Using GRASP Method to Compare Efficiently the Other Methods the Literature

Neemias Bucéli da Silva*, Eduardo Theodor Bogue†, Daniel de Leon Bailo da Silva‡ and Giuliano Oliveira de Macedo§
Department of Ponta Porã, University of Mato Grosso do Sul
Ponta Porã, Brasil
Email: *neemias.silva@aluno.ufms.br, †eduardo.theodor@ufms.br, ‡daniel.leon@aluno.ufms.br, §giovani@aluno.ufms.br

***Abstract*—Studies have shown that computer theory techniques has been bringing significant improvements. In this paper, a proposal was presented for knapsack 0-1 using a greedy randomized adaptive search procedure (GRASP) metaheuristic. To evaluate the proposed method, different algorithms were compared for evaluated the method proposal. The result showed then grasp algorithm algorithm was equal to the optimal algorithm using dynamic programming.**

***Index Terms*—GRASP, Dynamic Programming, Greedy**

I. INTRODUCTION

Along the time, may scientists have contributed a lot of heurists for solving problems that are in class NP-complete. As is the case with *Resende*researcher that created a denominated GRASP algorithm [1].

According *Festa and Resende*, GRASP is a mult-start or iterative process, in which each GRASP iteration consists of two phases, a **construction phase**, in which a feasible solution is produced, and a **local search** phase, in which a local optimum in the neighborhood of the constructed solution is sought. The best overall solution is kept as the result.

The unidimensional knapsack 0-1 [2] is defined by:

$$P = \sum_{i=1}^{n} p_i x_i$$

subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i = 0, 1 (i = 1, ..., n)$$

And the weight of the items cannot pass the capacity of the knapsack:

$$\sum_{i=1}^{n} w_i > W$$

$$w_i \leq W (i = 1, ..., n)$$

In this paper, we will util two algorithms to compare the grasp algorithms. The paper is divided into five sections, the first section is the introduction, the following section is the method which will talk the three algorithms utilized for this application, the third section is the experiment and results in this phase will go shown some metrics to evaluate the method proposed, the fourth section is the conclusion where will be determined which technique was the best or got the better result for the respective metrics proposed.

II. METHODS

A. GREEDY ALGORITHM

The greedy algorithm technique was proposed by the following scientists [3]–[5]. It has also been adapted to build several practical programs for *knapsack 0-1* problem [6], [7]. The main contribution of this paper is to use the greedy heuristic method to compare the method grasp and evaluated different instances for the knapsack 0-1 problem.

Basically, the greedy algorithm work of the following form *Algorithm* 1:

B. DYNAMIC PROGRAMMING ALGORITHM

A lot of researchers already implemented several efficient algorithms for the knapsack 0-1 problem, and many of these algorithms using dynamic programming [2], [8]. In this paper was implemented the dynamic programming approach top-down.

The dynamic algorithm is shown in the following procedure *Algorithm* 2:

C. GRASP ALGORITHM

A GRASP is a metaheuristic for combination optimization. Can be implemented using multi-start or iterative process, in which each GRASP iteration consists of two phases, in which a feasible solution is produced, and a local search phase, in which a local optimum in the neighborhood of the constructed solution is sought [1].

For this paper, was use the iterative phase which can be shown following form 3:

III. EXPERIMENTS AND RESULTS

Considering the analysis of the experiment was set that of different method two plots, the first metrics are defined coefficient determinist at use for find correlation in dynamic programming and greedy heuristic, and also grasp method metaheuristic [9]. The second plot is one analysis of behavior between the *Dynamic Programming*, *Greedy*, and *GRASP* method.

Algorithm 1 Knapsack Problems 0-1 Using Greedy Method

```
0: procedure KNAPSACK(Value, Weight, Capacity, item)
   // create the array ration which is sorted by value over weight
   max_value = 0
0:   while element of the ratio array is not visited do
0:     if ratio[i] <= capacity then
0:       capacity -= weigth[i]
0:       max_valeu += value[i]
0:   return max_value
=0
```

Algorithm 2 Knapsack Problems 0-1 Using Dynamic Programming Method

```
0: procedure KNAPSACK(w, n, cache, wVector, valVector)
0:   if n == 0 or w == 0 then
0:     return 0
0:   if cache[n − 1][w] != −1 then
0:     return cache[n − 1][w]
0:   if wVector[n − 1] > w then
0:     return knapSack(w, n − 1, cache, wVector, valVector)
0:   a= valVector[n − 1] + knapSack(w − wVector[n − 1], n − 1, cache, wVector, valVector)
0:   b= knapSack(w, n − 1, cache, wVector, valVector)
0:   return max(a, b)
=0
```

Algorithm 3 Knapsack Problems 0-1 Using GRASP Method

```
0: procedure KNAPSACK(Value, Weight, Capacity, item, max_iteration)
0:   best = 0
0:   for i = 1; i < max_iteration; i++ do
0:     solution = Greedy_Randomized_Construction(Value, Weight, Capacity, item)
0:     solution = local_search(Value, Weight, Capacity, item)
0:     best = max(best, cost(solution, Value, Weight, Capacity)
0:   return best
=0
```

According to Figure 1, we can do observe the behavior of the instance GRASP (axis X) in relation instance Dynamic Programming (axis Y). Can determine that as correlation and instance Grasp and instance optimal output value. Can go to more deep, can do to determine that are correct maximize value, for instance, greater than 140,000 value according to the optimal algorithm.

The behavior of the *Dynamic Programming*, *Greedy*, and *GRASP* method is shown of the Figure **??**. According the figure, one observed that GRASP algorithm is equal to the optimal algorithm (*Dynamic Programming*.

Another analysis is shown in Figure 2 where one can to view which instance that delayed and it can be noted that had the highly execution time was the Dynamic Programming Method that came about 6 min according the graphic.

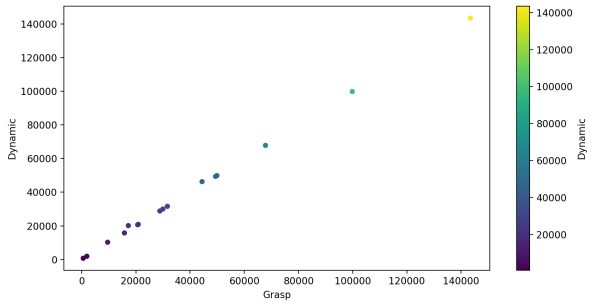

Fig. 1. Coefficient determinist (or score R^2) applied to output instance between the *Dynamic Programming* (axis Y) and *GRASP* method (axis X).

IV. CONCLUSION

Finally, one can conclused that metaheuristic *GRASP* was better that methods *Greedy* and *Dynamic Programming*, princepily the measured execution time according the Table I.

ACKNOWLEDGMENTS

Acknowledgments to teacher PhD Eduardo Theodoro who answered doubts about the GRASP implementations, to student Daniel Bailo was contributed to the timely execution of

TABLE I

RESULT OF INSTANCES FOR EVALUATED THE EXECUTION TIME USING FOR THE RESPECTIVELY METHODS: *Dynaminc Programing* (2TH COLUMN), *Greedy* (3TH COLUMN), *GRASP* (4TH COLUMN).

MEASURED EXECUTION TIME IN m/s			
Instance	**Dynamic Programming**	**Greedy Heuristic**	**GRASP Metaheuristic**
1	0.299604	0.000049	0.012054
2	0.831294	0.000045	0.035831
3	2.007281	0.000059	0.304420
4	2.014614	0.000049	0.082717
5	1.989233	0.000087	1.203739
6	26.858557	0.000148	5.439759
7	0.598913	0.000224	1.355621
8	0.785180	0.000289	3.156068
9	10.908195	0.000795	7.355839
10	4.878645	0.000086	0.129530
11	7.812857	0.000082	0.260361
12	2203.987961	0.009826	4115.226108
13	1043.048750	0.011252	789.132951
14	431.436847	0.004796	3905.787402
15	3.979783	0.000859	30.129698
16	4.037800	0.000832	22.595806

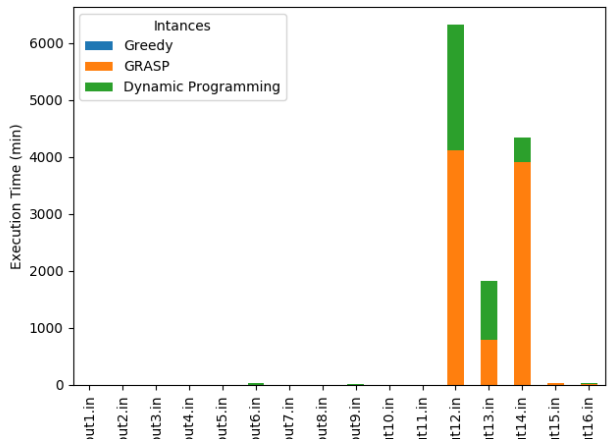

Fig. 2. Execution time analysis for respectively method: the *Dynamic Programming*, *Greedy*, and *GRASP*.

the algorithms, and lecture computer theory that proportionated the development for this paper [1].

REFERENCES

[1] Paola Festa and Mauricio GC Resende. An annotated bibliography of grasp–part i: Algorithms. *International Transactions in Operational Research*, 16(1):1–24, 2009.

[2] Joachim H Ahrens and Gerd Finke. Merging and sorting applied to the zero-one knapsack problem. *Operations Research*, 23(6):1099–1109, 1975.

[3] Webb Miller and Eugene W Myers. A file comparison program. *Software: Practice and Experience*, 15(11):1025–1040, 1985.

[4] Richard Burkhauser, Karen C Holden, and Daniel A Myers. Marital disruption and poverty: The role of survey procedures in artificially creating poverty. *Demography*, 23(4):621–631, 1986.

[5] Esko Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1-3):100–118, 1985.

[6] Yalçın Akçay, Haijun Li, and Susan H Xu. Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research*, 150(1):17, 2007.

[7] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[8] Paolo Toth. Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, 25(1):29–45, 1980.

[9] Minos Garofalakis and Amit Kumar. Deterministic wavelet thresholding for maximum-error metrics. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 166–176. ACM, 2004.