INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Multicriteria Knapsack Problem

Algorithmic Aspects and Computer Implementation

Eduardo Pombal Luila

May 2008

Abstract

Most of the existing papers regarding the Knapsack problem deal with a single criterion model. In this paper we study the knapsack problem from a multiple criteria perspective. There are not many articles published on this topic, though this model describes several real-life applications such as, capital budgeting problems, project selection problems and others. The motivation for this study was to explore efficiently the labeling algorithm in a multiple criteria approach, that enable us to obtain non-dominated solutions quickly and solve bigger instances than usually, because there is an emerging need for new algorithms able to compute non-dominated solutions quickly.

The aim of this paper is to implement an exact algorithm to solve the multiple criteria $0-1$ knapsack problem (more than two criteria). Beforehand, we convert the multiple criteria knapsack model into a multiple criteria longest path over an acyclic network. The methodology proposed in this paper uses efficiently the labeling algorithm and the decision maker does not have to judge the objectives about their priority. The advantage of this approach is to compute non-dominated solutions for multiple criteria $0-1$ knapsack problems quickly for certain instances, as shown by the computational results. However, this exact approach is not able to solve hard instances.

Keywords: Knapsack Problem, Multiobjective Combinatorial Optimization, Labeling Algorithm.

1. Introduction

At the beginning of the seventies, with the increasing complexity of the socio-economic environment that characterizes the modern technological societies, it was verified that almost all the problems involved some criteria, generally in conflict with each other, making it difficult to formulate, model and attain the solution of the problem. Decision makers are not just concerned with economic issues, but also with, social, and environmental issues, among others. The solution for some of these problems consists of transforming them into a multicriteria knapsack problem that is a multiobjective combinatorial optimization problem. The knapsack problems characterize a class of integer linear programming problem and are classified in the literature, according to its complexity, as *NP-Hard* problems.

Although many researchers have dedicated some of their careers to studying the combinatorial knapsack problem, the determination of an optimal solution remains a hard problem to solve. Besides, most of these studies had been dedicated to solving the monocriterion problem. The single criterion model cannot take into account many relevant aspects of reality that "must be" represented by several criteria. So, a multiple criteria model seems to be more realistic because, the real-life problems require, in general, an analysis explicitly considering several criteria. The aim of this paper is to develop an exact algorithm to solve the knapsack problem with more than two criteria, based in the conversion of the multiple criteria knapsack model into a multiple criteria longest path over an acyclic network, where the labeling algorithm calculates all the non-dominated solutions, for complete enumeration.

The main motivation or this study, was that we intended to make a study for the knapsack problem with more than two criteria. The knapsack problem with two criteria was already tested in Captivo *et al.* (2003). Our algorithm solves not only, the monocriterion, but also the bicriteria and multicriteria knapsack problem. The knapsack problem with more than two criteria had never been solved, in this paper we present its resolution for the first time.

More precisely, we studied the behaviour and the performance of the algorithm used in the resolution of this problem and its limitations. We present the results of this study in terms of the total number of vertices, total number of arcs, maximum label used number of nondominated solutions, total memory used and the total execution time for multicriteria instances generated randomly.

2. Knapsack Problem

The knapsack problem is one of the most important integer linear programming problems that have been extensively studied over the past forty years. The problem hypothetically arises as the situation of a hiker having to fill up his knapsack by selecting from among various possible objects, each of which has a specified weight and value. The capacity of the knapsack is limited and the hiker would like to maximize the total value of the objects in the knapsack.

The knapsack problems is a well known *NP – Hard* problem, but using dynamic programming algorithms, several problems of this class can be solved in pseudo-polynomial time. The implementation time, in these cases, is directly related to the size of the instance, the number of criteria and the size of

the knapsack. These surprising results come from several decades of research that have exposed the structural properties of the special knapsack problem, which make the problem relatively easy to solve.

2.1 Monocriterion Knapsack Problem

The monocriterion knapsack problem can be formally defined as follows: We are given an instance of the knapsack problem with object set N, consisting of n objects j with profit c_j and weight w_j, and the capacity value W (usually all these values are taken from the positive integer numbers). Then objective is to select a subset of N such that the total profit of the selected objects is maximized and the total weight does not exceed W.

Alternatively, a knapsack problem can be formulated as a solution of the following linear integer programming formulation:

$$Maximize \sum_{j=1}^{n} c_j \, x_j \qquad (2.1)$$

Subject to:

$$\sum_{j=1}^{n} w_j \, x_j \leq W \qquad (2.2)$$

$$x_j \in \{0,1\} \; ; \; j = 1, \dots, n \qquad (2.3)$$

Where x_j is a binary variable equalling 1 if object j should be included in the knapsack and 0 otherwise.

2.2 Multicriteria Knapsack Problem

In this paper, we solved one of the variants of the classic knapsack problem, which is the knapsack problem with various criteria and binary variables, known as multicriteria $0-1$ knapsack problem. This problem is not yet completely explored in the literature, and the available methods for its resolution are not able to solve problems with more than two criteria and they cannot be successfully applied yet to the resolution of problems of big dimensions.

The multicriteria knapsack problem is a well-known combinatorial optimization problem with a wide range of applications. Examples may be found in affordability analysis and capital budgeting where projects have to be chosen with respect to more than a single criterion, in transportation, investment, planning, or in conservation biology to model allocation aspects. In this paper we consider the integer multicriteria knapsack problem formulated as follows:

$$Max \; z_1(x_1, \dots, x_n) = \sum_{j=1}^{n} c_j^1 \, x_j$$

$$Max \; z_2(x_1, \dots, x_n) = \sum_{j=1}^{n} c_j^2 \, x_j \qquad (2.4)$$

$$\vdots$$

$$Max \; z_m(x_1, \dots, x_n) = \sum_{j=1}^{n} c_j^m \, x_j$$

Subject to:

$$\sum_{j=1}^{n} w_j \, x_j \leq W \qquad (2.2)$$

$$x_j \in \{0,1\}, \qquad j = 1, \dots, n. \qquad (2.3)$$

Where $x_j = 1, j = 1,\ldots,n$ if and only if object j is to be included in the knapsack, $W > 0$, c_j^k, $w_j > 0$ holds for all $j = 1,\ldots,n$ and $k = 1,\ldots,m$.

The notation concerning the knapsack problem includes:

- n: is the object number;
- m: is the number of criteria;
- W: is the capacity of the knapsack;
- c_j^k: is the k^{th} value (or the k^{th} profit) of the j object, for $k = 1,\ldots,m$ (c_j will be used when $m = 1$).

In these problems with more than one criterion does not exist, in general, a solution that optimizes simultaneously all the involved criteria. Thus, the notion of optimal solution does not make sense.

3. Multiobjective Combinatorial Optimization

Combinatorial Optimization is a field extensively studied by many researchers. Due to its potential for application in real world problems it has prospered over the last few decades. It is also well known that decision makers have to deal with several usually conflicting objectives. Multicriteria or Multiobjective Combinatorial Optimization has not been studied widely. A few papers in the area have been published in the seventies. During the last 15 years, specific methodologies have been developed, and the number of research papers on this topic has grown considerably.

Combinatorial problems are characterized by the possibility of combination of several variables. Usually there are various criteria to be used in evaluating a solution for a combinatorial problem, which makes the problem multiobjective (also known as multicriteria combinatorial optimization problem).

The goal of solving problems with one objective is to find the optimal solution, or a feasible solution that optimizes the objective function, whose value is unique, even if there are other optimal solutions. In problems with multiple objectives, this notion of optimality is not applicable, since a feasible solution that optimizes one of the objectives, do not optimize, in general, the others objectives. This concept is explained in the following topic.

3.1 Nondominated Solution

A nondominated point corresponds to an efficient solution in the decision space. An efficient solution is a feasible solution for which an improvement in one objective will always lead to deterioration in at least one of the other objectives. Mathematically, a nondominated solution can be defined as follows:

– Consider $x = (x_1, x_{2,} \ldots, x_n)^T$ and $y = (y_1, y_{2,} \ldots, y_n)^T$ two feasible solutions of a multiobjective problem ($x, y \in X$). Solution x dominate solution y if and only if:

$$\mathrm{f_j}(x_1, x_2, \ldots, x_n) \geq \mathrm{f_j}(y_1, y_2, \ldots, y_n) \text{ for } \forall \mathrm{j} \quad (3.1)$$

$$\mathrm{f_j}(x_1, x_2, \ldots, x_n) > \mathrm{f_j}(y_1, y_2, \ldots, y_n) \text{ for some } j \text{ with } j = 1, 2, \ldots, h \quad (3.2)$$

– Consider x a feasible solution of a multiobjective problem ($x \in X$). Solution x is nondominated if and only if these does not exist another feasible solution y ($y \in X$) that it dominates x.

3.2 Methodology To Solve Multiobjective Problems

In recent years considerable effort has been dedicated to the development of methods for the calculation of nondominated solutions, taking always in consideration the information provided by the decision maker, because this information can help in the search of such solutions, through his preferences system. In such a way, excluding the trivial case where a solution exists that optimizes simultaneously all the objectives, three approaches can be considered to solve multiobjective problem, through the preferences system of the decision maker are incorporated *a priori*, *a posteriori* or gradually, in the decision process:

The methodology that is used in this paper incorporates the preferences of the decision maker a posteriori, it initially determine all the nondominated solutions in the problem, which later are presented to the decision maker to proceed his selection. The advantage of this approach is that the decision maker does not have to judge the objectives about their priority. In addition it allows him to learn more about the trade-offs between the objectives. As disadvantages, there is a computational effort underlying the determination of all the nondominated solutions for certain instances, and it can become complicated for the decision maker to choose a solution in the nondominated solutions set, that can be very big, and where many times these solutions have many similar characteristics.

3.3 Solution Method For Multiobjective Combinatorial Optimization Problems

Many of the existing methods concern the biobjective case (to various extensions, depending on the problem). The multicriteria case is still hard to be solved, due, not only, to computational complexity, but also due to the higher number of more efficient solutions (supported and nonsupported) of the MOCO (Multiple Objective Combinatorial Optimization) problem. There are approaches that can be used for solving MOCO problems. These algorithms can be categorized into two major classes of algorithms, the exact and the approximate. Exact algorithms are guaranteed to find all efficient solutions, and these solutions are effectively nondominated. While for the approximate algorithms also called *heuristics,* the solutions are potentially nondominated.

Many of the exact methods used to solve MOCO problems essentially combine the multiple objectives into one single objective. The most popular and the one used first, is the weighted sum scalarization. These exist otherg techniques used in exact methods to solve MOCO problems, as ε-Constraint and minimization of the distance to a reference point to find all the nondominated solutions (see Steuer 1986).

In harder problems, with many objectives and large instances, the exact methods might not be able to solve them or when they do it they take too much time, which means, in some way, that they are not efficient. In such circumstances it is important to find a good feasible solution, which is at least approximate to an optimal solution. These methods are approximation methods, usually

called "*heuristics and metaheuristics*", such as genetic algorithms, simulated annealing, tabu search and artificial neural networks. In these methods it is intended to establish a commitment between the solutions quality and the time necessary to calculate them.

Recently, some methods were proposed for the generation of all nondominated solutions for bicriteria case, the method of Visée *et al.* (1998) and of Captivo *et al.* (2003). The method proposed in Captivo *et al.* (2003) converting the knapsack model into a bicriteria path problem over an acyclic network, and uses a special labeling algorithm to search for all the efficient solutions. The problem is transformed into a bicriteria shorter path problem. This method calculates indistinctly the supported and not supported solutions. In this paper the method used to solve the multicriteria $0-1$ knapsack problem, that is the multiobjective combinatorial optimization problem is an exact adaptation of the method proposed by Captivo *et al.* (2003) in a multicriteria approach, for the complete enumeration of all the nondominated solutions.

4. Implementation Of The Algorithm

The algorithm presented in this section and its respective procedure have been adapted from the article of Captivo et al. (2003) to the multicriteria case.

The algorithm described below is a particular implementation of the labeling algorithm for multiple criteria shortest path problems on acyclic networks.

Algorithm to determine efficient paths

```
BEGIN;
    s(1^0) ← {(0, ..., 0)} and s(1^{w_1}) ← {(v_1^1, ..., v_1^r)}
    T ← {0, w_1} and V ← {}
    FOR (j = 2) TO n DO
    BEGIN
        FOR (a = 0) TO W DO
        BEGIN
            IF (a ∈ T) THEN
            BEGIN
                V ← V ∪ {a};
                IF (a − w_j ∈ T) THEN {Two incoming arcs are defined}
                    S(j^a) ← Set nondominated of
                    (S((j − 1)^a) ∪ {(v_j^1, ..., v_j^r)} + S((j − 1)^{a−w_j}));
                ELSE {Only the arc ((j − 1)^a, j^a) is defined}
                    S(j^a) ← S((j − 1)^a);
            END
            ELSE
                BEGIN
                    IF (a − w_j ∈ T) THEN { Only the arcs ((j −
                    1)^{a−w_j}, j^a é is defined}
                    BEGIN
                        S(j^a) ← {(v_j^1, ... , v_j^r)} + S((j − 1)^{a−w_j});
                        V ← V ∪ {a};
                    END
                    {ELSE The node j^a does not exist}
                END
        END
        T ← V e V ← {};
    END
    S(z) ← Set of nondominated of ∪_{a=0}^{W} S(n^a);
END
```

This network is also generated, layer by layer; only knowledge of the nodes belonging to layer $j-1$ is required to construct the set of nodes of layer j and the arcs connecting these two consecutive layers. Thus, all the other nodes and arcs are not taken into account in this procedure. With this technique a lot of memory space is saved which is not the case for general networks.

Each node j^a, $a = 0, \dots, W$, has at most two incoming arcs $((j-1)^a, j^a)$ and $((j-1)^{a-w_j}, j^a)$. In this way, the labels of j^a can be easily obtained from the labels of nodes $(j-1)^a$ and $(j-1)^{a-w_j}$. So, the whole set of labels of j^a is generated in one iteration. In subsequent iterations it is not necessary to update the set of non-dominated labels for each node as it is usually the case when

applying labeling algorithms to general networks.

5. Computational Experiments And Results

This section deals with the computational behavior of the algorithm on certain sets of randomly generated instances.

All the instances used in this paper have been generated from an instances generator, based on the generation of random numbers from the generator proposed by Klingman (1974) for network flow problems. In this paper we use dynamic structures data, which allow us to make a better management of available memory, and the accesses to the elements of the structure are made efficiently and quickly as arrays. The results are presented in terms of the total number of vertices, total number of arcs, maximum label used, number of nondominated solutions, total memory used and the total execution time for multicriteria instances randomly generated. The instances presented were run on a computer Intel Core 2 Duo / 1.80 GHz / 2GB RAM.

5.1 Instance with three criteria

Initially, we generated a set of 30 randomly generated instances of 3 criteria and 35 objects from the generator of Klingman (1974), and we analyzed them. The results of these instances are presented in Table 5.1.

Instance	Knapsack capacity	Nº of nodes	Nº of arcs	Maximum label used	Execution time (s)	Memory used (Mb)	Nº of nondominated solutions
1	872	20052	38428	126786	2,15	53,09	83
2	902	20709	39637	106327	1,83	39,53	117
3	904	21682	41567	201963	3,76	77,19	131
4	1024	23591	45190	242342	6,22	101,30	169
5	847	18488	35301	148230	2,90	50,47	144
6	976	22682	43480	223412	5,13	87,39	179
7	946	22165	42474	192861	4,26	69,48	183
8	819	20095	38581	145699	2,67	49,71	162
9	895	21319	40881	223227	6,25	86,39	183
10	865	20471	39238	170601	2,93	60,14	199
11	841	19871	38091	94283	1,59	37,75	126
12	684	16357	31384	212896	5,05	72,02	183
13	1060	24435	46802	153255	4,50	65,12	169
14	960	22422	42954	119474	2,18	45,19	109
15	736	17250	33025	85412	1,37	35,36	62
16	909	22243	42708	300532	7,35	122,81	235
17	885	21259	40753	110995	2,14	44,87	130
18	882	21313	40915	340199	10,00	130,97	332
19	881	21387	41031	154692	2,65	56,66	169
20	1007	22920	43913	355092	8,66	121,59	192
21	783	18274	35003	163209	4,23	64,35	199
22	900	21253	40802	188967	3,92	57,62	196
23	850	20852	40060	166753	3,06	71,92	130
24	976	21024	40183	211954	4,56	75,01	160
25	896	21854	41929	124140	2,75	47,56	151
26	848	20425	39195	110618	1,76	36,51	64
27	971	22286	42692	197161	4,10	60,63	185
28	924	22159	42556	305307	6,86	125,00	275
29	850	20301	38979	254452	6,79	77,94	190
30	793	19304	37051	80893	1,20	30,43	42
	890	20948	40160	183724	4,09	68,47	162

Table 5.1 – Results of 30 instances with 3 criteria and 35 objects

The maximum label used represents the maximum dimension of the dominated and nondominated labels sets in each layer, this maximum label almost always belongs to the last but one layer $n-1$, then it is from this set where it verifies the nondominated labels that pass to final layer n and are nominated the solutions of the problem.

However, analyzing the number of dominated solutions (maximum labels used less the number of nondominated solutions) versus the number of nondominated solutions for each instance of Table 5.1, we verify that

there's an enormous difference between the two values.

The graph in Figure 5.1 shows that approximately 0.1% of the dominated and nondominated labels set represent the final solution set of the problem. Then we can draw the following conclusion, the program loses much time to find the few final nondominated solutions in a very big labels set, which in its great majority is composed of dominated labels.

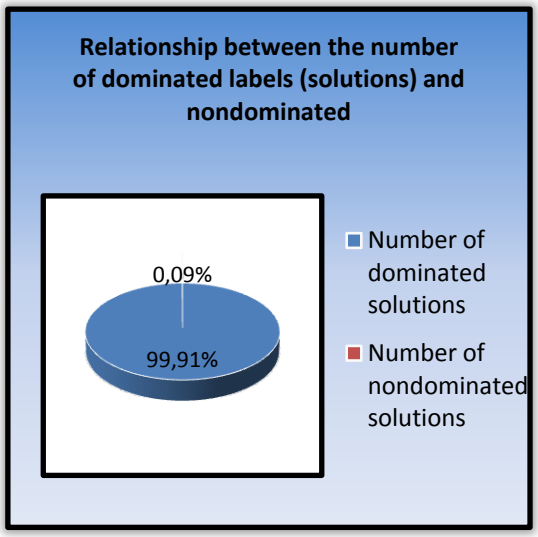

Figure 5.1 – Relationship between the number of dominated and nondominated solutions

We can see in Table 5.1 that the algorithm took $10.00s$ for the maximum time, $1.20s$ for the minimum time and $4{,}09s$ for the average execution time. Through the number of objects (items) and the number of criteria used, we consider these times good, because they are relatively low. But for instances of average dimension or with a high number of criteria, the execution time of the algorithm is very high.

The memory used indicates the stored and allocated memory by the variables during the program execution. Analyzing Figure 5.2, we verify that instances 4, 16, 18, 20 and 28 use considerable amounts of memory. The considerable amount of memory used is one of the factors that strongly conditioned the algorithm performance, being a clear limitation of the algorithm. As we will have the opportunity to see ahead, instances with big dimensions or with many criteria are normally more demanding in terms of memory therefore, and taking into consideration the computational limitations of the computer used, our algorithm is not able to solve these instances.

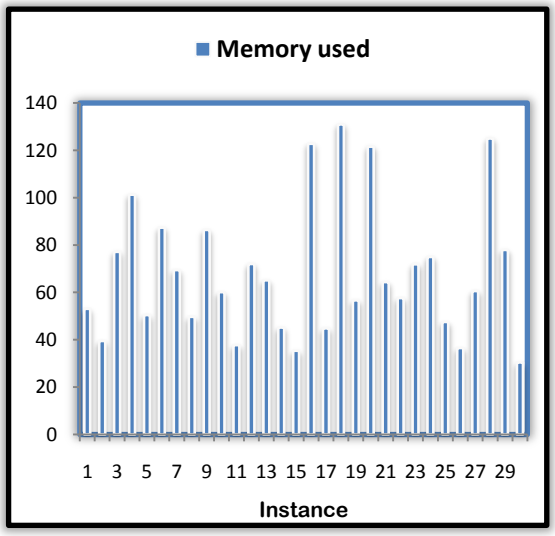

Figure 5.2 – Memory used by the algorithm

The average value of the memory used was $68.47Mb$ for 30 instances with 35 objects and 3 criteria what it is a matter, because when the number of objects or the number of criterion increases, the memory consumption increases significantly.

5.2 Multicriteria Instances

In this subsection we analyze the algorithm performance when the number of criteria increases gradually, and the consequences of it. Another aspect that should be taken into account in the resolution of multicriteria problems is the time necessary to calculate the nondominated solutions. In order to

analyze this aspect, we randomly generated various multicriteria instances of 25 objects. The average results of these multicriteria instances are presented in Table 5.2.

Criterion	Knapsack capacity	Nº of nodes	Nº of arcs	Maximum label used	Execution time (s)	Memory used (Mb)	Nº of nondominated solutions
2	632	9708	18069	10393	0,11	2,64	18
3	632	9708	18217	29690	0,39	7,81	70
4	638	9740	18272	62045	1,84	17,69	235
5	633	9663	18102	98406	6,61	28,83	455
6	626	9544	22374	151233	14,78	44,03	890
7	632	9644	18102	209541	44,81	70,27	1622
8	618	9473	19768	255478	84,60	96,20	2144
9	616	9509	17848	362159	184,63	131,13	3162
10	636	9432	17710	333400	201,47	136,71	2905
11	620	9025	18910	487761	339,74	179,98	4233
12	622	8955	17464	512673	477,54	198,38	4344
13	649	10512	19746	664703	583,68	259,42	5237
14	656	9909	18573	662980	605,46	299,23	6942
15	624	9594	18007	692452	799,53	358,72	7351

Table 5.2 – Results of multicriteria instances of 35 objects

The increase of more one or two criteria in a multicriteria knapsack problem is not a necessary for condition to the number of solutions to increase. But as we increase the knapsack problem dimension in terms of the increment of the number of criteria, the quantity of the solutions increases very quickly (see Figure 5.3).

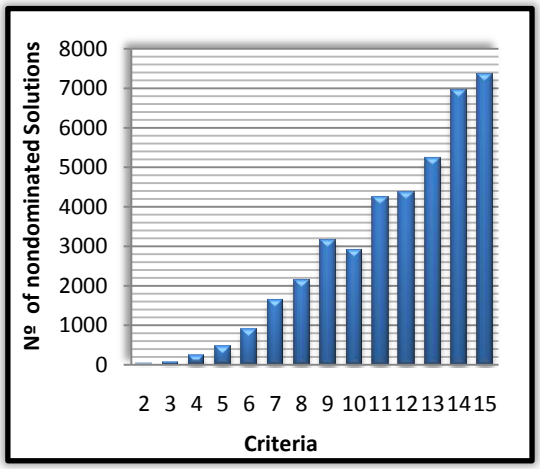

Figure 5.3 – Relationship between the number of nondominated solutions and the number of criteria

It can happen in some problems with a small number of criteria that they may have more efficient solutions than others, with more criteria. This situation can be verified in Table 5.2 between the instances with 10 criteria and 11 criteria.

The results in Table 5.2 show that it was only possible to solve instances with less than 16 criteria, in problems with 25 objects, because for instances with 25 objects and more than 15 criteria, the program exceeded the available memory that the computer provided to run the problems. That is, with only the increase of the number of criteria the computer quickly reaches the memory limit. Figure 5.4 shows the evolution of the memory used as the number of criteria in increased.

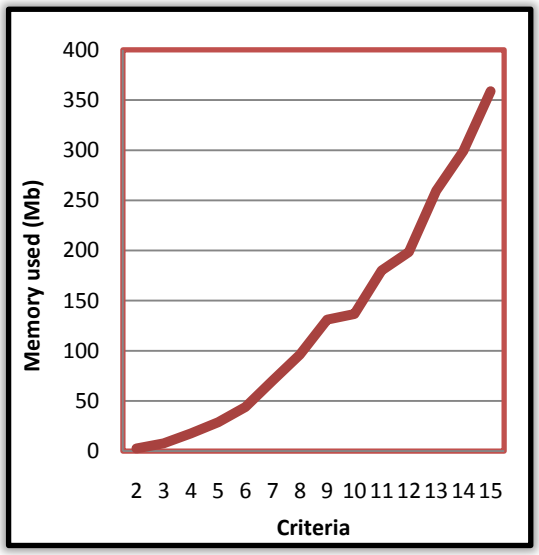

Figure 5.4 – Relationship between the memory used and the number of criteria

The high number of criteria is an important factor to take into account in the knapsack problems, for the reasons above mentioned, but it is not the only one. Another important aspect is the execution time (the time that an algorithm takes to calculate all the nondominated solutions). This time is directly related with the size of the instances

and the number of criteria. In Table 5.2 we keep constant the dimension of the instance, i.e., we consider all the instances with 25 objects but we gradually increase the number of criteria. We verify that, when we increase the number of criteria, the period of time necessary to find all the nondominated solutions for the problem also increases (see Figure 5.5).

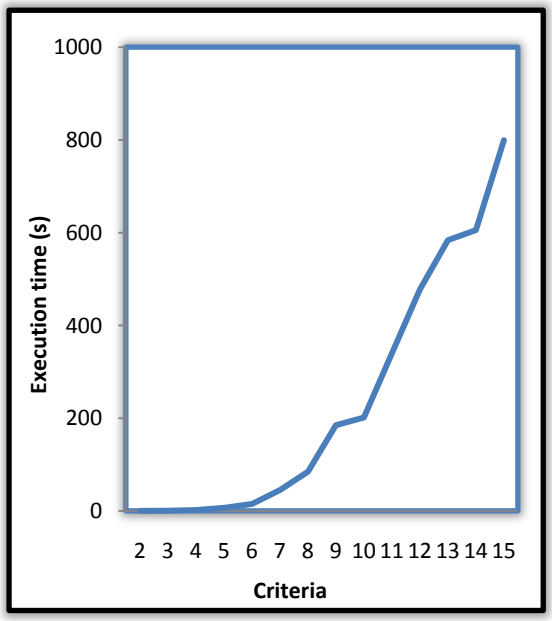

Figure 5.5 – Relationship between the execution time and the number of criteria

6. Conclusion

Many of the real-life problems that can be modelled through networks are of a multiobjective nature. The objects generally involved in these problems are the following: cost, time, distance, accessibility, satisfaction, environmental, risk, security, and others.

In these multiobjective problems the notion of an optimal solution is substituted by the notion of a nondominated solution. In a general way, there one three methodologies that can be used to solve multiobjective programming, through the preferences of the decision maker, that are included *a priori, a posterior* or gradually, in the decision process.

The methodology proposed in this paper, incorporates the preferences of the decision maker *a posteriori*, it initially determines all the nondominated solutions of the problem, which later are presented to the decision maker to proceed his selection. In this approach the decision maker does not have to judge the objectives about their priority. In addition it allows him to learn more about the trade-offs between the objectives.

The experiences and computational results presented in this paper showed that the exact approach of our algorithm, has been efficient enough to solve multicriteria instances of small dimensions. The memory used was one of the factors that conditioned strongly the algorithm performance, being a clear limitation of the algorithm to run instances of bigger dimensions. It should be noted that the memory capacity used depends on the number of labels in two consecutive layers. This fact explains why problems of larger dimension cannot be solved.

The quantity of solutions of the problem increased quickly with the increase of the dimension of the knapsack problem in terms of the number of criteria. The computer reached quickly the limit memory with the increase of the number of criteria. These facts confirm the limitations imposed by the computational memory.

Bibliography

Captivo, M., Clímaco, J., Figueira, J., Martins, E., Santos, J.L. (2003) "*Solving bicriteria* $0-1$ *knapsack problems using a labeling algorithm*", Computers & Operations Research, 30, pp. 1865 - 1886.

Klingman, J., Napier A., and Stutz, J. (1974) "*NETGEN – A program for generating large scale (un) capacitated assignment, transportation and minimum cost flow network problems*", Management Science, 20, pp 814 - 822.

Steuer, R. (1986) "*Multiple Criteria Optimization, Theory, Computation and Application*", John Wiley & Son, Inc.

Visée, M., Teghem, J., Ulungu, E.L. (1998) "*Two-Phases method and branch and bound procedures to solve the bi-objective knapsack problem*", Journal of Global Optimization, 12, pp. 139 - 155.