FINANCIAL PLANNING WITH 0-1 KNAPSACK PROBLEMS, PART II: USING DOMINATION RESULTS TO SOLVE KNAPSACK PROBLEMS

Daniel E. O'Leary

ABSTRACT

Part II uses domination results in knapsack problems to assist in generating solutions more rapidly and with less work. In particular, domination results are used to: (1) generate smaller problems so that less work is needed to solve the problems, and (2) speed up solution by pruning the variable sets that must be examined. Three solution processes are examined in Part II. Domination results are used on branch and bound algorithms, dynamic programming algorithm and epsilon-approximate algorithms. Use of domination concepts can substantially reduce the computational demands of a solution process.

Advances in Mathematical Programming and Financial Planning,
Volume 4, pages 151-169.

ISBN: 1-55938-724-6

VI. APPLICATION TO BRANCH AND BOUND ALGORITHMS

The domination results can be used to improve the branch and bound algorithms of Kolesar (1967) and Greenberg and Hegerich (1970), and other similar algorithms. This section extends the domination results, illustrates their use and provides some empirical evidence which illustrates the impact of incorporating the domination results in those algorithms.

A. Background

A number of branch and bound algorithms have been presented for the solution of the 0-1 knapsack problem. Perhaps the first branch and bound algorithm was that of Kolesar (1967), who sequentially branched on each variable, x_1, x_2, and so on. For each of the two branches coming out of a given node, the variable was set to 0 or 1. The upper bound was then examined to determine whether or not it exceeded the best known solution up to that branch in the tree. If it did then branching continued. If not branching stopped. Greenberg and Hegerich (1970) also developed a similar branch and bound algorithm to Problem (1). However, instead of branching on the variables in sequence, their approach was to branch on the one variable that was noninteger in the linear programming solution to Problem (1).

In these and other algorithms, the upper bound has been determined using a linear programming solution to Problem (1). In that computation, a linear programming approach is used to solve the problem that results from the remaining variables that have not been set equal to 0 or 1 in the branching and bound tree.

B. Revisions Due to Domination

Each of those algorithms (e.g., Kolesar, 1967 and Greenberg and Hegerich, 1970) can be improved by using the reduced problem of Theorem 7 instead of the original Problem (1). In addition, those algorithms can make use of the domination results in another fashion. Let

$$K^1 = \{i \mid x_1 \text{ is set at 1 at the node of the search tree under consideration}\}$$

$$K^0 = \{i \mid x_i \text{ is set at 0 at the node of the search tree under consideration}\}$$

$$K^2 = \{i \mid i \notin (K^1 \cup K^0 \cup I^1 \cup I^0\}$$

The sets K^1 and K^0 can be increased in size by using the following observation (A) and then observation (B). These additional branching rules, if used in conjunction with the algorithms, can substantially prune a search tree.

Observation (A)—Impact of Domination

Consider some nodes of the search tree.

a. If $j \in K^1$ then for $i \in \gamma_j$ let $i \in K^1$ at that node of the search tree and all nodes emanating from that node.

b. If $j \in K^0$ then for $i \in \Delta_j$ let $i \in K_0$ at that node of the search tree and all nodes emanating from that node.

Observation (B)—Setting Variables to 0 or 1

a. If for $j \in K^2$ at some node of the search tree

$$b \geq \sum_{i \in (K^1 \cup I^1)} a_i + \sum_{\substack{i \notin \Delta_j \\ i \in K^2}} a_i$$

then $j \in K^1$ at that node of the search tree and all nodes emanating from that node.

b. If for $j \in K^2$ at some node of the search tree

$$b < a_j + \sum_{i \in (K^1 \cup I^1)} a_i + \sum_{i \in \gamma_j} a_i$$

then $j \in K^0$ at that node and all nodes emanating from that node.

Observation C—Bounding

In terms of the above notation, both the above mentioned branch and bound algorithms use the linear programming solution to the following problem,

$$\max \sum_{i \in K^2} c_i x_i + \sum_{i \in K^1} c_i$$

$$\sum_{i \in K^2} a_i x_i \leq b - \sum_{i \in K^1} a_i$$

$$1 \geq x_i \geq 0, i \in K^2$$

as a means of establishing the upper bound at a given node of the search tree. Using observations (A) and (B), that bound can be improved by using the linear programming solution to the following problem as the upper bound.

$$\max \sum_{i \in (K^2 \cap I^2)} c_i x_i + \sum_{i \in (K^1 \cup I^1)} c_i$$

$$\sum_{i \in (K^2 \cap I^2)} a_i x_i \leq b - \sum_{i \in (K^1 \cup I^1)} a_i$$

$x_i \geq x_j$ If $i \in (K^2 \cap I^2)$ and $x_i \rightarrow x_j$ is in the reduced domination network

$$1 \geq x_i \geq 0.$$

C. Example

Consider the example in Kolesar, 1967 and Greenberg and Hegerich, 1970,

$$\max 60x_1 + 60x_2 + 40x_3 + 10x_4 + 20x_5 + 10x_6 + 3x_7$$

$$100 \geq 30x_1 + 50x_2 + 40x_3 + 10x_4 + 40x_5 + 30x_6 + 10x_7$$

$$x_i = 0 \text{ or } 1.$$

The original algorithm by Kolesar, (1967), used in this example, results in a branch and bound tree of 14 arcs, as seen in Figure 4. However, by incorporating the approach suggested by observations (A) and (B), that same problem and algorithm results in a branch and bound tree of 4 arcs, as seen in Figure 5, a substantial decrease in the amount of branching that must be done. In the first branch, by setting $x_1 = 0$, it is seen that because of the domination results x_2, x_3, x_5 and x_6 must also be 0, thus, eliminating the need to branch further in that direction. By setting $x_1 = 1$, and $x_2 = 1$, the optimal solution is found. By setting $x_2 = 0$ it is found by domination results that the solution is not optimal.

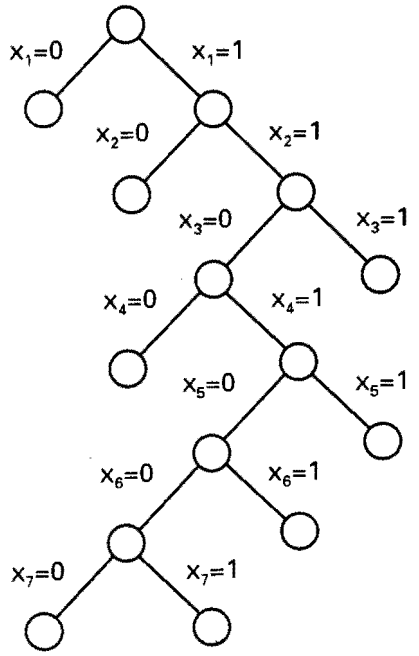

Figure 4. Branch and Bound Tree—Kolesar Algorithm

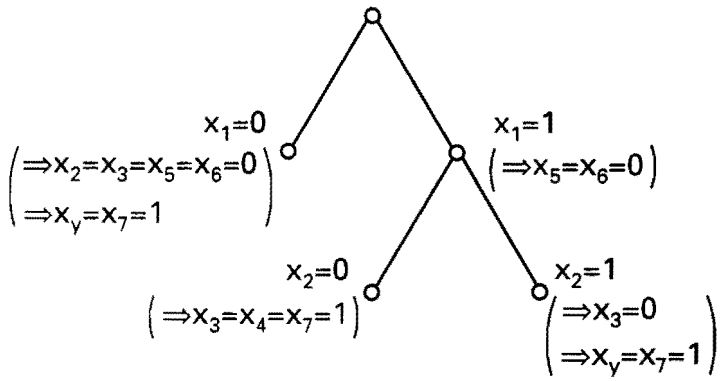

Figure 5. Branch and Bound Tree—Revised Kolesar Algorithm

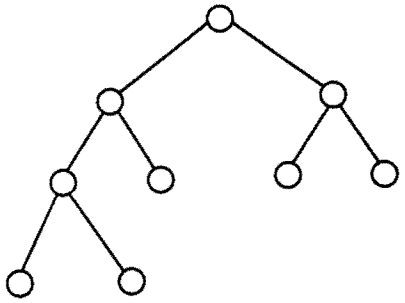

Figure 6. Branch and Bound Tree—Greenberg and Hegerich Algorithm

Similarly, the algorithm of Greenberg and Heinrich (1970) results in a branch and bound tree of 8 arcs, as seen in Figure 6. However, by incorporating the approach suggested by observations (A) and (B), that same example and algorithm results in a branch and bound tree of 2 arcs, as seen in Figure 7.

D. Empirical Tests

Empirical tests were used to study the impact of the reduced problem size and the suggested revisions to the algorithms. Kolesar's (1967) and Greenberg and Hegerich's (1970) algorithms were programmed. In addition, a revised version of Kolesar's (1967) algorithm was developed to account for observations (A), (B) and (C).

Twenty problems for each n, n = 10, 20, 30 and 40 were generated, with the c_i and a_i uniformly distributed between 0 and 1. B values were chosen as a function of the sum of the a_i's, where, $b = (\alpha * \Sigma a_i)$.

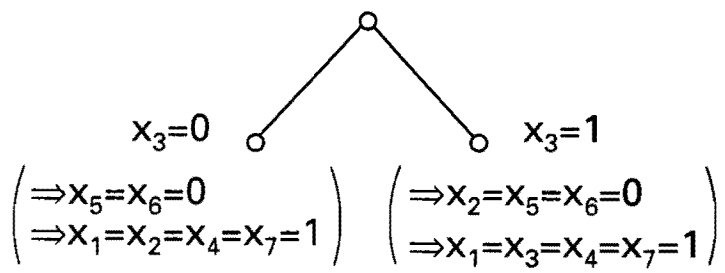

Figure 7. Branch and Bound Algorithm—Revised Greenberg and Hegerich Algorithm

Table 2. Average Size of Reduced Problem

		n			
		10	*20*	*30*	*40*
	.9	3.00	5.85	7.40	9.35
	.7	5.00	10.25	14.05	18.75
α	.5	6.30	12.85	18.65	24.90
	.3	5.90	12.80	18.45	25.65
	.1	2.85	8.45	13.30	18.35

Each problem was solved using both the original algorithms. After the problems were reduced in size using the domination results, each problem was solved using the same algorithm.

The average size of the reduced problems is contained in Table 2. As in Part I of the paper the results were substantial, with the greatest reduction occurring at the extreme values of $\alpha = .9$ and .1.

Each of the original and reduced problems was solved using the Kolesar (1967), see Table 3, and Greenberg and Hegerich (1970), see Table 4, algorithms. The reduced problems required far fewer branches than the original problems. For example, in the case of $\alpha = .1$ and $n = 10$, the number of branches required to solve the reduced problem was

Table 3. Average Number of Nodes in Search Tree Kolesar's Algorithm

		n			
		10	*20*	*30*	*40*
	.9	7.1[1]	16.1	19.9	26.1
		21.2[2]	47.6	68.2	98.6
	.7	15.1	35.8	52.9	94.6
		26.9	61.8	90.0	146.4
α	.5	25.0	54.6	83.9	159.2
		36.7	76.9	117.9	217.0
	.3	21.7	70.3	87.0	146.0
		41.0	112.4	129.3	206.6
	.1	5.4	28.1	59.6	92.3
		25.0	77.9	142.1	191.1

Notes: 1. Average search tree size of reduced problems.
2. Average search tree size of original problems.

Table 4. Average Number of Nodes in Search Tree Greenberg and Hegerich Algorithm

			n			
		N				
	X		*10*	*20*	*30*	*40*
	.9		4.2[1]	7.8	12.2	13.4
			6.3[2]	12.0	19.5	24.8
	.7		9.4	27.6	39.5	88.0
			13.7	41.6	52.0	115.6
α	.5		17.6	37.1	47.5	110.4
			25.6	48.6	61.8	137.4
	.3		14.9	53.2	69.5	118.6
			28.3	78.3	100.3	160.8
	.1		4.5	20.5	47.8	96.0
			19.8	56.8	104.1	195.5

Notes: 1. Average search tree size of reduced problems.
2. Average search tree size of original problems.

slightly more than 20% of the number of branches needed to solve the original problems.

The value of α seems to influence the number of branches required, particularly for the reduced problems. For example, for the Kolesar (1967) algorithm, the use of the original problem typically requires two to four times as many branches for $\alpha = .1$ or .9. However, for $\alpha = .5$, use of the original problem requires about 1.5 times as many branches.

The reduced problems seem to be a bit more difficult to solve than the original problems. Consider first Kolesar's (1967) algorithm. The average size of the reduced problems with $n = 20$ and $\alpha = .7$ was about 10. In that case the average number of branches was 35.8 compared to 26.9 of the problems where $n = 10$ and $\alpha = .7$. Similarly, the reduced problem size where $n = 40$ and $\alpha = .9$, was 9.35. In that case, the average number of branches was 26.1 compared to the 21.2 in those problems where $n = 10$ and $\alpha = .9$. Although they are more difficult to solve, the solution of the reduced problem always took fewer branches, in the case of both algorithms.

Next consider Greenberg and Hegerich's (1970) algorithm. For the case $n = 20$ and $\alpha = .7$ there were reduced problem sizes of about 10 and an average of 27.6 branches in the tree. For the comparative case, $n = 10$ and $\alpha = .7$, there was an average of 13.7 branches. Similarly, the reduced problem size where $n = 40$ and $\alpha = .9$ was 9.35 and there were an average

Table 5. Average Number of Nodes in Search Tree Revised Kolesar Algorithm

		n		
		10	*20*	*30*
	.9	6.5[1]	14.4	17.2
		20.6[2]	45.0	62.8
	.7	13.8	28.7	42.4
		24.0	48.6	74.4
α	.5	18.5	39.8	59.5
		24.0	52.2	78.3
	.3	19.2	54.1	64.0
		24.2	61.6	73.0
	.1	5.4	24.3	44.8
		12.6	30.3	48.9

Notes: 1. Average search tree size of reduced problems.
2. Average search tree size of original problems.

of 13.4 branches per tree. For comparative case of $n = 10$ and $\alpha = .9$, the average number of branches was 6.3.

Finally, in order to test the impact of observations (A), (B) and (C) on the branch and bound process, a revised version of Kolesar's algorithm was generated and used to solve the same problems. The results of those tests are summarized in Table 5.

When the results of Table 5 are compared with the results in Table 3, it can be seen that there are substantial differences that accrue with the use of domination to reduce the problem and assist in its solution. For example, for $n = 30$ and alpha = .3, the original algorithm required about 129 branches on the original problem and 87 on the reduced problem. The revised algorithm required 73 on the original problem and 64 on the reduced problem. By coupling problem reduction and accounting for domination in branching and bounding important increases in efficiency can be attained.

VII. APPLICATION TO DYNAMIC PROGRAMMING SOLUTIONS

The domination results also have some implications for dynamic programming. Theorem 5 can be used to reduce the dimensionality. Theorem 6 can be used to reduce the dimensionality and the state space. As a result, the domination results can be used to reduce the computations required if dynamic programming is used to solve Problem (1). In addition, the domination results can be embedded in dynamic programming ap-

proaches. This section develops those results and illustrates them in the example.

A. Impact of Domination Results

Let $f_i(y)$ be the optimal solution to

$$f_i(y) = \max \sum_{j=1}^{i} c_j x_j$$

$$y \geq \sum_{j=1}^{i} a_j x_j,$$

$$x_i = 0 \text{ or } 1,$$

where $f_0(y) = 0$ for all $y \leq b$ (in the reduced problem $f_0(y) = \Sigma_{i \in I'} c_i$ and $y \leq b - \Sigma_{i \in I'} a_i$. Theorems 5 and 8 can be restated to account for the structure of dynamic programming.

Theorem 5′. $x_i = 0$ in some optimal solution for $y < a_i + \sum_{j \in \gamma_i} a_j$.

Theorem 8′. For all y,

$$y < a_i + \sum_{j \in \gamma_i} a_j$$

$$f_i(y) = f_{i-1}(y)$$

in some optimal solution to Problem (1′).

A number of recursions are available for finding $f_i(y)$ (e.g., Weingartner and Ness, 1967 and Morin and Marsten, 1974). Each of those recursions (and others) can be structured to account for Theorems 5′ and 8′.

B. Example

Consider the example in Kolesar (1967),

$$\max 60x_1 + 60x_2 + 40x_3 + 10x_4 + 20x_5 + 10x_6 + 3x_7$$

$$100 \geq 30x_1 + 50x_2 + 40x_3 + 10x_4 + 40x_5 + 30x_6 + 10x_7$$

$$x_i = 0 \text{ or } 1.$$

Table 6. Dynamic Programming Solution of Non-Reduced Problem

$f_0(0) = 0$					
$f_0(1) = 0$				$f_4(1) = 10$	
$f_0(2) = 0$				$f_4(2) = 10$	$f_7(2) = 13$
$f_0(3) = 0$	$f_1(3) = 60$			$f_4(3) = 60$	$f_7(3) = 60$
$f_0(4) = 0$	$f_1(4) = 60$			$f_4(4) = 70$	$f_7(4) = 70$
$f_0(5) = 0$	$f_1(5) = 60$			$f_4(5) = 70$	$f_7(5) = 73$
$f_0(6) = 0$	$f_1(6) = 60$			$f_4(6) = 70$	$f_7(6) = 73$
$f_0(7) = 0$	$f_1(7) = 60$		$f_3(7) = 100$	$f_4(7) = 100$	$f_7(7) = 100$
$f_0(8) = 0$	$f_1(8) = 60$	$f_2(8) = 120$	$f_3(8) = 120$	$f_4(8) = 120$	$f_7(8) = 120$
$f_0(9) = 0$	$f_1(9) = 60$	$f_2(9) = 120$	$f_3(9) = 120$	$f_4(9) = 130$	$f_7(9) = 130$
$f_0(10) = 0$	$f_1(10) = 60$	$f_2(10) = 120$	$f_3(10) = 120$	$f_4(10) = 130$	$f_7(10) = 133$

Table 7. Dynamic Programming Solution of Reduced Problem

$f_0(0) = 60$				
$f_0(1) = 60$			$f_4(1) = 70$	
$f_0(2) = 60$			$f_4(2) = 70$	$f_7(2) = 73$
$f_0(3) = 60$			$f_4(3) = 70$	$f_7(3) = 73$
$f_0(4) = 60$		$f_3(4) = 100$	$f_4(4) = 100$	$f_7(4) = 100$
$f_0(5) = 60$	$f_2(5) = 120$	$f_3(5) = 120$	$f_4(5) = 130$	$f_7(5) = 120$
$f_0(6) = 60$	$f_2(6) = 120$	$f_3(6) = 120$	$f_4(6) = 130$	$f_7(6) = 130$
$f_0(7) = 60$	$f_2(7) = 120$	$f_3(7) = 120$	$f_4(7) = 130$	$f_7(7) = 133$

Table 8. Embedded State Space and Solution Domination Programming Solution

$f_1(0) = 0$	$f_2(0) = 0$	$f_3(0) = 0$	$f_4(0) = 0$	$f_5(0) = 0$	$f_6(0) = 0$	$f_7(0) = 0$
$f_1(3) = 60$	$f_2(3) = 60$	$f_3(3) = 60$	$f_4(1) = 10$	$f_5(1) = 10$	$f_6(1) = 10$	$f_7(1) = 10$
	$f_2(5) = 60$	$f_3(4) = 40$	$f_4(3) = 60$	$f_5(3) = 60$	$f_6(3) = 60$	$f_7(2) = 13$
	$f_2(8) = 120$	$f_3(7) = 100$	$f_4(4) = 70$	$f_5(4) = 70$	$f_6(4) = 70$	$f_7(3) = 60$
	$f_2(3) \rightarrow f_2(5))$	$f_3(8) = 120$	$f_4(7) = 100$	$f_5(7) = 100$	$f_6(7) = 100$	$f_7(4) = 70$
	$f_2(5)$ need not	$(f_3(3) \rightarrow f_3(4))$	$f_4(8) = 120$	$f_5(8) = 120$	$f_6(8) = 120$	$f_7(5) = 73$
	be stored	$f_3(4)$ need not	$f_4(9) = 130$	$f_5(9) = 130$	$f_6(9) = 130$	$f_7(7) = 100$
		be stored				$f_7(8) = 120$
						$f_7(9) = 130$
						$f_7(10) = 133$

Table 9. Embedded State Space and Variable Domination Dynamic Programming Solution

$f_1(0) = 0$	$f_2(0) = 0$	$f_3(0) = 0$	$f_4(0) = 0$	$f_7(0) = 0$
$f_1(3) = 60$	$f_2(8) = 120$	$f_3(7) = 100$	$f_4(1) = 10$	$f_7(2) = 13$
	$f_2(3) = 120$	$f_3(8) = 120$	$f_4(3) = 60$	$f_7(3) = 60$
	need only	$f_3(3) = 60$	$f_4(4) = 70$	$f_7(4) = 70$
	be stored	need only	$f_4(7) = 100$	$f_7(5) = 73$
		be stored	$f_4(8) = 120$	$f_7(7) = 100$
			$f_4(9) = 130$	$f_7(8) = 120$
				$f_7(9) = 130$
				$f_7(10) = 133$
				$f_7(1) = 10$
				need only
				be stored

The constraint can be written

$$10 \geq 3x_1 + 5x_2 + 4x_3 + 1x_4 + 4x_5 + 3x_6 + 1x_7.$$

The calculations that are required, given Theorem 8, for the full example are given in Table 6. For the reduced problem discussed in Section II, the results are given in Table 7. The blank area is the area in which no computations were required $i \geq 1$. In the first case a 56% decrease in the usage of $f_i(y)$ was achieved. In the second case, the reduced problem, a 43% decrease was achieved.

If the embedded state space approach is used in conjunction with solution domination concepts (see Morin and Marsten, 1974) the example is solved as in Table 8. Finally, if the embedded state space approach is used in conjunction with Theorem 8 the example is solved as in Table 9.

In each case the use of the reduced problem and revising the algorithm based on Theorems 5′ and 8′ yield approaches with substantially fewer computations.

VIII. APPLICATION TO AN ϵ-APPROXIMATE ALGORITHM

The domination results can be used in conjunction with the ϵ-approximate algorithm of Sahni (1975). Approximate algorithms can be used when optimal solutions are not required. After Theorems 5 and 6 are applied, both the dimensionality and state space are reduced. Thus,

Sahni's (1975) algorithm should be used on the reduced problem of Theorem 7. In addition, that algorithm can be altered to take into account the domination relationships.

C. Background: ϵ-Approximate Algorithm

Let C = $(c_1, c_2, ..., c_n)$ and A = $(a_1, a_2, ..., a_n)$. The ϵ-approximate algorithm of Sahni (1975, p. 117) without concern for domination results can be stated as follows.

Algorithm S (ϵ-Approximate)

Define the following terms
(1) The size of a combination is the number of objects in it; (2) the weight of a combination is the sum of the weights of the objects in that combination; (3) k is a non-negative integer which defines the order of the algorithm.

1. PMAX = 0
2. For all combinations I of size $\leq k$ and weight $\leq b$ do

$$c_I = \sum_{i \in I} c_i$$

PMAX = max {PMAX, c_I + L(I, C, A, b)
end
where Algorithm L(I, C, A, b) is as follows.

Algorithm L(I, C, A, b)

1. $L = 0; i = 1; w = b - \sum_{i \in I} a_i$

2. If $i \notin$ I and $a_i \leq w$ then do

$$L = L + c_i$$

$$W = W - a_i$$

end
3. $i = i + 1$; if $i \leq n$ then go to 2.
4. Return (L); end.

D. Accounting for Domination Results

A revised version of Algorithm S which does take into account domination relationships can be stated as follows.

Algorithm S'

1. PMAX = 0
2. For all combinations I of size k and weight $\leq b$, do 3.
3. If $\sum_{j \in I} a_j + \sum_{\substack{i \in (\cap \gamma_j) \\ j \in I}} a_i > b$

Go to 2. Otherwise do

$$i \in \text{I for } i \in (\cap \gamma_j), j \in \text{I}$$

$$c_I = \sum_{i \in I} c_i$$

PMAX = max {PMAX, c_I + L(I, C, A, b)}
end.

The algorithm S' will make no more calls for L(I, C, A, b) than algorithm S and possibly substantially fewer. In addition, algorithm S' brings to light a criterion for stopping when no solutions larger than PMAX exist. This criterion is given in the following theorem.

Theorem 9. If for some k and all combinations I of size k,

$$\sum_{j \in I} a_j + \sum_{\substack{i \in \gamma_j \\ j \in I}} a_i > b,$$

then PMAX is optimal.

E. Relationship to Greedy Algorithms

The solution from L(∅, C, A, b) is referred to as the greedy solution (see Magazine, et al., 1975). The domination results are closely related to the greedy solution.

Let

$$K_0 = \{i \mid x_i = 0 \text{ in } L(\varnothing, C, A, b)$$

$$K_1 = \{i \mid x_i = 1 \text{ in } L(\varnothing, C, A, b)$$

$$I^0 = \{i \mid x_i = 0 \text{ by Theorem 2 domination results}\}$$

$$I^1 = \{i \mid x_i = 1 \text{ by Theorem 2 domination results}\}.$$

The following observations can be made about the relationship between these sets.

Observation (a)

$I^0 \subseteq K^0$ and $I^1 \subseteq K_1$. Thus, the ϵ-approximate property of algorithm S is preserved in algorithm S′.

Observation (b)

If for each $i \in K_1$ and $i \in \gamma_j$ for each $j \in K_0$ then $L(\varnothing, C, A, b)$ is optimal.

Observation (b′)

If $i \in \gamma_{i+1}$, $i = 1, 2, \ldots, n-1$ then $L(\varnothing, C, A, b)$ is optimal.

Observation (c)

Suppose $L(\varnothing, C, A, b)$ is not optimal then there exists $i \in K_1$ and $j \in K_0$ such that $i \notin \gamma_j$.

Table 10. Probability of Optimality ($n = 2$)

0 - Change

m / *b*	*2*	*3*	*4*	*5*	*6*
1	1	1	1	1	1
2	1	.977778	.985294	.987692	.990991
3	1	.955556	.963235	.972308	.98048
4	1	1	.941176	.953846	.965465
5		1	.985294	.929231	.945946
6		1	1	.969231	.924925
7			1	.990769	.962462
8			1	1	.984985
9				1	.995495
10				1	1
11					1
12					1

Table 11. Probability of Optimality ($n = 3$)

	0 - Change			*1 - Change*		
b \ m	2	3	4	2	3	4
1	1	1	1	1	1	1
					1	1
2	1	.963636	.971814	1	1	1
					1	1
3	1	.957576	.942402	1	1	1
					1	1
4	1	.957576	.92402	1	1	1
					.993939	.997549
5	1	.981818	.946078	1	1	.998775
					.993939	.996324
6	1	.981818	.958333	1	1	.998775
					.987879	.988971
7		1	.974265		1	.997549
					1	.988971
8		1	.981618		1	1
					1	.987745
9		1	.996324		1	1
					1	.997549
10			1			1
						1
11			1			1
						1
12			1			1
						1

Observation (d)

If for each $i \in K_1$ and for some $j \in K_0$, $i \in \gamma_j$ then $L(\varnothing, C, A, b)$ cannot be improved by only placing j in K_1.

D. An Enumeration Analysis of the Probability of Optimality

One approach to determining the quality of the Sahni approximation algorithm, with and without the use of the domination reduction algorithm, is to enumerate all possible knapsack problems with integer coefficients (between 1 and m) for a particular n and solve the problems using both an algorithm guaranteed to provide an optimal solution and the Sahni approximation algorithm. For small values of n and m this

approach can be used to enumerate a probability of optimality using Sahni's approach.

This was done by allowing the a_i and c_i to take integer values between 1 and m, for m an integer. All possible problems were elicited and solved. The results are summarized in Tables 10–13. For each triple (b, n and m) there are two probabilities. The first probability is the probability of optimality when domination results of Theorem 2 are used to reduce the problem size. The second probability comes from using the original problem. If there is only one number, as is the case with 0-changes, then there is no difference.

The results in Tables 10–13 provide a number of findings and hypotheses. First, the problem has not been reduced to account for domination, so an 0-change is optimal for $b = 1, (m^*n–2), (m^*n–1), (m^*n)$; a j-change is optimal for $b = 1, 2, ..., 2^*j + 1$, and $(m^*n–2), (m^*n–1), (m^*n)$.

Table 12. Probability of Optimality ($n = 4$)

	0 - Change		*1 - Change*		*2 - Change*	
b \ m	2	3	2	3	2	3
1	1	1	1	1	1	1
				1	1	1
2	1	.957576	1	1	1	1
				1	1	1
3	1	.957576	1	1	1	1
				1	1	1
4	1	.947475	1	1	1	1
				.987879	1	1
5	1	.959596	1	1	1	1
				.989899	1	1
6	1	.963636	1	1	1	1
				.981818	1	.99798
7	1	.973737	1	1	1	1
				.987879	1	.99798
8	1	.989899	1	1	1	1
				.993939	1	.99798
9		.991919	1	1		1
				.993939		.99596
10		1		1		1
				1		1
11		1		1		1
				1		1
12		1		1		1
				1		1

Table 13. Probability of Optimality ($n = 5$)

	0 - Change		1 - Change		2 - Change	
b \ m	2	3	2	3	2	3
1	1	1	1	1	1	1
				1		1
2	1	.956488	1	1	1	1
				1		1
3	1	.956488	1	1	1	1
				1		1
4	1	.941725	1	1	1	1
				.983683		1
5	1	.954157	1	1	1	1
				.986014		1
6	1	.949495	1	1	1	1
				.978244	.995338	
7	1	.958819	1	1	1	1
				.982906	.996115	
8	1	.966589	1	1	1	1
				.983683		.994561
9	1	.97669	1	1	1	1
				.986791		.993784
10	1	.985237	1	1	1	1
				.990676		.995338
11		.994561		1		1
				.996115		.997669
12		.996115		1		1
				.996892		.997669
13		1		1		1
				1		1
14		1		1		1
				1		1
15		1		1		1
				1		1

Second, reduction does not affect the 0-change.

Third, since the problem has been reduced using Theorem 2 domination results:

- an 0-change is optimal for $m = 2$ (for $n = 2, ..., 5$)
- a 1-change is optimal for $m = 3$ (for $n = 2, ..., 5$)

This suggests that, since the problem has been reduced to account for domination results, an r-change is optimal for $m = r + 2$ for all n.

Fourth, since the problem has been reduced to account for domination results, a j-change is optimal for $j = 1, ..., 2^*j + 1$.

IX. SUMMARY AND EXTENSIONS

Part I of this paper developed the notions of domination in 0-1 knapsack problems. Part II of this paper has used the notions of domination in two distinct ways. First, it studied the impact of reducing the problem size. Second, key notions of domination for 0-1 knapsack problems were embedded in different algorithms and used to improve the speed with which a solution could be found.

This paper could be extended in a number of different ways. First, the domination results can be extended to other types of solution processes. The branch and bound algorithms and dynamic programming approaches provided here were for illustration purposes. Second, the results could be expanded to other problem types. For example, results could be developed for the bounded variable knapsack problem (e.g., Bruckner et al., 1975) and the multi-dimensional knapsack problems (e.g., Weingartner and Ness, 1967).

REFERENCES

Greenberg, H., and R. L. Hegerich, "A Branch and Search Algorithm for the Knapsack Problem," *Management Science*, Vol. 16, No. 5 (1970).

Kolesar, P., "A Branch and Bound Algorithm for the Knapsack Problem," *Management Science*, Vol. 13, No. 9 (1967).

Magazine, M. J., G. L. Nemhauser, , and L. E. Trotter, "When the Greedy Solution Solves a Class of Knapsack Problems," *Operations Research*, Vol. 23, No. 2 (1975).

Morin, T., and R. E. Marsten, "Branch and Bound Strategies for Dynamic Programming," September 1974, Discussion Paper 106, Center for Mathematical Studies in Economics and Management Science, Northwestern University.

Sahni, S., "Approximate Algorithms for the 0/1 Knapsack Problem," *Journal of the Association of Computing Machinery*, Vol. 22, No. 1 (1975).

Weingartner, H. M., and D. N. Ness, "Methods of Solutions of the Multidimensional 0/1 Knapsack Problem," *Operations Research*, Vol. 15, No. 1 (1967).

Weingartner, H. M., *Mathematical Programming and the Analysis of Capital Budgeting Problems*. Englewood Cliffs, NJ, Prentice-Hall, 1962.