รายงานวิจัยฉบับสมบูรณ์

โครงการ การแก้ปัญหาการตัดหนึ่งมิติกรณีที่ความต้องการไม่แน่นอน

Solving Stochastic One-Dimensional Cutting Stock Problem with Discrete Random Demands

โดย ผู้ช่วยศาสตราจารย์ ดร. ศิริรัตน์ วงศ์ประกรณ์กุล

เดือน เมษายน พ.ศ. 2553

สัญญาเลขที่ MRG5180209

รายงานวิจัยฉบับสมบูรณ์

โครงการ การแก้ปัญหาการตัดหนึ่งมิติกรณีที่ความต้องการไม่แน่นอน
Solving Stochastic One-Dimensional Cutting Stock Problem with Discrete Random Demands

ผศ.ดร.ศิริรัตน์ วงศ์ประกรณ์กุล มหาวิทยาลัยขอนแก่น

สนับสนุนโดยสำนักงานกองทุนสนับสนุนการวิจัย
(ความเห็นในรายงานนี้เป็นของผู้วิจัย สกว.ไม่จำเป็นต้องเห็นด้วยเสมอไป)

Abstract

Project Code: **MRG5180209**

Project Title: **Solving Stochastic One-Dimensional Cutting Stock Problem with Discrete Random Demands**

Investigator: **Asst. Prof. Sirirat Wongprakornkul**
Khon Kaen University
E-mail Address: sirwon@kku.ac.th

Project Period: **2 years**

Problem statement: One-Dimensional Cutting Stock Problem with discrete demands and capacitated planning objective is an NP hard problem. **Objectives:** The objectives of this study are 1) to propose a mathematical programming with column-generation technique for solving the stochastic one-dimensional cutting stock problem with discrete random demands, 2) to propose a heuristic for solving the stochastic one-dimensional cutting stock problem with discrete random demands, and 3) comparing two algorithms in computation times, and upperbound value. **Approach:** The mathematical model with column-generation technique by a branch-and-bound procedure and the heuristic based on the first fit decreasing method are proposed. Then, both approaches were compared and some characteristics were investigated such as upperbound value, percentage above lowerbound value, computation time, and number of patterns. **Results:** The 24 instances were examined. The proposed heuristic provides the upperbound value above the lowerbound around 0-13.16%. All upperbound values from column-generation and integer programming are better than the proposed heuristic but all computation times are higher. **Conclusion:** The proposed heuristic has consistently high performance in computation times. This particular information is worth for making decision to closely meet the appropriate planning of the One-Dimensional Cutting Stock Problem with discrete demands. **Next research:** A heuristic based on random fit decreasing method is interested to study in next step.

Key words: stochastic integer linear programming, cutting stock problem, column-generation technique, heuristics

ลงนาม .....
(ผศ.ดร.ศิริรัตน์ วงศ์ประกรณ์กุล)
ผู้วิจัย

ลงนาม .....
(รศ.ดร.พีรยุทธ ชาญเศรษฐิกุล)
ที่ปรึกษา

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SOLVING ONE-DIMENSIONAL CUTTING STOCK PROBLEM WITH DISCRETE DEMANDS AND CAPACITATED PLANNING OBJECTIVE

INTRODUCTION

More than four decades ago, mathematicians studied intensively and developed techniques involved with large-scale problems. The extensive work on large-scale mathematical programming has been initiated by (Dantzig and Wolfe, 1961). One of classical NP-hard problems which could not be solved within the polynomial computation time is a cutting stock problem (CSP). The finite number of cutting patterns may be very large. Gilmore and Gomory (1961, 1963) claimed that if the stock to be cut has length l = 200 inches and there are demands for 40 different lengths from 20 to 80 inches, the number of possible patterns could be exceed 10 to 100 million. The more cutting patterns and the more columns of constraints are involved. The most powerful algorithm for solving linear programs with many columns is the Column-Generation Procedure. To apply in the real world problem, uncertainty is inherent in many real combinatorial problems. Uncertainty seems to be an important issue unconstraint programming. The Stochastic CSP (SCSP) is a framework that can be used to model combinatorial decision problems involving uncertainty and probabilities recently. Therefore, the number of scenarios depends on the number of random variables and the discrete random choices. The increasing of number of scenarios is tremendously such, 5 variables and 2 discrete demand choices have made 32 scenarios, 10 variables and 2 discrete demand choices have made 1,024 scenarios, but 20 variables and 2 discrete demand choices have made 1,048,576 scenarios. So the specific algorithms must be concerned for solving it.

In this research, we investigate the mathematical model of 1-dimensional cutting stock problem with discrete random demands (1D-SCSP) and propose the column-generation technique for searching the effective cutting patterns. The model is based on “here and now” assumption with minimizing capacitated planning objective. The objectives of this study are to propose a mathematical programming with column-generation technique for solving the stochastic with discrete demands and capacitated planning objective, to propose a heuristic based on the first fit decreasing method and to compare between the mathematical programming method and the proposed heuristic method. These approaches are designed beneath the low cost and within a reasonable time. And finally, the obtained information from the solutions is used to plan the process strategy with known discrete random demands.

OBJECTIVES

In order to design the production strategy of the 1-dimensional cutting stock problem with known discrete random demands (1D-SCSP), two approaches are proposed for solving particular problems. Thus, the objectives of this study are as follows:

1) To propose a mathematical programming with column-generation technique for solving the stochastic with discrete demands and capacitated planning objective.
2) To propose a heuristic based on the first fit decreasing method.
3) To compare between the mathematical programming method and the proposed heuristic method.

SCOPE

The assumptions of the 1D-SCSP are 1) the material costs are linear, 2) there is no initial inventory, and 3) all discrete demands are known. This problem could be formulated as a mixed integer linear programming model where special techniques accommodating for this structure are valid to apply for solving Column-Generation techniques. Moreover, the proposed heuristic based on first fit decreasing method of Bin Packing problem (Brandimarte and Villa, 1995) is proposed. Two performances are measured: the computation time and the above lowerbound percentage.

The next section is the literature review. In this section, the past research is discussed, related to this study on the approaches to solve large-scale problems. Then, two proposed approaches for solving the 1D-SCSP are mentioned in the materials and methods section. The algorithms for solving problems are described. After that the results of all experiments are in the results and discussion sections.

LITERATURE REVIEWS

Beraldi et al. (2009) developed a two-stage stochastic programming formulation for solving the CSP under uncertainty. The objective function was to minimize both setup cost and raw material cost, subjected to uncertainty customer demands, fitting to the given width length, and the limitation of maximum number of products that can be cut from one pattern. The random vector of customer demands was created as uncertain scenarios. The conclusion claimed that the stochastic solution was more robust and provided recommendations to the decision maker with added value and better quality than the expected demand solution.

Birge and Louveaux (1997) described that a stochastic programming is a framework for modeling optimization problems that involve uncertainty. Two basic models of stochastic program are chance constraint model and two-stage model. For two-stage program, the decision maker makes a decision in the first stage then the random effects occur. After that the decision maker can make a second-stage decision that compensates for any bad effects from the first-stage decision. For chance constraint model, it does not require that our decisions are feasible for every outcome of the random parameters, but require feasibility with at least some specified probability. Dantzig (1955) was the first, considered the stochastic program with recourse under the name two-stage linear programs under uncertainty.

Approaches for solving stochastic program with reducing scenarios can be performed as Monte Carlo sampling based-methods (Ermoliev, 1983), importance sampling method (Dantzig and Glynn, 1990) and stochastic decomposition methods (Higle and Sen, 1991).

Many algorithms were developed for solving stochastic program with recourse such as (Elmaghraby, 1959 and 1960), (Ziemba, 1970), (Garstka and Rutenberg, 1973), and (Audsley et al., 2000).

Elmaghraby (1959) presented that production problem with stochastic demand, when distribution function demand is discrete, can be transformed to be the linear programming problem. The objective function is minimizing production cost plus storage cost plus shortage cost. The objective function was proved to be a piecewise convex function. To transform the stochastic problem to be linear programming problem, the objective function of minimizing cost is changed to be minimizing partial differentiate of original objective function. In addition, the numerical examples of transforming problem were applied. Moreover, Elmaghraby (1960) proposed the particular problem when the demand is in continuous distribution.

Ziemba (1970) modified the convex simplex method for solving convex stochastic program with simple recourse.

Garstka and Rutenberg (1973) presented a procedure for solving discrete stochastic programs with recourse. It views the m stochastic elements of the right-hand side vector as an m-dimensional space in which each combination of the discrete values is a lattice point. For a given second stage basis, certain of lattice points are feasible. A procedure is presented to delete infeasible points from this m-dimensional

space until only feasible points remain. Thus, the probability associated with feasible lattice for this basis can be enumerated, and used to weight the vector of dual variables defined by the current optimal basis. Finally, they presented a systematic procedure for changing optimal basis so that a feasible and optimal basis is found for every lattice point.

Audsley et al. (2000) presented a two-stage stochastic programming with recourse model for the problem of determining optimal planting plans for a vegetable crop. The first stage of the model relates to finding a planting plan. The second stage is concerned with deriving a harvest schedule for each scenario. They solved this problem by using decomposition technique, and compared their solutions with the solutions from deterministic equivalent model, which is solved by using interior point method. Their solutions were greater expected profit than the deterministic model and also more robust.

Ermoliev (1983) presented a stochastic branch and bound method for solving stochastic discrete programming problem. In this method, stochastic lower and upper bounds are calculated and combined with branch and bound method. In order to find stochastic lower and upper bounds, they applied two general ideas which are interchange of minimization and mathematical expectation operators, and dual estimates. As a result, when using stochastic bounds, the optimal solution will be reached faster than using deterministic bounds. Mukai and Yan (1992), and Andradottir (1996) presented researches which were related to stochastic discrete programming. These researches used random search techniques which aim at finding a nearly-optimal solution.

MATERIALS AND METHODS

MATERIALS

There are two algorithms proposed in this study which are the method of column-generation technique and the proposed heuristic. Both algorithms were coded in the C++ programming and run on Pentium 4, 256 MB, 2.4 GHz. The column-generation procedure is linked with LINGO 6.0 solver.

METHODS

THE MATHEMATICAL PROGRAMMING WITH COLUMN-GENERATION TECHNIQUE

Consider the 1D-CSP with discrete demands and minimized capacitated planning objective. The following assumptions are made.

1. There is different length of items to be cut from a stock.
2. Each item has associated a certain length.
3. Each cutting pattern for a stock are not limited in the number of knives, but the sum of length of items are not exceed a length of stock.
4. The discrete demands are considered.
5. The cost of capacity planning such as raw material or stock, inventory and backorder costs are considered.

To describe the problem, we introduce the following notations:

k = index for stock lengths (k = 1, ..., K),

m = index for retail items (m = 1, ..., M),

L_k = the length of stock k; (k = 1, ..., K),

l_m = the length of item m; (m = 1, ..., M),

g_k = the cost of stock k,

R_k = amount of stock k available,

p = index for patterns,

P_k = the number of feasible patterns for stock k,

a_{mkp} = the number of strips of item m cut in pattern p for stock length k, fulfilling $\sum_{m=1}^{M} l_m a_{mkp} \le L_k$,

s = index for scenario of discrete demands (s=1,…,S),

F_{ms} = the probability of demand of item m at a scenario s, it equals $\prod_{h=1}^{H} f_{hm}$,

h = index for choices of discrete demands (h=1,…, H),

f_{hm} = the probability of choice h of item m,

I_{ms} = the inventory units of item m at a scenario s,

T_{ms} = the inventory costs of item m at a scenario s,

B_{ms} = the backorder units of item m at a scenario s,

A_{ms} = the backorder units of item m at a scenario s,

D_{ms} = the demand of item m at a scenario s,

x_{kp} = decision variables that represent the number of times pattern p of stock k is used.

Minimize

$$\sum_{k=1}^{K}\sum_{p=1}^{P_k} g_k x_{kp} + \sum_{\mathrm{m}=1}^{\mathrm{M}} \sum_{\mathrm{s}=1}^{\mathrm{S}} \mathrm{F_{ms}}(\mathrm{T_{ms}I_{ms}} + \mathrm{A_{ms}B_{ms}}) \quad (1)$$

Subject to

$$\sum_{p=1}^{P_k} x_{kp} \le R_k\,,\ \forall \mathrm{k}, \quad (2)$$

$$\sum_{k=1}^{K} \sum_{p=1}^{P_k} a_{mkp} x_{kp} - I_{ms} + B_{ms} = D_{ms}\,, \forall \mathrm{m}, \forall \mathrm{s}, \quad (3)$$

$$x_{kp} \ge 0\,,\ \forall \mathrm{k},\ \mathrm{p}. \quad (4)$$

For a large instance, P_k is extremely large. To reduce the number of P_k, we are not adding all feasible patterns in the model, but selecting some necessary patterns p* that maximized dual objective function. To determine $a_{mk^*p^*}$ we develop the DUAL model by the simplex multipliers obtained from (1)-(4) as follows:

e_k = the simplex multipliers of the used stock k from constraints (2),

r_{ms} = the simplex multipliers of the item m from constraints (3).

$a_{mk^*p^*}$ = the obtained variable that defined number of strips of item m cut in pattern p* for stock length k*,

q_k = the used stock length; $q_k \in \{0,1\}$,

if q_k = 1 then the stock length k is used. Otherwise, the stock length k is not used.

$$\text{Maximize } V = \sum_{k=1}^{K} (g_k - e_k) q_k - \sum_{s=1}^{S} \sum_{m=1}^{M} r_{ms} a_{mk^*p^*} \quad (5)$$

Subject to

$$\sum_{m=1}^{M} l_m a_{mk^*p^*} \leq \sum_{k=1}^{K} L_k q_k \,, \quad (6)$$

$$\sum_{k=1}^{K} q_k = 1, \quad (7)$$

$a_{mk^*p^*} \geq 0$ and integer, m = 1, ..., M, (8)

$q_k \geq 0$ and binary, k = 1,...,K (9)

The obtained output from (5) – (9) is a new legitimate cutting pattern and introduced into model (1)-(4) as a new basis column. The new cutting pattern is to be adding until the objective dual values V from (5) $\leq$ 0, the current column is proven to be globally linear programming optimal point. For a profound description of the column-generation (CG) procedure for cutting stock problems, see Lasdon (1970). To illustrate this CG procedure, a flow chart is shown in Fig. 1.

Since the obtained x_{kp} from globally linear programming optimal point usually are not integer therefore, the branch-and-bound procedure is applied for selecting integer values of x_{kp} by the same basis column. The obtained integer solution from the branch-and-bound procedure by the same basis column or necessary patterns are not providing the optimality that proven by solving all the feasible patterns, but it nearly the optimal value than the other heuristics.

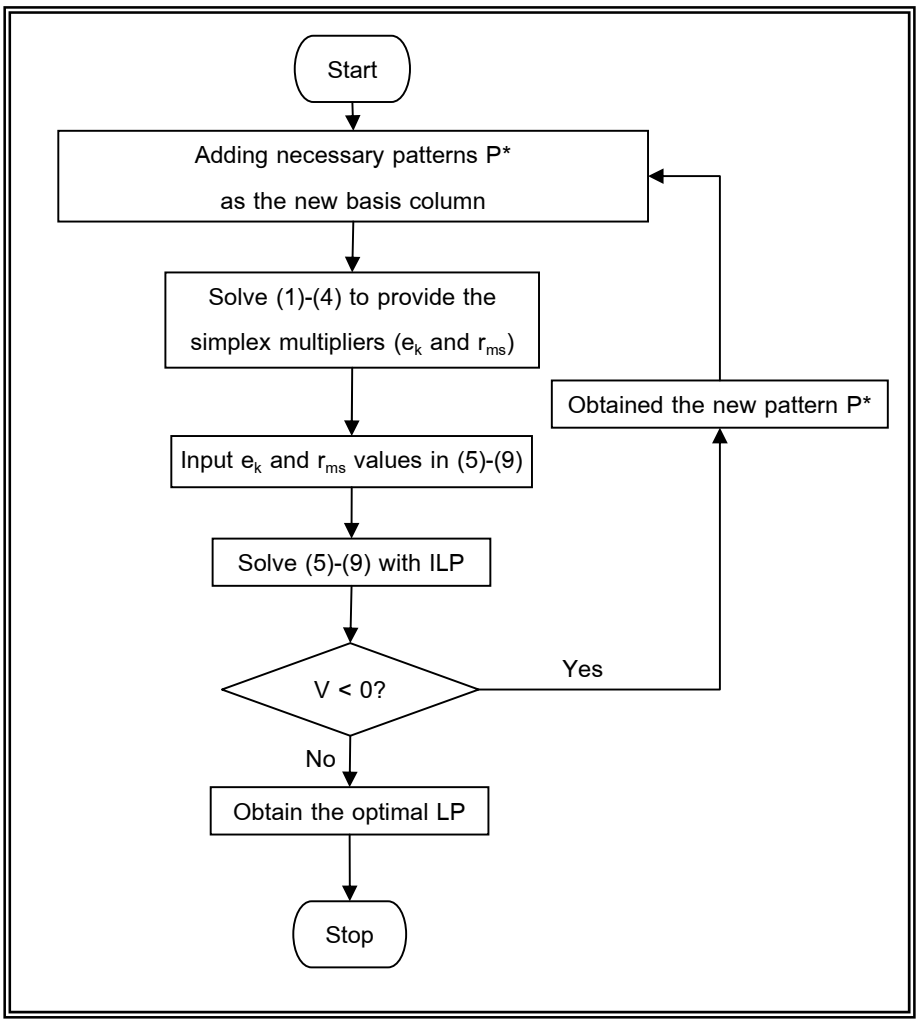


Figure 1. Flowchart of Column-Generation procedure for the problem

THE CONSTRUCTIVE HEURISTIC METHOD

Although the column-generation approach provides the nearly optimal solution, variables and constraints increase drastically when increasing the number of items and the chance of discrete demand. Therefore, the obtained solution from the column-generation approach is not always attainable within the allowable time. The proposed heuristic is based on first fit decreasing method of Bin Packing problem (Brandimarte and Villa, 1995). Items are sorted in such a way that the longer item is selected before others. The outline of the algorithm has been shown in Fig. 2.

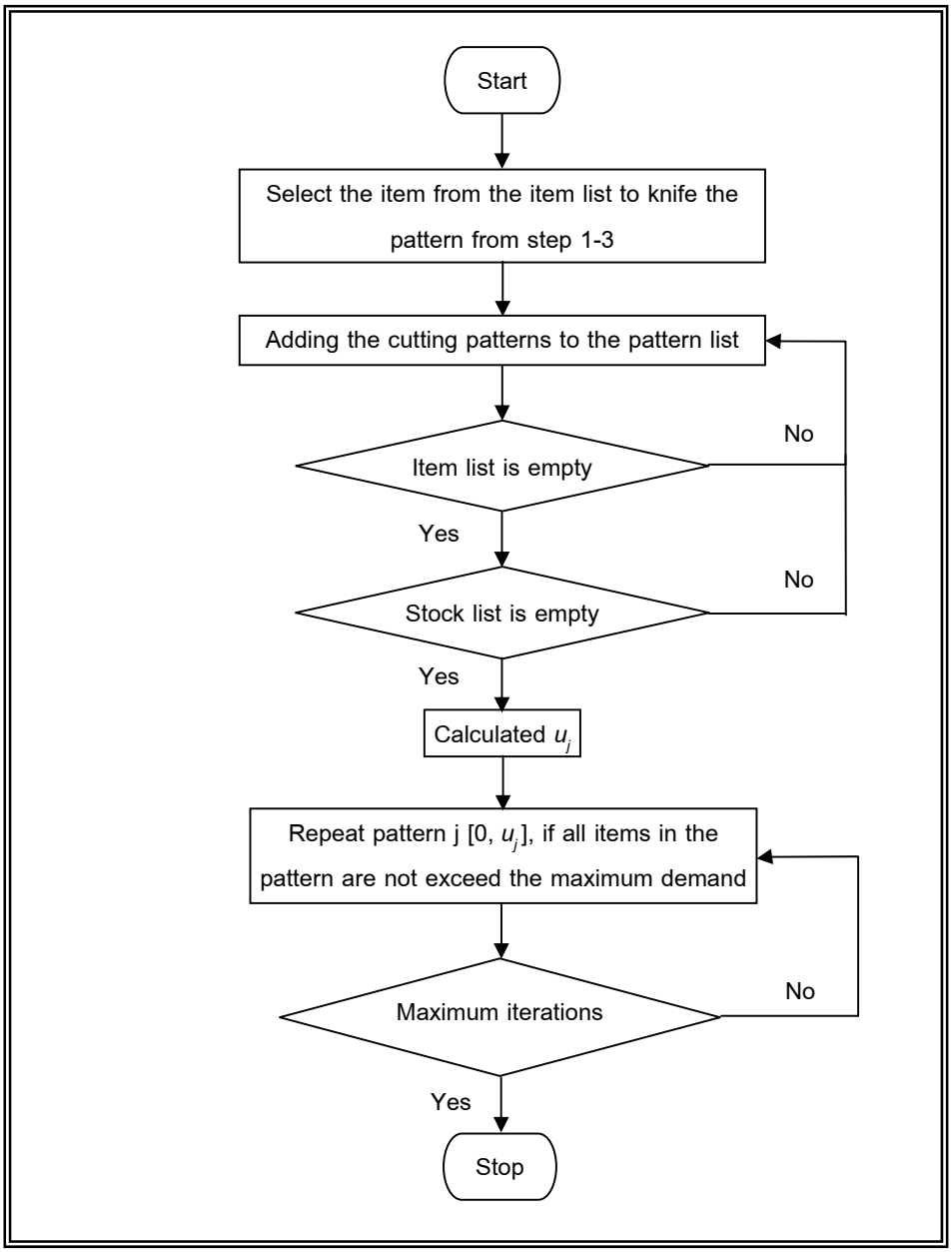


Figure 2. Flowchart of the heuristic for the problem

The algorithm is presented in the following steps.

1. Selected the longest length item from the item list and the stock size of lowest cost from the stock list. Go to step 2.

2. Knifed the selected item on the selected stock size from step 1 until the remainder length of stock size is not sufficient. Go to step 3.

3. From the remainder length of stock size, searching the longest length item that sufficient from all items and knives it as step 2. Repeated this step until the remainder length is zero or the item are not selected. Go to step 4.

4. Adding one cutting pattern from step 3 to the pattern list. Remove the longest length item from the item list. If the item list is empty go to step 5, else return to step 1.

5. Return all items into the item list but remove the stock size of lowest cost from the stock list. If the stock list is empty go to step 6, else return to step 1.

6. From all patterns in the pattern list, calculated the maximum usage value of pattern j (u_j) as:

$\bar{d}_z$ = average demand of item z,

n_z = number of knives of item z,

$$u_j = \frac{\bar{d}_z}{n_z} \tag{10}$$

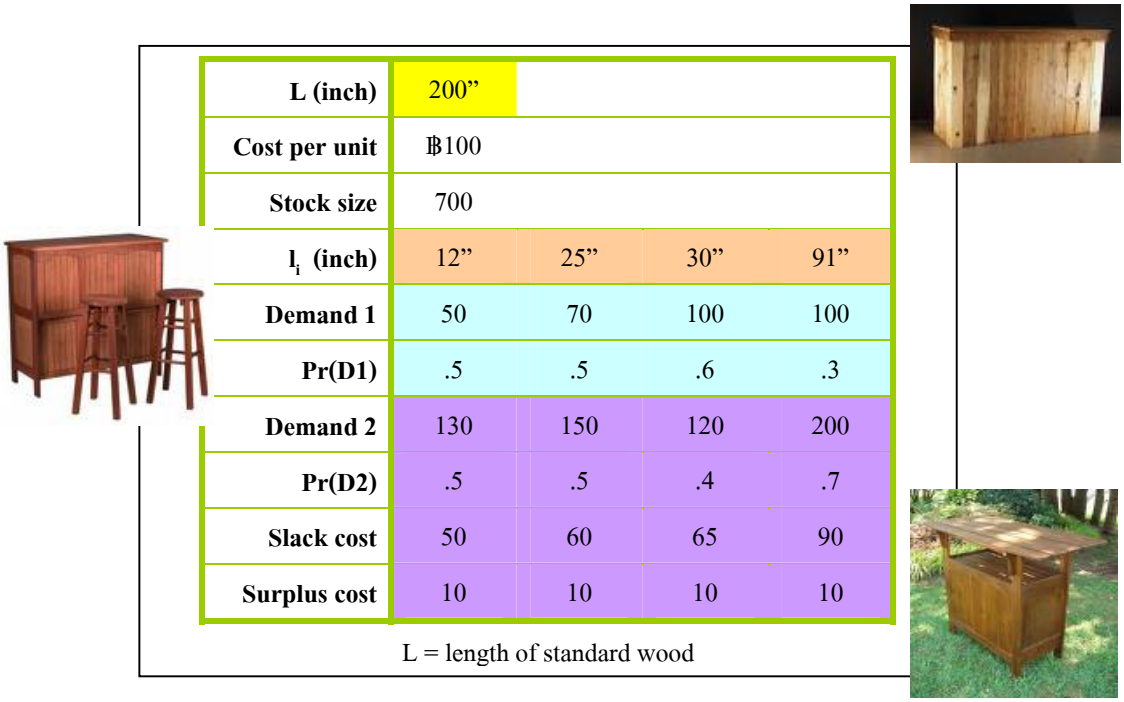
setting the iteration index as 1 and go to step 7.

7. Sorting the pattern from the pattern list as randomly. If all items in the selected pattern j are not exceed the maximum demand choices, then repeat the selected pattern j from the sorting order randomly between 0 and u_j. If the pattern in the last order is reached, then calculated cost as objective (1). Go to step 8.

8. If the maximum iteration is reached then STOP else increasing the iteration index and go to step 7.

AN INSTANCE OF STOCHASTIC CUTTING STOCK PROBLEMS (SCSP)

In a construction industry, Carpenter Company (assumed name) is a distributor and importer of wooden furniture. Wooden bar is the most famous product, various styles are produced. To assembly wooden bar, the processor need to cut standard squared woods with 200 inches in length to small pieces with size 12 inches, 25 inches, 30 inches, and 91 inches. The demands are not constant; it was up to customer's orders which were changed more rapid than production planning. The problem is if the processor produced more than demands (surplus), he or she will increase retention costs. But if the processor produced less than demands (slack), he or she will increase costs for buying small items to meet demands. Data were collected to see the probabilities of each demand. The lowest and the highest demands are defined as demand 1 and demand 2, consequently.



L (inch)	200"			
Cost per unit	฿100			
Stock size	700			
l_i (inch)	12"	25"	30"	91"
Demand 1	50	70	100	100
Pr(D1)	.5	.5	.6	.3
Demand 2	130	150	120	200
Pr(D2)	.5	.5	.4	.7
Slack cost	50	60	65	90
Surplus cost	10	10	10	10

L = length of standard wood

Figure 3. An instance of SCSP with Data

From Fig. 3, all combination sets of possible demands are 2^4 or 16 sets. Each set was observed to know its probability. Therefore, this problem is for finding what the best production planning is, to optimize stochastic one-dimensional cutting stock problem with discrete random demands. The information of all sets was shown as following:

Table 1. Expanding all combination scenarios of possible demands

Set	Pr(Set)	Retail size	Pr(D)	Demand	Set	Pr(Set)	Retail size	Pr(D)	Demand
1	.07	12	.5	130	9	.105	12	.5	130
		25	.5	150			25	.5	70
		30	.4	120			30	.6	100
		91	.7	200			91	.7	200
2	.07	12	.5	50	10	.03	12	.5	130
		25	.5	150			25	.5	70
		30	.4	120			30	.4	120
		91	.7	200			91	.3	100
3	.07	12	.5	130	11	.045	12	.5	130
		25	.5	70			25	.5	150
		30	.4	120			30	.6	100
		91	.7	200			91	.3	100
4	.105	12	.5	130	12	.105	12	.5	50
		25	.5	150			25	.5	70
		30	.6	100			30	.6	100
		91	.7	200			91	.7	200
5	.03	12	.5	130	13	.03	12	.5	50
		25	.5	150			25	.5	70
		30	.4	120			30	.4	120
		91	.3	100			91	.3	100
6	.07	12	.5	50	14	.045	12	.5	50
		25	.5	70			25	.5	150
		30	.4	120			30	.6	100
		91	.7	200			91	.3	100
7	.105	12	.5	50	15	.045	12	.5	130
		25	.5	150			25	.5	70
		30	.6	100			30	.6	100
		91	.7	200			91	.3	100
8	.03	12	.5	50	16	.045	12	.5	50
		25	.5	150			25	.5	70
		30	.4	120			30	.6	100
		91	.3	100			91	.3	100

SOLVING BY THE COLUMN-GENERATION PROCEDURE

For solving the problem from Fig. 3, we need to generate the problem in text file, named “dat1.txt”, as shown in Fig. 4 – 5. Then the program of the Column-Generation (CG) procedure, called “CGstochastic1.EXE” (see also Appendix A), was run. The solutions were shown in Table 4 and Fig. 6.

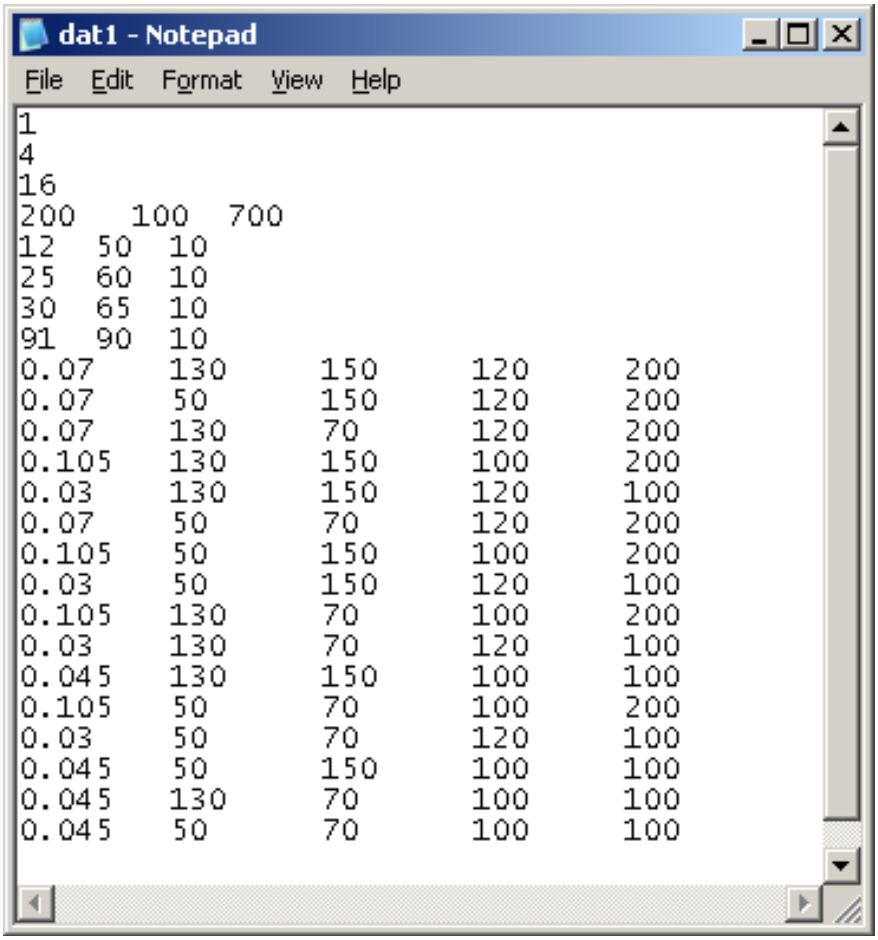


Figure 4. Problem in Fig. 3 was generated into the “dat1.txt”

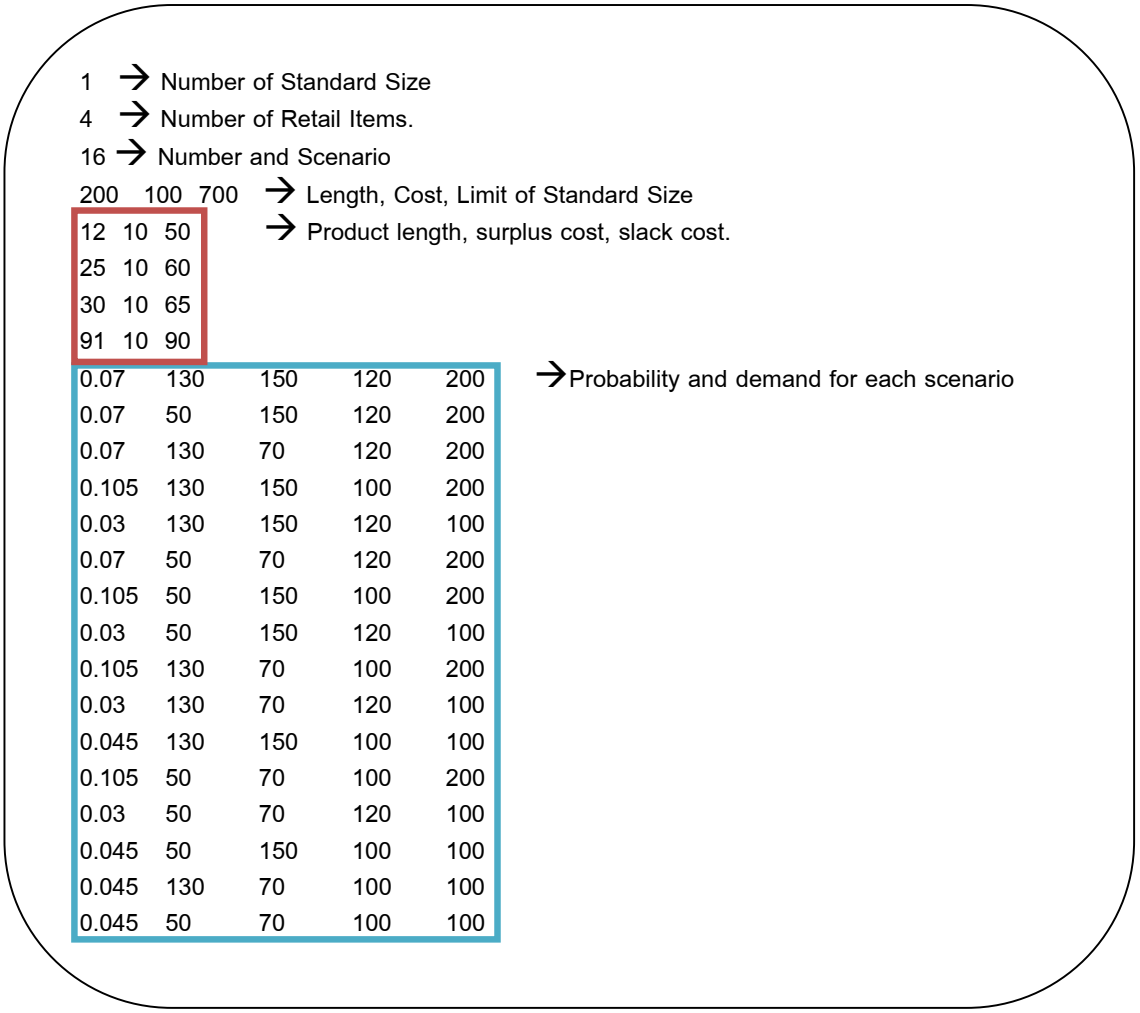


Figure 5. The way to create problem in text file

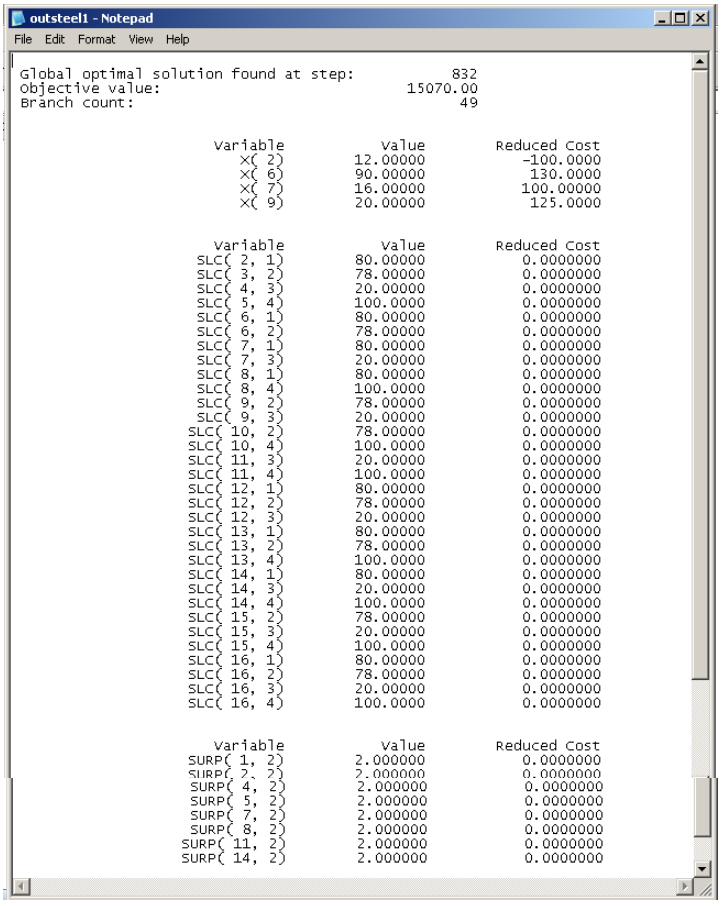
Table 2. Initial patterns

Pattern	Retail size (inch)			
	12”	25”	30”	91”
1	16	0	0	0
2	0	8	0	0
3	0	0	6	0
4	0	0	0	2

a) Active patterns

Pattern	Retail size (inch)			
	12	25	30	91
1	16	0	0	0
2	0	8	0	0
3	0	0	6	0
4	0	0	0	2
5	4	0	5	0
6	1	0	0	2
7	0	2	5	0
8	9	0	0	1
9	2	1	2	1

b) Output



c) Show the computation time

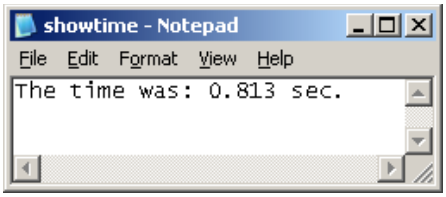


Figure 6. Solutions of the CG procedure

SOLVING BY THE PROPOSED HEURISTIC, BASED ON FIRST FIT DECREASING METHOD

For solving the problem from Fig. 3, we need to generate the problem in text file as shown in Fig. 7. Then the program of the proposed heuristic, called “stochasticcutting2.EXE” (see also Appendix B), was run. The solutions were shown in Fig. 8 - 9.

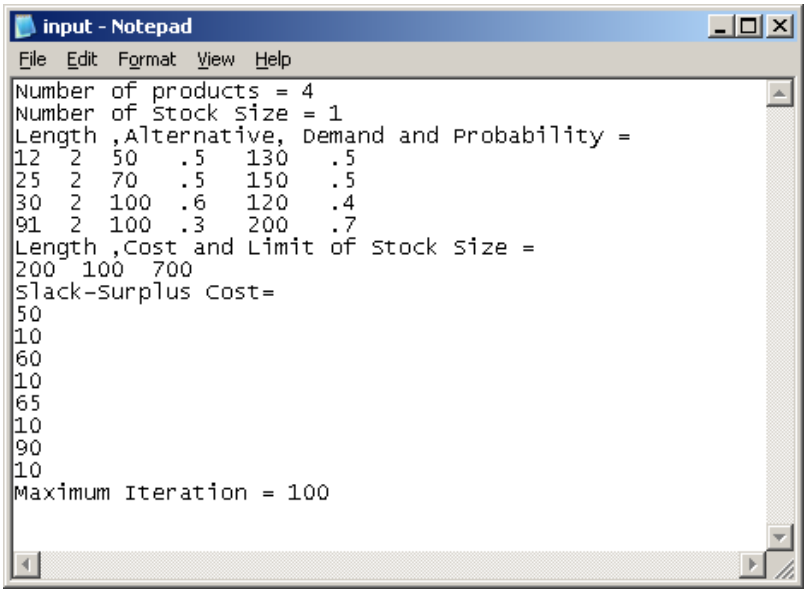


Figure 7. Problem in Fig. 3 was generated into the “input.txt”

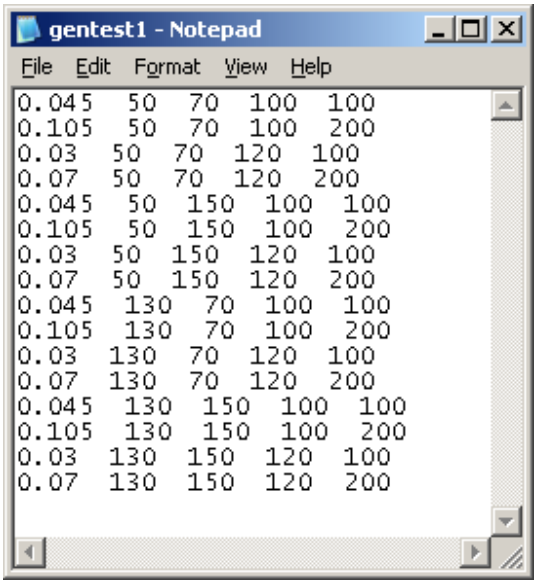
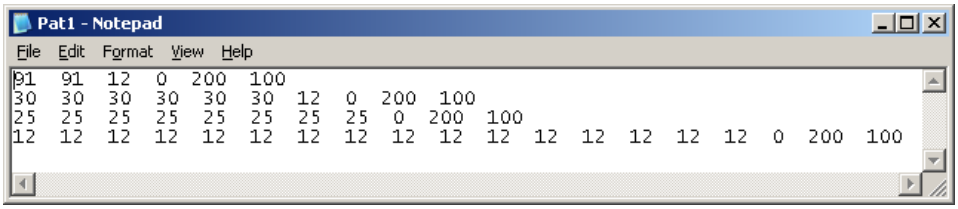
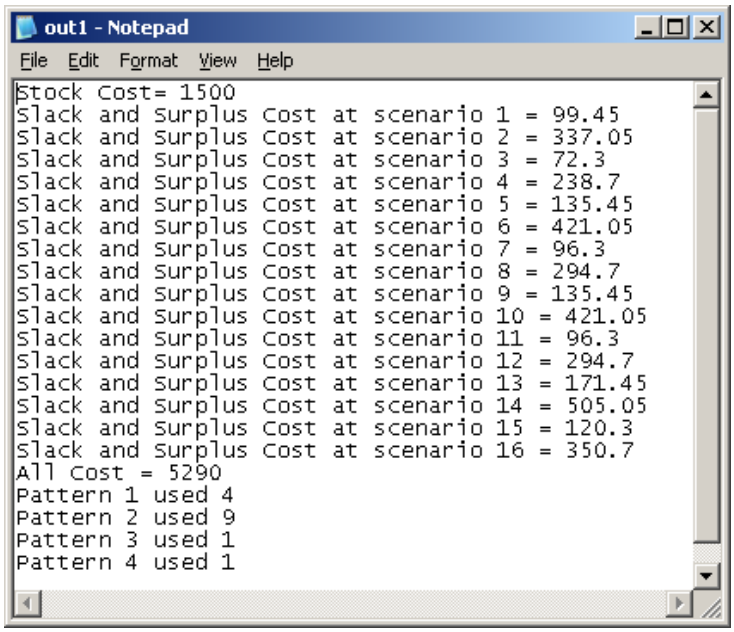


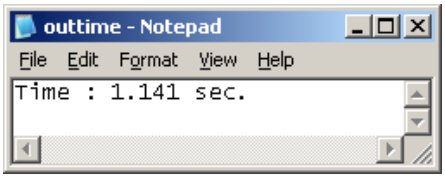
Figure 8. All possible scenarios were created and kept into “gentest1.txt”



a) Active patterns



b) Output



c) Show the computation time

Figure 9. Solutions of the proposed heuristic, based on first fit decreasing method

EXPERIMENTAL RESULTS

Both the method of column-generation technique and the proposed heuristic were coded in the C++ programming and run on Pentium 4, 256 MB, 2.4 GHz. The column-generation procedure is linked with LINGO 6.0 solver. The 24 instances were considered and separated into 3 groups of eight instances. The first group is performed with 10 retail items; all items are 2 choices of demands with 50% of probability. For each instance, all items have the same demand are such (50, 130), (50, 120), (50, 110), (50, 100), (50, 90), (50, 80), (50, 70), (50, 60). All possible scenarios are 1,024. The second group is performed with 6 retail items; all items are 3 choices of demands with probabilities 0.3, 0.3, and 0.4, respectively. For each instance, all items have the same demand are such (50, 90, 130), (50, 90, 120), (50, 90, 110), (50, 80, 130), (50, 80, 120), (50, 80, 110), (50, 70, 130), (50, 70, 120). All possible scenarios are 729. The last group is performed with 5 retail items; all items are 4 choices of demands with probabilities 0.3, 0.3, 0.2, and 0.2, respectively. For each instance, all items have the same demand are such (50, 70, 100, 130), (50, 70, 100, 120), (50, 70, 90, 130), (50, 70, 90, 120), (50, 70, 90, 110), (50, 80, 100, 130), (50, 80, 100, 120), (50, 80, 90, 120). All possible scenarios are 1,024. For all instances, the stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.

The lowerbound is computing by solving linear programming model (1-4). The limited time for a branch-and-bound procedure to obtain all integer solutions is 360 seconds. For the proposed heuristic, we set the maximum iterations of randomly generate the repeated cutting pattern to 10.

In order to compare the performance between column-generation technique and proposed heuristic, the upperbound value and computation times are shown in Tables 1-3. From the results, the proposed heuristic provides the upperbound value above the lowerbound 0.00-16.78% approximately. The upperbound value from column-generation and integer programming is better than the proposed heuristic but the computation times are higher.

Table 3. Eight instances with 10 retail items and 2 varied demands

			Column generation + ILP				Proposed heuristic			
Problem	Demand 1	Demand 2	UB	LB	Time (sec)	Patterns	UB	Time (sec)	Patterns	Above LB (%)
Test 1	50	130	35700	35687	360	19	39290	29	10	10.10
Test 2	50	120	33250	33212	360	18	36200	29	10	9.00
Test 3	50	110	30620	30620	153	19	33690	29	10	10.03
Test 4	50	100	28140	28140	83	18	31220	29	10	10.95
Test 5	50	90	25670	25670	74	18	28160	29	10	9.70
Test 6	50	80	23200	23200	79	18	26470	29	10	14.09
Test 7	50	70	20720	20720	82	18	23050	29	10	11.25
Test 8	50	60	18240	18240	76	18	21300	29	10	16.78

Note: Time (sec): Computing times in seconds; Patterns: The number of generated cutting patterns

Table 4. Eight instances with 6 retail items and 3 varied demands

				Column generation + ILP				Proposed heuristic			
Problem	Demand 1	Demand 2	Demand 3	UB	LB	Time (sec)	Patterns	UB	Time (sec)	Patterns	Above LB (%)
test 9	50	90	130	18374	18374	179	11	19596	29	6	6.65
test10	50	90	120	17174	17174	179	11	18350	29	6	6.85
test11	50	90	110	15974	15974	179	11	17050	29	6	6.74
test12	50	80	130	17940	17940	33	11	19586	29	6	9.18
test13	50	80	120	18034	18034	33	11	18034	29	6	0.00
test14	50	80	110	16656	16656	33	11	16656	29	6	0.00
test15	50	70	130	17560	17560	120	11	19010	29	6	8.26
test16	50	70	120	16360	16360	120	11	17964	29	6	9.80

Note: Time (sec): Computing times in seconds; Patterns: The number of generated cutting patterns

Table 5. Eight instances with 5 retail items and 4 varied demands

					Column generation + ILP				Proposed heuristic			
Problem	Demand 1	Demand 2	Demand 3	Demand 4	UB	LB	Time (sec)	Patterns	UB	Time (sec)	Patterns	Above LB (%)
test17	50	70	100	130	12686	12686	37	9	13956	29	5	10.01
test18	50	70	100	120	13192	13192	37	9	13192	29	5	0.00
test19	50	70	90	130	12186	12186	37	9	13312	29	5	9.24
test20	50	70	90	120	11686	11686	37	9	12756	29	5	9.16
test21	50	70	90	110	11186	11186	37	9	12052	29	5	7.74
test22	50	80	100	130	13950	13950	37	9	13950	29	5	0.00
test23	50	80	100	120	13306	13306	37	9	13306	29	5	0.00
test24	50	80	90	120	12870	12870	37	9	12870	29	5	0.00

Note: Time (sec): Computing times in seconds; Patterns: The number of generated cutting patterns

DISCUSSION

From Tables 1-3, 24 instances were tested. The computation times of all instances are not always related with various retail items and choices of demands. The column-generation and integer linear programming provides more patterns than the proposed heuristic. The numbers of scenarios for each table are 1,024, 729, and 1,024, respectively. That causes the computation times of all instances, solved by the proposed heuristic, are almost equal and also less than those solved by the column-generation and integer linear programming.

CONCLUSION

In this paper, we have examined the problem of minimizing the number of cutting stocks and capacitated planning cost such inventory and backorder. The 24 instances were experimented into three groups, varying by number of retail items and number of uncertain demands; (10, 2), (6, 3), and (5, 4). For the mathematical model, the number of variables and constraints after adding the basis column to linear optimal of all groups are (20,498, 10,241), (8,759, 4,375), and (10,249, 5,121). The problem is determined by column-generation technique and the proposed constructive heuristic. For the column-generation technique, the linear programming model is proposed as (1-4) and the new cutting pattern are adding as the new basis column by solving (5-9), the lowerbound is obtained when stop adding the new basis column and the upperbound is obtained by applied branch-and-bound steps for rounding variables to integer. For the proposed heuristic, it has consistently high performance in computation times but the obtained upperbounds are not closed to the lowerbound as the column-generation technique. But the proposed heuristic is appropriate to solve a large instance because of the quick response requirement.

ACKNOWLEDGEMENT

We would like to acknowledge the Thai Research Fund (TRF) for providing the grant to support this research.

LITERATURE CITED

Andradottir, S, 1996. A Global Search Method for Discrete Stochastic Optimization. SIAM Journal on Optimization; 513-530. DOI: 10.1137/0806027.

Audsley, E., S. Barker, K. Darby-Dowman and D. Parsons, 2009. A Two-Stage Stochastic Programming with Recourse Model for Determining Robust Planting Plans in Horticulture. The Journal of the Operational Research Society; 51: 83-89. http://www.jstor.org/pss/253950.

Beraldi, P., M. E. Bruni and D. Conforti, 2009. The Stochastic Trim-loss Problem. European Journal; of Operational Research; 197: 42-49. DOI: 10.1016/j.ejor.2008.04.042.

Birge, J. R. and F. Louveaux, 1997. Introduction to Stochastic Programming. New York: Springer.

Brandimarte, P. and A. Villa, 1995. Advanced Models for Manufacturing Systems Management. CRC press, Inc; U.S.A. ISBN: 0849383323.

Dantzig, G. B., 1955. Linear Programming under Uncertainty. Management Science; 1: 197-206. http://www.stanford.edu/class/msande348/papers/Dantzig_original_sp_paper.pdf.

Dantzig, G.B and P. Glynn, 1990. Parallel Processors for Planning Under Uncertainty. Annals of Operations Research; 22: 1-21. DOI: 10.1007/BF02023045.

Dantzig, G. B. and P. Wolfe, 1961. The decomposition algorithm for linear programs. Econometrica; 29:767-778. http://www.jstor.org/pss/1911818.

Elmaghraby, S. E., 1959. An Approach to Linear Programming under Certainty. Operations Research; 7: 208-216. DOI: 10.1287/opre.7.2.208.

Elmaghraby, S. E., 1960. Allocation under Uncertainty When the Demand has Continuous D.F. Management Science; 6: 270-294. http://www.jstor.org/pss/2627342.

Ermoliev, V. I. Norkin and A. Ruszczynski, 1998. An Optimal Allocation of Invisibles under Uncertainty. Operations Research; 46: 381-395. DOI: 10.1287/opre.46.3.381.

Ermoliev, Y. M., 1983. Stochastic Quasigradient Methods and Their Application to System Optimization. Stochastic; 9: 1-36. http://www.informaworld.com/smpp/content~db=all~content=a776560572.

Garstka, S. J. and D.P. Rutenberg, 1973. Computation in Discrete Stochastic Programs with Recourse. Operations Research; 21: 112-122. DOI: 10.1287/opre.21.1.112.

Gilmore, P. C. and R. E. Gomory, 1961. A linear programming approach to the cutting stock problem. Operations Research; 9: 849-859. DOI: 10.1287/opre.9.6.849.

Gilmore, P. C. and R. E. Gomory, 1963. A linear programming approach to the cutting stock problem - Part II. Operations Research; 11: 863-888. DOI: 10.1287/opre.11.6.863.

Higle, J. and S. Sen, 1991. Stochastic Decomposition: An Algorithm for Two Stage Linear Programs with Recourse. Mathematics of Operations Research; 16: 650-669. http://mor.journal.informs.org/cgi/content/abstract/16/3/650.

Lasdon, S. L., 1970. Optimization theory for large systems. New York: Macmillan publishing, NY, U.S.A. ISBN-10: 0486419991.

Mukai, H. and D. Yan, 1992. Stochastic Discrete Optimization. SIAM Journal on Control and Optimization; 30: 594-612. DOI: 10.1137/0330034.

Ziemba, W.T., 1970. Computational Algorithm for Convex Stochastic Programs with Simple Recourse. Operations Research; 18: 414-431. DOI: 10.1287/opre.18.3.414.

APPENDICES

APPENDIX A

The CGstochastic1.EXE

```
#include <stdio.h>      // for file i/o
#include <conio.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <fstream.h>
#include <windows.h>
#include <string.h>
#include <process.h>
clock_t start, end;
int inf=30000,btect,*ccol,*cxcol;
typedef void(LINGOCALL)(void*[],int*,int*);
void main()
{ char fg[5];

  void initial();

  start = clock();
  initial();
  cout<<"Complete Solution "<<endl;
  end = clock();
  ofstream vv("showtime.txt");
  vv<<"The time was: "<<((end - start) / CLK_TCK);
  vv.close();
  getch();
}
void initial()
{ char fg[5];
  void solvedual(int,int ,int ,int& ,int *,int *, int *,bool &);
  void genfeas(int ,int ,int *,int *,int *);
  void LPsolve()   ;
  void ILPsolve();
  void setmemory();
  int i,j,k,m,l,ii,jj,*leg,*cleg,*tleg,**dem,*cap,h;
  float *cslc,*csurp,*probs;

  ifstream ff("c:\\twostages\\dat1.txt");
   ff>>k;
   ff>>m;
   ff>>h;
   tleg=new int[m+1];
   ccol=new int[m+1];
   cxcol=new int[m+1];
    cslc=new float[m+1];
     csurp=new float[m+1];
   leg=new int[k+1];
   cleg=new int[k+1];
   dem=new int*[h+1];
   cap=new int[k+1];
   probs=new float[h+1];
   for(j=1;j<=h;j++)
    dem[j]=new int[m+1];

    for(jj=1;jj<=k;jj++)
```

```
      { ff>>leg[jj];    ff>>cleg[jj];
        ff>>cap[jj];
      }


    for(ii=1;ii<=m;ii++)
    { ff>>tleg[ii]; ff>>cslc[ii]; ff>>csurp[ii];

    }
   for(jj=1;jj<=h;jj++)
   { ff>>probs[jj];
     for(ii=1;ii<=m;ii++)
     {
       ff>>dem[jj][ii];
     }
   }
    ff.close();

    genfeas(k,m,leg,cleg,tleg);
    ofstream e1("c:\\twostages\\addint.txt");
    e1.close();
    ofstream ee("steelcut1.lng");
    ee<<"Model:"<<endl;
   ee<<"SETS:"<<endl;
ee<<"k/1.."<<k<<"/;"<<endl;
ee<<"l/1..@file('c:\\twostages\\l1.txt')/;"<<endl;
ee<<"m/1.."<<m<<"/;"<<endl;
ee<<"h/1.."<<h<<"/;"<<endl;
ee<<"ccost(l):x,a;"<<endl;
ee<<"tcost(m):c,y,cslc,csurp;"<<endl;
ee<<"patt(l,m):b;"<<endl;
ee<<"u(h,m):slc,surp;"<<endl;
ee<<"pr(h):ps;"<<endl;
ee<<"s(k):r;"<<endl;
ee<<"t(l,k):w;"<<endl;
ee<<"dm(h,m):d;"<<endl;
ee<<"ENDSETS"<<endl;
 ee<<"
Min=@sum(ccost:a*x)+@sum(u(i,j):ps(i)*((cslc(j)*slc(i,j))+(csurp(j)*surp(i
,j))));"<<endl;

ee<<"@for(s(kk): "<<endl;
ee<<" [duo1]@sum(l(q):x(q)*w(q,kk))<=r(kk););"<<endl;

ee<<"@for(u(hh,mm): "<<endl;
 ee<<"[duo2]@sum(l(q):b(q,mm)*x(q))-slc(hh,mm)+surp(hh,mm)=d(hh,mm););
"<<endl;
ee<<"@file('c:\\twostages\\addint.txt')"<<endl;
 ee<<"Data:"<<endl;
ee<<"r="<<endl;
for(ii=1;ii<=k;ii++)
{ee<<cap[ii]<<" "; if ( ((ii%8)==0)||(ii==k)) ee<<endl;}
ee<<";"<<endl;
ee<<"d="<<endl;
for(j=1;j<=h;j++)
```

```
{
for(ii=1;ii<=m;ii++)
{ //cout<<dem[ii]<<endl; getch();
  ee<<dem[j][ii]<<" "; if ( ((ii%8)==0)||(ii==m)) ee<<endl;
}
}
ee<<";"<<endl;

ee<<"ps="<<endl;
for(j=1;j<=h;j++)
{
  ee<<probs[j]<<" "; if ( ((j%8)==0)||(j==h)) ee<<endl;
}
ee<<";"<<endl;

ee<<"cslc="<<endl;
for(j=1;j<=m;j++)
{
  ee<<cslc[j]<<" "; if ( ((j%8)==0)||(j==m)) ee<<endl;
}
ee<<";"<<endl;

ee<<"csurp="<<endl;
for(j=1;j<=m;j++)
{
  ee<<csurp[j]<<" "; if ( ((j%8)==0)||(j==m)) ee<<endl;
}
ee<<";"<<endl;

ee<<"b=@file('c:\\twostages\\pp1.txt');"<<endl;
ee<<"a=@file('c:\\twostages\\costpp1.txt');"<<endl;
ee<<"w=@file('c:\\twostages\\wpp1.txt');"<<endl;
ee<<"@text('c:\\twostages\\duo1.txt')=@dual(duo1);"<<endl;
  ee<<"@text('c:\\twostages\\duo2.txt')=@dual(duo2);"<<endl;
  ee<<"ENDDATA"<<endl;

ee<<"End";
  ee.close();
  ifstream tt("c:\\twostages\\l1.txt");
  tt>>l;
  tt.close();
  setmemory();
  bool detect=0;
  do{ detect=0;
  LPsolve();

  solvedual(h,k,m,l,cleg,tleg,leg,detect);
  /*cout<<"check "<<endl; getch();
  cout<<detect<<endl; */
   if (btect==0) break;
  }while(detect==1);
  ILPsolve();
}
```

```
void solvedual(int h,int k,int m,int& tl,int *cleg,int *tleg, int
*leg,bool &detect)
{ int ii,ik;
  void gkpsolve();
  float *alp,*p,**ep;
alp=new float[k+1];
ep=new float*[h+1];
p=new float[m+1];
for(ii=1;ii<=h;ii++)
{ep[ii]=new float[m+1];
 p[ii]=0.0;
}

char fg[5];
 ifstream hhh("duo1.txt");
 for(ik=1;ik<=k;ik++)
   {hhh>>alp[ik];
   if (alp[ik]<0.00001) alp[ik]=0.00;
   cout<<alp[ik]<<endl;
   }
 hhh.close();
  ifstream hh("duo2.txt");
for(ii=1;ii<=h;ii++)
{ for(ik=1;ik<=m;ik++)
   {//hh>>fg[5];
    hh>>ep[ii][ik];
   //if (ep[ii][ik]<0.00001) ep[ii][ik]=0;
   p[ik]+=ep[ii][ik];
  cout<<"p[ik] "<<p[ik]<<endl;
   }
 }
 hh.close();

 int pi=0;
 ofstream ai("gkp1.txt");
  ai<<"Model:"<<endl;
  ai<<"SETS:"<<endl;
  ai<<"mat/1.."<<m<<"/:b;"<<endl;
  ai<<"stock/1.."<<k<<"/:y;"<<endl;
  ai<<"ENDSETS"<<endl;
ai<<"[obj]Max=  "<<endl;
pi=0;
for(ik=1;ik<=m;ik++)
    {  //if (p[ik] >0.00001)
    if (pi!=0) ai<<"+"; ai<<(-p[ik])<<"*b("<<ik<<")"; pi++;

          if (((pi%4)==0) ||(ik==m))ai<<endl;
   }

for(ik=1;ik<=k;ik++)
{  pi++; ai<<"-"<<(alp[ik]+cleg[ik])<<"*y("<<ik<<")";  if
(((pi%4)==0)||(ik==k)) ai<<endl;
  if (ik==k) ai<<";"<<endl;
}
```

```
for(ik=1;ik<=m;ik++) {ai<<tleg[ik]<<"*b("<<ik<<")";   if (ik!=m) ai<<"+";
if (((ik%4)==0)||(ik==m)) ai<<endl;   }
for(ik=1;ik<=k;ik++) { ai<<"-"<<leg[ik]<<"*y("<<ik<<")";  if
(((ik%4)==0)||(ik==k)) ai<<endl;}
ai<<"<=0; "<<endl;
pi=0;
 for(ik=1;ik<=k;ik++)
    {         ai<<"y("<<ik<<")"; pi++;
        if (ik!=k) ai<<"+";  if (((pi%4)==0) ||(ik==k))ai<<endl;
   }
  ai<<"=1;"<<endl;   ai<<"@for(mat:@GIN(b););
@for(stock:@GIN(y););"<<endl;
  ai<<"DATA:"<<endl;
  ai<<"@text('objgkp.txt')=obj;"<<endl;
  ai<<"@text('patt1.txt')=b;"<<endl;
  ai<<"@text('istock.txt')=y;"<<endl;
  ai<<"ENDDATA"<<endl;
ai<<"End"<<endl;
  ai.close();
  gkpsolve();
float vi=0.00000;
ifstream nn("objgkp.txt");
nn>>vi;
nn.close();

if (vi>=0.001) { detect=1;
float jk;
int dk;

   ofstream mm("pp1.txt",ios::app);
ifstream ll("patt1.txt");
for(ik=1;ik<=m;ik++)
{ ll>>jk;  mm<<jk<<"  ";
  ccol[ik]=jk;
}
mm<<endl;
ll.close();
mm.close();
btect=0;
for(ik=1;ik<=m;ik++)
{ if(ccol[ik]!=cxcol[ik]) {btect=1; break;}
}
if (btect==1)
{
  for(ik=1;ik<=m;ik++)
  {
    cxcol[ik]=ccol[ik];
  }
}

   ofstream mm1("wpp1.txt",ios::app);
ifstream ll1("istock.txt");
for(ik=1;ik<=k;ik++)
{ ll1>>jk;  mm1<<jk<<"  "; if (jk==1.0000) {dk=ik;}  }
```

```
mm1<<endl;
ll1.close();
mm1.close();


   ofstream mm2("costpp1.txt",ios::app);
mm2<<cleg[dk]<<endl;
mm2.close();
 }//endif
else{
int dk;
float jk;

   ofstream mm("pp1.txt",ios::app);

for(ik=1;ik<=m;ik++)
{ mm<<"0  ";}
mm<<endl;

mm.close();


   ofstream mm1("wpp1.txt",ios::app);

for(ik=1;ik<=k;ik++)
{  mm1<<"0  "; }
mm1<<endl;
mm1.close();


   ofstream mm2("costpp1.txt",ios::app);
mm2<<inf<<endl;
mm2.close();


}//endelse


tl++;
ofstream mm3("l1.txt");
mm3<<tl<<endl;
mm3.close();

}
void genfeas(int k,int m,int *leg,int *cleg,int *tleg)
{  void minc(int*,int,int,int*,int&); void maxl(int,int*,int&);
     int t,p,ii,iv;
   maxl(m,tleg,t);
   minc(leg,k,tleg[t],cleg,p);
float ppp;
int ik;


   ofstream mm("pp1.txt");
  for(iv=1;iv<=m;iv++)
```

```
{  for(ik=1;ik<=m;ik++)
 {    if(iv!=ik) mm<<"0 "; else{ ppp=leg[p]/tleg[iv]; mm<<floor(ppp)<<"
";}
  } mm<<endl;
  }
mm.close();

   ofstream mm1("costpp1.txt");
  for(iv=1;iv<=m;iv++)
     mm1<<cleg[p]<<endl;
mm1<<endl;
mm1.close();

   ofstream mm2("wpp1.txt");
for(ik=1;ik<=m;ik++)
for(iv=1;iv<=k;iv++)
{  if(iv==p) mm2<<"1 "; else mm2<<"0 ";
   mm2<<endl;
  }
 mm2.close();

ofstream mm3("l1.txt");
mm3<<m<<endl;
mm3.close();
}

void maxl(int m,int *tleg,int &t)
      { int max=0,i;
        for(i=1;i<=m;i++)
        {  if (tleg[i]>max){ max=tleg[i]; t=i;}}


      }
void minc(int*leg,int k,int t,int *cleg,int &p)
      {  int min1=inf,i;
        for(i=1;i<=k;i++)
        { if (leg[i] <t) continue;
          if (cleg[i]<min1) { min1=leg[i]; p=i;}
        }
      }

void LPsolve()
{    LINGOCALL* m_pLINGOCaLL;
HINSTANCE m_hInstLINGO;
  m_hInstLINGO = LoadLibrary( "lingodll.dll");
   m_pLINGOCaLL =(LINGOCALL*) GetProcAddress( m_hInstLINGO, "LGCSCRIPT");
    void* pArgs[4];
     int nArgs = 4;
   int nErrorCode;

   pArgs[0] = (void*)
"SET CUTAPP 0\nSET HEURIS 0\nSET TERSEO 1\nTAKE steelcut1.lng\nDIVERT
outsteel1.txt\ngo\nnonz x\nnonz slc\nnonz surp\nRVRT\nQUIT\n";
   pArgs[1] = (void*) "cutout.txt";
   pArgs[2] = NULL;
```

```
   pArgs[3] = NULL;
   (*m_pLINGOCaLL)( pArgs, &nArgs, &nErrorCode);
   FreeLibrary(m_hInstLINGO);
}

void gkpsolve()
{    LINGOCALL* m_pLINGOCaLL;
HINSTANCE m_hInstLINGO;
  m_hInstLINGO = LoadLibrary( "lingodll.dll");
   m_pLINGOCaLL =(LINGOCALL*) GetProcAddress( m_hInstLINGO, "LGCSCRIPT");
    void* pArgs[4];
     int nArgs = 4;
   int nErrorCode;


   pArgs[0] = (void*)
"SET CUTAPP 0\nSET HEURIS 0\nSET TERSEO 1\nTAKE gkp1.txt\ngo\nQUIT\n";

   pArgs[1] = (void*) "cutout.txt";
   pArgs[2] = NULL;
   pArgs[3] = NULL;
   (*m_pLINGOCaLL)( pArgs, &nArgs, &nErrorCode);
   FreeLibrary(m_hInstLINGO);
}
void ILPsolve()
{ void LPsolve();
  rename("c:\\twostages\\outsteel1.txt","c:\\twostages\\outsteel.txt");
ofstream m2("c:\\twostages\\addint.txt");
  m2<<"@for(ccost:@gin(x););"<<endl;

  m2.close();
  LPsolve();

}
void setmemory()
{    LINGOCALL* m_pLINGOCaLL;
HINSTANCE m_hInstLINGO;
  m_hInstLINGO = LoadLibrary( "lingodll.dll");
   m_pLINGOCaLL =(LINGOCALL*) GetProcAddress( m_hInstLINGO, "LGCSCRIPT");
    void* pArgs[4];
     int nArgs = 4;
   int nErrorCode;
     pArgs[0] = (void*)"SET MXMEMB 2000\nILFTOL 0.0003\nSET FLFTOL
0.00001\nFREEZE\nQUIT\n";
     pArgs[1] = (void*) "cutout.txt";
     pArgs[2] = NULL;
     pArgs[3] = NULL;

    (*m_pLINGOCaLL)( pArgs, &nArgs, &nErrorCode);


   FreeLibrary(m_hInstLINGO);
}
```

APPENDIX B

The stocasticcutting2.exe

```
#include <stdio.h>      // for file i/o
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <fstream.h>
#include <string.h>
#include <process.h>
clock_t start, end;
char a[5];
int n,l,i,j,npat;
float *b,*s,*c,**pr,*slc,*surp,*vs,*kvs,*cc,atcost;
int **d,*limit,*v,*al,*tal,maxiter;
long sce=0;
void swapp(int&,int&);
void calcost(int);
void calpat();
void main()
{ void genpat();
  void readdat();
   readdat();
   ofstream a3("Pat1.txt");
   a3.close();
   npat=0;
   start=clock();
   genpat();
   al=new int[npat+1];
   tal=new int[npat+1];
   float tcost,mincost;
   int ii,k;
   //cout<<"1111 "<<endl;
   atcost=0;
   calpat();
   for(ii=1;ii<=npat;ii++)
   tal[ii]=al[ii];
    mincost=1000000;
  for(k=1;k<=maxiter;k++)
  { tcost=0;  sce=0;
   for(ii=1;ii<=npat;ii++)
      tcost+=al[ii]*cc[ii];

      ofstream a12("solution1.txt");
      a12<<"Stock Cost= "<<tcost<<endl;
      atcost=tcost;
   a12.close();

   ofstream b2("gentest1.txt");
   b2.close();

   calcost(0);

   if (atcost<mincost)
   {  mincost=atcost;
      remove("out1.txt");
      rename("solution1.txt","out1.txt");
```

```
      ofstream gf("out1.txt",ios::app);
      {
       gf<<"All Cost = "<<mincost<<endl;
       for(ii=1;ii<=npat;ii++)
       {  gf<<"Pattern "<<ii<<" used "<<al[ii]<<endl;}

      }
      gf.close();
   }
     for(ii=1;ii<=npat;ii++)
   {  al[ii]=random(tal[ii]+1);}
     cout<<"Iteration "<<k<<"   "<<atcost<<"  "<<mincost<<endl;
    }// end k
   //cout<<"3333 "<<endl;
   //spawnlp(P_NOWAIT,"Ordinary1.exe","Ordinary1.exe",NULL);
   end = clock();
  ofstream hj("outtime.txt");
  hj<<"Time : "<<((end - start) / CLK_TCK);
  hj.close();

}
void readdat()
{
  ifstream jk("input.txt");
  jk>>a[1];
  if (a[1]!='='){
  do{
     jk>>a[1];
    }while(a[1]!='=');}

  jk>>l;
  jk>>a[1];
  if (a[1]!='='){
  do{
     jk>>a[1];
    }while(a[1]!='=');}

  jk>>n;

  b=new float[l+1];
  v=new int[l+1];
  s=new float[n+1];
  c=new float[n+1];
  d=new int*[l+1];
  pr=new float*[l+1];
  limit=new int[n+1];
  slc=new float[l+1];
  surp=new float[l+1];
  vs=new float[l+1];
  kvs=new float[l+1];

  jk>>a[1];
  if (a[1]!='='){
  do{
     jk>>a[1];
```

```
    }while(a[1]!='=');}

  for(i=1;i<=l;i++)
  {
    jk>>b[i]; jk>>v[i];
    d[i]=new int[v[i]+1];
      pr[i]=new float[v[i]+1];
    for(j=1;j<=v[i];j++)
    {

    jk>>d[i][j]; jk>>pr[i][j];

    }

  }

  jk>>a[1];
  if (a[1]!='='){
  do{
     jk>>a[1];
    }while(a[1]!='=');}
  for(i=1;i<=n;i++)
  {
   jk>>s[i]; jk>>c[i]; jk>>limit[i];
  }

  jk>>a[1];
  if (a[1]!='='){
  do{
     jk>>a[1];
    }while(a[1]!='=');}
  for(i=1;i<=l;i++)
  {
   jk>>slc[i]; jk>>surp[i];
  }
  jk>>a[1];
  if (a[1]!='='){
  do{
     jk>>a[1];
    }while(a[1]!='=');}
  jk>>maxiter;
  jk.close();
}
void genpat()
{ int *dt,*vt,bmax,bm,k,h,*t,*e,*pp,x,**d,*st,*ut,**ss,*lk;
  float qq,*vp;
  dt=new int[l+1];
  vt=new int[l+1];
for(i=1;i<=l;i++)
{ vt[i]=i; dt[i]=0;}
 k=0; bmax=0;
 do{
 for(i=1;i<=l;i++)
   { if (dt[i]==1) continue;
    if (b[i]>bmax) {bmax=b[i]; bm=i;}
```

```
   }
   dt[bm]=1;
   k++; vt[k]=bm;  bmax=0;
  }while(k<l);

for(j=1;j<=n;j++)
  {
  for(h=1;h<=l;h++)
  {  qq=s[j];
  for(i=1;i<=l;i++)
 { if (i<h) continue;
  do
   {
     if (b[vt[i]]<=qq) {qq-=b[vt[i]];
                  ofstream sa("Pat1.txt",ios::app);
                  sa<<b[vt[i]]<<"  ";
                  sa.close();
                 }
                 else {i++;}
    if (qq==0){

                    break;
              }

   }while(i<=l);
 }
   ofstream a2("Pat1.txt",ios::app);
                  a2<<"0  ";
                  a2<<s[j]<<"  "<<c[j]<<endl;
                  npat++;
                  a2.close();


  }// end h
 }// end j
 t=new int[l+1];
 e=new int[l+1];
 pp=new int[npat+1];
 vp=new float[npat+1];
 st=new int[npat+1];
 ut=new int[npat+1];
 lk=new int [npat+1];
 cc=new float[npat+1];

/* for(i=1;i<=l;i++)
  for(j=1;j<=n;j++)
   d[i][j]=0; */
  ofstream a7("Pat2.txt");
  a7.close();
  ifstream a6("Pat1.txt");
  for(i=1;i<=npat;i++)
  { for(j=1;j<=l;j++)
    { e[j]=0; t[j]=0;}
   do{
      a6>>x;
```

```
      if (x>0)
      { for(j=1;j<=l;j++)
        {
          if (b[j]==x) {if (e[j]==1) {t[j]++;}  else {e[j]=1; t[j]=1;
pp[i]++;}
                        }
        }
      }
     }while(x!=0);
     a6>>x;
     vp[i]=x;
     a6>>x;
     cc[i]=x;
     ofstream a8("Pat2.txt",ios::app);
     for(j=1;j<=l;j++)
     {a8<<t[j]<<"  ";
     }
     //a8<<vp[i]<<endl;
     a8<<endl;
     a8.close();
  }// end i
}
void calcost(int z)
{  float **tt,vk,tcost,ual,pkvs;
   int ii,sk,jj;
   tt=new float*[npat+1];
   for(i=1;i<=npat;i++)
    tt[i]=new float[l+1];

   z++;
   for(jj=1;jj<=v[z];jj++)
      { vs[z]=d[z][jj];
        kvs[z]=pr[z][jj];
        if (z==l)
        {
          ifstream a9("Pat2.txt");
            for(ii=1;ii<=npat;ii++)
            { //vk=0;
              for(i=1;i<=l;i++)
              {a9>>tt[ii][i];
              //if (t[i]>vk) {vk=t[i]; sk=i;}
              }
             /* al[ii]=ceil(vs[sk]/t[sk]);
              ofstream a10("solution1.txt",ios::app);
              a10<<"Pattern "<<ii<<" = "<<al[ii]<<endl;
              a10.close();*/
            }
            a9.close();
             tcost=0;

             for(i=1;i<=l;i++)
             { ual=0;
               for(ii=1;ii<=npat;ii++)
               { ual+=al[ii]*tt[ii][i];
```

```
                 //tcost+=al[ii]*c[i];
                }

                if ((ual-vs[i]) < 0) {  tcost+= surp[i]*abs(ual-vs[i]);

                                      }
                                      else
                                          {
                                            tcost+= slc[i]*abs(ual-vs[i]);

                                          }
              }// end i
               pkvs=1.00;
               for(i=1;i<=l;i++)
               {tcost=kvs[i]*tcost;
                 pkvs=pkvs*kvs[i];
               }
               atcost+=tcost;
               sce++;
               ofstream a11("solution1.txt",ios::app);
              a11<<"Slack and Surplus Cost at scenario "<<sce<<" =
"<<tcost<<endl;
              a11.close();

              ofstream b1("gentest1.txt",ios::app);
              b1<<pkvs<<"  ";
              for(i=1;i<=l;i++)
              {b1<<vs[i]<<"  "; }
              b1<<endl;
              b1.close();

        }
        else calcost(z);

      }//end jj

}
void calpat()
{  float *vss,tcost=0.0;
   int *t,vk,sk,ii;
   t=new int[l+1];
   vss=new float[l+1];

   for(i=1;i<=l;i++)
   { vss[i]=0;
     for(j=1;j<=v[i];j++)
     {
       vss[i]+=pr[i][j]*d[i][j];
     }
   }
  ifstream a13("Pat2.txt");
            for(ii=1;ii<=npat;ii++)
            { vk=0;
              for(i=1;i<=l;i++)
              {a13>>t[i];
```

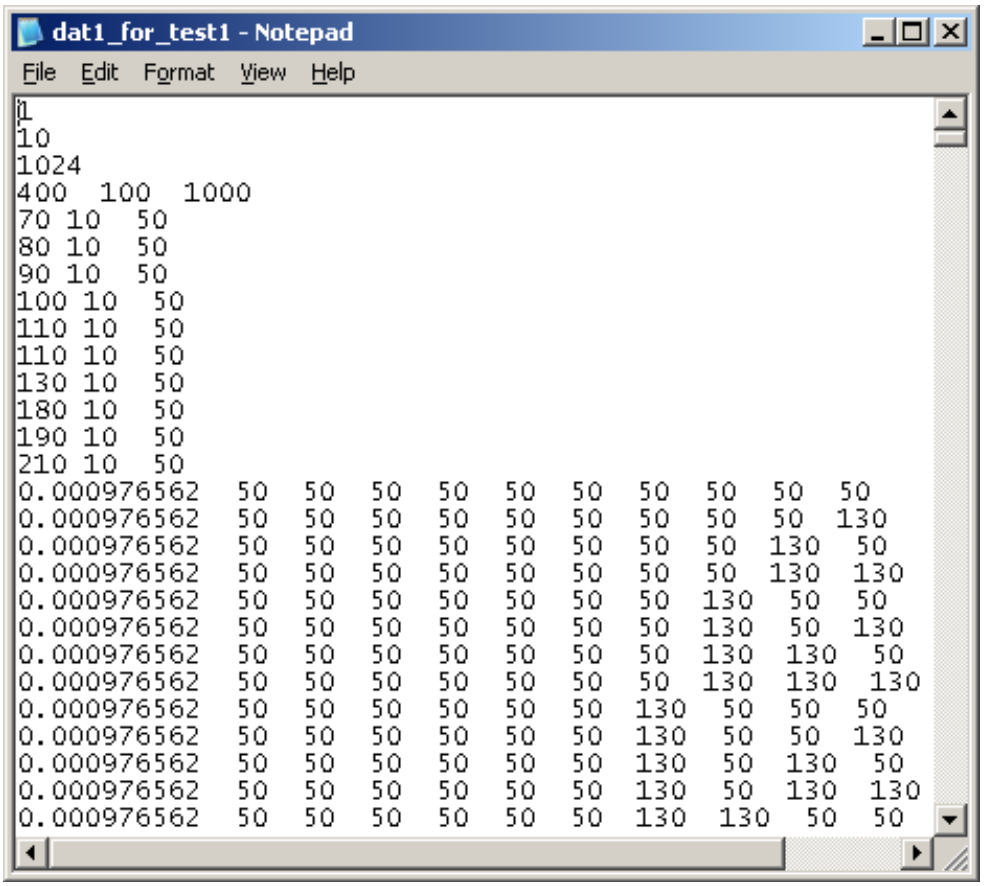
```
              if (t[i]>vk) {vk=t[i]; sk=i;}
              }
              al[ii]=ceil(vss[sk]/vk);
            }
            a13.close();


}
void swapp(int &x,int &y)
{ int z;
  z=x;
  x=y; y=z;

}
```

APPENDIX C

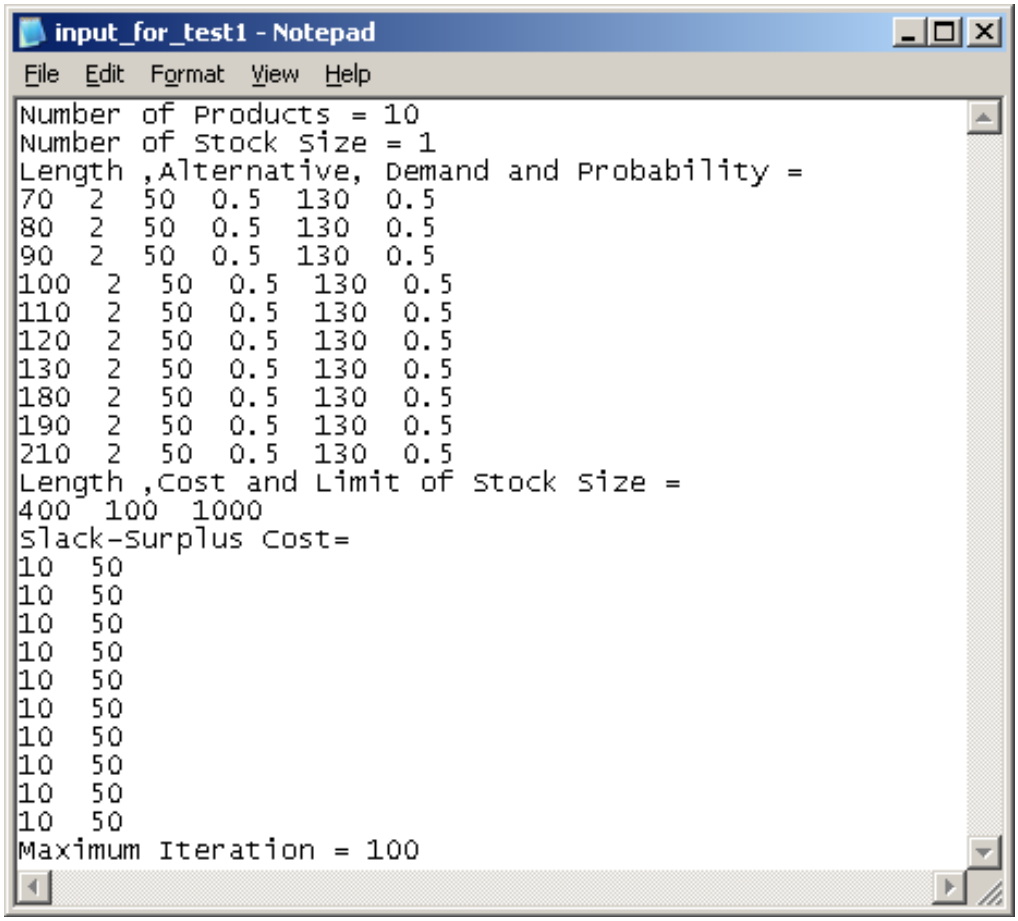
The 8 experimental instances are shown as example to see how to input data in the CGstochastic1.EXE and the stochasticcutting2.EXE.

THE 1st INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 130). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



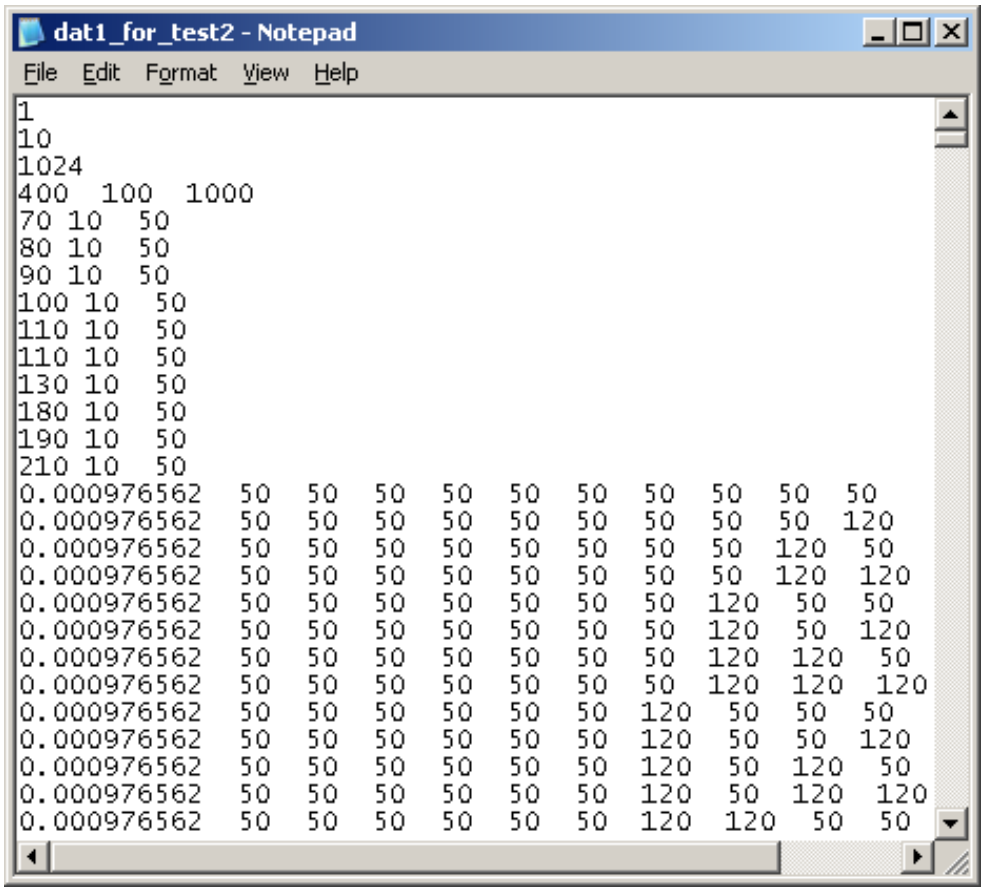
Data file for solving with the CGstochastic1.EXE or the column-generation procedure



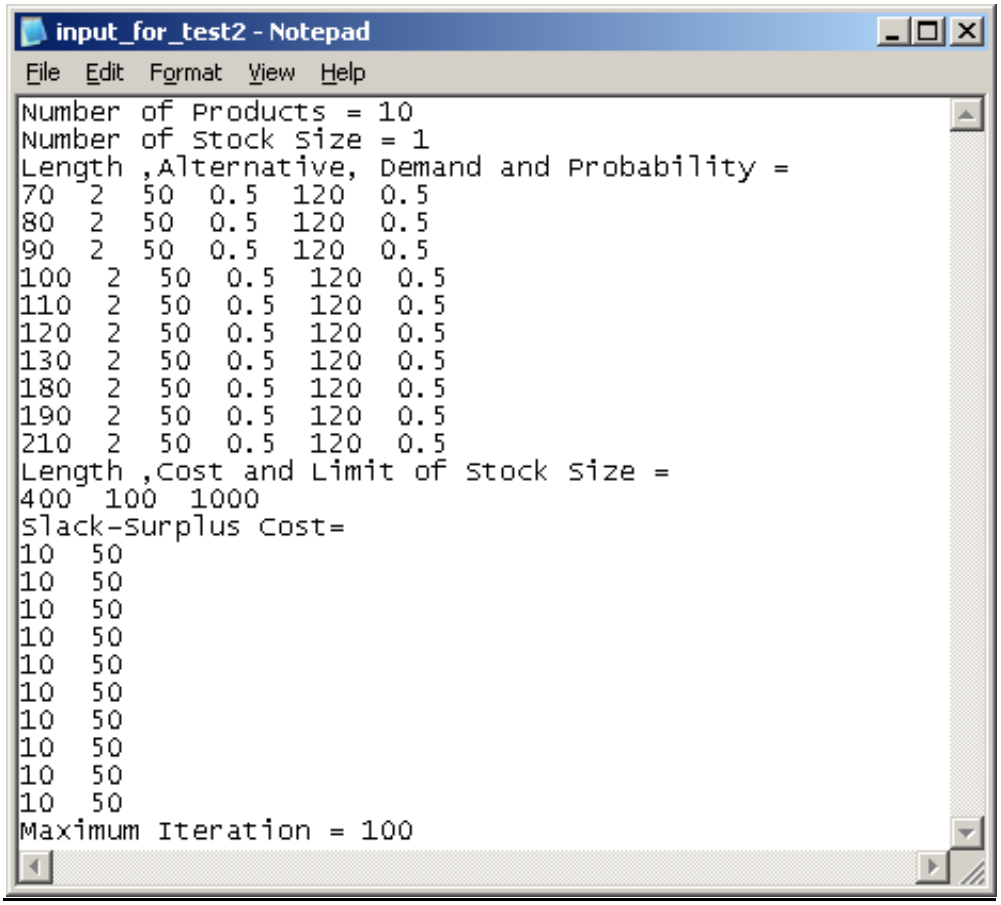
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

THE 2nd INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 120). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



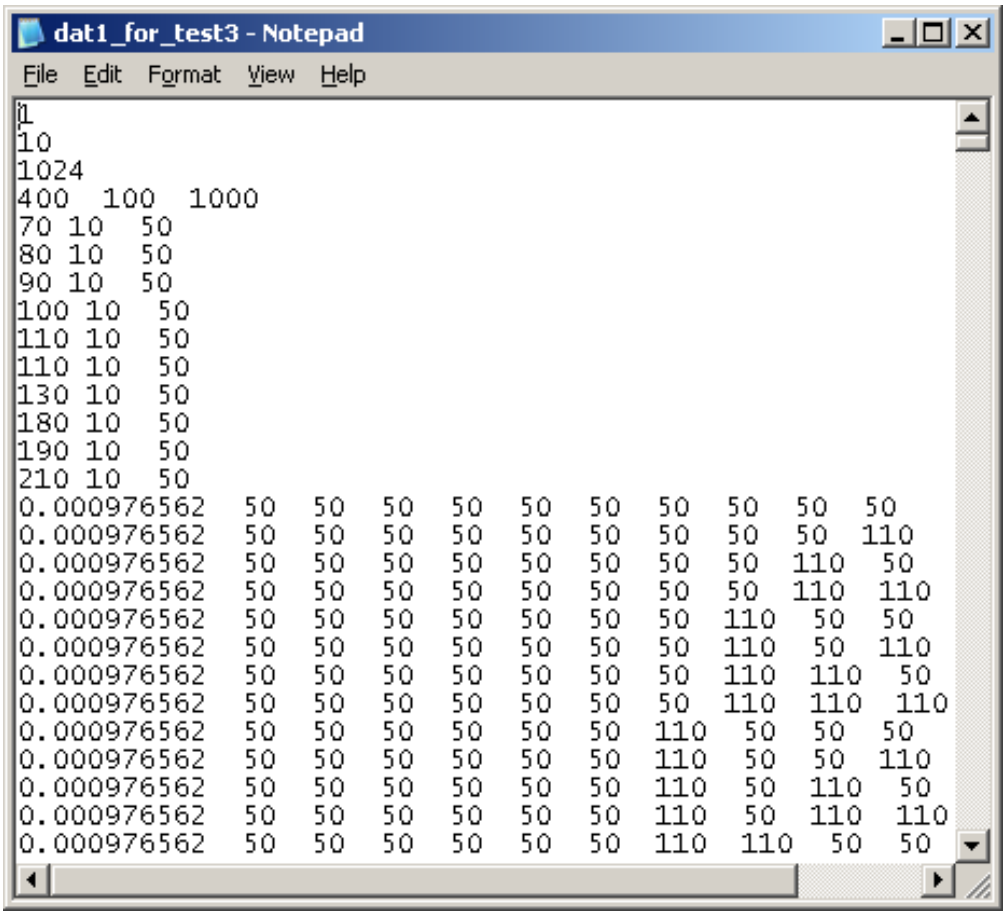
Data file for solving with the CGstochastic1.EXE or the column-generation procedure



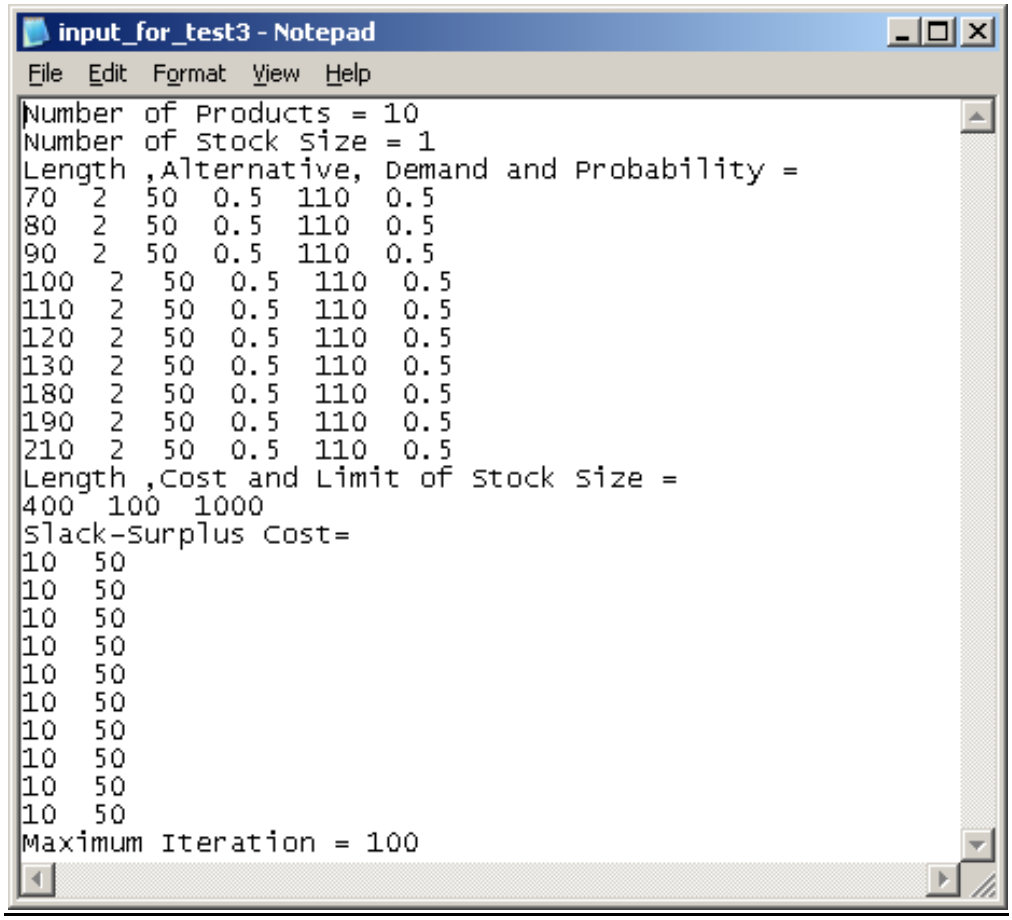
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

THE 3rd INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 110). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



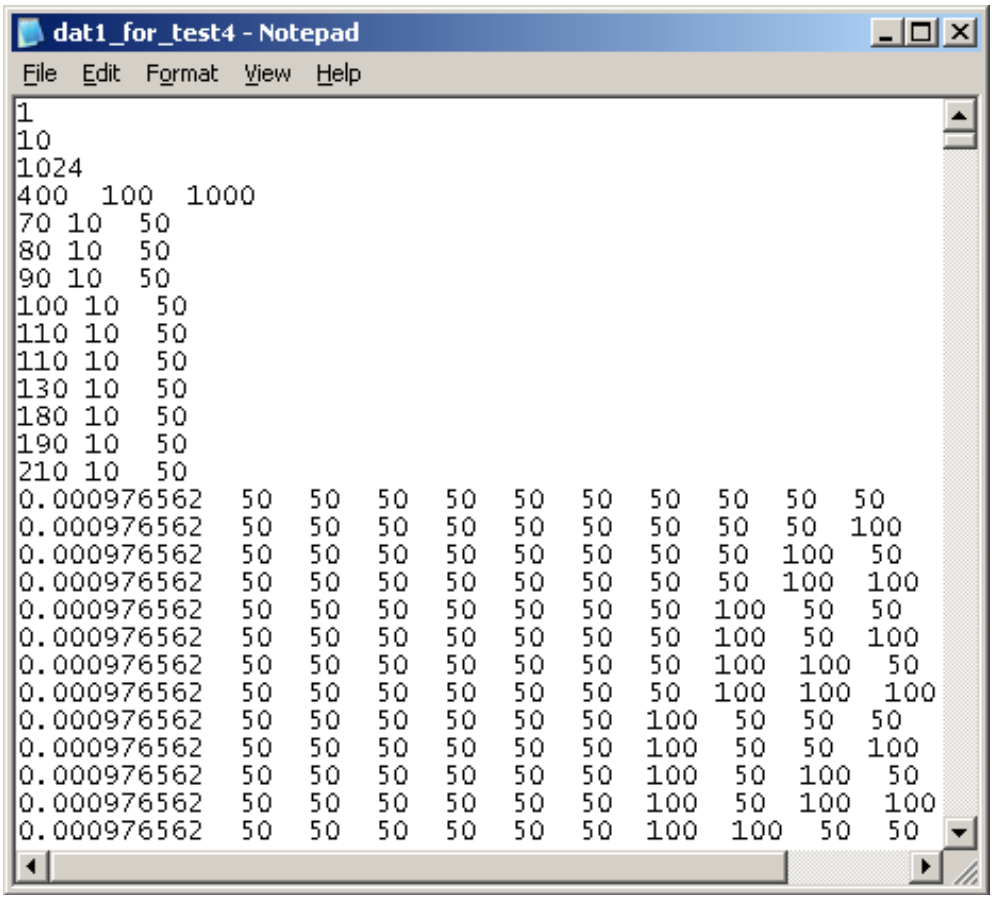
Data file for solving with the CGstochastic1.EXE or the column-generation procedure



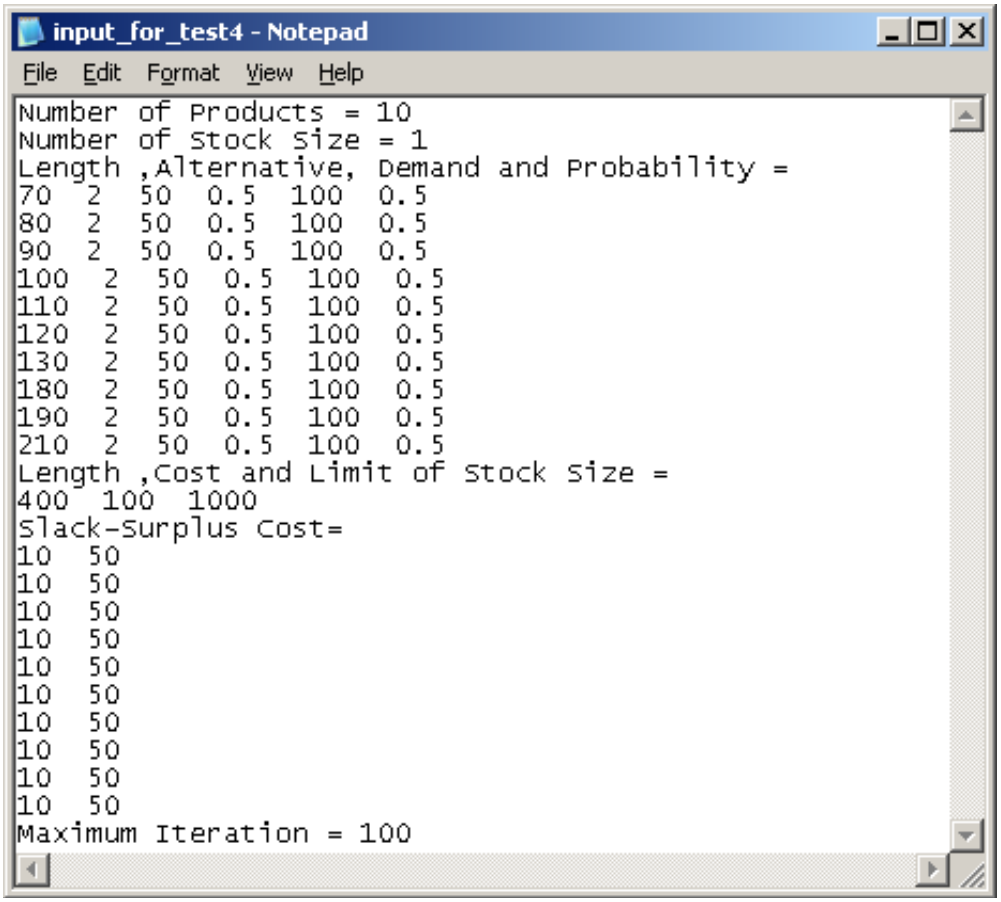
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

THE 4th INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 100). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



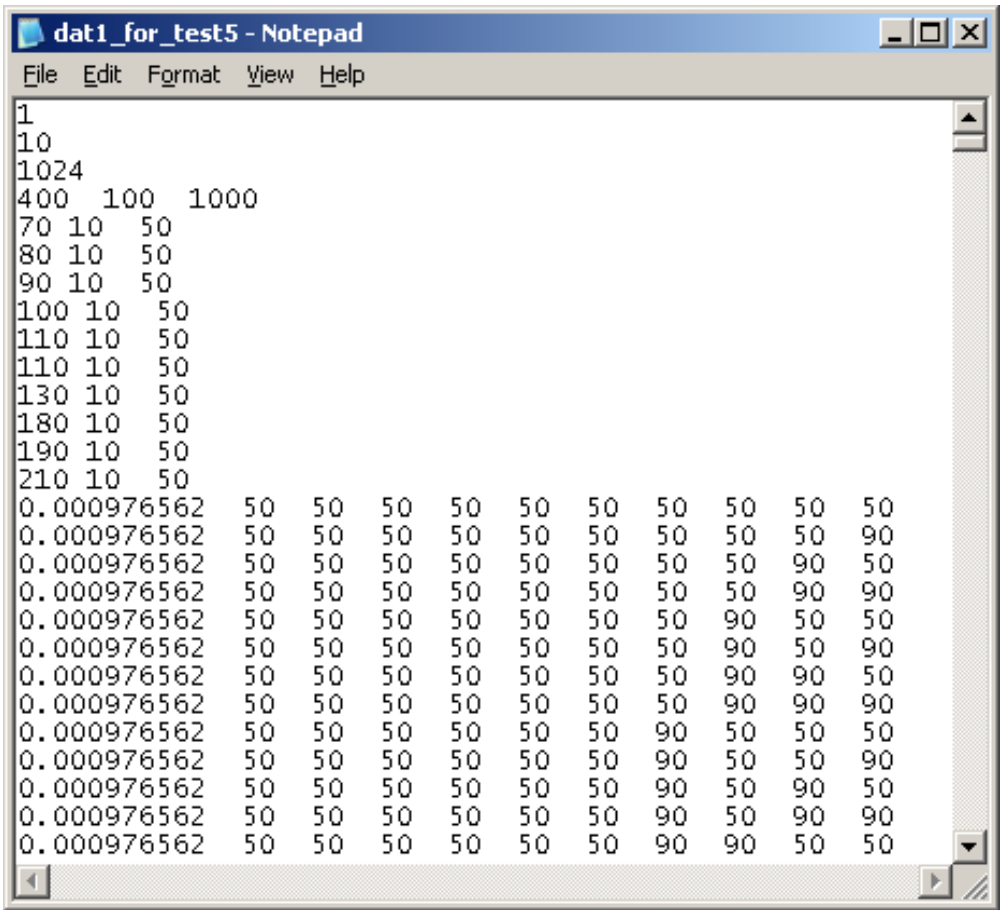
Data file for solving with the CGstochastic1.EXE or the column-generation procedure



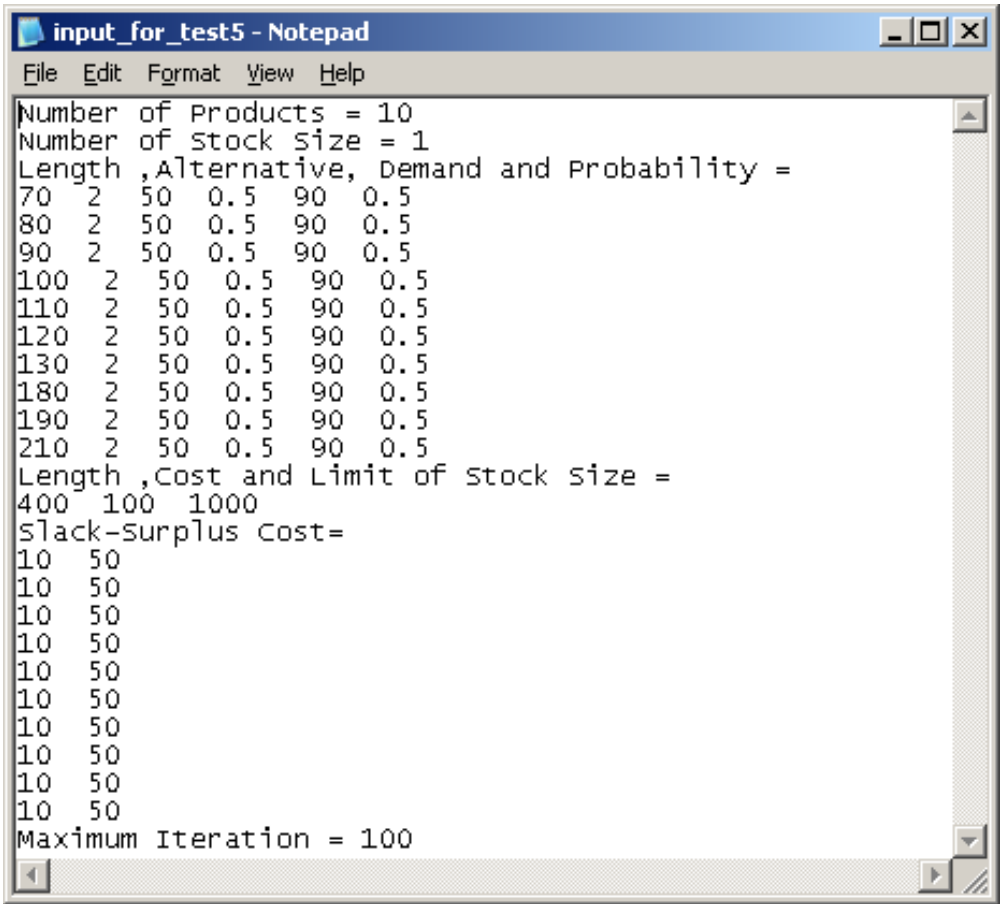
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

THE 5th INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 90). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



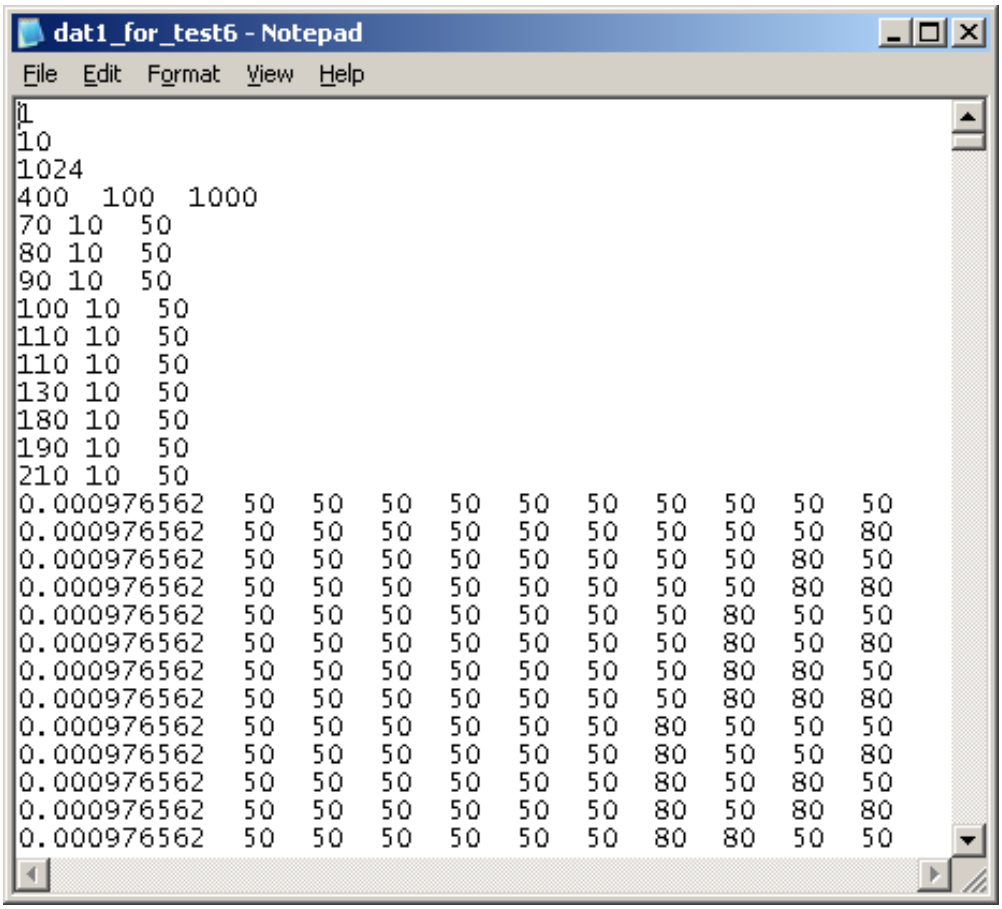
Data file for solving with the CGstochastic1.EXE or the column-generation procedure



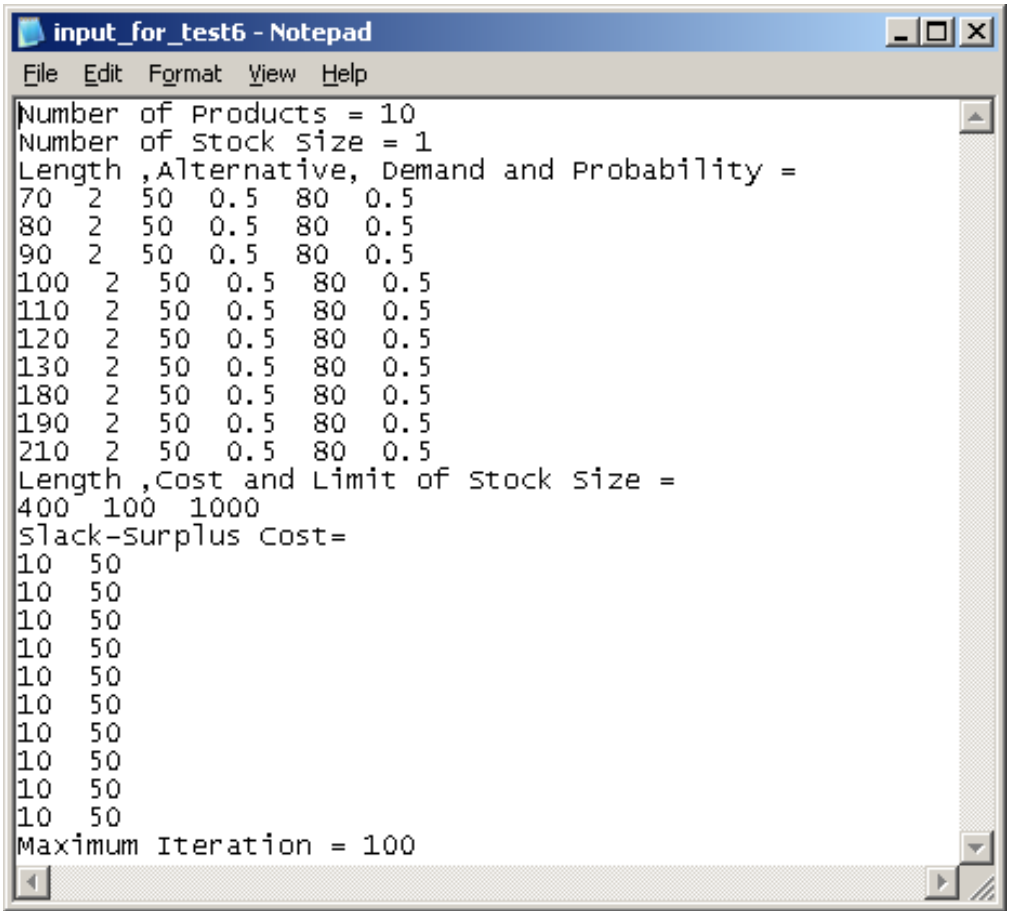
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

THE 6th INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 80). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



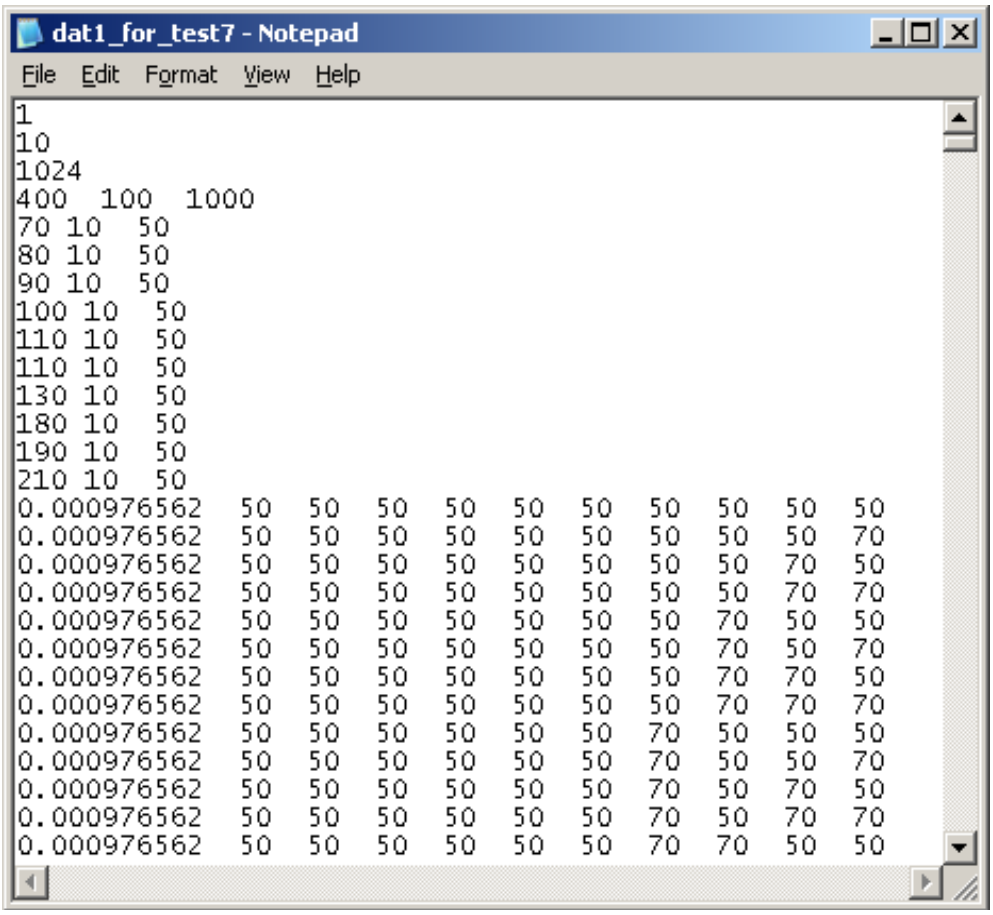
Data file for solving with the CGstochastic1.EXE or the column-generation procedure



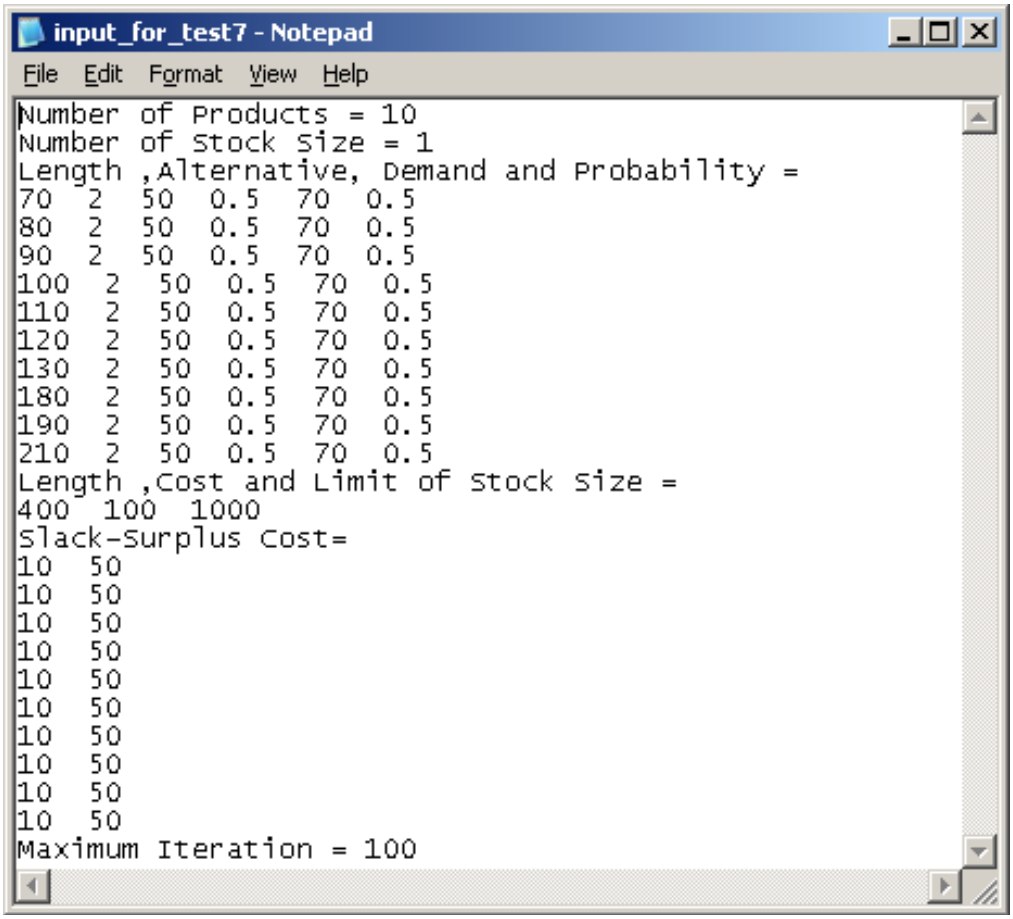
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

THE 7th INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 70). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



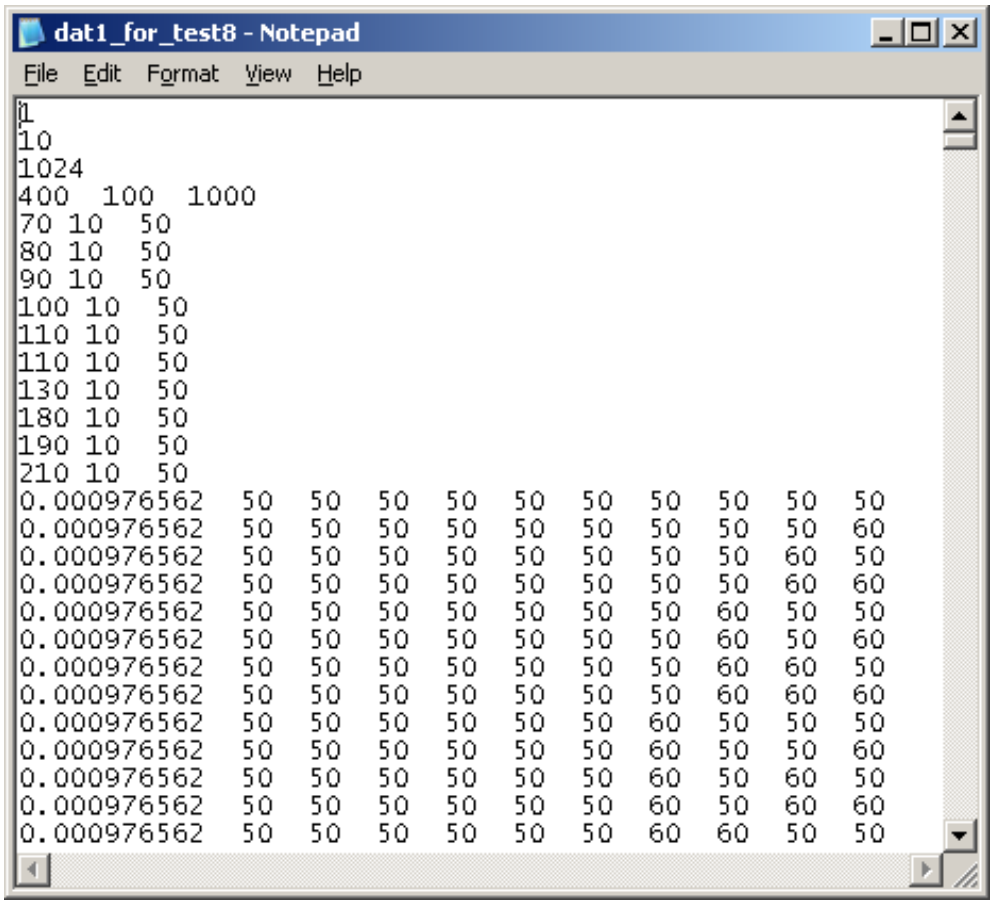
Data file for solving with the CGstochastic1.EXE or the column-generation procedure



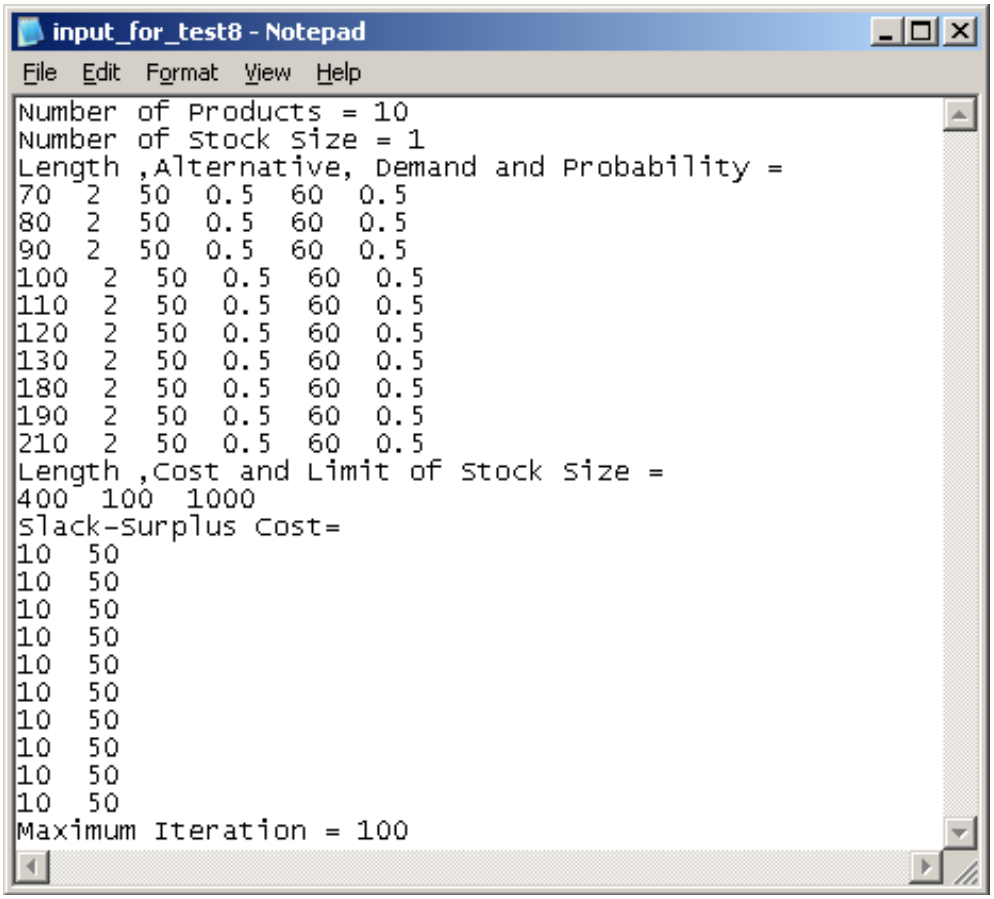
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

THE 8th INSTANCE

The number of items is 10; all items are 2 choices of demands with 50% of probability. All items have the same demands are such (50, 60). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.



Data file for solving with the CGstochastic1.EXE or the column-generation procedure



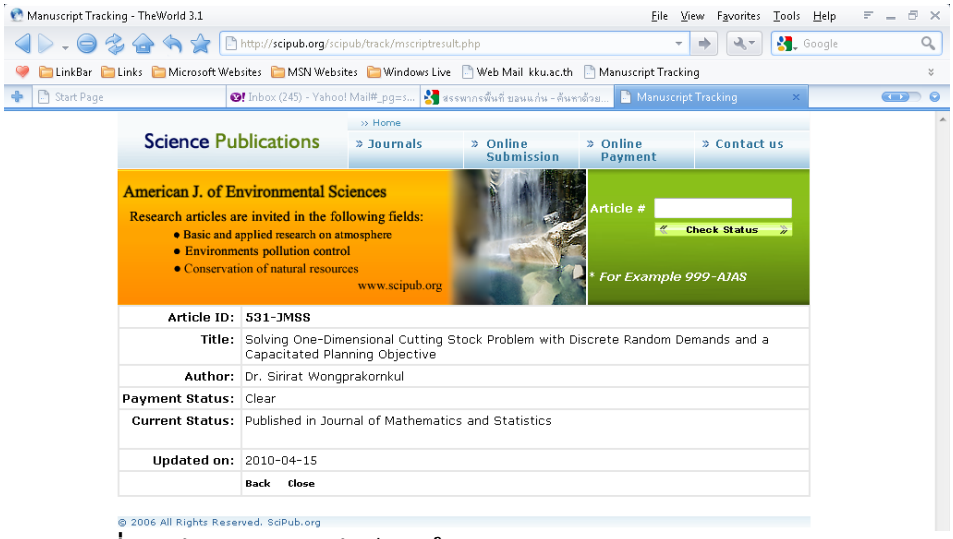
Data file for solving with the stochasticcutting1.EXE or the proposed heuristic

เอกสารแนบหมายเลข 3

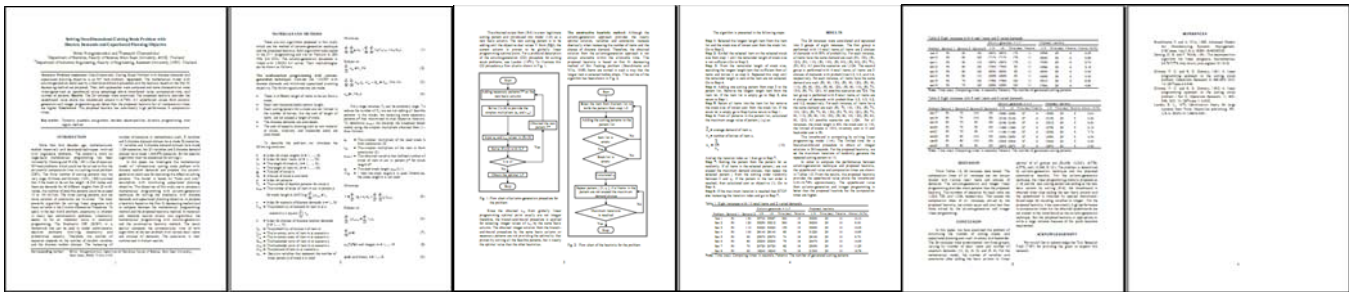
เอกสารแนบหมายเลข 3

Output จากโครงการวิจัยที่ได้รับทุนจาก สกว.

1. ผลงานตีพิมพ์ในวารสารวิชาการนานาชาติ (ระบุชื่อผู้แต่ง ชื่อเรื่อง ชื่อวารสาร ปี เล่มที่ เลขที่ และหน้า) หรือผลงานตามที่คาดไว้ในสัญญาโครงการ: ได้รับการตอบรับตีพิมพ์ผลงานใน Journal of Mathematics and Statistics ดังแสดงในรูปที่ 1. และ บทความที่ได้รับการพิจารณาตีพิมพ์ ดังแสดงตัวอย่างในรูปที่ 2. ทั้งนี้ ได้แนบเอกสารบทความที่ได้รับการตีพิมพ์ ดังแสดงในเอกสารแนบ ก.



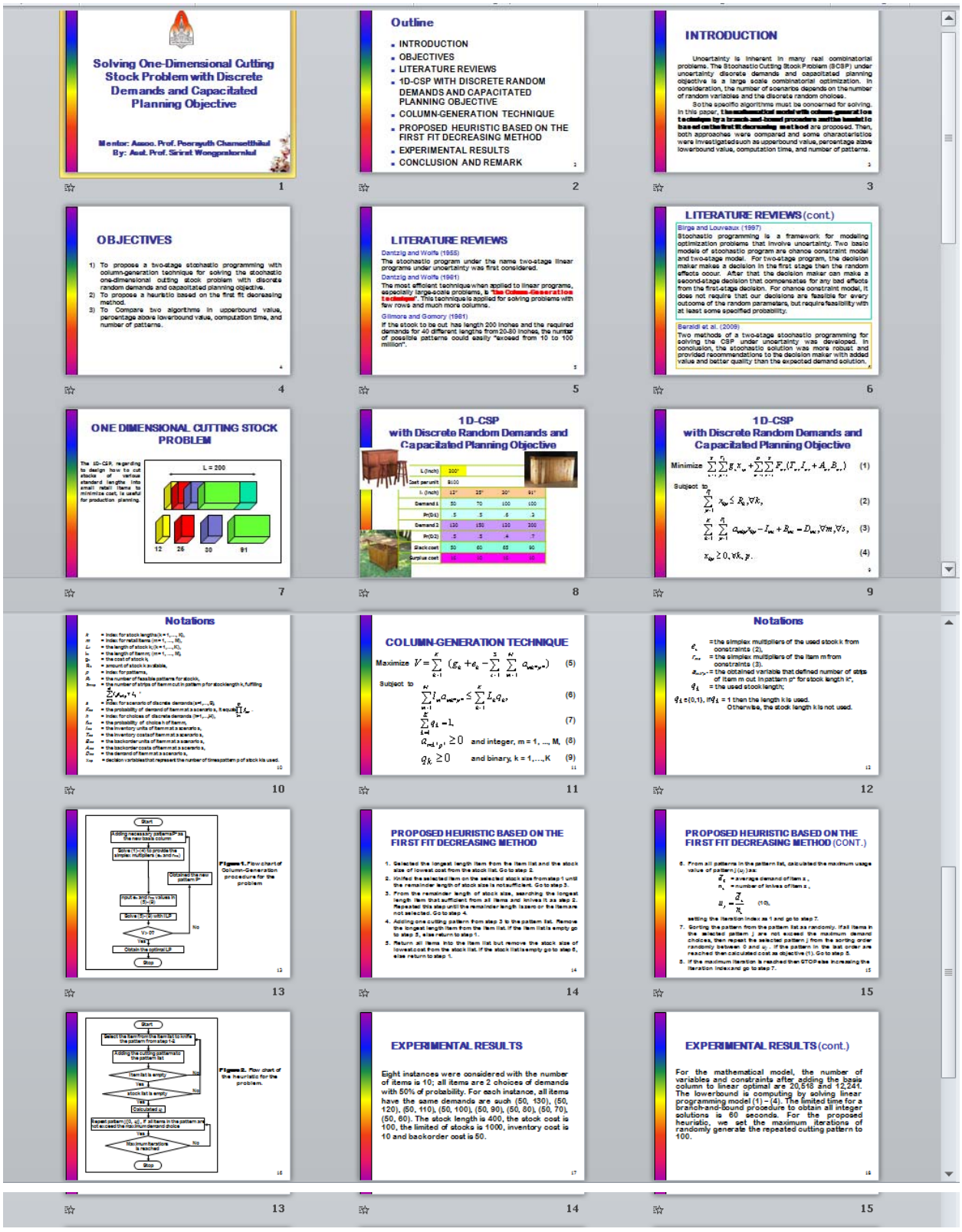
รูปที่ 1. หลักฐานการตอบรับตีพิมพ์ใน Journal of Mathematics and Statistics



รูปที่ 2. ตัวอย่างบทความที่ได้รับการพิจารณาตีพิมพ์

2. การนำผลงานวิจัยไปใช้ประโยชน์
   - เชิงวิชาการ นำองค์ความรู้เรื่อง One-Dimensional Cutting Stock Problem สอนให้นักศึกษาระดับปริญญาตรี นำไปประยุกต์ใช้กับปัญหาจริง เรื่อง การวางแผนการผลิตประตูรั้วสแตนเลส ด้วยการโปรแกรมเชิงเส้นจำนวนเต็ม กรณีศึกษา : ร้านเว้งอัลลอย (Productive planning of stainless steel gate by using integer linear programming: Case study Vaing Alloy Store) จัดทำโดย นางสาวกิ่งกมล นุ้ยห้วยแก้ว และนายกรวิชญ์ จันทร์ชูวงศ์ ภาควิชาสถิติ คณะวิทยาศาสตร์ มหาวิทยาลัยขอนแก่น ปีการศึกษา 2552

3. อื่นๆ ได้แก่ การเสนอผลงานแบบบรรยายในที่ประชุมนักวิจัยรุ่นใหม่ พบ เมธีวิจัยอาวุโส สกว. ครั้งที่ 9 ระว่างวันที่ 15-17 ตุลาคม 2552 โรงแรมฮิลิเดย์อินน์ รีสอร์ท รีเจนท์ บีช ชะอำ จังหวัดเพชรบุรี เรื่อง Solving One-Dimensional Cutting Stock Problem with Discrete Demands and Capacitated Planning Objective โดยมี PowerPoint ประกอบการนำเสนอ ดังนี้



Solving One-Dimensional Cutting Stock Problem with Discrete Demands and Capacitated Planning Objective

Mentor: Assoc. Prof. Peerayuth Charnsethikul
By: Asst. Prof. Sirirat Wongpaisarnsiri

1

Outline

- INTRODUCTION
- OBJECTIVES
- LITERATURE REVIEWS
- 1D-CSP WITH DISCRETE RANDOM DEMANDS AND CAPACITATED PLANNING OBJECTIVE
- COLUMN-GENERATION TECHNIQUE
- PROPOSED HEURISTIC BASED ON THE FIRST FIT DECREASING METHOD
- EXPERIMENTAL RESULTS
- CONCLUSION AND REMARK

2

INTRODUCTION

Uncertainty is inherent in many real combinatorial problems. The Stochastic Cutting Stock Problem (SCSP) under uncertainty discrete demands and capacitated planning objective is a large scale combinatorial optimization. In consideration, the number of scenarios depends on the number of random variables and the discrete random choices.

So the specific algorithms must be concerned for solving. In this paper, the multi-cut L-shaped model with column-generation technique by a branch-and-bound procedure and the heuristic based on the first fit decreasing sort tool are proposed. Then, both approaches were compared and some characteristics were investigated such as upperbound value, percentage above lowerbound value, computation time, and number of patterns.

3

OBJECTIVES

1) To propose a two-stage stochastic programming with column-generation technique for solving the stochastic one-dimensional cutting stock problem with discrete random demands and capacitated planning objective.
2) To propose a heuristic based on the first fit decreasing method.
3) To Compare two algorithms in upperbound value, percentage above lowerbound value, computation time, and number of patterns.

4

LITERATURE REVIEWS

Dantzig and Wolfe (1955)
The stochastic program under the name two-stage linear programs under uncertainty was first considered.

Dantzig and Wolfe (1961)
The most efficient technique when applied to linear programs, especially large-scale problems, is "the Column Generation technique". This technique is applied for solving problems with few rows and much more columns.

Gilmore and Gomory (1961)
If the stock to be cut has length 200 inches and the required demands for 40 different lengths from 20-80 inches, the number of possible patterns could easily "exceed from 10 to 100 million".

5

LITERATURE REVIEWS (cont.)

Birge and Louveaux (1997)
Stochastic programming is a framework for modeling optimization problems that involve uncertainty. Two basic models of stochastic program are chance constraint model and two-stage model. For two-stage program, the decision maker makes a decision in the first stage then the random effects occur. After that the decision maker can make a second-stage decision that compensates for any bad effects from the first-stage decision. For chance constraint model, it does not require that our decisions are feasible for every outcome of the random parameters, but require feasibility with at least some specified probability.

Beraldi et al. (2009)
Two methods of a two-stage stochastic programming for solving the CSP under uncertainty was developed. In conclusion, the stochastic solution was more robust and provided recommendations to the decision maker with added value and better quality than the expected demand solution.

6

ONE DIMENSIONAL CUTTING STOCK PROBLEM

The 1D-CSP, regarding to design how to cut stocks of various standard lengths into small retail items to minimize cost, is useful for production planning.

7

1D-CSP with Discrete Random Demands and Capacitated Planning Objective

8

1D-CSP with Discrete Random Demands and Capacitated Planning Objective

$$\text{Minimize} \sum_{s=1}^{S}\sum_{j=1}^{n} g_s x_{js} + \sum_{s=1}^{S}\sum_{i=1}^{m} F_{is}(I_{is}^{-}, I_{is}^{+}, A_{is}, B_{is}) \quad (1)$$

Subject to

$$\sum_{j=1}^{n} x_{js} \le R_s, \forall s, \quad (2)$$

$$\sum_{c=1}^{K}\sum_{j=1}^{n} a_{mcjs} x_{js} - I_{ms}^{-} + B_{ms} = D_{ms}, \forall m, \forall s, \quad (3)$$

$$x_{js} \ge 0, \forall k, p. \quad (4)$$

9

Notations

- k = Index: for stock lengths ($k = 1, \ldots, K$),
- m = Index: for retail items ($m = 1, \ldots, M$),
- L_k = the length of stock k ($k = 1, \ldots, K$),
- l_m = the length of item m; ($m = 1, \ldots, M$),
- g_k = the cost of stock k,
- R_s = amount of stock k available,
- p = index: for patterns,
- n = the number of feasible patterns for stock k,
- a_{mcjs} = the number of strips of item m cut in pattern p for stock length k, fulfilling $\sum_{m=1}^{M}(a_{mcjs} \cdot l_m) \le L_k$,
- s = index: for scenario of discrete demands ($s = 1, \ldots, S$),
- P_{ms} = the probability of demand of item m at a scenario s, it equals $\prod_{m=1}^{M} P_{ms}$,
- i = index: for choices of discrete demands ($i = 1, \ldots, M$),
- P_{mi} = the probability of choice i of item m,
- I_{ms}^{-} = the inventory units of item m at a scenario s,
- T_{ms} = the inventory cost of item m at a scenario s,
- B_{ms} = the backorder units of item m at a scenario s,
- A_{ms} = the backorder costs of item m at a scenario s,
- D_{ms} = the demand of item m at a scenario s,
- x_{js} = decision variables that represent the number of times pattern p of stock k is used.

10

COLUMN-GENERATION TECHNIQUE

$$\text{Maximize } V = \sum_{k=1}^{K}(g_k + e_k - \sum_{c=1}^{3}\sum_{m=1}^{N} a_{mcjs} r_{mc}) \quad (5)$$

Subject to

$$\sum_{m=1}^{N} l_m a_{mcjs} \le \sum_{k=1}^{K} L_k q_k, \quad (6)$$

$$\sum_{k=1}^{K} q_k = 1, \quad (7)$$

$$a_{mcjs} \ge 0 \text{ and integer, } m = 1, \ldots, M, \quad (8)$$

$$q_k \ge 0 \text{ and binary, } k = 1, \ldots, K \quad (9)$$

11

Notations

- e_c = the simplex multipliers of the used stock k from constraints (2),
- r_{mc} = the simplex multipliers of the item m from constraints (3),
- a_{mcjs} = the obtained variable that defined number of strips of item m cut in pattern p for stock length k,
- q_k = the used stock length;

$q_k \in \{0,1\}$, if $q_k = 1$ then the length k is used. Otherwise, the stock length k is not used.

12

Figure 1. Flow chart of Column-Generation procedure for the problem

13

PROPOSED HEURISTIC BASED ON THE FIRST FIT DECREASING METHOD

1. Selected the longest length item from the item list and the stock size of lowest cost from the stock list. Go to step 2.
2. Knifed the selected item on the selected stock size from step 1 until the remainder length of stock size is not sufficient. Go to step 3.
3. From the remainder length of stock size, searching the longest length item that sufficient from all items and knifes it as step 2. Repeated this step until the remainder length is zero or the item are not selected. Go to step 4.
4. Adding one cutting pattern from step 3 to the pattern list. Remove the longest length item from the item list. If the item list is empty go to step 5, else return to step 1.
5. Return all items into the item list but remove the stock size of lowest cost from the stock list. If the stock list is empty go to step 6, else return to step 1.

14

PROPOSED HEURISTIC BASED ON THE FIRST FIT DECREASING METHOD (CONT.)

6. From all patterns in the pattern list, calculated the maximum usage value of pattern j (u_j) as:

$\bar{d}_i$ = average demand of item i,
n_{ij} = number of knives of item i,

$$u_j = \frac{\bar{d}_i}{n_{ij}} \quad (10),$$

setting the iteration index as 1 and go to step 7.

7. Sorting the pattern from the pattern list as randomly. If all items in the selected pattern j are not exceed the maximum demand choices, then repeat the selected pattern j from the sorting order randomly between 0 and u_j. If the pattern in the last order are reached then calculated cost as objective (1). Go to step 8.
8. If the maximum iteration is reached then STOP else increasing the iteration index and go to step 7.

15

Figure 2. Flow chart of the heuristic for the problem.

13

EXPERIMENTAL RESULTS

Eight instances were considered with the number of items is 10; all items are 2 choices of demands with 50% of probability. For each instance, all items have the same demands are such (50, 130), (50, 120), (50, 110), (50, 100), (50, 90), (50, 80), (50, 70), (50, 60). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.

14

EXPERIMENTAL RESULTS (cont.)

For the mathematical model, the number of variables and constraints after adding the basis column to linear optimal are 20,516 and 12,241. The lowerbound is computing by solving linear programming model (1) – (4). The limited time for a branch-and-bound procedure to obtain all integer solutions is 60 seconds. For the proposed heuristic, we set the maximum iterations of randomly generate the repeated cutting pattern to 100.

15

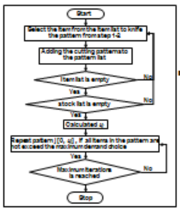


Figure 2. Flow chart of the heuristic for the problem.

EXPERIMENTAL RESULTS

Eight instances were considered with the number of items is 10; all items are 2 choices of demands with 50% of probability. For each instance, all items have the same demands are such (50, 130), (50, 120), (50, 110), (50, 100), (50, 90), (50, 80), (50, 70), (50, 60). The stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.

EXPERIMENTAL RESULTS (cont.)

For the mathematical model, the number of variables and constraints after adding the basis column to linear optimal are 20,518 and 12,241. The lowerbound is computing by solving linear programming model (1) – (4). The limited time for a branch-and-bound procedure to obtain all integer solutions is 60 seconds. For the proposed heuristic, we set the maximum iterations of randomly generate the repeated cutting pattern to 100.

Table 1. The results from the column-generation techniques and the proposed heuristic.

Problem	Demand 1	Demand 2	Column Generation + ILP				Proposed heuristic			
			LB	UB	time(s)	pattern	UB	time(s)	pattern	% above LB
Test 1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test 2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test 3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test 4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test 5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test 6	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test 7	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Test 8	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Note: time(s) = computing times in seconds.
pattern = the generated cutting patterns.

CONCLUSION AND REMARK

In this paper, we have examined the problem of minimizing the number of cutting stocks and capacitated planning cost such inventory and backorder. The problem is determined by column-generation technique and the proposed constructive heuristic. The proposed heuristic has consistently high performance in computation times but the obtained upperbounds are not closed to the lowerbound as the column-generation technique. However the proposed heuristic is appropriate to solve a large instance because of the quick response requirement.

Be not afraid of growing slowly,
Be afraid only of standing still.

THANK YOU

<u>เอกสารแนบ ก.</u>

บทความที่ได้รับการพิจารณาตีพิมพ์

Journal of Mathematics and Statistics 6 (2): 79-83, 2010
ISSN 1549-3644

Solving One-Dimensional Cutting Stock Problem with Discrete Demands and Capacitated Planning Objective

[1]Sirirat Wongprakornkul and [2]Peerayuth Charnsethikul
[1]Department of Statistics, Faculty of Science, Khon Kaen University, 40002, Thailand
[2]Department of Industrial Engineering, Faculty of Engineering,
Operations Research and Management Science Units, Kasetsart University, 10900, Thailand

Abstract: Problem statement: One-dimensional cutting stock problem with discrete demands and capacitated planning objective is an NP hard problem. **Approach:** The mathematical model with column-generation technique by a branch-and-bound procedure and the heuristic based on the first fit decreasing method are proposed. Then, both approaches were compared and some characteristics were investigated such as upper-bound value, percentage above lower-bound value, computation time, and number of patterns. **Results:** The 24 instances were examined. The proposed heuristic provides the upper-bound value above the lower-bound around 0-16.78%. All upper-bound values from column-generation and integer programming are better than the proposed heuristic but all computation times are higher. **Conclusion:** The proposed heuristic has consistently high performance in computation times.

Key words: Stochastic integer linear programming, cutting stock problems, column-generation technique, heuristics

INTRODUCTION

More than four decades ago, mathematicians studied intensively and developed techniques involved with large-scale problems. The extensive work on large-scale mathematical programming has been initiated by (Dantzig and Wolfe, 1961). One of classical NP-hard problems which could not be solved within the polynomial computation time is a cutting stock problem (CSP). The finite number of cutting patterns may be very large. Gilmore and Gomory (1961; 1963) claimed that if the stock to be cut has length l = 200 inches and there are demands for 40 different lengths from 20-80 inches, the number of possible patterns could be exceed 10-100 million. The more cutting patterns and the more columns of constraints are involved. The most powerful algorithm for solving linear programs with many columns is the Column-Generation Procedure. To apply in the real world problem, uncertainty is inherent in many real combinatorial problems. Uncertainty seems to be an important issue in constraint programming. The Stochastic CSP (SCSP) is a framework that can be used to model combinatorial decision problems involving uncertainty and probabilities recently. Therefore, the number of scenarios depends on the number of random variables and the discrete random choices. The increasing of number of scenarios is tremendously such, 5 variables and 2 discrete demand choices have made 32 scenarios, 10 variables and 2 discrete demand choices have made 1,024 scenarios, but 20 variables and 2 discrete demand choices have made 1,048,576 scenarios. So the specific algorithms must be concerned for solving it.

In this study, we investigate the mathematical model of 1-dimensional cutting stock problem with discrete random demands and propose the column-generation technique for searching the effective cutting patterns. The model is based on "here and now" assumption with minimizing capacitated planning objective. The objectives of this study are to propose a mathematical programming with column-generation technique for solving the stochastic with discrete demands and capacitated planning objective, to propose a heuristic based on the first fit decreasing method and to compare between the mathematical programming method and the proposed heuristic method. In materials and methods shows two algorithms; the mathematical programming with column-generation and the constructive heuristic method. The result compares the computational time of both algorithms on the test problem with various retail items and choices of demands. Sequentially, conclusions are drawn.

MATERIALS AND METHODS

There are two algorithms proposed in this study which are the method of column-generation technique and the proposed heuristic. Both algorithms were coded

Corresponding Author: Sirirat Wongprakornkul, Department of Statistics, Faculty of Science, Khon Kaen University, Khon Kaen, 40002, Thailand

in the C++ programming and run on Pentium 4, 256 MB, 2.4 GHz. The column-generation procedure is linked with LINGO 6.0 solver. Their methodologies can be shown as follows:

The mathematical programming with column-generation technique: Consider the 1D-CSP with discrete demands and minimized capacitated planning objective. The following assumptions are made:

- There is different length of items to be cut from a stock
- Each item has associated a certain length
- Each cutting pattern for a stock are not limited in the number of knives, but the sum of length of items are not exceed a length of stock
- The discrete demands are considered
- The cost of capacity planning such as raw material or stock, inventory and backorder costs are considered

To describe the problem, we introduce the following notations:

k = Index for stock lengths ($k = 1, \ldots, K$)
m = Index for retail items ($m = 1, \ldots, M$)
L_k = The length of stock k; ($k = 1, \ldots, K$)
l_m = The length of item m; ($m = 1, \ldots, M$)
g_k = The cost of stock k
R_k = Amount of stock k available
p = Index for patterns
P_k = The number of feasible patterns for stock k
a_{mkp} = The number of strips of item m cut in pattern p for stock length k, fulfilling $\sum_{m=1}^{M} l_m a_{mkp} \le L_k$
s = Index for scenario of discrete demands ($s = 1,., S$)
F_{ms} = The probability of demand of item m at a scenario s, it equals $\prod_{h=1}^{H} f_{hm}$
h = Index for choices of discrete random demands ($h = 1,\ldots, H$)
f_{hm} = The probability of choice h of item m
I_{ms} = The inventory units of item m at scenario s
T_{ms} = The inventory costs of item m at scenario s
B_{ms} = The backorder units of item m at scenario s
A_{ms} = The backorder units of item m at scenario s
D_{ms} = The demand of item m at scenario s
x_{kp} = Decision variables that represent the number of times pattern p of stock k is used

Minimize:

$$\sum_{k=1}^{K}\sum_{p=1}^{Pk} g_k x_{kp} + \sum_{m=1}^{M}\sum_{s=1}^{S} F_{ms}(T_{ms}I_{ms} + A_{ms}B_{ms}) \quad (1)$$

Subject to:

$$\sum_{p=1}^{P_k} x_{kp} \le R_k \ \forall k \quad (2)$$

$$\sum_{k=1}^{K}\sum_{p=1}^{P_k} a_{mkp}x_{kp} - I_{ms} + B_{ms} = D_{ms}, \ \forall m, \forall s \quad (3)$$

$$x_{kp} \ge 0, \ \forall k, p \quad (4)$$

For a large instance, P_k can be extremely large. To reduce the number of P_k, we are not adding all feasible patterns in the model, but selecting some necessary patterns p* that maximized its dual objective function. To determine a_{mk*p*} we develop the knapsack based model using the simplex multipliers obtained from (1-4) as follows:

e_k = The simplex multipliers of the used stock k from constraints (2)
r_{ms} = The simplex multipliers of the item m from constraints (3)
a_{mk*p*} = The obtained variable that defined number of strips of item m cut in pattern p* for stock length k*
q_k = The used stock length; $q_k \in \{0,1\}$
if q_k = 1 then the stock length k is used. Otherwise, the stock length k is not used

Minimize:

$$V = \sum_{k=1}^{K} (g_k - e_k)q_k - \sum_{s=1}^{S}\sum_{m=1}^{M} r_{ms}a_{mk*p*} \quad (5)$$

Subject to:

$$\sum_{m=1}^{M} l_m a_{mk*p*} \le \sum_{k=1}^{K} L_k q_k \quad (6)$$

$$\sum_{k=1}^{K} q_k = 1 \quad (7)$$

$a_{mk*p*} \ge 0$ and integer, $m = 1, ..., M$ (8)

$q_k \ge 0$ and binary, $k = 1,...,K$ (9)

The obtained output from (5-9) is a new legitimate cutting pattern and introduced into model (1-4) as a

new basis column. The new cutting pattern is to be adding until the objective dual values V from (5)≥0, the current column is proven to be globally linear programming optimal point. For a profound description of the column-generation (CG) procedure for cutting stock problems, Lasdon (1970). To illustrate this CG procedure, a flow chart is shown in Fig. 1.

Since the obtained x_{kp} from globally linear programming optimal point usually are not integer therefore, the branch-and-bound procedure is applied for selecting integer values of x_{kp} by the same basis column. The obtained integer solution from the branch-and-bound procedure by the same basis column or necessary patterns are not providing the optimality that proven by solving all the feasible patterns, but it nearly the optimal value than the other heuristics.

The constructive heuristic method: Although the column-generation approach provides the nearly optimal solution, variables and constraints increase drastically when increasing the number of items and the chance of discrete demand. Therefore, the obtained solution from the column-generation approach is not always attainable within the allowable time.

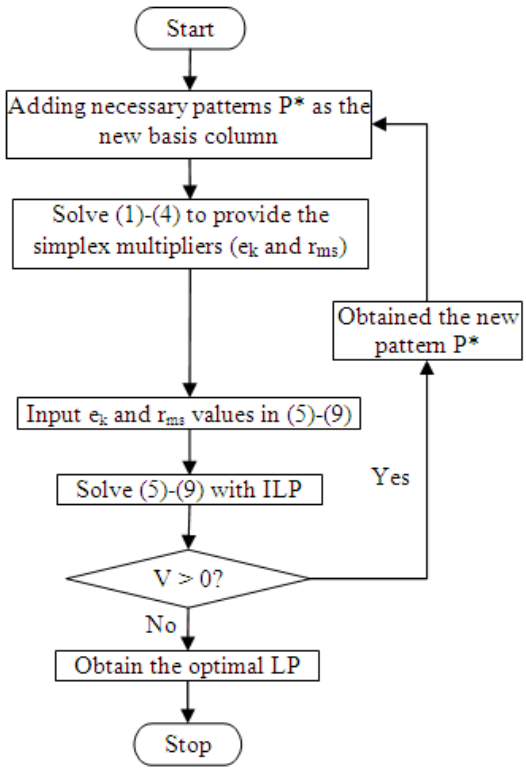


Fig. 1: Flow chart of column-generation procedure for the problem

The proposed heuristic is based on first fit decreasing method of Bin Packing problem (Brandimarte and Villa, 1995). Items are sorted in such a way that the longer item is selected before others. The outline of the algorithm has been shown in Fig. 2.

The algorithm is presented in the following steps:

Step 1: Selected the longest length item from the item list and the stock size of lowest cost from the stock list. Go to Step 2

Step 2: Knifed the selected item on the selected stock size from step 1 until the remainder length of stock size is not sufficient. Go to Step 3

Step 3: From the remainder length of stock size, searching the longest length item that sufficient from all items and knives it as step 2. Repeated this step until the remainder length is zero or the item are not selected. Go to Step 4

Step 4: Adding one cutting pattern from step 3 to the pattern list. Remove the longest length item from the item list. If the item list is empty go to Step 5, else return to Step 1

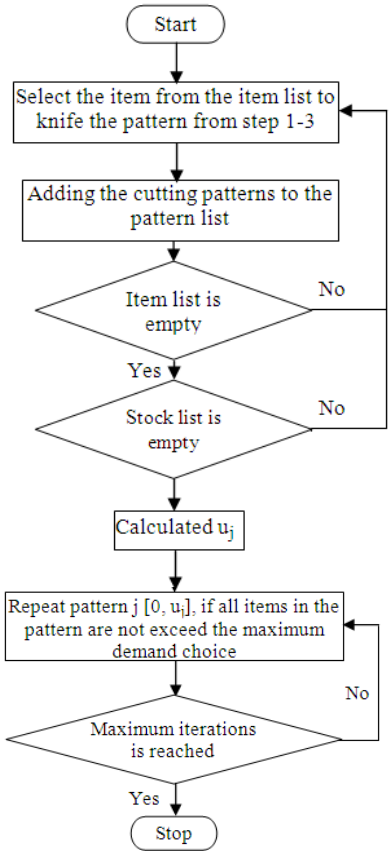


Fig. 2: Flow chart of the heuristic for the problem

Step 5: Return all items into the item list but remove the stock size of lowest cost from the stock list. If the stock list is empty go to Step 6, else return to Step 1

Step 6: From all patterns in the pattern list, calculated the maximum usage value of pattern j (u_j) as:

$\bar{d}_z$ = Average demand of item z

n_z = Number of knives of item z

$$u_j = \frac{\bar{d}_z}{n_z} \quad (10)$$

And set the iteration index as 1 then go to Step 7.

Step 7: Sorting the pattern from the pattern list as randomly. If all items in the selected pattern j are not exceed the maximum demand choices, then repeat the selected pattern j from the sorting order randomly between 0 and u_j. If the pattern in the last order is reached, then calculated cost as objective (1). Go to Step 8

Step 8: If the maximum iteration is reached then STOP else increasing the iteration index and go to Step 7

RESULTS

The 24 instances were considered and separated into 3 groups of eight instances. The first group is performed with 10 retail items; all items are 2 choices of demands with 50% of probability. For each instance, all items have the same demand are such (50, 130), (50, 120), (50, 110), (50, 100), (50, 90), (50, 80), (50, 70), (50, 60). All possible scenarios are 1,024. The second group is performed with 6 retail items; all items are 3 choices of demands with probabilities 0.3, 0.3, and 0.4, respectively. For each instance, all items having the same demand are such (50, 90, 130), (50, 90, 120), (50, 90, 110), (50, 80, 130), (50, 80, 120), (50, 80, 110), (50, 70, 130), (50, 70, 120). All possible scenarios are 729. The last group is performed with 5 retail items; all items have 4 choices of demands with probabilities 0.3, 0.3, 0.2, and 0.2, respectively. For each instance, all items have the same demand are such (50, 70, 100, 130), (50, 70, 100, 120), (50, 70, 90, 130), (50, 70, 90, 120), (50, 70, 90, 110), (50, 80, 100, 130), (50, 80, 100, 120), (50, 80, 90, 120). All possible scenarios are 1,024. For all instances, the stock length is 400, the stock cost is 100, the limited of stocks is 1000, inventory cost is 10 and backorder cost is 50.

The lower-bound is computed by solving linear programming model (1-4). The limited time for a branch-and-bound procedure to obtain all integer solutions is 360 seconds. For the proposed heuristic, we set the maximum iterations of randomly generate the repeated cutting pattern to 10.

In order to compare the performance between column-generation technique and proposed heuristic, the upper-bound value and computation times are shown in Table 1-3. From the results, the proposed heuristic provides the upper-bound value above the lower-bound 0.00-16.78% approximately. The upper-bound value from column-generation and integer programming is better than the proposed heuristic but the computation times are higher.

Table 1: Eight instances with 10 retail items and 2 varied demands

			Column generation + ILP				Proposed heuristic			
Problem	Demand 1	Demand 2	UB	LB	Time (sec)	Patterns	UB	Time (sec)	Patterns	Above LB (%)
Test 1	50	130	35700	35687	360	19	39290	29	10	10.10
Test 2	50	120	33250	33212	360	18	36200	29	10	9.00
Test 3	50	110	30620	30620	153	19	33690	29	10	10.03
Test 4	50	100	28140	28140	83	18	31220	29	10	10.95
Test 5	50	90	25670	25670	74	18	28160	29	10	9.70
Test 6	50	80	23200	23200	79	18	26470	29	10	14.09
Test 7	50	70	20720	20720	82	18	23050	29	10	11.25
Test 8	50	60	18240	18240	76	18	21300	29	10	16.78

Note: Time (sec): Computing times in seconds; Patterns: The number of generated cutting patterns

Table 2: Eight instances with 6 retail items and 3 varied demands

				Column generation + ILP				Proposed heuristic			
Problem	Demand 1	Demand 2	Demand 3	UB	LB	Time (sec)	Patterns	UB	Time (sec)	Patterns	Above LB (%)
Test 9	50	90	130	18374	18374	179	11	19596	29	6	6.65
Test 10	50	90	120	17174	17174	179	11	18350	29	6	6.85
Test 11	50	90	110	15974	15974	179	11	17050	29	6	6.74
Test 12	50	80	130	17940	17940	33	11	19586	29	6	9.18
Test 13	50	80	120	18034	18034	33	11	18034	29	6	0.00
Test 14	50	80	110	16656	16656	33	11	16656	29	6	0.00
Test 15	50	70	130	17560	17560	120	11	19010	29	6	8.26
Test 16	50	70	120	16360	16360	120	11	17964	29	6	9.80

Note: Time (sec): Computing times in seconds; Patterns: The number of generated cutting patterns

Table 3: Eight instances with 5 retail items and 4 varied demands

					Column generation + ILP				Proposed heuristic			
Problem	Demand 1	Demand 2	Demand 3	Demand 4	UB	LB	Time (sec)	Patterns	UB	Time (sec)	Patterns	Above LB (%)
Test 17	50	70	100	130	12686	12686	37	9	13956	29	5	10.01
Test 18	50	70	100	120	13192	13192	37	9	13192	29	5	0.00
Test 19	50	70	90	130	12186	12186	37	9	13312	29	5	9.24
Test 20	50	70	90	120	11686	11686	37	9	12756	29	5	9.16
Test 21	50	70	90	110	11186	11186	37	9	12052	29	5	7.74
Test 22	50	80	100	130	13950	13950	37	9	13950	29	5	0.00
Test 23	50	80	100	120	13306	13306	37	9	13306	29	5	0.00
Test 24	50	80	90	120	12870	12870	37	9	12870	29	5	0.00

Note: Time (sec): Computing times in seconds; Patterns: The number of generated cutting patterns

DISCUSSION

From Table 1-3, 24 instances were tested. The computation times of all instances are not always related with various retail items and choices of demands. The column-generation and integer linear programming provides more patterns than the proposed heuristic. The numbers of scenarios for each table are 1,024, 729, and 1,024, respectively. For the mathematical model, the number of variables and constraints after adding the basis column to linear optimal of all groups are (20,498, 10,241), (8,759, 4,375), and (10,249, 5,121) respectively. That causes the computation times of all instances, solved by the proposed heuristic, are almost equal and also less than those solved by the column-generation and integer linear programming.

CONCLUSION

In this study, we have examined the problem of minimizing the number of cutting stocks and capacitated planning cost such inventory and backorder. The 24 instances were experimented into three groups, varying by number of retail items and number of uncertain demands; (10, 2), (6, 3), and (5, 4). All test problems are solved by column-generation technique and the proposed constructive heuristic. For the column-generation technique, the linear programming model is proposed as (1-4) and the new cutting pattern are adding as the new basis column by solving (5-9), the lower-bound is obtained when stop adding the new basis column and the upper-bound is obtained by applied branch-and-bound steps for rounding variables to integer. For the proposed heuristic, it has consistently high performance in computation times but the obtained upper-bounds are not closed to the lower-bound as the column-generation technique. Nevertheless, the proposed heuristic should be more appropriate to solve a large instance because of the quick response requirement.

ACKNOWLEDGEMENT

We would like to acknowledge the Thailand Research Fund (TRF) for providing the grant to support this research.

REFERENCES

Brandimarte, P. and A. Villa, 1995. Advanced Models for Manufacturing Systems Management. 1st Edn., CRC Press, Inc., USA., ISBN: 0849383323.

Dantzig, G.B. and P. Wolfe, 1961. The decomposition algorithm for linear programs. Econometrica, 29: 767-778. http://www.jstor.org/pss/1911818

Gilmore, P.C. and R.E. Gomory, 1961. A linear programming approach to the cutting stock problem. Operat. Res., 9: 849-859. DOI: 10.1287/opre.9.6.849

Gilmore, P.C. and R.E. Gomory, 1963. A linear programming approach to the cutting stock problem-part II. Operat. Res., 11: 863-888. DOI: 10.1287/opre.11.6.863

Lasdon, S.L., 1970. Optimization Theory for Large Systems. 1st Edn., Macmillan Publishing, New York, USA., ISBN: 10: 0486419991.