and $\bar{D}$ because the q(1-q) term will usually be insignificant in comparison with the variations due to a unit change in $\bar{y}$. Thus, the overall search is computationally reasonable.

Thus, we can obtain an optimal packetization strategy for any given $\bar{D}$. Computational experience indicates that for small $\bar{D}$ and large c this optimal strategy outperforms ordinary packetization schemes by as much as 20%.

C. PROBABILISTIC ANALYSIS OF ALGORITHMS

We have obtained results in this area indicating that even relatively simple algorithms will, with high probability, obtain near optimal solutions to certain difficult optimization problems. This is encouraging in that it implies that problems, previously considered intractable, may in fact be solvable, at least with high expectation of success. The results are also useful in guiding heuristics towards potentially fruitful areas within the solution space and in avoiding unnecessary effort in areas where little further progress can be expected.

Specifically, we have analyzed the intrinsic probabilistic behavior of the 3-Satisfiability Problem and found that, independent of the size of the problem, there is only a limited region of uncertainty, i.e., only a finite set of values where even the simplest sensible algorithm can make an error. This problem, which is described in detail below, is representative of a very large class of difficult combinatorial problems and can, in fact, be used as a vehicle for solving many other problems. Thus, we believe these results can be extended to other problems as well, most notably, to problems in network design such as facility location and link topology optimization. We are currently in the process of investigating such extensions and of examining the probabilistic performance of several specific heuristic algorithms.

C.1. Probabilistic Behavior of the 3-Satisfiability Problem

The 3-Satisfiability Problem has a simple and homogeneous structure which makes it easy to develop and study a probabilistic model. It also does not involve any numerical data which would otherwise obscure the nature and generality of the results which we obtain. These results can, however, be directly extended to optimization problems which do involve numerical data, as we will see in the following discussion.

In the ordinary satisfiability problem, we are given a Boolean expression, E, over m Boolean variables $v_1, v_2, \ldots, v_m$ and we ask if there exists a set of truth values for the variables which will result in E taking the value True. Thus, each of the v_i can take either the value True or False (alternatively denoted by 1 or 0) and E will then take either the value True or False based on the values of the v_i. E is usually given in disjunctive normal form, i.e., as a set of clauses all of

which must be True in order for E to be true. Each clause contains one or more variables and is said to be true if at least one of the variables in the clause is assigned the value it takes in the clause. Thus for example, $E = (v_1 + v_2)(\bar{v}_1 + v_3)$ has 2 clauses. The first, $(v_1 + v_2)$ is true if either v_1 is True or v_2 is True. The second is True if either v_1 is False or v_3 is True. Thus E is satisfiable as the values v_1 = True, v_2 = True, and v_3 = True satisfy both clauses and hence E. There are several other assignments of truth values v_1, v_2, and v_3 which will satisfy this E. The expression $(v_1)(\bar{v}_1 + \bar{v}_2)(v_2)$ on the other hand is not satisfiable.

The 3-Satisfiability Problem is a version of the ordinary Satisfiability Problem where all clauses contain exactly 3 variables. Garey and Johnson, in their book Computers and Intractability, show this problem is NP-complete by showing that any ordinary Satisfiability Problem can be transformed into a corresponding 3-Satisfiability Problem of roughly the same size. Thus the two problems are equivalent.

Many other problems can also be transformed into corresponding 3-Satisfiability Problems. In particular, combinatorial optimization problems such as the Traveling Salesman Problem or the problem of locating earth stations in a satellite network can be so transformed. Thus, we can produce a Boolean expression which corresponds to a particular optimization problem in the sense that if the expression is satisfiable then the optimization problem has a solution with a value less than or equal to a given constant (for minimization problems) or greater than or equal to a given value (for maximization problems). Furthermore, the satisfying truth assignment can be used to obtain the solution to the corresponding optimization problem directly. By altering the

value of the constant in the above transformation the optimal solution to the corresponding optimization problem can be found via binary search; i.e., if we know the optimum lies between values c_1 and c_2, we try the value $c_1 + c_2/2$. This technique is known as thresholding.

One can determine if an expression is satisfiable by assigning all 2^m possible truth values to its variables. This approach is, of course, not practical for large values of m. There are many sophisticated techniques for answering the question whether or not a given expression is satisfiable, but all have running times which ultimately grow exponentially with the number of variables and thus are limited to problems of modest size.

We obtain here a technique which yields the a priori probability of an expression with a given structure being satisfiable given a probabilistic model of the space from which the problem is drawn. For problems where this probability, P_s, is very close to 1 or very close to zero, we need seek no further. Only for problems where P_s is significantly different from 0 or 1 need the question be investigated any further. In some cases, the P_s itself is all that we need. For example, if we have found, using a heuristic algorithm, a solution of value c_1 to a given maximization problem, and have determined that P_s is less than .01 for an expression corresponding to the existence of a solution of value greater than or equal to $c_1 + a$ (for a much smaller than c_1), then we are reasonably certain that there is not much to be gained from an attempt at further optimization. This is very important because it has often turned out that it is much harder to verify the optimality (or near-optimality) of a solution than to find an optimal solution especially when the solution space is rich and there exist many alternate near-optimal solutions.

We consider a uniform probabilistic model for the 3-Satisfiability Problem where all clauses are equally likely. Thus a problem is totally characterized by v and b, the number of variables and clauses, respectively. We will consider two models which are nearly equivalent. In one case, clauses are picked without replacement, i.e., a given clause can appear at most once in an expression. In the other case, clauses are picked with replacement.

Duplicate clauses do not affect the satisfiability of an expression. We can thus relate problems where the clauses are chosen with replacement to problems where the clauses are chosen without replacement by saying that a problem with v variables and b clauses chosen with replacement is the same as a problem with v variables and b´ clauses chosen without replacement if the number of distinct clauses in the first problem equals b´. The relationship between the number of clauses chosen with replacement, b, and the average number of distinct clauses is given by the following argument.

Let X_b be the number of distinct clauses when b clauses are selected with replacement. Then $X_b \leq b$ and $X_b = X_{b-1} + \alpha$, where $\alpha = 1$ with probability P_{new} and $\alpha = 0$ with probability $1 - P_{new}$. P_{new} is the probability that the b^{th} clause is different from all the first b - 1. P_{new} is simply the ratio of the number of unchosen clauses to the total number of possible clauses. The number of possible clauses is

$$B = 8\binom{v}{3}$$

since each clause contains 3 variables each of which can take either of two values. Thus P_{new} is given by:

$$P_{new} = \frac{B - X_{b-1}}{B} \quad . \qquad \text{(C-1)}$$

Let f(b) be the average number of distinct clauses. Then

$$E(X_b|X_{b-1}) = X_{b-1} + P_{new} \qquad \text{(C-2)}$$

and

$$f(b) = E(X_b) = f(b - 1) + \frac{B - f(b-1)}{B} \quad . \qquad \text{(C-3)}$$

Thus

$$f(b) = f(b-1) \times \frac{B-1}{B} + 1 \quad . \qquad \text{(C-4)}$$

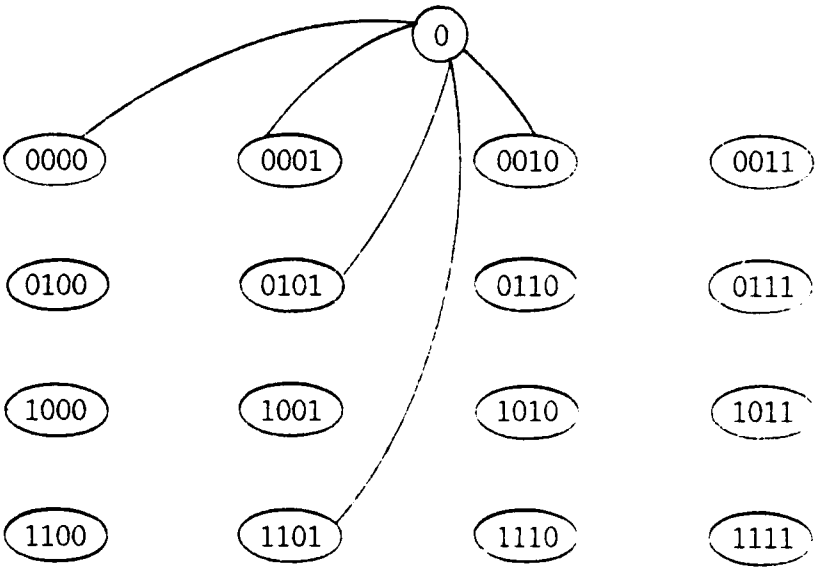
As an example, if v = 10, then B = 960 and f(50) = 48.7. As we will see, we are most interested in values of b near 5v. For v = 20 and b = 100, f = 99.5. As v increases, f/b approaches 1 very quickly. Thus, there is little difference between choosing clauses with or without replacement.

C.2. Analytical Method 1

We now turn to the problem of estimating $P_s(v, b)$, the probability of expression E, with v variables and b clauses, being satisfiable. Given a problem with v variables and b clauses, we define the truth assignment graph, G, associated with that problem to be a graph with $2^v + 1$ nodes. Nodes 1 through 2^v correspond to the possible truth values for the v variables and node 0 is a distinguished pseudonode. Each clause in E is inconsistent with one eighth of the truth assignments, specifically those in which all three variables in the clause have values opposite to their values in the clause. G contains an arc between node 0 and any node corresponding to a truth assignment which does not satisfy E. We thus proceed through E clause by clause and add arcs between node 0 and nodes which do not satisfy the clause. There are 2^{v-3} arcs corresponding to each clause. In general, arcs

"belonging" to different clauses overlap, i.e., there will be truth values inconsistent with more than one clause. We do not show the multiplicity of these arcs. Thus, G will in general contain fewer than $b \times 2^{v-3}$ arcs. We are interested in G because P_s, the probability E is satisfiable, is precisely equal to the probability that G is not connected. Figure C.1 shows a truth assignment graph.

The problem of whether a random graph is connected or not has been studied extensively by Erdos and Renii. We will proceed along similar lines.



$$E = (v_1 + v_2 + v_3)\ (v_1 + v_2 + v_4)\ (\bar{v}_2 + v_3 + \bar{v}_4)$$

FIGURE C.1. A TRUTH ASSIGNMENT GRAPH

We define A(v, k, b) to be the number of Boolean expressions whose corresponding truth assignment graphs contain k or more isolated nodes (i.e., nodes with no incident links). $P_s(v, b)$ is then given by

$$P_s(v, b) = 1 - \sum_{k=0}^{2^v} (-1)^k \frac{A(v, k, b)}{\binom{B}{b}} \qquad (C\text{-}5)$$

where $B = 8\binom{v}{3}$.

This relation follows from the fact that if an expression is satisfiable then its corresponding truth assignment graph must contain at least one isolated node. Note that the alternating sum is required in order to account for the situations where graphs with more than k isolated points are included in terms with k or more isolated points. We seek the number of expressions with graphs with one or more isolated points. The alternating sum gives us the number with exactly zero isolated points.

We can write A(v, k, b) as

$$A(v, k, b) = \sum_{i=0}^{v} \binom{v}{i} 2^i \, NM(v, i, k) \, B(v, i, k, b) \qquad \text{(C-6)}$$

where NM(v, i, k) is the number of ways of selecting k clauses which match in i specific variables (i.e., k clauses in which i specific variables take the same truth value in all k clauses) and B(v, i, k, b) is the number of Boolean expressions whose corresponding truth assignment graphs have k or more isolated points and the isolated points match in i variables.

$$NM(v, i, k) = \sum_{m=0}^{v-i} (-1)^m \binom{2^{v-i-m}}{k} \binom{v-i}{m} 2^m \cong \frac{1}{k!} (2^k - 2)^{v-i} \qquad \text{(C-7)}$$

where the approximation is made by replacing

$$\binom{2^{v-i-m}}{k} \text{ by } \frac{(2^{v-i-m})^k}{k!}$$

Now, $B(v, i, k, b) = \binom{c(v,i,k)}{b}$ where c(v, i, k) is the total number of clauses which are permissible in the sense that they give

rise to graphs that have the requisite number of isolated points and matched variables. c(v, i, k) can be estimated by

$$c(v, i, k) \cong 8\binom{v}{3} \exp\left[-\frac{2^k\, k(v-i)}{(2^k-2)8v}\right] \qquad \text{(C-8)}$$

The details of this approximation, which are somewhat lengthy, are given in W. Chuang's thesis.

Using the above approximations and Sterling's approximation

$$n! \cong \sqrt{2\pi}\, N^{N+\frac{1}{2}}\, e^{-N}$$

we find by algebraic manipulation that

$$P_s(v, b) \cong 1 - \exp[-\exp(-c)] \qquad \text{(C-9)}$$

where $b = 8(\ell n\, 2^v + c)$

and c is a constant.

Thus, only for $b \cong 5.5\ v$ does $P_s(v, b)$ take values significantly different from 0 or 1 since for any value of c less than -2, P_s is nearly 1 and for any value of c greater than +4, P_s is nearly 0. Figure C.2 summarizes the relationship between P_s and c. Note that this is not a function of v or b directly. The approximations used in obtaining this analytic form are, however, functions of v as we will see. The approximation turns out to be reasonable for modest values of v (e.g., v in the range 10 to 30) which we are interested in. Note especially the sharp drop from 1 to 0 over the interval c = -2 to c = +4.

C.3. Analytic Method 2

Because of the approximations made in the previous analysis, we sought a second method to independently verify the approach. We

consider the case now where clauses are selected with replacement. We note that all clauses are equally likely and that a given clause eliminates 1/8 of the possible truth assignments. Indeed, because of this symmetry, on the average the b^{th} clause in an expression eliminates 1/8 of the remaining truth assignments which satisfy the first b-1 clauses. We say a truth assignment is eliminated if it does not satisfy an expression. Thus, if we define $N_s(v, b)$ as the number of truth assignments satisfying an expression with v variables and b clauses, we have the recurrence relation

$$\bar{N}_s(v, b) = \bar{N}_s(v, b-1) - 2^{v-3} + \frac{2^v - \bar{N}_s(v, b-1)}{2^v} \; 2^{v-3}$$

$$= \frac{7}{8} \bar{N}_s(v, b-1) = (\frac{7}{8})^b \bar{N}_s(v, 0) \qquad \text{(C-10)}$$

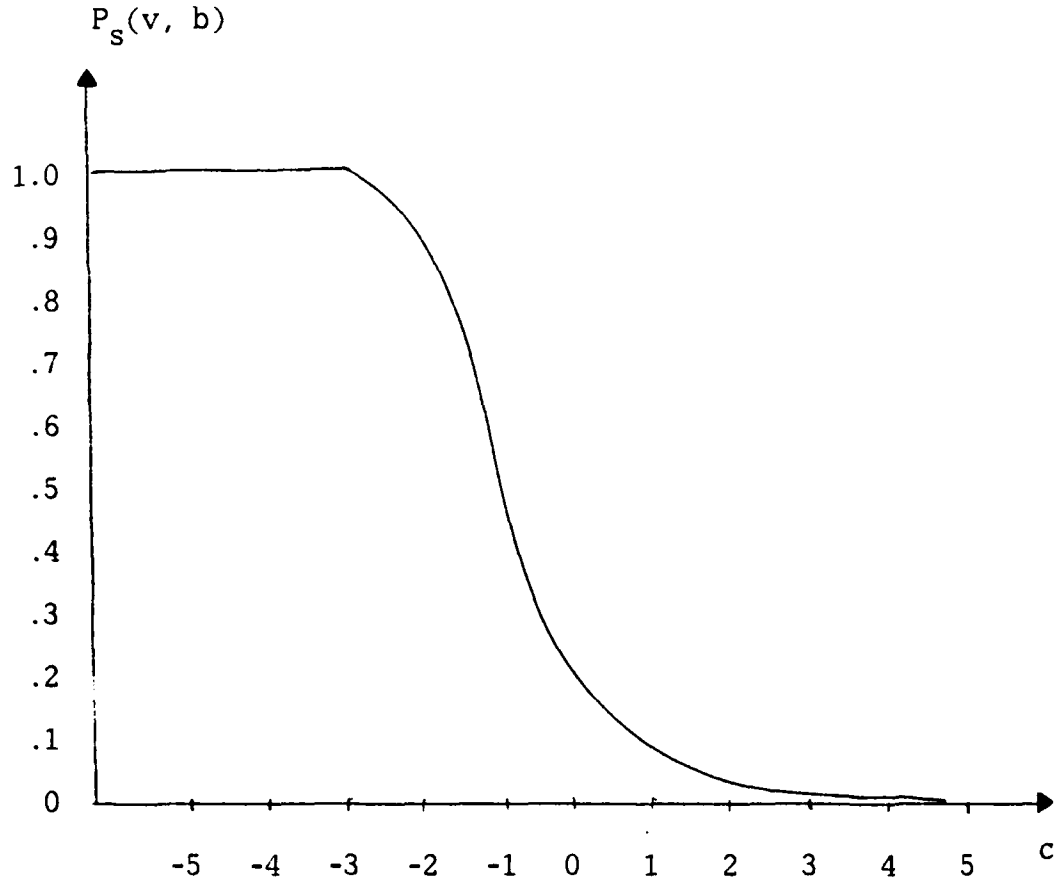
where $\bar{N}_s$ is the expected value of N_s.

Since $\bar{N}_s(v, 0) = 2^v$, i.e., all truth assignments satisfy an expression with no clauses, we have

$$\bar{N}_s(v, b) = (\frac{7}{8})^b \; 2^v \qquad \text{(C-11)}$$

$P_s(v, b)$ is by definition the probability that $N_s(v, b)$ is greater than zero. Unfortunately, we only have an expression for $\bar{N}_s$, the average value of N_s and not for the value of N_s itself. If $\bar{N}_s(v, b) >> 1$, then we would expect $P_s(v, b) \cong 1$. If $\bar{N}_s(v, b) < 1$, then we may assume that $N_s(v, b)$ is either 0 or 1. Then $P_s(v, b) = P(N_s(v, b) = 1) = \bar{N}_s(v, b)$. Thus we use the approximation

$$P_s(v, b) = \begin{cases} 1 \;, & \bar{N}_s(v, b) \geq 1 \\ \bar{N}_s(v, b), & \bar{N}_s(v, b) \leq 1 \;. \end{cases} \qquad \text{(C-12)}$$



c	P_s
-3	.999999998
-2	.9994
-1	.934
0	.632
1	.308
2	.127
3	.049
4	.018
5	.007

FIGURE C.2. $P_s(v, b)$ AS A FUNCTION OF c.

This approximation is reasonably good. We first find b such that $N_s(v, b) = 1$:

$$(\frac{7}{8})^b \, 2^v = 1$$

$$b = \frac{v}{\log_2 8 - \log_2 7} \cong 5.2 \, v \qquad \text{(C-13)}$$

We then have

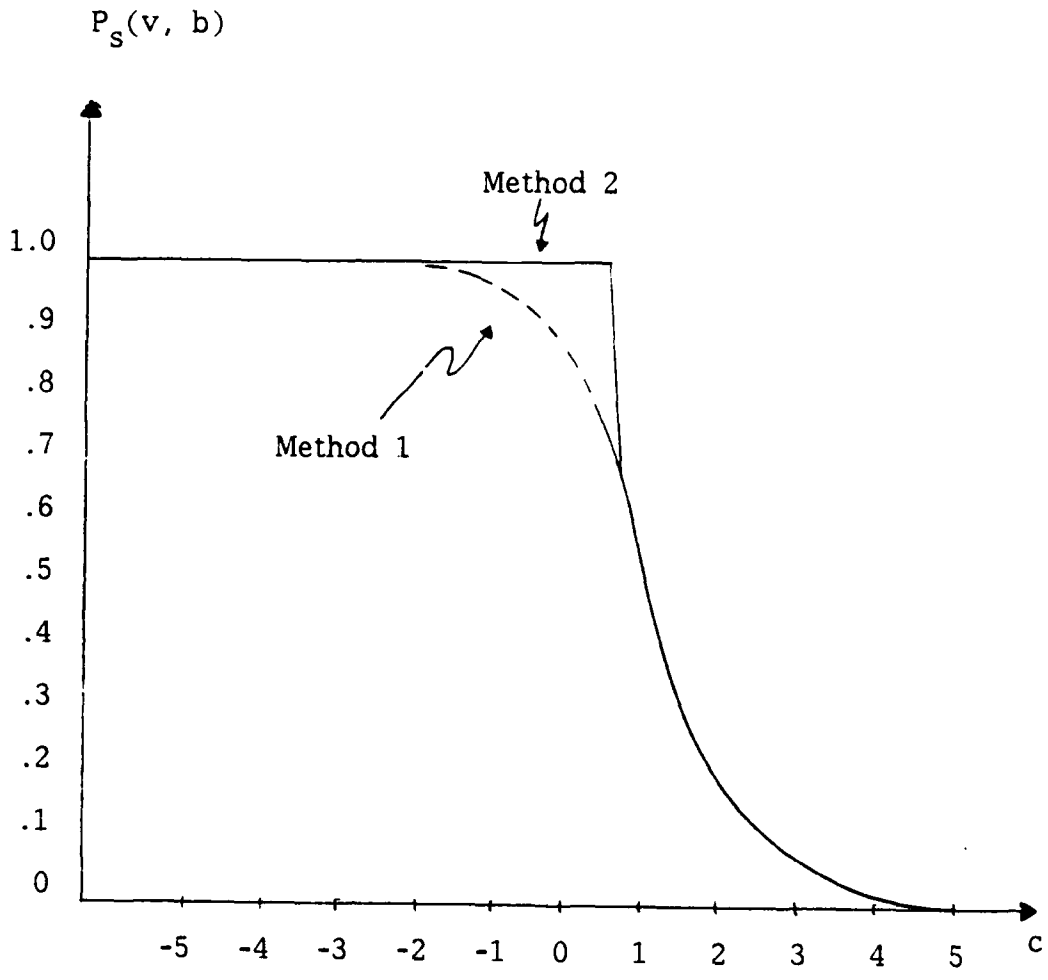
$$P_s(v, b) = \begin{cases} 1 & b \leq 5.2v \\ (\frac{7}{8})^{b-5.2v} & b \geq 5.2v \end{cases} \qquad \text{(C-14)}$$

This function is very similar to the one obtained by the first analytic method and again the region over which $P_s(v, b)$ drops effectively from 1 to 0 is limited again to a small range and is independent of v and b. Figure C.3 illustrates the relationship between P_s and c (where $c = \frac{b - 5.2v}{8}$).

The dotted line of Fig. C.3 shows P_s as found by Method 1 for the same values of c. As can be seen, for $c \geq 1$ the curves are virtually identical. There is, however, a 5% difference between the constants relating v and b in the two approximations. Nevertheless, the two analyses corroborate eath other and give rise to the same analytical behavior of P_s; i.e., v and b are linearly related at the point where P_s drops and P_s decays exponentially.

C.4. Simulation

Finally, in order to compare the accuracy of the two analytic methods, we performed a simulation to measure $P_s(v, b)$ directly. In the simulation, clauses were generated at random with replacement, all clauses being equally likely. Each time a clause was generated, the truth values it eliminated were eliminated. This procedure was con-



c	P_s
0	1.632
1	.344
2	.118
3	.041
4	.014
5	.005

FIGURE C.3. A COMPARISON OF THE FIRST AND SECOND METHODS FOR $P_s(v, b)$

tinued until no truth values remained. For each value of v, the value of b at which the last remaining truth value was eliminated was recorded and became a sample, b_i. An estimate of $P_s(v, b)$ is then formed by

$$P_s(v, b) \cong \frac{(\#\text{ of } b_i > b)}{(\text{total } \#\text{ of } b_i)} \qquad \text{(C-15)}$$

Figure C.4 gives the results of this simulation for v = 10, 20 and 30. As can be seen, the true value of $P_s(v, b)$ lies between the N values predicted by the two analytic methods and is somewhat closer to that predicted by the second method. The important property of rapid decay from 1 to 0 is borne out.

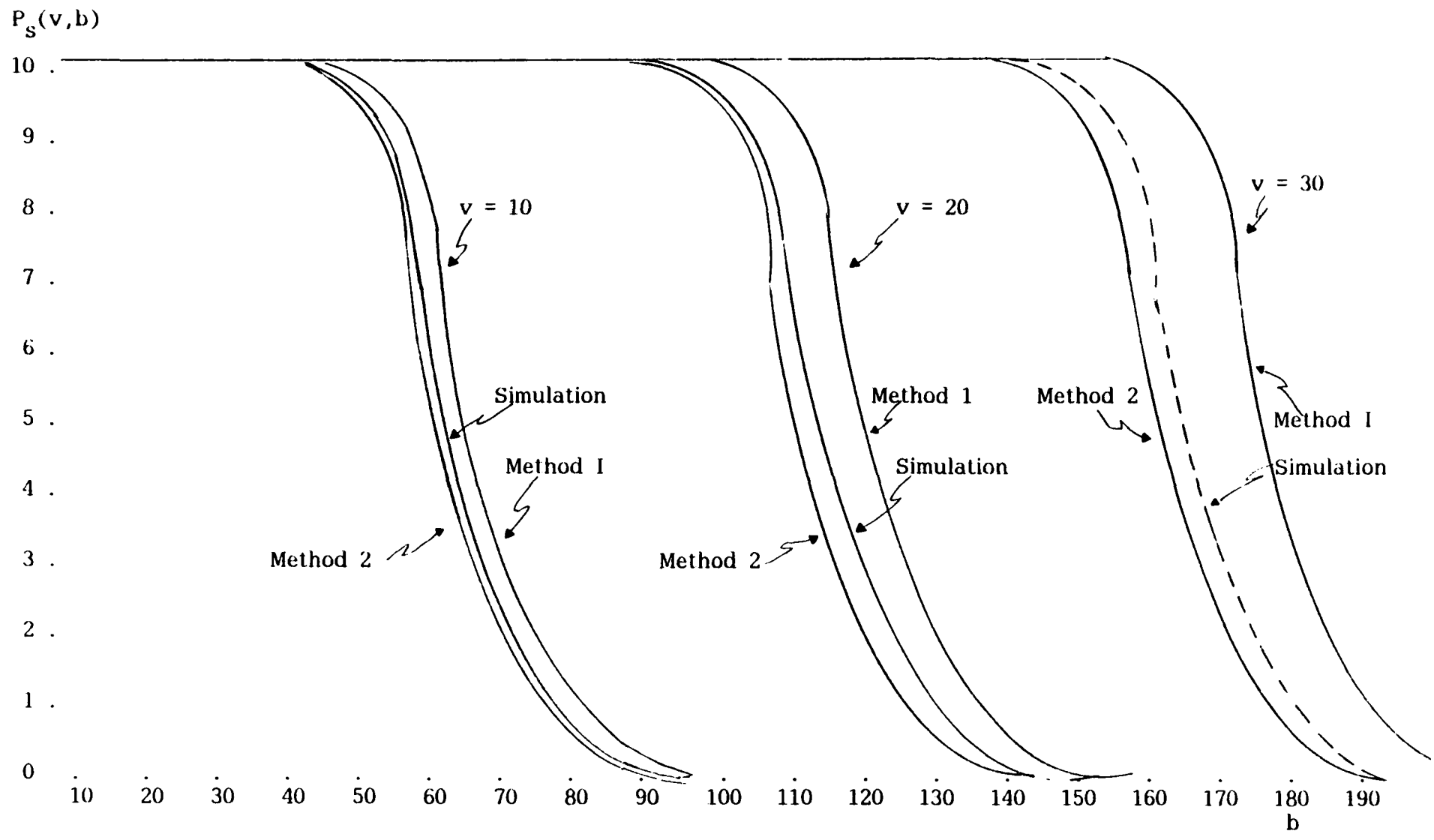


FIGURE C.4 COMPARISON OF SIMULATION AND TWO ANALYTICAL METHODS

PERSONNEL

R. Boorstyn, Professor

A. Kershenbaum, Associate Professor

Students

Paul Chu (Adaptive Routing)

Veli Sahin (Multihop Packet Radio)

David Tsao (New Switching Techniques)

William Chuang (Algorithms)

ACTIVITIES

The following four Ph.D. theses were successfully defended.

Paul Chu, A Dynamic Routing Scheme in Packet Switched Networks

Veli Sahin, Analysis of Multihop Packet Radio Networks

David Tsao, Comparison of Switching Techniques

William Chuang, Probabilistic Analysis of Algorithms.

Our paper on A Technique for Adaptive Routing in Networks appeared in the April 1981 issue of IEEE Transactions on Communications, Special Issue on Congestion Control in Computer Networks.

A paper on Generalized Augmenting Paths was presented at the 10th IFIPS Conference on System Modeling and Optimization in New York, August 1981. The paper will appear in the proceedings to be published later.

A paper on a Simulation of a Dynamic Routing Scheme will be presented at the IEEE NTC '81 Conference in New Orleans, November 1981.

A talk on Dynamic Routing was given at Carleton University, Toronto, Canada at an all-day seminar on Networks co-sponsored by Carleton University, IEEE Toronto Chapter, and Bell Northern Research.

APPENDIX

A Shortest Path Algorithm for the Solution of the Simple Knapsack Problem and Extensions.

A paper to be submitted for publication.

A SHORTEST PATH ALGORITHM FOR THE SOLUTION OF THE SIMPLE KNAPSACK PROBLEM AND EXTENSIONS

Aaron Kershenbaum*

Department of Electrical Engineering and Computer Science
Polytechnic Institute of New York
Brooklyn, NY 11201

ABSTRACT

We consider several versions of the Knapsack Problem and show that they can be solved using a shortest path algorithm requiring both storage and running time which are polynomial functions only of the number of types of object and the size (weight) of a single object. This is a significant improvement over previous algorithms for the solution of the Knapsack Problem using shortest paths, which typically require storage and runtime which are functions of the total knapsack size.

I. INTRODUCTION

The Knapsack Problem, which is interesting in its own right, has also received much attention recently[3] as a means of solving integer programming problems via a relaxation technique using group theory. Formally, the Knapsack Problem can be stated as:

$$\text{Maximize } z = \sum_{i=1}^{M} p_i x_i$$

*This work was partially supported by NSF under Grant ENG 7908120 and by U.S. Army CECOM under Contract DAAK-80-80-K-0579.

$$\text{subject to} = \sum_{i=1}^{M} w_i x_i = W$$

$$x_i \geq 0 \text{ for } i = 1,2,\ldots M$$

$$x_i \text{ integer}$$

Thus, we are given M types of object with weight w_i and profitability p_i per object of type $_i$. We are required to maximize the total profit subject to a constraint that the sum of the weights of the objects selected equals W. In the simple (unbounded) version of the problem there is no restriction on the number of objects of each type which may be included in the solution.

Sometimes an inequality constraint is used in place of the equality constraint, i.e., one seeks objects whose aggregate weight does not exceed W. Such a problem can be transformed into a problem with an inequality constraint by including an additional object with unit weight and zero profitability. Here we will concentrate primarily on problems with equality constraints.

We assume that W and the w_i are non-negative integers (or, more generally, that they are commensurate) and that the p_i are non-negative.

Previous approaches to the solution of this problem include those reported by Horowitz and Sahni[1] using dynamic programming and implicit enumeration. These approaches have been found to be effective for a wide range of problems but have exhibited excessive runtimes for problems with a large number of nearly equally valuable (in terms of cost per unit weight) objects. Also, their worst case running times are exponential in the number of object types. Denardo

and Fox[2] have developed an approach which overcomes these objections by solving the problem as a shortest path problem in a network with W nodes and W × M arcs. Thus, their approach requires storage proportional to W and runtime proportional to W × M. If W and M are sufficiently small, their approach is extremely attractive. If W is large, however, the storage, and to a lesser extent the runtime, become prohibitive.

Denardo and Fox present their algorithm in the context of using a Knapsack Problem as an integer programming relaxation. In such cases, W is generally of the same order as the w_i. We extend their approach to the case where W is very large but the individual w_i are not, and exploit the underlying cyclic group structure of the problem to develop an algorithm with both runtime and storage requirements a function only of the weight of a single object and the number of object types. Garfinkle and Nemhauser[4] point out the relationship to the group structure but do not discuss application of the structure to develop an algorithm as efficient as the one presented here.

II. BASIC PROCEDURE

For the sake of clarity, we begin by describing the basic procedure and justifying it. In later sections, extensions and refinements of this procedure are considered.

We begin by defining a network K = (N,A,L) corresponding to the Knapsack Problem (KP). The node set, N, contains W + 1 nodes numbered 0 through W corresponding to the aggregate weight of the objects selected thus far. At each node, i, there are outgoing arcs

corresponding to feasible objects which may be selected to augment a partial solution with weight i. Thus, at node i there will be arcs $(i, i + w_K)$ for all K such that $i + w_k \leq W$. The length of an arc corresponding to an object of type K is p_k. We thus have:

$N = \{0,1,\ldots W\}$

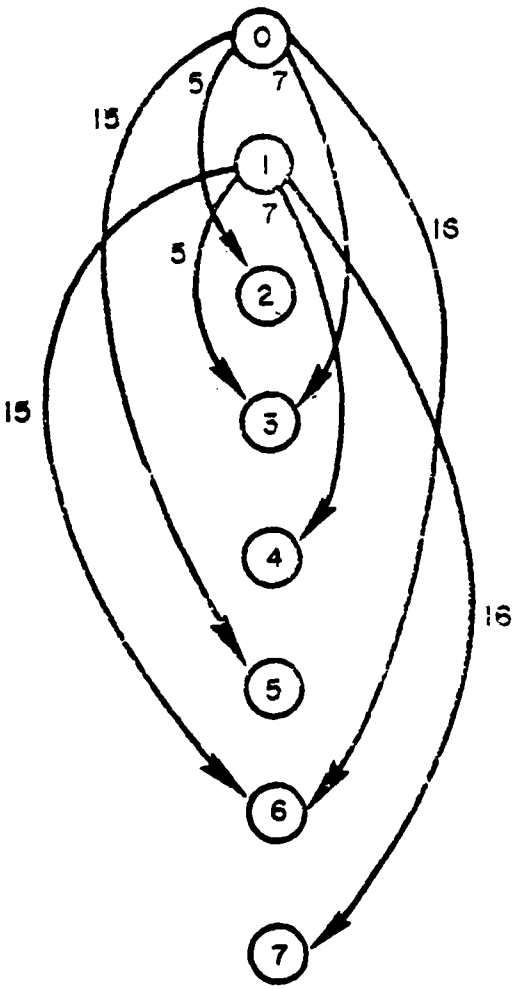
$A = \{(i, i + w_k)!\ i = 0,1,\ldots W$, for all k such that $i + w_k \leq W\}$

$L + \{l_{ij}\ !\ (i,j)\varepsilon A$, where $l_{ij} = p_k$ for arcs corresponding to type k objects$\}$

The network thus contains approximately MW arcs. We will assume that $w_{k_1} \neq w_{k_2}$ for $k_1 \neq k_2$ as if two types of objects have the same weight the one with the smaller profit can be ignored. We also assume $w_k > 0$ and $p_k \geq 0$ for all k. We will refer to arcs corresponding to a type k object simply as type k arcs. Figure 1 illustrates part of a network corresponding to a KP with 4 types of object, described in the table accompanying Figure 1. For the sake of clarity, only nodes 0 through 7 and arcs from nodes 0 and 1 are shown in Figure 1.

Paths from node 0 to node W in such networks correspond to solutions of the associated KP. The length of a path, defined as the sum of the lengths of the arcs in the path, is the total profit of the solution. Thus, the longest path corresponds to the optimal solutions. Note that, by the assumption that $w_k \geq 0$ for all k, the network is acyclic and so the problem of finding the longest path from 0 to W is solvable using Bellman's shortest path algorithm suitably modified to find longest paths. Defining d_i to be the length of the current estimate of the longest path from node 0 to node i, this amounts simply to:

that $w_{k_1} \neq w_{k_2}$ for $K_1 \neq K_2$ as



i	P_i	W_i
1	2	5
2	3	7
4	5	15
4	6	16

Figure 1- Standard Network Representation Knapsack Problem

Algorithm 1

Step 0: $d_0 \leftarrow 0$

$d_i \leftarrow \infty \quad i = 1,....W$

Step 1: For i = 1 to W

For all j such that $(i,j)\varepsilon A$

$\{d_j = \max (d_j, d_i + l_{ij})$

One can also keep track of the solution itself by setting $a_j \leftarrow i$ whenever the value of d_j is updated. At the end of the algorithm one can trace the path back from node W to node 0 using the a_j. We refer to d_j as the label on node j and to a_j as the predecessor of node j. Step 1 is known as a scan of each node, i.

Algorithm 1 is the simplest possible shortest path algorithm. The nodes are scanned in numerical order and each node is scanned only once. The algorithm has a running time on the order of MW operations and a storage requirement on the order of W. If W is very large, the approach becomes unattractive. The approach described below avoids this problem by solving the problem using a network which is, in general, much smaller. The key to reducing the size of the network is embodied in the following lemmas. Define $\hat{w} = \max_k (w_k)$ and $w^* = w_j$ such that $\frac{p^*}{w^*} = \max_k \frac{p_k}{w_k}$. Thus $\hat{w}$ is the weight of the "heaviest" type of object and w^* is the weight of the "best" type of object.

The proofs of some of the lemmas rely upon elementary properties of cyclic groups. The interested reader is referred to [6] for more details.

Lemma 1: An optimal KP solution exists containing no set of objects whose aggregate weight is a multiple of w*.

Proof: Given an optimal KP solution containing a set of objects of aggregate weight nw*, we can replace these objects by n "best" objects without altering the total weight. The net change is profit, Δp is defined by:

$$\Delta p = np^* - \sum_j p_j$$

$$\sum_j \left(\frac{w_j}{w^*} p^* - p_j\right)$$

By the definition of p*, each term of the sum is non-negative and hence $\Delta p \geq 0$. We can thus replace the set without destroying optimality and the lemma is proven.

The terms in the above sum can be interpreted as the amount of profit "wasted" by using a type j object instead of a best object. We can in fact redefine the arc lengths in the network corresponding to a given KP to be precisely these quantities. The optimal KP solution then corresponds to the shortest path from 0 to W. This can be seen to be true by observing that the above transformation of the arc lengths affects the lengths of all paths from 0 to W in precisely the same way. If the length of a 0 to W path in the original network was p the length of the same path in the new network is $p^* \frac{W}{w^*} - p$ Thus the relative length of all 0 to W paths are reversed. Alternatively, one can observe that the optimal path is the one which wastes the least profit. In the sequel, we use this form of the length function as it will allow us to improve the efficiency of the algorithm. We refer to the network using this length function as K_1.

In Figure 1, the new lengths of arcs of types 1 through 4 are 1,2,0, and 2, respectively. Note that type 3 arcs are best and that best arcs have 0 length. All arcs have non-negative length. If the best type of arc is not unique (e.g. if $w_2 = 9$ in the above example), it is possible for the length of more than one type of arc to be 0. To avoid confusion, we assume that the best type of object is unique. The extension of the algorithms to this case is straight forward.

Lemma 2: An optimal solution exists containing no more than w* - 1 non-best objects.

Proof: Any set of w* or more objects must contain a subset of weight nw* for some $n > 0$. To see this, consider a set of w* objects, $i_1, i_2, \ldots i_{w^*}$. From the sums

$$S_j = \left(\sum_{k=1}^{i} W_{i_k} \right) \text{ modulo } W^*$$

$$\text{If } S_j = 0 \text{ then } \sum_{k=1}^{i} W_{i_k} = nw^*.$$

$$\text{If } S_j = S_\ell \text{ for } \ell > j \text{ then } \sum_{k=j}^{\ell} W_{i_k} = nw^*.$$

But there are w* S_j's and only w* - 1 nonzero values of modulo w*. Hence, either $S_j = 0$ or $S_j = S_\ell$ for some ℓ and the set contains a subset of weight nw*. From the proof of Lemma 1 we see that we can replace any such subset by n best elements without decreasing the profit. Thus, we need only consider sets containing w* - 1 or fewer non-best elements together with zero or more best elements. An optimal solution can always be obtained in this way.

But since all elements have weight no greater than $\hat{w}$, the aggregate weight of the non-best elements in any such optimal solution

will not exceed $\hat{w}^*(w^* - 1)$. If $W \geq \hat{w}w^*$, we need not consider W explicitly as a constraint. We need simply apply the above algorithm to get d_j for $j \leq \hat{w}w^*$, and then complete the solution by adding the appropriate number of best objects.

Indeed, we are only interested in the weight modulo w* of a partial solution since any two solutions whose weights differ by a multiple of w* differ only by one or more best objects. We thus have:

Lemma 3: If $W \geq \hat{w}w^*$, the optimal KP solution can be found by adding zero or more best objects to the set of objects defining the shortest path in the network formed by merging all nodes with the same weight modulo w* as described above.

Proof: Consider the original network before the nodes were merged. Consider the 0 to W path corresponding to an optimal solution. By Lemma 1, we may assume this path contains no set of arcs (correspond to objects) of weight nw* except for best arcs. Since the network contains paths corresponding to picking these objects in all possible orders, we may assume that the path under consideration has all the best arcs at the end. Consider only the initial subpath corresponding to the non-best arcs.

Now, consider what the transformation does to this initial subpath. All the nodes in this path have different weights modulo w*. Thus the transformation changes this optimal 0 to W path in K_1 to a 0 to W´ path in the network with merged nodes, where W´ = W modulo w*. Note that other paths, including some optimal ones, become complex paths containing cycles because they contain subsets whose aggregate weights are a multiple of w*. The path we are focusing on

however remains a simple path and can thus be found using an ordinary shortest path algorithm.

No new paths are created by the transformation. Thus, an optimal KP solution can be obtained by appending 0 or more best arcs to the shortest 0 to W´ path in the new network, and the lemma is proven.

It is thus possible to define a new network

$K_2 = (N,A,L)$:

$$N = \{0,1,\ldots w^*-1\}$$

$$A = \{(i,j) \mid j = (i + w_k) \text{ modulo } w^* \text{ for some } K\}$$

$$L = \{\ell_{ij} \mid (i,j)\varepsilon\ A,\ \ell_{ij} = \frac{w_k}{w^*}\ p^* - p_k \text{ for } k \text{ defining } (i,j)\}$$

This network has only w* nodes instead of W. Furthermore, we note that if $(W_{k_1} = W_{k_2})$ modulo w*, then type k_1 and type k_2 arcs are parallel and the arc corresponding to the object with the smaller ℓ_{ij} can be ignored. Thus, there is at most 1 outgoing arc from each node to each other. The total number of arcs is thus at most w*(w* - 1). We have thus removed dependency on both M and W and have a procedure whose running time and storage are a polynomials only in w*.

ALGORITHM DESCRIPTION

We now describe the actual procedure in more detail and analyze its storage and runtime.

The following is the essence of the procedure:

Step 0: (Initialization)

$d_0 = 0$

$d_j = \infty \quad j = 1, \ldots w^*-1$

where d_j is the current estimate of the length of the shortest path from 0 to j.

Step 1: Find i, the best node to scan next. This is discussed in detail below.

Step 2: (Scan node i)

For $k = 2, \ldots M$

Let $j = (i + w_k)$ modulo w^*

If $d_j > d_i + \ell_{ij}$ then $d_j = d_i + \ell_{ij}$ and $PR_j = k$

where ℓ_{ij} is the length of an arc corresponding to a type k object ($\ell_{ij} = w_k(p^*/w^* - p_k)$ as defined above) and PR_j records the type of the arc used to label node j and hence keep track of the optimal solution.

Step 3: Return to Step 1 if any nodes remain to be scanned; otherwise stop.

It is clear that the storage required for this procedure is linear in $w^* + M$. If Step 1 is sufficiently simple and each node is only scanned once, the procedure's runtime will be dominated by Step 2 and will be proportional to Mw^*. This is in fact the case, as we will see.

A node, j, must be scanned if its label, d_j, is improved (reduced) as it may then improve the labels of other nodes. If we are to ensure that each node is scanned at most once, we must defer

scanning it until its label cannot be improved. When all arc lengths are positive, as they are here, this objective is achieved by scanning the nodes in ascending order of d_j. This is Dijkstra's well known shortest path algorithm.

In the general case, Dijkstra's algorithm requires that in Step 1 we find the as yet unscanned node with the smallest label. If this were done naively, it would require an examination of w* labels and the overall procedure would require $(w^*)^2 + Mw^*$ operations. In this case, however, we can implement Step 1 carefully and do much better.

We form "buckets" of nodes to be scanned and place nodes to be scanned together in the same bucket if they have similar labels. If the width of the buckets is q, then we place a node with label d, where $(k-1) \leq d < kq$ in the k^{th} bucket. We then simply work our way through the buckets in ascending order of bucket number scanning the nodes in each bucket in arbitrary order. Thus, there is no explicit search in Step 1. If the width of the bucket is no greater than the length of the shortest arc in the graph, then any node labeled by the node currently being scanned will reside in a higher numbered bucket and hence, each node will be scanned at most once.

The only problem with this procedure, which is well known, is that the number of buckets itself may become very large, in particular much larger than the number of nodes, if the longest arc is many times greater than the shortest. In this case, a great deal of storage and running time may be wasted in dealing with empty buckets. We now show that in this case fewer than w* buckets are required in all cases.

We begin by reindexing the arcs out of each node corresponding to the M - 1 remaining types of object in increasing order of length. (Recall that the arc corresponding to the best type of object was removed from explicit consideration.) Thus, L_1, is the length of the arc which corresponds to the second best type of object, etc. The w_i and p_i are reindexed correspondingly. We will refer to arcs of length L_i, weight w_i, and profit p_i after this reindexing as arcs of type i (or as arcs corresponding to objects of type i). For simplicity, we will assume $L_i \neq L_j$ for $i \neq j$ although the procedure does not require this.

Consider the case where w* is relatively prime to w_1. By the definition above, arcs of length L_1 are the shortest arcs in the graph. Since w_1 and w* are relatively prime, a path containing at most w* - 1 arcs comprised entirely of arcs of type 1 exists from 0 to each other node. Thus, we know that no node need have a label greater than $L_1(w^* - 1)$. So, w* - 1 buckets of width L_1 suffice since the shortest path will be no longer than the aforementioned one.

In the case where w* and w_1 are not relatively prime, the above path will loop back to node 0 before reaching many of the other nodes. In this case, the following preliminary computation is carried out before setting up the buckets:

```
r = w*
For j = 1, M-1
    g_j = r/GCD(r,w_j)
    r = GCD(r,w_j)
```

where GCD(i,j) is the greatest common divisor of the integers i and j and can be found using Euclid's Algorithm, whose running time is bounded by the following Lemma:

Lemma 4: Euclid's Algorithm has a worst case running time of order log N when applied to find the GCD of N and M, for N > M.

Proof: For N > M, Euclid's Algorithm replaces a problem on N and M by one one M and N - qM. The worst case for this, i.e., the one which converges most slowly, is when q = 1 at all stages. This results in a Fibonacci sequence. It is well known [1] that the K^{th} Fibonacci number, F_K is given by:

$$F_K = \frac{1}{\sqrt{5}} \left\{ \left(\frac{1 + \sqrt{5}}{2} \right)^K - \left(\frac{1 - \sqrt{5}}{2} \right)^K \right\}$$

$$\frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^K$$

Thus, the runtime of the algorithm is of order K for N of order F_K, i.e., K is of order log N.

We observe that, corresponding to the paths comprised entirely of type 1 arcs above, there is now a tree rooted at 0, spanning all the nodes, and containing paths with at most $g_j - 1$ arcs of length L_j. An illustration of such a tree is given in Figure 4. In this case only nodes 0i8, and 4 can be reached using only arcs of type 1 before the path cycles back to node 0. The number of nodes so reachable is g_1 (three in this case). The original set of w* nodes is partitioned into g_1 parts which are then each partitioned into g_2 smaller parts corresponding to the nodes which are now reachable via arcs of type 2. The partitioning continues until all nodes are reached. The above can be proven rigorously in the general case using the properties of cyclic groups.

We now define buckets of variable width corresponding to the longest path in the tree defined above. Thus, there are $g_1 - 1$

buckets of width l_1 followed by $g_2 - 1$ buckets of width l_2, etc. A total of $B = \sum_i (g_i - 1)$ buckets is required. Since the product of the g_i's is w*, $B \leq w^*$.

We have shown that less than w* buckets are required. We now show that the width of each bucket is no greater than the length of the shortest arc still permitted in a path.

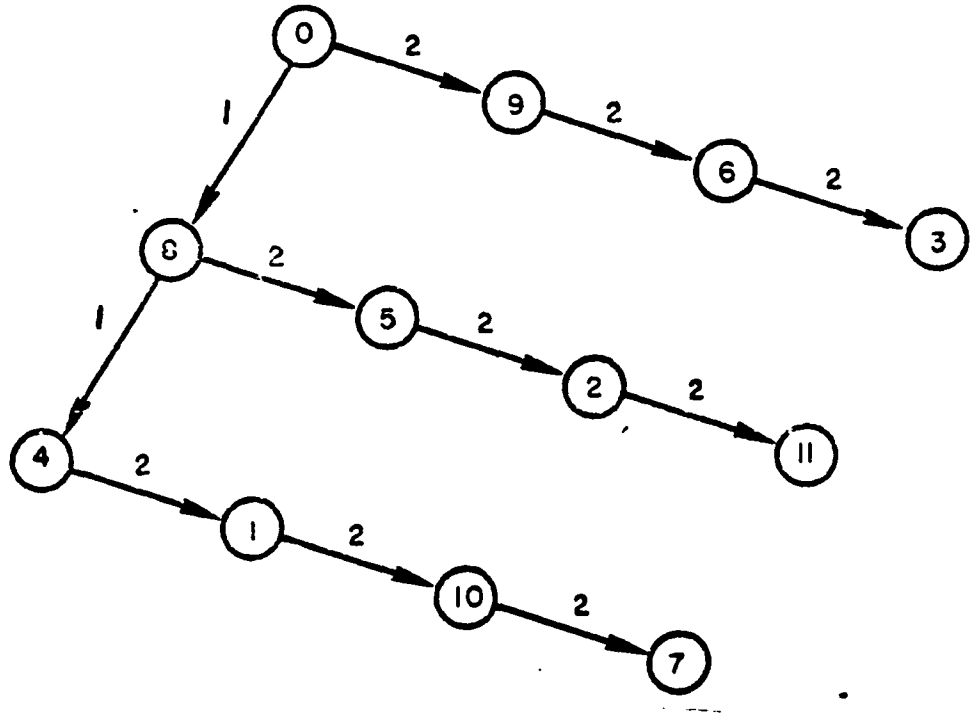


Figure 4: A spanning tree bounding the lengths of shortest paths

Note that the paths from 0 to j correspond to ordered selections of objects of aggregate weight j; i.e., there are distinct paths corresponding to the selection of the same objects in different orders. We can, and should, consider only one path corresponding to each distinct collection of objects. This is easily done by keeping track of the last type of arc in the path (which we are doing with the variable

PR_j) and only considering arcs with the same or higher index (object type). Thus, we will find solutions corresponding to a selection of objects in non-decreasing order of type.

Now, consider a node in bucket i of width w_j. This node was either reached via a path in the tree considered above or via an arc whose length is greater than that of any in that tree. In latter case, the length of the smallest arc under consideration exceeds the width of any bucket. In the former case, since we are considering arcs in increasing order of length, again a node can only enter a bucket via an arc at least as long as the width of the bucket.

Extensions

If $W < w^*\hat{w}$ then it is possible for w_i to be equal to w_j and for objects i and j not to dominate one another. In particular, if $w_i > w_j$ and $\ell_i < \ell_j$ neither i nor j may dominate. This is because the constraint on W may prohibit the use of a sufficient number of objects with the smaller length. As an example of this, consider a problem with W = 24 and (W_i,P_i) of (5,50), (6,58), (11,109), (1,0). Here, the ℓ_i are 0,2,1, and 10, and $w_2 = w_3$ modulo 5. Neither object 2 nor object 3 dominates the other, however. In fact, the optimal solution is 4 type-2 objects in this case. For W = 21 or W = 22, however, type-3 objects would be used in the optimal solution. For W = 23, both type-2 and type-3 objects would be used. Lemma 3 can thus not be extended directly to the case whose $W < w^*\hat{w}$.

Thus, it is not possible to make the final reduction to a network with w^* nodes in this case. However, when the individual w_i are small, which is the principal case of interest here, $w^*\hat{w}$ will be small as well. The above algorithm can then still be applied effectively to

the network with W nodes. Alternatively, one can view the following procedure as working on the reduced graph with w* nodes but allowing multiple labels on each node. We adopt this latter point of view in describing the modified procedure below.

A label on a node, i, is now a couple (d_i, c_i) where d_i is the amount of wasted profit to get to node i, i = 1,2,... w*-1, and c_i is the cycle number of this label. The c_i are defined by the relation

$$i = j - w^* c_i$$

where j is the node in the network K_1 (with W nodes) which would have received the same label. A node i in K_2 can thus have several labels corresponding to different c_i, but for any two labels (d_i, c_i) and (d_i', c_i') on the same node i,

$$d_i' > d_i \rightarrow c_i' < c_i$$

since as we noted above the only reason for considering a larger d_i would be to obtain a smaller c_i. Similarly, for any pair of objects with weights having the same residue modulo w* the object with the larger weight must have the smaller length.

There are several limitations on the maximum number of objects of any given type. In particular, m_i, the maximum number objects of type i, is bounded by

$$m_i \leq \min \left\{ \left\lfloor \frac{W}{w_i} \right\rfloor , \frac{w^*}{GCD(w^*, r_i)} - 1 \right\}$$

where r_i is the residue of w_i modulo W*. The first term on the right hand side follows from the fact that any larger number of type i objects have weight greater than W. The second term follows by the some reasoning is Lemma 2.

Proceeding along the lines similar to those in [3] we create $(\log_2 M_i) + 1$ objects corresponding to $1,2,4,8,\ldots$ and 2^b objects of type i, where b is largest power of 2 that is less than M_i. We thus now have defined a new problem with at most $M \log_2 W$ objects. At most one object of each type is permitted in the solution.

Proceeding along lines similar to those in [3], we can create $(\log_2 M_i) + 1$ objects corresponding to $1,2,4,8\ldots2^b$ objects of type i, where b is the largest power of 2 that is less than M_i. We then have a problem with at most $M^1 = M\log_2(w^*)$ objects where at most one object of each type need be considered. Thus, we could create M^1 buckets and proceed with the algorithm as above.

One can refine this procedure by eliminating dominated objects from consideration. In particular, if two objects i and j, have the same residue modulo w^*, and $w_i > w_j$ and $\ell_i \leq \ell_j$ then object j may be eliminated from further consideration. One must be careful, however, not to allow one copy of an object to eliminate another copy of the same object.

For example suppose there were initially 4 types of object with (W_i,P_i) equal to (10,20), (3,5), (7,4) and (1,0) $W = 10$ and $W < \hat{w}w^*$, respectively. If $W = 37$ we must consider the modified procedure. We thus create objects with (W_i,P_i) equal to (6,10), (12,20), and (6,10) corresponding to 2,4, and 2 objects of type 2. These three new objects together with the original object of type 2 can be used to select anywhere from 0 to 9 copies of the type 2 object. The first new object is identical to the third and would dominate it unless we specifically prohibited this.

If the W_i and P_i are drawn independently from uniform distributions, this latter refinement will reduce the average number of objects to no more than $w^* \ell n(M)$ objects and buckets as we see from Lemma 5.

Lemma 5: There will be on the average on the order of no more than:

$$W^* \ell n \left[\frac{M \log_2(W^*)}{W^*} \right]$$

objects left undominated after dominated objects are removed in the above procedure.

Proof: As observed above, there are approximately $M\log_2(W^*)$ or fewer objects before dominance is checked. Suppose these objects divide evenly into residue classes. There would then be $M\log_2(w^*)/w^*$ objects in each residue class.

An object can be dominated by any other object in the same class. If the objects are ordered in increasing order of ℓ_i, an object will be dominated unless its w_i is less than the w_i of all objects preceding it on the list since the weights are uniformly distributed, the K^{th} item in the list will be undominated with probability $\frac{1}{K}$. Thus, given a list of length L the expected number of undominated elements remaining on the list is

$$\sum_{j=1}^{L} \frac{1}{j} \quad \ell n(L)$$

The result follows directly by substituting for L. If the residue classes do not all contain the same number of elements, the result is still true since we observe that the quantity

$$\ell n(L_1) + \ell n(L_2) \text{ for } L_1 + L_2 = 2L$$

is maximized for $L_1 = L_2 = L$, that is, equal sized residue classes yield the maximum number of objects.

Thus, even for $W < w^*\hat{w}$, we have a number of buckets polynomial in w^*, not W. The running time is thus at worst of order$Ww^* < (w^*)^2\hat{w}$.

It is possible to extend the procedure still further to take into account restrictions on the number of permissible objects of each type, for some or all of the objects. First, if one is given an a priori restriction, U_i, on the number of objects of type i, one need only set

$$M_i = \text{Min } (M_i, U_i)$$

where M_i is the maximum number of objects of type is permitted and the M_i on the right hand side is computed as above. If the a priori maximum permissible number of best objects is less than W/w^* than a fictitious best object with $w^* = W + 1$ must be added to the problem. This maintains the validity of the procedure but may reduce its efficiency considerably if the original w^* was much smaller than W.

CONCLUSIONS

We have presented an algorithm for the solution of the simple (unbounded) Knapsack Problem whose running time and storage are functions only of the size of the "best" object weight, W^* in the case where the overall weight constraint is sufficiently large. In the cases where the overall weight constraint is smaller or where restrictions exist on the number of objects of a given type, we present an extension of this procedure which in practice is often very efficient and whose worst case running time is no greater than $(w^*)^2\hat{w}$ where $\hat{w}$ is the weightiest object.

REFERENCES

[1] Horowitz, E. and S. Sahni Fundamentals of Computer Algorithms, Computer Science Press (1978).

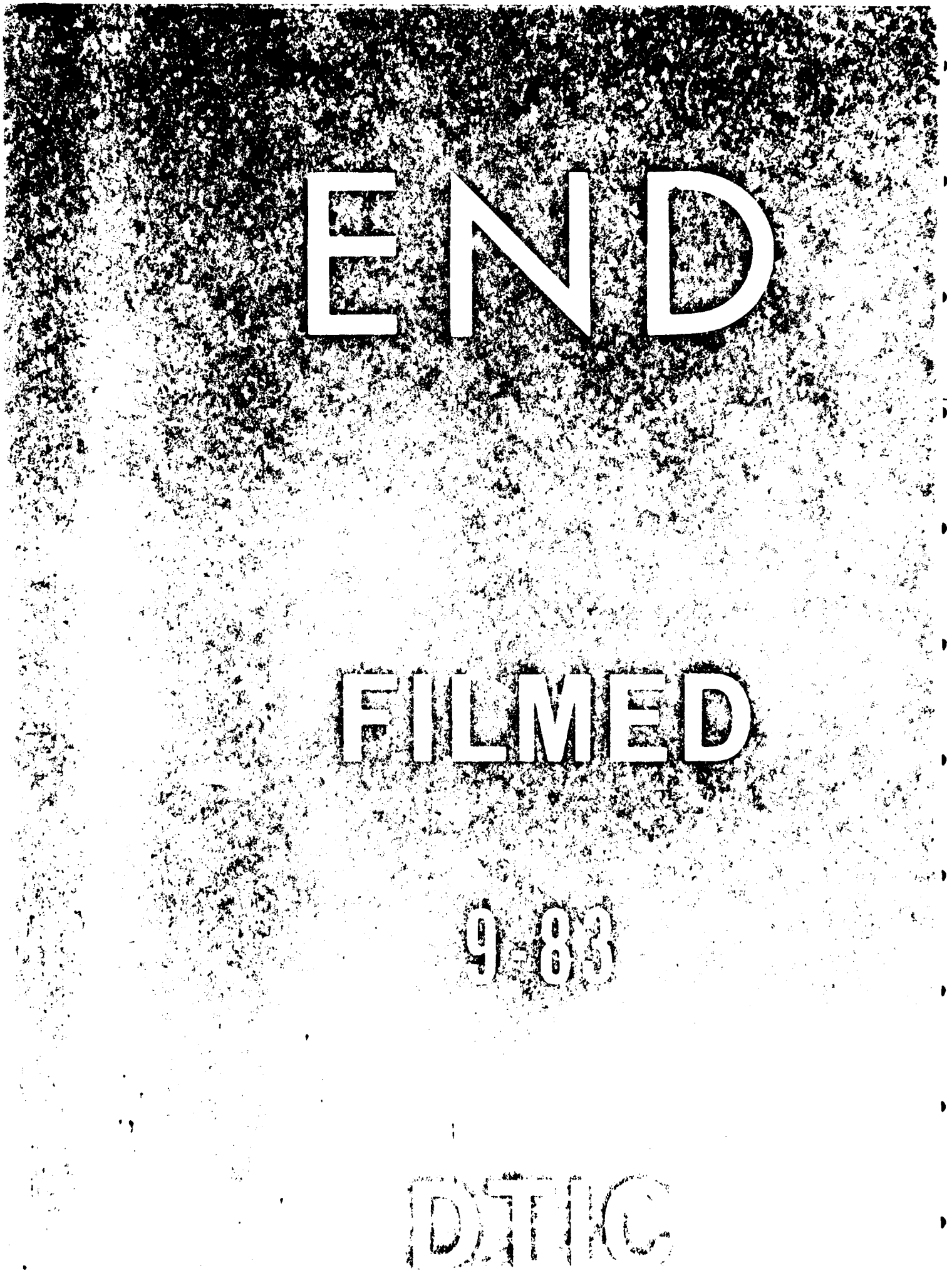
[2] Denardo, E. and B.L. Fox, "Shortest Route Methods 2: Group Knapsacks, Expanded Networks, and Branch-and-Bound," Operations Research, Vol. 27, No. 3 (May-June 1979).

[3] Johnson, E.L. "Integer Programming: Facets, Subadditivity, and Duality for Group and Semi-Group Problems," IBM Research Report, RC-7450 (12/78).

[4] Garfinkle, R.S. and G.L. Nemhauser, Integer Programming, Wiley (1972).

[5] Bellman, R.E. "On a Routing Problem," Ann. Quart. Appl. Math Vol. 16, pp. 87-90, (1958).

[6] Shapiro, J. Mathematical Programming Structures and Algorithms, Wiley, (1979).


END
FILMED
9-83
DTIC