Mathematical Programming 11 (1976) 150–157.
North-Holland Publishing Company

SHORTEST PATH ALGORITHMS FOR KNAPSACK TYPE PROBLEMS

A.M. FRIEZE
University of London, London, England

Received 13 February 1975
Revised manuscript received 16 January 1976

The group knapsack and knapsack problems are generalised to shortest path problems in a class of graphs called knapsack graphs. An efficient algorithm is described for finding shortest paths provided that arc lengths are non-negative. A more efficient algorithm is described for the acyclic case which includes the knapsack problem. In this latter case the algorithm reduces to a known algorithm.

1. Introduction

The name group knapsack problem has been given to

$$\text{minimise} \quad \sum_{j=1}^{n} c_j x_j, \tag{1.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_j g_j = g_0, \tag{1.2}$$

$$x_1, \ldots, x_n \text{ non-negative integers.}$$

The elements $g_0, \ldots, g_n$ are a subset of the elements of a finite additive abelian group H and $c_1, \ldots, c_n$ are non-negative reals.

This problem was first considered by Gomory [4] and arises in a pure-integer programming problem when the non-negativity constraints are relaxed on an optimal set of basic variables for the associated LP problem.

Algorithms for solving this problem have been described by Gomory [5], Shapiro [8, 9], Hu [6] and others.

It can be formulated as a shortest path problem in the following way:

Let G_1 be the graph with nodes H and arcs of the form $(h, h + g_j)$ h an arbitrary element of H and $j = 1, \ldots, n$. The length of such an arc is c_j. Let P be a path from 0 to g_0 in G_1 then if x_j is the number of arcs of the form $(h, h + g_j)$ in P then $(x_1, \ldots, x_n)$ is a solution to (1.2) and the length of P is (1.1). Conversely if $(x_1, \ldots, x_n)$ satisfies (1.2), then one may construct a set of paths from 0 to g_0 of the same length. Thus the problem becomes that of finding a shortest path from 0 to g_0. In this paper we give a new algorithm for solving this problem.

The name knapsack problem applies to

$$\text{maximise} \quad \sum_{j=0}^{n} c_j x_j, \tag{1.3}$$

$$\text{subject to} \quad \sum_{j=0}^{n} w_j x_j = W, \tag{1.4}$$

$$x_0, x_i, \ldots, x_n \text{ non-negative integers,}$$

where $c_0 = 0$, $c_1, \ldots, c_n$ are positive reals, $w_0 = 1$ and $w_1, \ldots, w_n$, W are positive integers.

One can formulate a knapsack problem as a longest path problem defining the graph G_2 with nodes $0, 1, \ldots, W$ and arcs of the form $(w, w + w_j)$ of length c_j. The knapsack problem is then equivalent to that of finding a longest path from 0 to W.

Gilmore and Gomory [2] describe an algorithm for solving trim loss problems which solve a sequence of knapsack problems. A more efficient algorithm for solving the knapsack sub-problems is given in Gilmore and Gomory [3].

2. An algorithm

The graphs G_1 and G_2 of the previous section are examples of a class of graphs which for the purposes of this paper we call knapsack graphs.

Definition. A graph G with nodes N and arcs A is a *knapsack graph* if:
(2.1) The arcs A can be partitioned into n disjoint sets $A_1, \ldots, A_N$;
(2.2) the length of each arc belonging to A_j is l_j;
(2.3) let $P = (i_0, i_1, \ldots, i_p)$ be a path between an arbitrary pair of nodes i_0, i_p. Suppose that $(i_{t-1}, i_t) \in A_{m_t}$ for $t = 1, \ldots, p$. Then for any re-ordering $n_1, \ldots, n_p$ of the indices $m_1, \ldots, m_p$ there exists a path $Q = (j_0, j_1, \ldots, j_p)$ where $j_0 = i_0, j_p = i_p$ and $(j_{t-1}, j_t) \in A_{n_t}$ for $t = 1, \ldots, p$.

For shortest path problems with non-negative arc lengths an efficient algorithm is that described by Dijkstra [1]. We describe a modification of this algorithm applicable to a group knapsack problem which takes advantage of property (2.3) of knapsack graphs. The algorithm finds a shortest path from an origin node s to all other nodes.

Algorithm 1

The algorithm uses a set of labels (d_j, p_j) for each node j such that when a label is made 'permanent' by the algorithm d_j is the length of a shortest path TP_j from s to j and p_j is the predecessor of j on TP_j. Define a_j by arc $(p_j, j) \in A_{a_j}$ and note that for a group knapsack problem one can dispose with p_j and use labels (d_j, a_j). Finally if a label is not currently permanent it is referred to as temporary.

Step 0. Put $(d_s, p_s) = (0, s)$, $a_s = n$ and $(d_j, p_j) = (\infty, s)$ for $j \neq s$.

Step 1. If all labels are now permanent terminate, otherwise let $d_k = \min(d_j \mid j$ has a temporary label) make the label (d_k, p_k) permanent.

Step 2. For $r \leq a_k$ and $(k, j) \in A_r$, calculate $d_k + l_r$ and if $d_k + l_r < d_j$ replace the label of j by $(d_k + l_r, k)$. Go to step 1.

The improvement of the above algorithm over the more general Dijkstra algorithm is that in the latter algorithm one would have replace $r \leqslant a_k$ by $r \leqslant n$ in step 2.

Before proving that this modification is valid we introduce some notation.

For an arbitrary path P we denote its length by $l(P)$. If $P = (i_1, \ldots, i_p)$ and $(i_{p-1}, i_p) \in A_m$ we define $\delta(P) = m$.

At any stage of the algorithm if a node k has label (d_k, p_k) with $d_k \neq \infty$, then one can construct a path $Q = (j_0, \ldots, j_q)$ from s to k of length d_k by tracing back from k with $j_t = p_{j_{t+1}}$. If k has a permanent label we refer to this path as the tree path TP_k.

Given a path $P = (i_0, i_1, \ldots, i_p)$ with $(i_{t-1}, i_t) \in A_{m_t}$ we say that P conforms if $m_1 \geqslant m_2 \geqslant \cdots \geqslant m_p$.

Theorem 2.1. *Algorithm 1 finds a shortest path from s to all other nodes provided that $l_j \geqslant 0$ for $j = 1, \ldots, n$.*

Proof. We shall prove this inductively. We assume that when a label (d_k, p_k) is about to be made permanent that we have found shortest paths for the set of permanently labelled nodes $L \not\ni k$. This is trivially true initially.

We note first that if $i \in L$, $j \notin L$ and $(i,j) \in A_t$, then $t \leqslant \delta(TP_i)$ implies that $l(Q) \geqslant d_k$ where Q is TP_i followed by the arc (i,j).

Now let $P = (i_0, i_1, \ldots, i_p)$ be a path from s to k, and let i_t be the first node of P not belonging to L. Define P_1 to be $TP_{i_{t-1}}$ followed by the arc (i_{t-1}, i_t). Clearly $l(P_1) \leqslant l(P)$ and if P_1 conforms then $l(P_1) \geqslant d_k$ by the note at the beginning of the theorem. In this case $l(P) \geqslant d_k$ as is to be proved. Conversely suppose that $P_1 = (j_0, j_1, \ldots, j_q)$ and that $(j_{u-1}, j_u) \in A_{m_u}$ for $u = 1, \ldots, q$ and $m_q > m_{q-1}$. Define $r \geqslant 0$ by $m_1 \geqslant \cdots \geqslant m_r \geqslant m_q > m_{r+1} \geqslant \cdots \geqslant m_{q-1}$. By (2.3) applied to $(j_r, j_{r+1}, \ldots, j_q)$ there exists a path $(j_r, j'_{r+1}, \ldots, j'_q)$ from j_r to j_q such that $(j_r, j'_{r+1}) \in A_{m_q}$ and $(j'_t, j'_{t+1}) \in A_{m_t}$ for $t > r$. Let $Q = (j_0, \ldots, j_r, j'_{r+1}, \ldots, j'_q)$ and let j'_ρ be the first node of Q that does not belong to L where clearly $\rho \geqslant r+1$. Define the path P_2 to be $TP_{j'_{\rho-1}}$ followed by $(j'_{\rho-1}, j'_\rho)$. Clearly $l(P_2) \leqslant l(P_1)$. If $\rho = r+1$ then since $m_r \geqslant m_q$ we can deduce that $l(P_2) \geqslant d_k$ and the proof is complete. If $\rho > r+1$, then by the definition of r we have $\delta(P_2) < \delta(P_1)$. If P_2 conforms, then $l(P_2) \geqslant d_k$, otherwise we continue the above process to define paths $P_1, P_2, \ldots, P_n$, satisfying $l(P) \geqslant l(P_1) \geqslant l(P_2) \geqslant \cdots \geqslant l(P_n) \geq \cdots$ and $\delta(P_1) > \delta(P_2) > \cdots > \delta(P_n) > \cdots$. No path can be repeated in this process and we must ultimately terminate with a path P_N that conforms and has $l(P_N) \geqslant d_k$. Thus $l(P) \geqslant d_k$ and so a shortest path has been found.

We complete the proof by showing that any node reachable by a path from s will get a permanent label. Assuming the contrary there exists a path P from s to a node k that does not receive a finite permanent label. Using an almost identical argument to that above we can prove the existence of an infinite sequence of non-conforming paths $P_1, P_2, \ldots, P_n$, such $\delta(P_{n+1}) < \delta(P_n)$ for all n. Each path P_j being a tree path followed by an arc to a node not receiving a finite permanent label. This is clearly impossible.

The efficiency of the algorithm will depend on the ordering implied by

$A_1, \ldots, A_n$. A good ordering we feel is one satisfying

$$l_1 \leqslant l_2 \leqslant \cdots \leqslant l_n. \tag{2.4}$$

Ordering the arcs as in (2.3) does not minimise the total number of operations required by the algorithm for all sets of data. However it is a reasonable assumption that shortest paths will have short arcs and so such an order will tend to reduce the values of a_k in step 2.

The following theorem gives an upper bound to the number of operations required if (2.4) is satisfied.

We make an assumption for this theorem that for $t = 1, \ldots, n$ there exists a node σ_t such that $(s, \sigma_t) \in A_t$. This holds for example in the case of the group knapsack problem. In view of (2.3) if σ_k does not exist for some k, then no arc belonging to A_k lies on any path from s and so such arcs can be deleted.

Theorem 2.2. *Assume that* (2.4) *holds. For arbitrary* $k \neq s$ *let* $a(k)$ *denote the value of* a_k *in step* 2 *of the algorithm when the label of* k *is made permanent. Let* $s, i_1, \ldots, i_m$ *be an ordering of the nodes of* G *such that* $a(i_1) \leqslant a(i_2) \leqslant \cdots \leqslant a(i_m)$, *then* $a(i_t) \leqslant t$ *for* $t = 1, \ldots, m$.

Proof. Define the set of nodes $S_t = \{k \mid a(k) \leqslant t\}$ then we shall prove

$$|S_k| \geqslant k \quad \text{for } k = 1, \ldots, n. \tag{2.5}$$

It is clear that if we prove (2.5) we have proved the theorem. Let $\sigma_1, \ldots, \sigma_n$ be defined as above, then from (2.4) we deduce that $a(\sigma_t) \leqslant t$. Therefore $S_k \supseteq (\sigma_1, \ldots, \sigma_k)$ and the theorem is proved.

Consider now the group knapsack problem. Suppose that the group under consideration has D elements. Then the maximum number of group additions needed to find shortest paths to all non-zero nodes is

$$\sum_{r=1}^{n-1} r + (D - n - 1)n. \tag{2.6}$$

Note that to obtain (2.6) we use the fact that there is no need to carry out step 2 for the last node to be permanently labelled.

This is a pessimistic upper bound only being achieved if $D = n + 1$ and c_k is the shortest path from 0 to k for $k = 1, \ldots, n$. We note that $(D-1)(D-2)/2$ is the number required for the algorithm of Hu [6] and so our algorithm cannot be less efficient than this.

At the other extreme, if the group is cyclic with a generator g_1 with $c_1 = 0$ and $c_j > 0$ for $j \neq 1$, then the algorithm requires exactly D-2 group additions.

We can say that two paths in a knapsack graph G are equivalent if one can be obtained from the other by an application of (2.3). This divides the paths of G into equivalence classes. The efficiency of algorithm 1 rests on the fact that only one path from each equivalence class is considered throughout.

It is noted in Hu [6] that the Dijkstra algorithm can be readily adapted to solve problems where the length of a path is a more general function $\phi(P)$ satisfying

$$\phi(P) \geqslant \phi(Q) \quad \text{if } Q \text{ is a sub-path of } P. \tag{2.7}$$

We note that the proof of Theorem 2.1 only needs this property of paths in a graph with non-negative arc-lengths. Thus algorithm 1 can be modified in an obvious way to find minimum paths provided that (2.7) is satisfied.

(A further necessary condition is that if $P = (s, i_1', \ldots, i_p)$ and if Q is a minimum ϕ path from s to i_q where $q < p$, then we must have $\phi(Q, i_{q+1}, \ldots, i_p) \leqslant \phi(P)$).

We note that the Moore algorithm [7], where a node becomes a 'candidate' processing in step 2 each time its label changes can be modified in the same way we have modified the Dijkstra algorithm. We simply replace step 1 by: "Let k be any node which has not been chosen in step 2 since it last had its label altered. If no such k exists terminate". This algorithm terminates provided that the given graph has no negative cycles.

3. Acyclic knapsack graphs

A significant amount of time in algorithm 1 will be spent in finding the node k chosen in step 1 to have its label made permanent. It is clearly an advantage if there is a prior order in which the nodes can be chosen. For an acyclic graph one can use the topological order. We therefore assume that the nodes of the graph G have been numbered $0, 1, \ldots, m$ such that if (i, j) is an arc of G then $i < j$. The graph G_2 for the knapsack problem is already topologically ordered $0, 1, \ldots, W$. In this case step 1 of algorithm 1 can be replaced by "choose the next node in the order". Alternatively we can use the recurrence relation

$$d_j = \min(d_i + l_k \,|\, (i, j) \in A_k, k \leqslant a_i), \quad j = 0, 1, \ldots, m \tag{3.1}$$

where p_j is the node i giving the minimum in (2.9) and $(p_j, j) \in A_{a_j}$.

Note that l_k can be negative in this algorithm.

Theorem 3.1. *The values d_j defined by* (3.1) *are the lengths of shortest paths from* 0 *to j.*

Proof (outline). We proceed inductively assuming the propositions validity for $j < k$. Let P be a path from 0 to k. Let $i < k$ be the penultimate node of this path. Define P_1 to be TP_i followed by (i, k). Then $l(P_1) \leqslant l(P)$ and if P_1 conforms, then $d_k \leqslant l(P_1)$. If P_1 does not confirm then we can 're-sort' the arcs as in Theorem 2.1 to produce a path P_1'. Let i' be the penultimate node of P_1' and let P_2 be $TP_{i'}$ followed by (i', k). Then $l(P_2) \leqslant l(P_1)$ and $\delta(P_2) < \delta(P_1)$ and either P_2 conforms or we continue.

When applied to the knapsack problem by replacing min by max in (3.1), this is the algorithm of Gilmore and Gomory [3].

4. Non-linear knapsack problems

We consider next the problem

$$\text{minimise} \quad f_1(x_1)+\cdots+f_n(x_n), \tag{4.1}$$

$$\text{subject to} \quad w_1x_1+\cdots+w_nx_n = W, \tag{4.2}$$

where $x_1,\ldots,x_n$ are non-negative integers, $w_1,\ldots,w_n$, W are assumed to be positive integers and $f_j(0)\geq 0$ for all j.

We replace the above problem by a shortest path problem. Let G be the directed graph with nodes $(0,1,\ldots,W)$ and arcs of the form

$$(w, w+kw_j) \quad \text{for } j=1,\ldots,n \quad \text{and} \quad 0\leq k\leq[(W-w)/w_j]. \tag{4.3}$$

The length of each arc in (4.3) is $f_j(k)$ and the arcs of G are partitioned into $B_1,\ldots,B_n$ where B_t is the collection of arcs defined as in (4.3) with $j=t$.

Problem 4.1 is then equivalent to finding a shortest path from 0 to W, which uses at most one arc from each set B_j. For general functions $f_1,\ldots,f_n$ our formulation offers no advantage over the normal dynamic programming recussion.

$$g_r(w)=\min(f_r(x_r)+g_{r-1}(w-w_rx_r)\,|\,0\leq x_r\leq[w/w_r]), \tag{4.4}$$

where $g_r(w)$ is the optimum in 4.1 when n is replaced by r and W is replaced by w. This is essentially because subpaths of shortest paths are not necessarily shortest paths when the restriction of at most one arc from each B_j is applied.

There is however a class of functions for which the above property is true.

Definition. A function f is *super-additive* if

$$f(x)+f(y)\geq f(x+y) \quad \text{for all } x,y\geq 0. \tag{4.5}$$

It can be shown for example that if f is concave and $f(0)\geq 0$ then f is super additive over the positive reals.

Assuming 4.5 we can implicitly relax the restriction of at most one arc from each B_j and apply the normal acyclic shortest path algorithm with the refinements available to knapsack graphs. In the algorithm of course the relaxation is not made as by (4.5) there is in fact no need. We are thus led to the following algorithm for solving (4.1) if all f_j are superadditive.

We use a triple label scheme (d_w, p_w, q_w) where at the termination of the algorithm d_w is the minimum objective value if W is replaced by w, $p_w=t$ where t is the smallest index j such that the minimal path from 0 to w contains an arc of B_j and $q_w=k$ indicating that this arc is $(w-kw_t, w)$.

Step 0. $d(0)=0$ and $d_w=\infty$ for $w=1,\ldots,W$, $r=0$ and $p_0=n+1$.

Step 1. For $t=1,\ldots,p_r-1$ and $s=1,\ldots,[(W-r)/W_t]$ calculate $d_t+f_t(s)$ and if $d_t+f_t(s)<d_{r+sw_t}$, relabel $r+s\ w_t$ with $(d_t+f_t(s), t, s)$.

Step 2. $r=r+1$, if $r<W$ go to step 1, otherwise terminate.

Theorem 4.1. *The above algorithm finds a solution to* (4.1) *for all right-hand sides* $w=0,1,\ldots,W$ *of* (4.2).

Proof (outline). We can clearly proceed inductively assuming the theorems truth for $w < k$. For an arbitrary path P define $\epsilon(P)$ by the last arc of P belongs to $B_{\epsilon(P)}$. Now let P_1 be any path from 0 to k and let i_1 be the penultimate node of P_1. If $\epsilon(P) < p(i_1)$ then clearly $l(P_1) \geq d(k)$. Using our given shortest path to i_1 and arc of P from i_1 to k, resorting the arcs as in Section 2 and combining two arcs from $B_{\epsilon(P_1)}$ if necessary and using 4.5 we obtain a path P_2 from 0 to k such that $l(P_2) \leq l(P_1)$ and $\epsilon(P_2) < \epsilon(P_1)$. The proof continues as in Theorem 3.1.

We have carried out some limited experiments with this algorithm using randomly generated problems where f_j had the form

$$f_j(0) = 0,$$
$$f_j(x) = a_j + b_j x \quad \text{if } x > 0.$$

Prior to applying the algorithm we sorted the functions so that

$$a_j / W + b_j / w_j \leq a_{j+1} / W + b_{j+1} / w_j,$$

i.e. in order of increasing cost per unit length in the range $[0, W]$. We wished to make a comparison with the dynamic programming algorithm of (4.4) and so we compare the number of function evaluations used in our procedure with the number that would have been needed in (4.4) (see Table 1).

The parameters in the table of results are as follows: (n, W: are as in 4.1 and 4.2).

wa the integers w_j were uniformly randomly generated in the range $(1, wa)$,
a the integers a_j were uniformly randomly generated in the range $(0, a)$,
b the integers b_j were uniformly randomly generated in the range $(1, b)$,
e_1 the number of function evaluations needed by the algorthm,
e_2 the number of function evaluation needed by (4.4).

Table 1.

n	W	wa	a	n	e_1	e_2
25	250	8	4	5	5955	273438
25	250	8	4	5	6016	260764
25	250	8	4	5	7849	275276
25	250	8	4	5	7798	255851
25	250	20	4	5	6734	102071
25	250	20	4	5	5979	171755
25	250	20	4	5	6595	135748
25	250	20	4	5	7364	155193
25	1000	25	10	10	30153	1470256
25	1000	25	10	10	23906	2307246
25	1000	25	10	10	28833	1965540
25	1000	25	10	10	29136	2058319

Table 1 demonstrates the clear superiority of our algorithm over (4.4) in terms of computation. Another point in the algorithm's favour is that the amount of storage needed is $3W$ whereas (4.4) requires a minimum $(n+2)W$.

The ordering of the function f_i is clearly important and failing any other information we could order them after carrying out the first iteration of the algorithm by sorting them in decreasing order of the number of $p(w)$ with a given value.

References

[1] E.W. Dijkstra, "A note on two problems m connection with graphs", *Numerische Mathematik* 1 (1959) 269–271.

[2] P.C. Gilmore and R.E. Gomory, "Multistage cutting stock problems of two and more dimensions", *Operations Research* 13 (1965) 94–120.

[3] P.C. Gilmore and R.E. Gomory, "The theory and computation of knapsack functions", *Operations Research* 14 (1966) 1045–1074.

[4] R.E. Gomory, "On the relation between integer and non-integer solutions to linear programs", *Proceedings of the National Academy of Sciences* 53 (1965) 250–265.

[5] R.E. Gomory, "Some polyhedra related to combinatorial problems", *Linear Algebra and its Applications* 2 (1969) 451–558.

[6] T.C. Hu, *Integer programming and network flows* (Addison-Wesley, Reading, Mass., 1969).

[7] E.P. Moore, "The shortest path through a maze", in: *Proceedings of an international symposium on the theory of switching*, Part II, Apr. 2–5, 1957 (Harvard University Press, Cambridge, Ma., 1959).

[8] J.F. Shapiro, "Dynamic programming algorithms for the integer programming problem I: The integer programming problem viewed as a knapsack problem", *Operations Research* 16 (1968) 103–131.

[9] J.F. Shapiro, "Group theoretic algorithms for the integer programming problem II: Extension to a general algorithm", *Operations Research* 16 (1968) 928–947.