HARD PROBLEM GENERATION FOR MKP

María A. Osorio[1] and Germán Cuaya
School of Computer Sciences, Universidad Autónoma de Puebla. Ciudad Universitaria, Puebla 72560, México

ABSTRACT
We developed generators that produce challenging MKP instances. Our approaches uses independently exponential distributions over a wide range to generate the constraint coefficients, and the corresponding average for each variable is used to calculate directly correlated coefficients in the objective function. RHS values are a percentage of the sum of constraint coefficients. We present a comparative table with the average performance of the most important generators reported in the literature and our generators over a wide range of parameters and instances in the OR Library.

Key words: Multidimensional Knapsack Problem, Hard Problems Generation, Integer Programming.

RESUMEN
Desarrollamos generadores que producen alternativas que constituyen un reto para el MKP. Nuestro enfoque utiliza independientemente distribuciones exponenciales sobre una gama amplia para generar los coeficientes de las restricciones, y el promedio correspondiente para cada variable se usa para calcular directamente los coeficientes correlacionados en la función objetivo. Los valores de RHS son un porcentaje de la suma de coeficientes de las restricciones. Presentamos una tabla comparativa de la media del desempeño de los generadores más importantes reportados en la literatura y nuestros generadores sobre una gama amplia de parámetros y alternativas en la Biblioteca de IO.

MSC: 90C10.

1. INTRODUCTION

In this paper, we present a methodology for hard problems generation for the NP-hard, Multidimensional Knapsack Problem (MKP), which can be formulated as:

$$\text{Maximize } z = \sum_{j \in N} c_j x_j \tag{1}$$

$$\text{subject to } \sum_{j \in N} a_{ij} x_i \le b_i \quad i \in M \tag{2}$$

$$x_j \in \{0, 1\}, \quad j \in M \tag{3}$$

where $N = \{1,2 ,\ldots, n\}$, $M = \{1,2 ,\ldots, m\}$, $c_j \ge 0$, for all $j \in N$, $a_{ij} \ge 0$, for all $i \in M$, $j \in N$. Each of the m constraints of (2) is called a knapsack constraint.

The Multidimensional Knapsack Problem has received wide attention from the operations research community, because it embraces many practical problems (see Dantzig [1957], S. Gass [1997], Jeroslow and Lowe [1988]). *Applications include resource allocation in distributed systems, capital budgeting and cutting stock problems* (Pirkul [19871]). *In addition, the MKP can be seen as a general model for any kind of binary problems with positive coefficients* (see Kochenberger **et al**. [974]).

Most of the research on knapsack problems deals with the much simpler single constraint version (m = 1). For the single constraint case the problem is not strongly NP-hard and effective approximation algorithms have been developed for obtaining near-optimal solutions. A good review of the single knapsack problem and its associated exact and heuristic algorithms is given by Martello and Toth [1990]). Important recent advances are found in Martello, Toth and Pisinger [1999] and Pisinger [1999].

E-mail: [1]aosorio}@cs.buap.mx

A critical point in algorithm development issue usually arises with the need of comparing their performance. The set of benchmark problems is often comparatively small, and it is hard to be sure that a good performance is not the result of good luck. Formal theorems on performance are solid, but are often hard to come by and may not be relevant to practical problems. Algorithms can, of course, be tested on random problems, but there is justifiable suspicion of such results, as random problems are not meaningful in themselves. This work can be seen as a contribution in the sense of the "empirical science of algorithms" [Hooker (1994)].

While the way we generate the problems have got attention from the AI community (see Cheeseman, Kanefsky and Taylor [1991], Garey and Johnson [1979], Gent and Walsh [1993], Gent and Walsh (1994]), must people in the OR community seems to ignore these results and continue testing algorithms in random generated instances. However, the concept of using the number of nodes in a searching tree as a measure of hardness, has the origin in mathematical programming concepts and relies on the integer programming formulation of the problem. For this reason, it has already been used with the same purpose by people from the OR community (see Hooker **et al**. [1999] and Osorio **et al**. [2002]).

The main purpose of this work is to bring together ideas from both communities and apply the concepts of distribution used by the OR community to generate hard instances for MKP.

To relate the experience obtained in this research, we structured the present paper in the following way. In section 2, we introduce the problem generation methodologies used in the literature to generate MKP instances. Section 3 present the generator proposed. In section 4, we present the computational experiments performed and, in section 5, the conclusion.

2. PROBLEM GENERATION METHODOLOGIES

Problem generation methodologies are currently a very important topic of research. A main part of applying new techniques consists of empirically testing algorithm performance across a representative range of problems. An inadequate set of test problems can provide misleading information about algorithm performance.

MKP problem generation has been intensively studied in the last decade. Using the gap as a measure of hardness, Pirkul [1990] concludes that for a given number of constraints and a given degree of difficulty, the gap reduces as the number of variables increases. For a given number of variables and a given degree of difficulty, the gap increases as the number of constraints increases. Finally, holding the number of variables and constraints constant, the gap increases as the degree of difficulty increases (in particular, as the constraints become tighter).

Martello **et al**. [1988] proposed a generator, for a single knapsack problem, that correlated the knapsack coefficients with the objective function coefficients, inside a random uniform distribution. The objective function values are obtained from a uniform distribution with a very wide range and the knapsack coefficients have a variant of 10 over the objective value. The $c_j = U(0,1000)$ and the $a_{ij} = U(c_j - 10, c_j + 10)$. The RHS is obtained multiplying the sum of the knapsack coefficients by 0.8.

Frèville and Plateau [1990] suggested a procedure for generating hard MKP problems. This procedure was adopted by Chu and Beasley [1998] in the generation of the MKP dataset in the OR library. The approach algorithm uses independently uniform random generation with an interval of (0,1000) for the coefficients in the knapsacks, making $a_{ij} = U(0,1000)$. It also averages the constraint coefficients of each variable to generate the coefficient in the objective function, getting, in this way, a positive correlation between the constraint coefficients and the objective. The çjare equal to $(\sum_{i \in M} a_{ij})/m + 500\ U(0,10)$. This way of generating the knapsacks provides dependence and a wide range in the coefficients, characteristics that contribute to the hardness of these problems. The positive correlation between the objective and the knapsack coefficients, even if highly influenced by the number of knapsacks in the problem, contributes for the problems hardness (see Hill and Reilly [10]).

Glover and Kochenberger [2000] used a uniform distribution in the range (0,10), multiplied by 51 and added 50. The knapsacks coefficients were obtained combining the objective coefficients with numbers drawn form the uniform distribution with a range (0,10). The $c_j = 51\ U(0,10) + 50$ and the $a_{ij} = U(0,10) + 0.1\ c_j\ (1+U(0,10))$.

Hill and Reilly [1996] presented an empirical study of the effects of coefficient correlation structure and constraint slackness settings on the performance of solution procedures. This work examined synthetic two-dimensional knapsack problems (2KP), and conducted tests on representative branch-and-bound and heuristic procedures. Population correlation structure, and in particular the interconstraint component of the correlation structure, was found to be a significant factor influencing the performance of algorithms. In addition, the interaction between constraint slackness and population correlation structure was found to influence solution procedure performance and independent sampling seems to provide information closer to median performance. These results agree with prior studies. Tighter constraints and constraint coefficients with a wider range of values yield harder problems, especially when both of these elements are combined in a problem.

In a different setting, Laguna **et al**. [1995] developed a procedure to create hard problem instances for MGAP (Multilevel Generalized Assignment Problems). This approach embodies a random problem generator, labeled E, that draws constraint coefficients from an exponential distribution and generates the coefficients in the objective function to be inversely correlated. The effectiveness of this approach for generating difficult problems led us to consider adapting some of its features to the MKP setting.

3. HARD PROBLEM GENERATORS PROPOSED

Combining all the ideas described above for generating hard problems, we developed several generators that produces challenging MKP problems.

For the first generator developed and introduced in Osorio **et al**. [2002], the approach uses independently exponential distributions over a wide range to generate the constraint coefficients, and the corresponding average for each variable is used to calculate directly correlated coefficients in the objective function. The RHS values are set to 0.25 of the sum of the corresponding constraint coefficients for the first part of the experiment, using the concept that tighter constraints yield difficult problems (Pirkul [1987] and Hill [2000]). In the second the experiment, this multiple was set at 0.25, 0.5 and 0.75. For the third experiment, we added 0.8 to the tests.

The first problem generator we used to create the random instances of MKPs is designed as follows. The a_{ij} are integer numbers drawn from the exponential distribution $a_{ij} = 1.0 - 1000 \ln(U(0,1))$, $i \in M$, $j \in N$. Again, for each m-n combination, the right-hand side coefficients are set using the relation, $b_i = \alpha \sum_{j \in N} a_{ij}, I \in M$, where α is a tightness ratio and $\alpha = .25$ for the first part of the experiment. In the second part, $\alpha = .25$ for the first ten problems, $\alpha = .5$ for the next ten problems, $\alpha = 0.75$ for the next ten problems and $\alpha = 0.8$ for the remaining ten problems. The objective function coefficients $(c_j's)$ were correlated to a_{ij} and generated as: $c_j = 10 \left(\sum_{i \in M} a_{ij} / m\right) + 10\ U(0,1)$, $j \in N$. In all cases, U(0,1) is a real number drawn form the continuous uniform generator. This generator will be labeled as OSORIO in Tables 3 and 4.

Looking for another generator with similar characteristics but with different correlation value, the problem generator we used to create the random instances of MKPs is modified. The a_{ij} are integer numbers drawn, again from the exponential distribution $a_{ij} = 1.0 - 1000 \ln(U(0,1))$, $i \in M$, $j \in N$. For each m - n combination, the right-hand side coefficients are set using the relation, $b_i = \alpha \sum_{j \in N} a_{ij}$, where α is a tightness ratio $\alpha = .25$ for the first ten problems, $\alpha = .5$ for the next ten problems, $\alpha = 0.75$ for the next ten problems and , $\alpha = 0.8$ for the remaining ten problems. The objective function coefficients $(c_j's)$ were correlated to a_{ij} and include a logarithmic element, now. They were generated as: $c_j = 100\left(\sum_{i \in M} a_{ij}\right)/m - 10 \ln(U(0,1)$. In all cases, U(0,1) is a real number drawn form the continuous uniform generator. This generator will be labeled CUAYA in Tables 3 and 4.

Finally, we decided to substitute the logarithm function by a square root to create the random instances and study its behavior. The a_{ij} are integer numbers drawn from the distribution $a_{ij} = 32000 * U(0,1)^{1/2}$, $i \in M$, $j \in N$. For each m-n combination, the right-hand side coefficients are set using the relation, $b_i = \alpha \sum_{j \in N} a_{ij}, i \in M$, where α is a tightness ratio and $\alpha = .25$ for the first part of the experiment. In the second part, $\alpha = .25$ for the first ten problems, $\alpha = .5$ for the next ten problems and $\alpha = 0.75$ for the remaining ten problems. The objective

function coefficients ($c_j's$) were correlated to a_{ij}. They were generated as: $c_j = 10((\sum_{i \in M} a_{ij})/m + 10\ U(0,1)$. In all cases, U(0,1) is a real number drawn form the continuous uniform generator. This generator will be labeled CUAYA1 in Tables 3 and 4.

4. COMPUTATIONAL RESULTS

We report three experiments. The first experiment was performed on a Pentium III with 500 MHz and 128 Mb in RAM, using CPLEX V6.5.1 installed on Windows 98. This experiment used the OSORIO generator. We used the number of nodes in the searching tree, needed by CPLEX as measures of hardness for the instances generated in this way. The average number of nodes, in 10 instances, needed to solve a problem with 100 variables, 5 constraints and a tightness of 0.25 for the OR-Library problems is 36,434.2 and the solution time 54.234 seconds. For a problem with the same characteristics using the OSORIO generator, the average number of nodes is 1,313,857 and the average solution time 14,304.5 seconds.

For the first experiment we chose problems consisting of 100 variables, 5 constraints and α = 0.25. These settings generate problems that can usually be solved to optimality in less than three hours. We generated 30 problems and allowed each one to run without a limit on solution time. From the 30 instances, CPLEX could solve only 19 instances to optimality in less than three hours. In table 1, we show the sample characteristics and the average number of nodes and the CPU time needed by CPLEX to get optimality. The average solution time needed by CPLEX is 15937.49.

Table 1. Results for OSORIO with instances with 100 variables, 5 constraints and a tightness of 0.25.

Statistics	Corr. Coef.	Objective	NODES	CPUTime
AVERAGE	0.448163	2514.2	1312649	15937.49
STD. DESV.	0.032846	109.5	573885.9	20596.75
MAXIMUM	0.498274	2634	2745544	62161.9
MINIMUM	0.376742	2243	643594	229.87
SUM		75425	39379470	478124.7

For the second the experiment, we concentrated in problems with 5 constraints and 100, 250 and 500 variables, and examined tightness values of 0.25, 0.5 and 0.75 for OSORIO's generator. These problems were solved using CPLEX V 6.5.2 without modifying its default parameters. We allowed a maximum solution time of 3 hours (10,800 secs) and a memory tree size of 250 Mb. We reported the objective value obtained, the number of problems solved to optimality with a gap of 0.0001 and the number of problems finished when the time or the tree size memory were exceeded. We also reported the average gap value.

Table 2. Computational results for OSORIO.

Number Variables	α	Obj. Value	GAP	# Prob Finish.	Finished by	
					Time	Memory
100	0.25	2504	0.0024	5	5	0
100	0.5	5081	0.0059	0	0	10
100	0.75	7761	0.0048	0	0	10
250	0.25	2665	0.0044	0	0	10
250	0.5	5312	0.0024	0	1	9
250	0.75	7928	0.0022	0	0	10
500	0.25	2875	0.0026	0	0	10
500	0.5	5674	0.0014	0	0	10
500	0.75	8380	0.0010	5	6	10
Total		48192		5	6	79
Average		5355	0.0030			

The third experiment was performed using the most representative generators in the literature (Frèville and Plateau [1990], Glover and Kochenberger [1996], Martello and Toth [1990], and the generators proposed here with its variants, labeled as OSORIO, CUAYA and CUAYA1). For Martello's generator, we made an adaptation from problems with a single knapsack to problems with multiple knapsacks, and tested it with different values of α (he only reported values of 0.8). This experiment was performed in a Pentium IV with 256 Mb RAM and 1.8 GHz using CPLEX V 8.1. We tested 10 instances with 100 variables and 10 constraints for a tightness (α) equal to 0.25, 0.5, 0.75, and 0.8 and reported average values. For each m-n combination, the right-hand side coefficients are set using the relation, $b_i = \alpha \sum_{j \in N} a_{ij}, i \in M$, where α is a tightness ratio. The equations used to obtain the $a'_{ij}s$ and the $c'_j s$ in each case, were:

Table 3. Equations to generate coefficients aij and xj with different generators.

GENERATOR	$a_{ij}, i \in M, j \in N$	$c_j, j \in N$
Martello	$U(c_j - 10, c_j + 10)$	U(0,1000)
Frèvile	U(0,1000)	$(\sum_{i \in N} a_{ij})/m + 500\ U(0,10)$
Glover	$U(0,10) + 0.1c_j(1 + U(0,10)$	51 U(0,10) + 50
OSORIO	1.0 – 1000 ln(U(0,1))	$10(\sum_{i \in M} a_{ij})/m + 10\ U(0,1)$
CUAYA	1.0 – 1000 ln(U(0,1))	$100(\sum_{i \in M} a_{ij})/\tilde{m} + 10 \ln(U(0,1)$
CUAYA1	$32000\ U(0,1)^{½}$	$10(\sum_{i \in M} a_{ij})/m + 10\ U(0,1)$

We allowed a total time of 3 hours of CPU for each problem and reported the number of problems solved to optimality, the number of problems that could not be solved in three hours and the number of problems that could not be solved because of the lack of memory in the computer. Results for these generators are shown in Table 4.

We are using the number of nodes as a measure of the "hardness" of the problems. This measure correspond to the general measure of hardness of an integer problem, studied by Hooker [1994],[2003], y Hooker and Osorio[1999]. He concluded that the search tree size, expressed by the number of nodes, is relatively an intrinsic measure because it does not depend on most of the details of the software. The main problem is that it depends on the branching rule, but we can solve problems using the same branching rule and platform. It may be, sometimes, done for two or three branching rules to see if it affects the relative results.

On the other hand, the gap is the relative distance between the best integer found in the branch and bound tree and the LP value of the best unexplored node in the tree. It may be used as an approximation to know the quality of the solution when the process is interrupted. The average gap of the instances that have to be interrupted after three hours of CPU is presented in Table 4.

The results obtained with all the generators presented, enforce the same ideas presented in the paper and obtained by other authors [1994] about the elements that make a problem hard to solve. From all the characteristics mentioned in the paper, the exponential distribution in the coefficients made the biggest contribution to the hardness of the problems presented. Generators OSORIO, CUAYA and CUAYA1 were, systematically, generating instances that had a very big number of nodes in the searching tree and could not be totally solved in less than three hours of CPU.

Table 4. Results for instances with 100 variables and 5 constraints.

Generator	α	Correlation	Solution time	Nodes	GAP	# Prob Finish.	Finished by	
							Time	Memory
Frèville	0.25	0.3192356	35.116	47000.7		10	0	0
Glover	0.25	0.703748	2.246	3081.7		10	0	0
Martello	0.25	0.703748	35.451	49464.8		10	0	0
OSORIO	0.25	0.4674919	12361.2	9950703.9	0.001026	0	8	2
CUAYA	0.25	0.2173587	12300	9857093.3	0.000943	0	9	1
CUAYA1	0.25	- 0.8531521	12750.9	9893667.2	0.000583	0	8	2
Frèville	0.5	0.3192356	23.986	31203.6		10	0	0
Glover	0.5	0.703748	2.21	3027.7		10	0	0
Martello	0.5	0.9998064	26.393	38761.9		10	0	0
OSORIO	0.5	0.4674919	12503.7	9896030.7	0.000471	0	8	2
CUAYA	0.5	0.2173587	12853.5	9824380.3	0.000449	0	9	1
CUAYA1	0.5	- 0.8531521	12267.1	9982488.3	0.00032	0	9	1
Frèville	0.75	0.3192356	13.643	17635.1		10	0	0
Glover	0.75	0.703748	1.549	1922.8		10	0	0
Martello	0.75	0.9998064	5.069	7580.9		10	0	0
OSORIO	0.75	0.4674919	11907.8	9234448.1	0.0002912	1	9	0
CUAYA	0.75	0.2173587	12280.6	9742883.5	0.000315	0	10	0
CUAYA1	0.75	- 0.8531521	11108.8	8587458.6	0.0002074	1	9	0
Frèville	0.8	0.3192356	10.05	13231.1		10	0	0
Glover	0.8	0.703748	0.824	940.6		10	0	0
Martello	0.8	0.9998064	6.512	9429.4		10	0	0
OSORIO	0.8	0.4674919	12700.3	9606857.2	0.0002855	1	9	0
CUAYA	0.8	0.2173587	12361.3	9554086.9	0.000271	0	10	0
CUAYA1	0.8	- 0.8531521	12303.1	9698323.3	0.0002015	0	10	0

5. CONCLUSIONS

The results show that the problems generated with exponential distributions, usually need a greater number of nodes in the searching tree and CPU time to be solved to optimality for most of the instances. These results highlight the need to combine efforts from the AI and OR fields to explore more in detail the behavior of combinatorial problems to get more information about the settings that make “hard” problems really hard for MKP.

REFERENCES

CHEESEMAN, P.; B. KANEFSKY and W.M. TAYLOR, (1991):"Where the Really Hard Problems Are", in **Proceedings of the 12th JCAI, International Joint Conference on Artificial Intelligence,** 331-337.

CHU, P. and J. BEASLEY (1998): “A Genetic Algorithm for the Multidimensional Knapsack Problem”, **Journal of Heuristics**, 4, 63-86.

DANTZIG, G.B. (1957): “Discrete Variables Problems”, **Operations Research** 5, 266-277.

FRÉVILLE, A. and G. PLATEAU (1990): "Hard 0-1 Multiknapsack Test Problems for Size Reduction Methods", **Investigación Operativa** 1, 251-270.

GAREY, M. and D. JOHNSON (1979): **Computers and Intractability**, W.H. Freeman, San Francisco.

GASS, S. (ed.), (1977): **Encyclopedy of Operations Research and Management Sciences**, Kluwer Academic Publishers, New York.

GENT, I.P. and T. WALSH, (1993): "An empirical analysis of search in GSAT", **Journal of Artificial Intelligence Research**, 1 47-59.

GENT, I.P. and T. WALSH (1994): "Easy problems are sometimes hard", **Artificial Intelligence** 70, 335-345.

GLOVER, F. and G.A. KOCHENBERGER (1996): “Critical Event Tabu Search for Multidimensional Knapsack Problems”. In I.H. Osman and J.P. Kelly (eds.), **Meta-Heuristics: Theory and Applications**. Kluwer Academic Publishers, 407-427.

HILL, R. and CH. REILLY (2000): "The Effects of coefficient correlation Structure in Two-Dimensional Knapsack Problems on solution Procedure Performance", **Management Science** 46, 302-317.

HOOKER, J.N. (1994): “Logic-based methods for optimization”, in A. Borning, ed., **Principles and Practice of Constraint Programming, Lecture Notes in Computer Science** 874, 336-349.

_____ (1994): “Needed: An Empirical Science of Algorithms”, **Operations Research** 42, 201-212.

_____ (2003): “A Framework for combining solution methods”, Working Paper, Carnegie Mellon University

HOOKER, J.N. and M.A. Osorio (1999): “Mixed Logical/Linear Programming”, **Discrete Applied Mathematics** 96-97, 395-442.

JEROSLOW, R.E. and J. K. LOWE (1984): "Modeling with integer variables", **Mathematical Programming Studies** 22, 167-184.

KOCHENBERGER, G.; G. McCARL and F. WYMANN (1974): “A Heuristic for General Integer Programming”, **Decision Sciences** 5, 36-44.

LAGUNA, M.; J.P. KELLY; J.L. GONZÁLEZ-VELARDE and F. GLOVER (1995): "Tabu search for the multilevel generalized assignment problem", **European Journal of Operational Research** 82, 176-189.

MARTELLO, S. and P. TOTH (1990): **Knapsack Problems: Algorithms and Computer Implementations**. John Wiley & Sons , New York.

MARTELLO, S.; D. PISINGER and P. TOTH (1999): “New Trends in Exact Algorithms for the 0-1 Knapsack Problem”, **European Journal of Operational Research** 123(1999), 325-336.

OSORIO, M.A.; F. GLOVER and P. HAMMER (2002): “Cutting and Surrogate Constraint Analysis for Improved Multidimensional Knapsack Solutions”, **Annals of Operations Research** 117, 71-93.

PIRKUL, H. (1987): “A Heuristic solution Procedure for the Multiconstraint Zero-One Knapsack Problem”, **Naval Research Logistics** 34, 161-172.

PISINGER, D. (1999: “Contributed Research Articles: A Minimal Algorithm for the Bounded Knapsack Problem”, **ORSA Journal on Computing** 12, 75-84.