A Polynomial-Time Algorithm for the Knapsack Problem with Two Variables

D. S. HIRSCHBERG AND C. K. WONG

IBM Thomas J Watson Research Center, Yorktown Heights, New York

ABSTRACT. The general knapsack problem is known to be NP-complete In this paper a very special knapsack problem is studied, namely, one with only two variables. A polynomial-time algorithm is presented and analyzed However, it remains an open problem that for any fixed $n > 2$, the knapsack problem with n variables can be solved in polynomial time.

KEY WORDS AND PHRASES: knapsack problem, polynomial-time algorithm, integer optimization, continued fraction approximation

CR CATEGORIES 5.25, 5.30

1. *Introduction*

The knapsack problem considered here is the following:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} c_i X_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_i X_i \leq b \\ \text{and} \quad & X_i \text{ nonnegative integers,} \end{aligned} \tag{1}$$

where a_i, c_i, b are positive real numbers. It can be shown [6] that this problem is NP-complete in the sense of Karp [4, 5]. Recently, polynomial-time *approximate* solutions to the special case when X_i is restricted to 0, 1 have been obtained [7] In fact, polynomial-time approximate solutions to the general case when X_i is not restricted can be obtained in a similar fashion.

In this paper we follow another approach; namely, we fix the number of variables n and look for polynomial-time *optimum* solutions to the knapsack problem of n variables Although our attempt to solve this problem in general has been unsuccessful, we have found an algorithm for the case $n = 2$, which runs in polynomial time of the length of the input. More specifically, we shall solve the following problem:

$$\begin{aligned} \text{maximize} \quad & c_1X_1 + c_2X_2 \\ \text{subject to} \quad & a_1X_1 + a_2X_2 \leq b \\ \text{and} \quad & X_1, X_2 \text{ nonnegative integers} \end{aligned} \tag{2}$$

Note that the naive approach (test $X_1 = i$, $X_2 = \lfloor (b - a_1 i)/a_2 \rfloor$ as i takes on integer values from 0 to $\lfloor b/a_1 \rfloor$) takes time proportional to b/a_1 which is exponential in the length of the input

Authors' present addresses. D S Hirschberg, Department of Electrical Engineering, Rice University, Houston, TX 77001, C. K. Wong, Computer Sciences Department, IBM Thomas J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598

It is hoped that such a modest beginning will attract more attention to this problem and hence bring about its eventual solution.

2. *Preliminary Algorithms*

We first present an algorithm (Algorithm 1) which yields approximations to a given real number within a required accuracy with a fraction of minimal denominator. It solves a special form of (2), namely, when $c_1 = a_1 = 1$ and $c_2 = a_2$, and motivates the procedures in Algorithm 2, which will be a major part of the final algorithm. It will also be used in the proof of Theorem 2.

ALGORITHM 1

```
Given $\mu > 0$
(1) Initialize $m = \lfloor \mu \rfloor$, $n = 1$, $M = 1$, $N = 0$.
(2) Let $\delta_1 = -m + n\mu$, $\delta_2 = M - N\mu$.
    (i) If $\delta_1 \geq \delta_2$, set $m \leftarrow M + m$, $n \leftarrow N + n$
    (ii) If $\delta_1 < \delta_2$, set $M \leftarrow M + m$, $N \leftarrow N + n$.
(3) Halt if $\delta_1 = 0$, otherwise repeat (2) until required accuracy is satisfied
```

Remarks. Note that at all times except possibly the last step, δ_1 and δ_2 are positive and $M/N > \mu > m/n$. Consequently, $M/N > (M + m)/(N + n) > m/n$.

In (i), $\delta_1 \geq \delta_2$ implies that $M/N > \mu \geq (M + m)/(N + n)$ Execution of this step results in $\delta_1 \leftarrow \delta_1 - \delta_2$. Similarly, in (ii), $(M + m)/(N + n) > \mu > m/n$ and after execution, $\delta_2 \leftarrow \delta_2 - \delta_1$.

THEOREM 1. *M/N, m/n are the best approximations to μ for that size denominator and the first better approximation does not occur until $N + n$. This can be stated formally as:*

(a) for all $v < N + n$, $u/v \geq M/N$ or $u/v \leq m/n$,

(b) for all $v < N$, $u/v > M/N$ or $u/v \leq m/n$,

(c) for all $v < n$, $u/v \geq M/N$ or $u/v < m/n$.

PROOF. By induction on the steps of the algorithm. The theorem is obviously true at the initial step. Assume that it was true at the last step. If we last had condition (i) (i.e. n changed), then (b) is still true since the condition $u/v \leq m/n$ was eased, the condition $u/v > M/N$ was unchanged, and N did not change. (c) is true from previous (a) and (c) and the fact that the new m/n is larger than the old m/n. Similarly if we last had condition (ii) then (*b*) and (*c*) are now true.

We now show that (a) is true From (b) and (c) we have that for all $v < \max(n, N)$, $u/v \geq M/N$ or $u/v \leq m/n$ The case $v = \max(n, N)$ is trivial. Thus, we need only show that for all v, $\max(n, N) < v < N + n$, $u/v \geq M/N$ or $u/v \leq m/n$. Assume it is not true, i.e there exist u, v such that $\max(n, N) < v < N + n$, $M/N > u/v > m/n$. Let $v = N + v' = n + \hat{v}$, $0 < v' < n$, $0 < \hat{v} < N$. Let $u = M + u' = m + \hat{u}$. Then

$$M/N > (M + u')/(N + v') = u/v = (m + \hat{u})/(n + \hat{v}) > m/n. \tag{3}$$

From the left-hand side of (3), it follows that $M/N > u'/v'$. Therefore, by condition (c), $u'/v' < m/n$, hence $(m - u')/(n - v') > m/n$. Again by (c), $(m - u')/(n - v') \geq M/N$. On the other hand, the right-hand side of (3) implies that $\hat{u}/\hat{v} > m/n$. Therefore by (b), $\hat{u}/\hat{v} > M/N$, which can be written as $[M + (u' - m)]/[N + (v' - n)] > M/N$. Combining these inequalities, we have

$$M/N = \{[M + (u' - m)] + (m - u')\}/\{[N + (v' - n)] + (n - v')\} > M/N,$$

a contradiction. □

COROLLARY 1. *For all $v < N + n$, let $\delta(v) = -u + v\mu$. Then $\delta(v) \geq \delta_1$ or $\delta(v) < 0$.*

PROOF. From Theorem 1(a), for all $v < N + n$, $u/v \geq M/N$ or $u/v \leq m/n$. The former implies $u/v > \mu$, hence $\delta(v) < 0$.

For the latter case we shall first consider $v \geq n$. We have $u/v \leq m/n \leq \mu$, which implies $\mu - u/v \geq \mu - m/n \geq 0$. We can multiply on the left by v and on the right by n, getting

$\delta(v) \geq \delta_2$. We now consider $v < n$ (which is a nonempty case only for $n > 1$). If $m = 0$, then n must be one. If $u = 0$, then $m/(n-1) > (m/n)$ and hence, by Theorem 1, $m/(n-1) > \mu$. $m/(n-v) \geq m/(n-1)$ and so $m/(n-v) > \mu$ from which follows that $\delta(v) > \delta_1$. The only other possibility is that u and m are positive. In this case, inverting the inequality gives $(v/u)\mu \geq (n/m)\mu$, and multiplying by -1 and adding 1 yields $1 - (v/u)\,\mu \leq 1 - (n/m)\,\mu$, both positive numbers since $u/v \leq m/n \leq \mu$ Since $v < n$, it follows that $u < m$ and we can multiply (strengthening the inequality) yielding $u - v\mu < m - n\mu$ and thus $\delta(v) > \delta_1$. □

Consider the problem $\mu X \leq 1 \pmod 1$, where μ is a given positive real number. X is restricted to nonnegative integers. Then the sequence $\{N\}$ generated by Algorithm 1 when applied to the number μ has the property that $\{\mu N\}$ forms a sequence of closer approximations to 1 (mod 1) from below.

We now present Algorithm 2, which will do the same for the problem $\mu X \leq q \pmod 1$, where $0 < q \leq 1$ and $X \leq b_{\max}$, a given positive number. This algorithm will be a major part of our final algorithm

ALGORITHM 2

```
Given $\mu$, $q$, $b_{\max}$ $(0 < q \leq 1)$.
(1) Initialize $a = 0,\ b = 0,\ c = \lfloor\mu\rfloor,\ d = 1,\ e = 1,\ f = 0$.
(2) Let $\delta_B = a - b\mu + q,\ \delta_D = -c + d\mu,\ \delta_F = e - f\mu$
    Iterate testing to determine which of conditions (i), (ii), or (iii) occurs and taking the action
    indicated for that condition.
        (i)   $\delta_B \geq \delta_D$                          $a \leftarrow a + c,\ b \leftarrow b + d$
        (ii)  $\delta_B < \delta_D,\ \delta_D \geq \delta_F$    $c \leftarrow c + e,\ d \leftarrow d + f$.
        (iii) $\delta_B < \delta_D < \delta_F$.                 $e \leftarrow c + e,\ f \leftarrow d + f$.
(3) Halt whenever $b + d > b_{\max}$.
```

Remarks. After execution of (i), $\delta_B \leftarrow \delta_B - \delta_D$. Similarly, after (ii), $\delta_D \leftarrow \delta_D - \delta_F$ and after (iii), $\delta_F \leftarrow \delta_F - \delta_D$.

Note that the sequence of b's (d's, f's) generated by the algorithm is strictly increasing and the corresponding sequence of δ_B's (δ_D's, δ_F's) is strictly decreasing.

Also note that at all times $0 \leq \delta_B < 1$, $0 \leq \delta_D < 1$, $0 < \delta_F \leq 1$, and that $0 \leq -a + b\mu \leq q$

For integer β, define $\delta_B(\beta) = \alpha - \beta\mu + q$ where α is the integer such that $0 \leq \delta_B(\beta) < 1$. For β in the sequence of b's generated by the algorithm, α will be the associated a-value.

THEOREM 2. *The sequence of b's generated by Algorithm 2 is such that b_{i+1} is the min β for which $0 \leq \delta_B(\beta) < \delta_B(b_i)$. Formally:*

(a) for all $\beta < b$, $\delta_B(\beta) > \delta_B(b)$,

(b) for all $\beta < b + d$, $\delta_B(\beta) \geq \delta_B(b)$,

where d is the d-value at the next occurrence of condition (i).

PROOF. Since the above assertions are concerned with condition (i), we shall prove them by induction on the initial step and occurrences of condition (i). Initially, (a) is true. If initially (i) occurs, then (b) is also true; otherwise assume that (b) is not true, i.e. there exists $\beta < b + d$ such that $0 \leq \alpha - \beta\mu + q = \delta_B(\beta) < \delta_B(b) = a - b\mu + q$. We can assume $\beta > b$ from (a) (the case $\beta = b$ is trivial), so let $\beta = b + \hat{d}$, where $0 < \hat{d} < d$. Then $0 \leq \delta_B(b) + (\alpha - a) - \hat{d}\mu < \delta_B(b)$. Consequently, $-1 < -\delta_B(b) \leq (\alpha - a) - \hat{d}\mu < 0$. On the other hand $0 \leq \delta_D(\hat{d}) = -\hat{c} + \hat{d}\mu < 1$ determines the integer $\hat{c}$ uniquely; therefore $\hat{c} = \alpha - a$. Let d be obtained in the algorithm by $d = d' + f'$; then $\delta_D(d') > \delta_D(d)$ and $\delta_D(d') > \delta_B(b)$ since d is the first d-value obtained in the algorithm with $\delta_D(d) \leq \delta_B(b)$. Note that as far as c, d, e, f are concerned, their generation is exactly like that of m, n, M, N in Algorithm 1; hence Corollary 1 applies and $\hat{d} < d' + f'$ implies

$$-\delta_D(\hat{d}) = (\alpha - a) - \hat{d}\mu \leq c' - d'\mu = -\delta_D(d') < -\delta_B(b) \quad \text{or} \quad -\delta_D(\hat{d}) > 0,$$

either of which contradicts our assumption that (b) was not true.

As for the inductive argument, (a) follows from the previous (b) and the fact that $\delta_B(b') < \delta_B(b)$ (b' is the next b-value generated), and (b) follows from exactly the same argument used in the initial proof. □

COROLLARY 2. *If $b \leq b_{max}$ and for all $\beta < b$, $\delta_B(\beta) > \delta_B(b)$, then b is in the set of b-values generated by Algorithm 2.*

PROOF Assume not, i.e. there exists $\hat{b} \leq b_{\max}$ such that for all $\beta < \hat{b}$, $\delta_B(\beta) > \delta_B(\hat{b})$ and $\hat{b}$ is not generated by Algorithm 2. If there are b-values b_1 and b_2 that have been consecutively generated by Algorithm 2 (by $b_2 = b_1 + d$) such that $b_1 < \hat{b} < b_2$, then by Theorem 2(b) $\delta_B(b) \geq \delta_B(b_1)$, which contradicts our assumption. Otherwise, $\hat{b} > b_0$ (the last b generated by the algorithm) and $b_0 + d_0$ has exceeded $b_{\max}$. We note that the proof of Theorem 2(b) holds for all d occurring before condition (i) recurs Then $b_0 < \hat{b} \leq b_{\max} < b_0 + d_0$ and by Theorem 2(b) $\delta_B(\hat{b}) \geq \delta_B(b_0)$. Again our assumption has been contradicted □

3 *The Knapsack Problem*

We are now in a position to discuss the knapsack problem with two variables We consider the following two problems first.

Problem 1. Given μ, ω Find nonnegative integers i and j that maximize $i + \mu j$ subject to $i + \mu j \leq \omega$

Problem 2. Given μ, $\pi < \mu$, ω. Find nonnegative integers i and j that maximize $i + \pi j$ subject to $i + \mu j \leq \omega$.

Note that for both problems it suffices to find the optimal value of j since the optimal value of i can be obtained by $i = \lfloor \omega - \mu j \rfloor$.

THEOREM 3. *The solution value of j in Problem 1 will be the last b-value not exceeding b_{max} that is generated by Algorithm 2 with inputs q equal to the fractional part of ω such that $0 < q \leq 1$ and $b_{max} = \omega/\mu$.*

PROOF. Let j be the last b-value not exceeding $b_{\max}$ generated by Algorithm 2 From Theorem 2(b) (as extended in proof of Corollary 2) we have: for all $\beta \leq b_{\max} < j + d$, $\delta_B(\beta) > \delta_B(j)$, which can be rewritten as: for all α, β, $\alpha - \beta\mu + q > a - j\mu + q \geq 0$, or $\alpha - \beta\mu + q < 0$.

For α specified in the definition of $\delta_B(\beta)$, the first inequality will apply. For any other α, one of the above two inequalities will apply. Let $k = \omega - q$ (and thus an integer), then multiplying the previous set of inequalities by -1 and adding ω gives:

$$\text{for all } \alpha, \beta, \ (k - \alpha) + \beta\mu < (k - a) + j\mu \leq k + q = \omega, \quad \text{or} \\ (k - \alpha) + \beta\mu > k + q = \omega. \quad (4)$$

Also $a - j\mu + q = \delta_B(j) < 1$ so $(k - a) + j\mu > \omega - 1$, from which we see that $k - a > \omega - j\mu - 1 \geq -1$ since $j\mu \leq \omega$. $(k - a)$ is an integer greater than -1, thus greater than or equal to 0 and so by (4) we have that $(k - a, j)$ is a feasible solution point for Problem 1. Also from (4), we note that for all α, β, either $(k - \alpha) + \beta\mu < (k - a) + j\mu$ or $(k - \alpha) + \beta\mu > \omega$, hence $(k - a, j)$ is optimal. □

THEOREM 4. *The solution value of j in Problem 2 is among the b-values generated by Algorithm 2 (using as inputs q equal to the fractional part of ω and $b_{max} = \omega/\mu$) and thus can be found by simply testing all the b-values generated to see which yields the maximum value of $i + \pi j$.*

We first need the following lemma:

LEMMA 1. *If $\beta > b$ and $\delta_B(\beta) \geq \delta_B(b)$ then (ρ, β) is not the solution to Problem 2 for any ρ.*

PROOF. We can assume that $\beta \leq \omega/\mu$, otherwise there would be no feasible ρ. Rewriting the assumption of the lemma, we have

$$1 > \delta_B(\beta) = \alpha - \beta\mu + q \geq a - b\mu + q = \delta_B(b) \geq 0.$$

Let $k = \omega - q$ (k integer), then multiplying the above inequality by -1 and adding ω will give

$$\omega - 1 < (k - \alpha) + \beta\mu \leq (k - a) + b\mu \leq \omega. \tag{5}$$

Note that the integer $\rho = (k - \alpha)$ cannot be increased without violating the upper bound ω.

We also have the following inequality: $(k - \alpha) + (\beta - b)\pi < (k - \alpha) + (\beta - b)\mu \leq k - a$. The first part is true since $\pi < \mu$ and the second part comes from (5). From this we see that $(k - \alpha) + \beta\pi < (k - a) + b\pi$ and thus $(k - a, b)$ is a feasible solution which gives a greater value of the function $i + \pi j$ than that of (ρ, β) for the maximum ρ possible (and hence for all feasible ρ). □

Proof of Theorem 4. From Lemma 1 we see that for (i, j) to be a solution to Problem 2 we must have that for every $\beta < j$, $\delta_B(\beta) > \delta_B(j)$. Also, for (i, j) to be a solution, $j \leq \omega/\mu$; otherwise the value of i would necessarily be negative to satisfy the inequality and hence infeasible. By Corollary 2, j will be among the b-values generated by Algorithm 2.

4. *A Faster Algorithm*

We now present Algorithm 3 (the last one in this paper), which is Algorithm 2 with a few modifications that will speed up its execution performance For instance, in Algorithm 2, condition (i) would cause $a \leftarrow a + c$, $b \leftarrow b + d$ and thus $\delta_B \leftarrow \delta_B - \delta_D$. Condition (i) would recur until δ_B was reduced below δ_D. This would have occurred after $\lfloor\delta_B/\delta_D\rfloor$ steps. It is now done in one step.

```
ALGORITHM 3

Given $\mu$, $q$, $b_{\max}$ $(0 < q \leq 1)$
(1) Initialize $a = 0$, $b = 0$, $c = \lfloor\mu\rfloor$, $d = 1$, $e = 1$, $f = 0$.
(2) Let $\delta_B = a - b\mu + q$, $\delta_D = -c + d\mu$, $\delta_F = e - f\mu$.
    Iterate testing to determine which of conditions (i), (ii), or (iii) occurs and taking the action
indicated for that condition.
      (i)   $\delta_B \geq \delta_D$.                  $a \leftarrow a + \eta c$, $b \leftarrow b + \eta d$, where $\eta = \min(\lfloor\delta_B/\delta_D\rfloor, \lfloor(b_{\max} - b)/d\rfloor)$.
      (ii)  $\delta_B < \delta_D$, $\delta_D \geq \delta_F$ ·  $c \leftarrow c + \gamma e$, $d \leftarrow d + \gamma f$, where $\gamma = \min(\lfloor\delta_D/\delta_F\rfloor, \lceil(\delta_D - \delta_B)/\delta_F\rceil)$.
      (iii) $\delta_B < \delta_D < \delta_F$            $e \leftarrow e + \lfloor\delta_F/\delta_D\rfloor c$, $f \leftarrow f + \lfloor\delta_F/\delta_D\rfloor d$.
(3) Halt whenever $b + d > b_{\max}$.
```

Theorem 5 *Algorithm 3 generates the solutions to Problems 1 and 2 with inputs q equal to the fractional part of ω, $b_{max} = \omega/\mu$*

Lemma 2. *Let $b_k = b + kd$, $k = 0, \cdots, K$, such that $1 > \delta_B(b_k) > \delta_B(b_{k+1}) \geq 0$. Then either (i) none of $b_1, \cdots, b_{k-1}$ is a solution j-value to Problem 2 or (ii) one of b_0 or b_k is a solution j-value to Problem 2.*

Proof. Rewriting the assumed inequalities we have

$$1 > a_0 - b_0\mu + q > \cdots > a_k - b_k\mu + q > \cdots \geq 0.$$

Multiplying the above set of inequalities by -1 and adding ω gives

$$\omega - 1 < \omega - q - a_0 + b_0\mu < \cdots < \omega - q - a_k + b_k\mu < \cdots \leq \omega. \tag{6}$$

From the definition of b_k we have

$$b_{k+1}\mu - b_k\mu = d\mu. \tag{7}$$

Let $i_k = \omega - q - a_k$, $0 \leq k \leq K$ (which are integers since $\omega - q$ is an integer and so is a_k) and let $t = \lfloor d\mu \rfloor$. We have.

Sublemma. $i_{k+1} = i_k - t$.

Proof of Sublemma. Assume that for some k, $i_{k+1} \geq i_k - t + 1$. Then adding $b_{k+1}\mu$ to both sides gives

$$i_{k+1} + b_{k+1}\mu \geq i_k - t + 1 + b_k\mu + d\mu = i_k + b_k\mu + 1 + (d\mu - t).$$

$\lfloor d\mu \rfloor = t$ and so $d\mu - t \geq 0$. Also, from (6), $i_k + b_k\mu > \omega - 1$ and thus $i_k + b_k\mu + 1 >$ $.i_k$ Together this implies that $i_{k+1} + b_{k+1}\mu > \omega$, which contradicts (6). Therefore, since ω_{+1} is an integer, $i_{k+1} \leq i_k - t$.

Assume that for some k, $i_{k+1} \leq i_k - t - 1$. From (6) and (7) we have

$$i_k + b_k\mu < i_{k+1} + b_{k+1}\mu \leq (i_k + b_k\mu) + [(d\mu - t) - 1].$$

But this implies $1 < d\mu - t$, which contradicts $t = \lfloor d\mu \rfloor$. This leaves $i_{k+1} = i_k - t$ as the only possible alternative.

CONTINUING WITH PROOF OF LEMMA 2. If $d\pi < t$, then for all k, $i_k + b_k\pi = i_{k+1} + t + b_k\pi = i_{k+1} + t + b_{k+1}\pi - d\pi > i_{k+1} + b_{k+1}\pi$. Therefore $\{b_k\}$, $k > 0$, do not maximize the objective function $i + j\pi$ Thus (i) is true.

If $d\pi > t$, then for all k, $i_k + b_k\pi < i_{k+1} + b_{k+1}\pi$ and $\{b_k\}$, $k < K$, do not maximize the objective function. Thus (i) is true

If $d\pi = t$, then for all k, $i_k + b_k\pi = i_{k+1} + b_{k+1}\pi$ and $b_0, \cdots, b_K$ all give the same value for the objective function. So either all are solution j-values to Problem 2, in which case (ii) is true, or all are not solutions, in which case (i) is true. □

PROOF OF THEOREM 5. The flows of conditions (i), (ii), and (iii) in Algorithms 2 and 3 are shown in Figure 1.

It is seen that the modifications simply skip over the self-loops and make (in one shot) all the changes in the variables that would have occurred one at a time

The test for $\lfloor (b_{\max} - b)/d \rfloor$ ensures that the last b-value less than or equal to $b_{\max}$ that would have occurred in Algorithm 2 does occur in Algorithm 3 and is the solution of Problem 1.

As shown in Lemma 2, the skipped b's are not needed to ensure that the solution to Problem 2 which is generated by Algorithm 2 (see Theorem 4) is also generated by Algorithm 3. □

Next we will show that the number of iterations in Algorithm 3 is $O(\log b_{\max})$. First, referring to Figure 2 for the flow of conditions in Algorithm 3, we have:

LEMMA 3 *Branch Z_1 cannot occur more than log $b_{\max}$ times.*

PROOF. Z_1 occurs only if condition (iii) is followed by condition (ii). Let d and f be values when (iii) occurs; d', f' after (iii); d'', f'' after (ii).

Then $d' = d$, $f' \geq f + d$ (see Algorithm 3), $d'' \geq d' + f'$, $f'' = f'$, and thus we have $d'' \geq 2d$.

Thus each time Z_1 occurs, d at least doubles d is initialized at 1. After log $b_{\max}$ times, d would have value greater than $b_{\max}$ and the algorithm would have halted. □

LEMMA 4. *Branches Z_2 and Z_3 cannot occur more than log $b_{\max}$ times.*

PROOF. Z_1 is the only exit from (iii) and, by Lemma 3, cannot occur more than log $b_{\max}$ times. □

LEMMA 5. *The chain of branches $Z_4 - Z_5 - Z_4 - Z_5$ cannot occur.*

PROOF. Assume it can occur. This is equivalent to assuming that the following se-

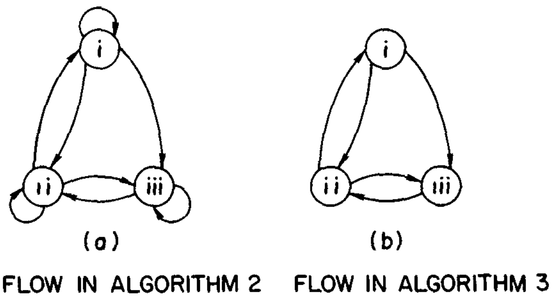
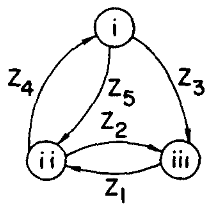


FIG. 1



FLOW OF CONDITIONS IN ALGORITHM 3

FIG. 2

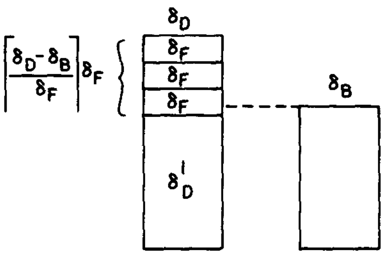


FIG 3

quence of conditions can occur: (ii) (i) (ii) (i) (ii). Initially $\delta_B < \delta_D$, $\delta_D \geq \delta_F$ and (ii) occurs. As a result of (ii), $\delta_D' \geq \delta_D - \lceil(\delta_D - \delta_B)/\delta_F\rceil\delta_F$ since (i) occurs next (see Algorithm 3).

Also, that (i) occurs next implies $\delta_D' \leq \delta_B$. Furthermore, we have

$$\delta_B - \delta_D' = \delta_B - (\delta_D - \lceil(\delta_D - \delta_B)/\delta_F\rceil\delta_F) = \lceil(\delta_D - \delta_B)/\delta_F\rceil\delta_F - (\delta_D - \delta_B) < \delta_F .$$

Together this means that after (ii) and before (i), δ_D was reduced below δ_B but just below δ_B (see Figure 3). After (i) occurs, $\delta_B' < \delta_D'$ and also $\delta_B' < \delta_F$ since $\delta_B - \delta_D' < \delta_F$. Since (ii) occurs next, $\delta_D' \geq \delta_F$. Thus, at this point, $\delta_D' \geq \delta_F > \delta_B'$. After (ii), we have $\delta_D'' \leq \delta_B'$ and thus (i) will occur next with $\delta_D'' \leq \delta_B' < \delta_F$. After (i), $\delta_B'' < \delta_D''$ and thus $\delta_B'' < \delta_D'' < \delta_F$. But this means (iii) and not (ii) occurs next. □

LEMMA 6. *Branches Z_4 and Z_5 cannot occur more than 6 log b_{max} times*

PROOF. At least one of branches Z_1, Z_2, or Z_3 must occur for each two occurrences of branches Z_4 or Z_5 since there are no self-loops and we cannot loop Z_4 and Z_5 for more than two occurrences of Z_4. Since the sum of the number of occurrences of Z_1, Z_2, and Z_3 is less than 3 log $b_{\max}$ (Lemmas 6 and 7), Z_4 and Z_5 cannot occur more than 6 log $b_{\max}$ times. □

THEOREM 6. *The number of iterations in Algorithm 3 is O(log b_{max}).*

PROOF. It follows from Lemmas 3, 4, and 6. □

COROLLARY 3. *Algorithm 3 can be used to solve Problem 1 or Problem 2 in $O(|\log \omega| + |\log \mu|)$ iterations*

PROOF. For Problem 1, the last b-value generated is the solution j-value. To get this it will take $O(\log b_{\max}) \leq O(|\log \omega| + |\log \mu|)$ iterations. The solution i-value is simply $i = \lfloor \omega - \mu j \rfloor$.

For Problem 2, we need simply look at the b-values generated (less than or equal to $O(\log b_{\max})$ of them) and test for maximizing $i + \pi j$ with $j = b$, $i = \lfloor \omega - \mu j \rfloor$. □

Finally, we can address the original problem:

Problem 3. Given c_1, c_2, a_1, a_2, b. Find integers X_1, $X_2 \geq 0$ so that $c_1X_1 + c_2X_2$ is maximized and $a_1X_1 + a_2X_2 \leq b$.

COROLLARY 4. *Algorithm 3 can be used to solve Problem 3 in $O(|\log c_1| + |\log c_2| + |\log a_1| + |\log a_2| + |\log b|)$ iterations.*

PROOF. Without loss of generality, we can assume that $c_1/a_1 \geq c_2/a_2$.

If $c_1/a_1 > c_2/a_2$, then Problem 3 is equivalent to Problem 2 with $\pi = c_2/c_1$, $\mu = a_2/a_1$, $\omega = b/a_1$.

If $c_1/a_1 = c_2/a_2$ then Problem 3 is equivalent to Problem 1 with μ and ω as above.

In either case the algorithm will terminate in $O(|\log \omega| + |\log \mu|)$ iterations and $|\log \omega| \leq |\log b| + |\log a_1|$, $|\log \mu| \leq |\log a_1| + |\log a_2|$.

Since the number $\pi = c_2/c_1$ is used, the term $O(|\log c_1| + |\log c_2|)$ is also needed. The corollary thus follows. □

Since in each iteration multiplication of $O(n)$ precision numbers is being performed and such a multiplication takes time $O(n \log n \log \log n)$, it follows that the asymptotic time complexity of our algorithms is $O(n^2 \log n \log \log n)$.

5. *Concluding Remarks*

In this paper we propose an algorithm to solve the knapsack problem with two variables. This algorithm originates from a well-known continued fraction method and runs in polynomial time of the input length. We conjecture that for any fixed $n > 2$, the knapsack problem with n variables may be solved in polynomial time The proof seems very difficult and generalization of the method used in this paper does not seem to work. It is hoped that the modest beginning presented in this paper will draw the attention of more researchers and will bring about the eventual solution of this problem

REFERENCES

(Note. References [1-3] are not cited in the text.)

1. HARDY, G. H , AND WRIGHT, E M *An Introduction to the Theory of Numbers*, 3rd ed Oxford U. Press, London, 1956, pp. 129-153
2. HOROWITZ, E , AND SAHNI, S. Computing partitions with applications to the knapsack problem *J ACM 21*, 2 (April 1974), 277-292.
3. HU, T C. *Integer Programming and Network Flows* Addison-Wesley, Reading, Mass , 1969, pp. 311-316
4. KARP, R M Reducibility among combinatorial problems In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., Plenum Press, New York, 1972, pp 85-103.
5. KNUTH, D. E. A terminological proposal. *SIGACT News 6*, 1 (Jan. 1974), 12-18
6. LUEKER, G S Two polynomial complete problems in nonnegative integer programming. Computer Science TR-178, Princeton U , Princeton, N J., March 1975.
7. SAHNI, S. Approximate algorithms for the 0/1 knapsack problem. *J ACM 22*, 1 (Jan. 1975), 115-124

RECEIVED AUGUST 1974; REVISED MARCH 1975