Hybrid Evolutionary Algorithm

Valentina Yusty Mosquera
vyustym@eafit.edu.co

Department of Mathematical Sciences
School of Sciences
Universidad EAFIT
Medellín – Colombia

Abstract

This article proposes the use of a hybrid evolutionary algorithm (HEA) to obtain a set of non-dominated solutions for the multi-objective multidimensional knapsack problem (MOMKP), a derivation of the knapsack problem (KP). The KP and its derivations are widely known for their vast extent of applications including, but not limited to, project management, resource management, energy management, among many others. The development of the HEA took into account evolutionary search strategies and methods based on local search.

Key words: Heuristics, HEA, Optimization, MOMKP.

1 Introduction

In this article, a hybrid evolutionary algorithm (HEA) is going to be developed in order to obtain a Pareto Frontier for the MOMKP. The development of the HEA took into account the genetic algorithm (GA) as the evolutionary search strategy and the iterated local search (ILS) as the methods based on local search. The KP problem is an NP-hard problem that has been extensively studied. As defined in Weisstein (1988), a problem is classified as NP-hard if "an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time problem)." Since the problem is classified as NP-Hard, a metaheuristic algorithm is developed to obtain a high-quality solution in a reasonable time. Lastly, as mentioned in Biglar (2018), some applications of the KP include: resource management, energy management, power allocation management, production planning problems, cognitive radio networks, among others.

In regards to the ILS algorithm, according to Hoos and Stützle (2000), local search algorithms have a variety of applications that are commonly used in fields such as Artificial Intelligence and Operations Research to solve hard combinatorial problems. In addition, the local search algorithms are commonly known as Monotonous Search (descending or ascending) and Climbing Algorithms. While performing local search, each instance is linked with a finite set of feasible solutions. When developing such algorithms, the goal is to obtain a Pareto Frontier with a series of feasible solutions and their corresponding costs. Obtaining these costs is fundamental since depending on the optimization problem, these must be maximized or minimized. As described in Hoos and Stützle (2000), the structure of the local search algorithm is the following: "starting from an independently obtained initial solution, we repeatedly replace the current solution by a neighboring solution of better value, until no such neighboring solution exists, at which point we have identified a solution that is locally optimal." Note that for many initial solutions that exist, the previous process will be processed. Additionally as mentioned in Rivera (2020), local search functions under the principle of Near-Optimality: "good solutions often have a similar structure." Inclusively, as mentioned in Rivera (2020), there are some benefits and disadvantages related to the development of a local search algorithm. The advantages include the ability to find solutions rapidly, small quantity of parameters and requires limited memory usage. On the other hand, some disadvantages

include that the final solution depends heavily on the initial solution and that it is easily caught in local maximum/minimum.

A variety of local search algorithms have been developed: tabu search, simulated annealing, iterated local search, evolutionary local search, GRASP, among many others. For example, the GRASP algorithm is used to solve combinatorial optimization problems of elevated complexity and serves as an iterative multi-start metaheuristic. Also, it is worth noting that as mentioned Rajkumar et al. (2011), the GRASP iteration consists of two phases: a construction phase where the initial solution is obtained and then the local search phase where the objective is to obtain the local optimum of the neighborhood. The iterated local search algorithm, on the other hand, consists of iteratively improving the solution by constantly perturbing the current solution with the objective of building a Pareto Frontier.

The genetic algorithms, as described in Rivera (2020), " try to model the laws of the natural evolution of living beings, in particular genetic inheritance and adaptation to the environment." As described in Mallawaarachchi (2019), the genetic algorithm is a " search heuristic that is inspired by Charles Darwin's theory of natural evolution." Hence, phenomenons such as the survival of the fittest are going to be present. As thoroughly described in Mallawaarachchi (2019), the fittest "produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving." The process previously described is going to successively iterate until the generation with the fittest individuals is found. Hence, as described in Mallawaarachchi (2019), this process is going to be constituted by five phases: the initial population, fitness function, selection, crossover and mutation. It is fair to note that the genetic algorithm has been wisely used in search problems and has applications in areas such as science, engineering, business and social sciences.

The article is organized as follows: Section 2 presents a review of the literature on related problems. In Section 3 the problem is formally described as well as its mathematical formulation. The proposed mathematical algorithm is presented in Section 4. Finally, in Section 5 computational experiments are presented and discussed while in Section 6 some conclusions and future work are introduced.

2 State of the art

A variety of algorithms have been used to solve the knapsack problem and its derivations. For example, the research conducted by Labrada-Nueva (2007) uses ILS, based on deterministic heuristics, to explore regions of feasibility in the KP. The article begins by mentioning the wide variety of applications that the KP has. However, the paper's focus revolves around finding a proper distribution of products in a warehouse in order to maximize profits. According to Labrada-Nueva (2007), "the analyzed method is a stochastic local search method that iteratively applies local searches to perturbations of the current point (initial solution) in a random way in a space of optimal solutions." As opposed to the algorithms implemented in this paper, whose initial solution is obtained from that of a constructive method, the initial solution from this paper is obtained through the use of stochastic processes. Furthermore, the article justifies the use of ILS algorithms since it has proven to be an effective strategy to escape the local optimal solutions. Therefore, as with the algorithms implemented in this paper, once a local optimal value is obtained, it is regarded as the current solution and the algorithm will then iteratively try to improve the solution until termination criteria are met. Note that if in the succeeding iteration a better solution is found as compared to the current solution, then the solution is replaced; otherwise, a perturbation is made with the objective to find a better solution than of the current one. The results obtained from the paper show that as the computing time increases, a better solution is obtained while still not violated the wight restrictions. Therefore, in terms of applicability, the approach of using ILS in the knapsack problem to solve problems related to the distribution of products has proven to be a fine way to obtain better solutions.

The HEA has also been used to solve the KP and its derivations. For example, Bercachi (2010) presents a hybrid evolutionary algorithm by considering both Variable Neighborhood Search (VNS) and an evolutionary algorithm to solve the multidimensional knapsack problem. A hybrid approach was taken with the objective of obtaining a better solution than that which was initially established. The paper

states that the VNS was chosen to tackle this problem since "a favorable characteristics of the current solution will be often kept and used to obtain promising solutions Bercachi (2010)." This paper has been predominant and relevant in the study of the VNS, since it began investigating the impact of how the order in which the neighborhoods are visited in a VNS impacts the result obtained. The paper states that in the VNS algorithm, "neighborhoods are sorted in order to increase time complexity of searching for the best moves Bercachi (2010)." However, figuring out the order in which the neighborhoods are visited can be misleading and even challenging to estimate. To further describe the HEA presented, Bercachi (2010) mentions that "solutions are encoded with binary strings of length n equal to 500. The set of neighborhoods is composed by different gene mutation ratios k = 10. Every time a k-exchange neighborhoods is occurred, this means that a new bit-flip mutation rate is applied to each individual in the population during the reproduction phase. Gene mutation ratios were varied within a fixed-length interval [0.01 : 0.1] with a step equal to 0.01." The paper concludes that the ordering of the neighborhoods does have an impact on the results obtained. The paper's results arrive to the following conclusion: the algorithm deployed, instead of sticking to a predetermined path, considered multiple potential choices of neighborhood ordering. Also, the paper concludes that promising results can be obtained when we consider the union of metaheuristics on a parallel and dynamic models.

Hybrid genetic algorithms are also used to solve the KP problem and its derivations. To solve the multidimensional knapsack problem, Djannaty and Doostdar (2008) propose a HGA that follows the following outline: a stochastic sampling method for the generation of the initial population, a M-point crossover operator, a mutation operator and a penalty function. Lets now expand on the previously mentioned phases. Firstly, a heuristic method, denoted as the Dantzig algorithm, is used in order to generate the initial population. An extensive description of the Dantzig algorithm can be found in Djannaty and Doostdar (2008). Afterwards, when dealing with the crossover, the paper considers the M-point crossover. As described in Djannaty and Doostdar (2008), "the M-point crossover uses two parents and creates two children. After selecting M crossover points, the resulted children are built in the following manner: the first child gets the even segments from the first parent, and the odd segments from the second parent. For the second child the complement holds: it gets the odd segments from the first parent, and the even segments from the second parent. The child with better fitness is inserted into the population." The paper indicates that the mutation operator is used as a mutation variable rate. To further expand, as described in Djannaty and Doostdar (2008), "the mutation rate is initially small and as iterations are continued, the above rate is also increased. For each gene in the chromosome, a random number r in [0, 1] is produced. If the current mutation rate is greater than r, this gene is mutated, otherwise no change is made." The algorithm also uses a penalty function to discard infeasible solutions. The paper indicates that competitive solutions, in terms of quality and execution time, are obtained.

Table 1 presents a summary of the literature review classified according to the type of solution methods used.

Table 1: Summary of literature references

Methods	***References***
Metaheurísticos	Yang (2010), Dorigo and Di Caro (1999)
Local Search	Hentenryck and Michel (2009), Aarts et al. (2003), Lourenço et al. (2003)
VNS	Laabadi et al. (2018), Puchinger et al. (2006), Drake et al. (2013)
HEA	Kafafy et al. (2011), Knowles (2002), Tan (2007)

As will be further explained in the proceeding sections, for the development of some algorithms, an initial solution is needed. In this case, as proposed by Hentenryck and Michel (2009), the initial solution is obtained from a previously designed constructive algorithm. Also, ideas and insights for the development of the neighborhoods of the VNS algorithm are extracted from Drake et al. (2013) and

Laabadi et al. (2018). Additionally, given that the algorithms developed return a Pareto Frontier, a Pareto dominance relation test is performed between the solutions obtained. The Pareto dominance relation test implemented was based on that explained in Dorigo and Di Caro (1999).

3 Description of the problem and mathematical formulation

As previously mentioned, the algorithms described in the previous sections are applied to the KP problem. The KP is an NP-hard problem that can be described through the lenses of figurative imagery as an analogous situation of filling a backpack. A backpack supports a maximum weight and also has a maximum capacity in which it can be filled. Therefore, given a set of items with their respective value, the backpack will be unable to support more than a given weight, with all or part of a set of items. It is important to note that items added to the knapsack should be added based on a criterion: maximize the total value backpack without over passing the allotted weight. The KP problem has applications in variety of field including: project management, resource management, energy management, among many others. In the previous scenarios, a set of elements must be chosen by taking into consideration that they posses a respective value and that there is finite capacity of resources.

The formulation of the MOMKP, obtained from Rivera (2020), can be formally described as follows. Given a set $J = \{1, 2, 3, .., n\}$ items, we want to select a subset of items with the following characteristics:

- Given a set of m types of capacity resources R, the amount of resources used of each type k for each item $i(a_{ik})$ cannot exceed the given capacity b_k.
- Each item can be included or not, but cannot be split or overrun.
- There is a set F of p objective functions to optimize where w_{ih} is the weight of item i in the objective function h.

The problem can be formulated mathematically as the following:

$$\max \sum_{i \in J} w_{ih} x_i \quad \forall h \in F \tag{1}$$

$$\sum_{i \in J} a_{ik} x_{ij} \leq b_k, \quad \forall k \in R \tag{2}$$

$$x_i \in \{0, 1\}, \quad \forall i \in J \tag{3}$$

The mathematical formulation of the problem described above is presented by Equations (1) through (3). Equation (1) represents the set of objective functions. Equation (2) represents the set of restrictions. Finally, Equation (3) defines the domain of the decision variables. To further expand, x_i is a binary decision variable, which takes the value $x_i = 1$ if and only if item i $\in$ J is included in the knapsack, and $x_i = 0$ otherwise.

The surge of the MKP relates to the context of capital budgeting presented by Lorie (1955) and Manne (1957). In 2004, Kellerer et al. (2004) and Fréville (2004) presented an all-inclusive overview of the MKP, including the practical and theoretical components. As mentioned in Jakob Puchinger (2010), applications of MKP can also be seen in Gomory (1966), Chu and Beasley (1998), among others.

4 Solution algorithms

In this section, the solution algorithms are presented. Constructive algorithms are presented in Section 4.1. Randomized algorithms are presented in Section 4.2. ILS and VNS algorithms are presented in Section 4.3 and Section 4.4, respectively. HEA is presented in Section 4.5. Lastly, indexes I_1 and I_2 are described in Section 4.6.

4.1 Constructive methods

Constructive methods are also employed. These are iterative procedures that, step by step, build the solution to a problem. They are usually deterministic methods that look for the element with the best evaluation in each iteration. Advantages in regards to this type of methods is that they have low implementation time, low computation time and low memory requirement. Disadvantages include that heuristic methods are limited to providing a solution, sometimes good, but not necessarily optimal. In this article, two constructive algorithms were developed. For the following methods, a set of considerations should be taken into account: the objective function was obtained by the Weighted Sum Method, in case there were infeasible solutions, a penalty factor for the objective function was considered, the algorithm terminates after each item in the candidate set has been traversed, taking into account the criteria used by the selection function.

Let the first constructive method developed be denoted as MC1. This method consists on the following premise: elements are selected from the candidate set based, as a selection function, on the weighted weight of the objective function elements. As a feasibility function, it was considered that a selected element from the set of candidates can only be added if the number of restrictions satisfied by adding it does not decrease.

Let the second constrictive method developed be denoted as MC2. This method runs different segments of the code depending on an assessed criteria. The first case is executed if there are negative values in the constraint limits. In this scenario, elements are selected from the set of candidates based, as a selection function, on adding those elements that have the greatest negative weight to satisfy the constraint with the greatest negative value in the constraint limit. Once the constraint with the greatest negative value in the constraint limit has been unsatisfied, it is determined if there are other restrictions with negative values in the restriction limits that have not been satisfied. If so, the (unsatisfied) constraint that has the largest negative value in the constraint limit is identified. Such process iteratively continues, until the constraints with negative values in the constraint limits have been satisfied. The second case is executed if there are no negative values in the constraint limits. In this case, elements are selected from the candidate set based, as a selection function, on the element with the highest coefficient in the objective functions. As a feasibility function, it was considered that a selected element from the set of candidates can only be added if the number of restrictions satisfied by adding it does not decrease.

4.2 Randomized methods

Two algorithms based on random search were implemented. The first algorithm consists of a noisy construction. Such noise was randomly generated with a uniform distribution and can be parameterized to be executed with different values of each parameter as the case may be. By using this randomized method, MC1N and MC2N were obtained. MC1N uses the first constructive method described in Section 4.1 while MC2N uses the second constructive method described in Section 4.1.

The second algorithm consists of a construction with the GRASP method. According to Rivera (2020), "GRASP is an iterative procedure consisting of a Construction phase and a Local Search phase." In this implementation, the cardinality k was taken into account, which can be parameterized to execute with different values of each parameter, even if it is the case. When we are based on cardinality, we will get a list containing k number of candidates to be considered in each iteration. Once the randomness occurs for the previous methods described,the process becomes deterministic. By using the construction withe the GRASP method, MC1G and MC2G were obtained. MC1G uses the first constructive method described in Section 4.1 while MC2G uses the second constructive method described in Section 4.1.

Note that for the noisy construction and the GRASP construction algorithms, the described process was repeated for 30 iterations and Pareto Frontier was selected from the solutions obtained.

4.3 ILS algorithm

The ILS algorithm consists of iteratively improving and perturbing the solution. The development of the ILS method was based on Figure 1, extracted from Rivera (2020).

Algorithm 1 Iterated local search

1: $s_0 = \text{GenerateInitialSolution}$
2: $s^* = \text{LocalSearch}(s_0)$
3: **repeat**
4: $\quad s' = \text{Perturbation}(s^*, history)$
5: $\quad s^{*\prime} = \text{LocalSearch}(s')$
6: $\quad s^* = \text{AcceptanceCriterion}(s^*, s^{*\prime}, history)$
7: **until** termination condition met

Figure 1: Algorithm ILS

To begin, the proposed functions are used to develop the ILS algorithm:

- Change a percentage of the initial solution (1) : This method receives an initial solution denoted as s_o. The local search algorithm deployed is based on the idea of altering a predetermined percentage of s_o. The altering process consists on the following: if an item in s_o is assigned a value of 1, it will now have a value of 0, and vice-versa. The altering process ends until s_o is changed by the preestablished predetermined percentage and let the solution obtained be denoted as n_o. Then, function (3) is evaluated with n_o and s_o, respectively. If such procedure returns true, the method returns n_o. On the other hand, the method returns s_o.

- Change ones to zeros and vice-versa (2) : This method receives an initial solution denoted as s_o and randomly generates an index. If the value of the randomly generated index in s_o has an associated value of 1, it will now have a value of 0, and vice-versa. Let the new solution obtained be denoted as n_o. Then, function (3) is evaluated with n_o and s_o, respectively. If such procedure returns true, the method returns n_o. On the other hand, the method returns s_o.

- Restriction satisfaction test and Pareto dominance relation test (3) : This method receives two solutions, n_o and s_o, respectively. Two criterion are established. The first makes a comparison between the number of satisfied restrictions between n_o and s_o. If the number of satisfied restrictions does not decrease, then a second criteria is tested. The second criteria, a Pareto dominance relation test, is evaluated between n_o and s_o. If n_o is a nondominated the method will return true. Otherwise, the method will return false.

In regards to the development of the algorithm, lets take into consideration the following premises. The initial solution utilized, shown in Figure 1 as s_o, was obtained by using a constructive algorithm. Afterwards, function (1) is applied to s_o and the solution obtained is denoted as s^*. Then, a termination condition is then established in which the algorithm is going to stop once it has met a time limit of 5 minutes. Afterwards, s' is obtained from applying function (2) to s^*. Then, function (1) is applied on s' to obtain $s^{*'}$. Afterwards, function (3) is evaluated on $s^{*'}$ and s^*. If function (3) returns true, s^* is updated to $s^{*'}$ and $s^{*'}$ is added to a list which contains a feasible set of solutions for the Pareto Frontier. On the other hand, if s^* dominates $s^{*'}$, s^* is not updated. In the previous scenarios presented, the time is updated in order to prevent the termination criteria from being violated. The algorithm is going to iteratively repeat the process, encapsulated after the repeat statement shown in Figure 1, until the termination criteria is met. Once the termination criteria is met, the algorithm applies a Pareto dominance relation test to the list of feasible solutions. Hence, a set of solutions conforming the Pareto Frontier are obtained.

4.4 VNS algorithm

The latent structure of the VNS algorithm is the following: if the solution has been improved, the current solution is replaced; otherwise, the neighborhood structure is modified. Note that the proposed function (3) from Section 4.3 is utilized for the development of the algorithm. The development of the VNS method is based on Figure 2, extracted from Rivera (2020).

```
procedure VNS()
    s ← initial_solution()
    j = 1;
    while j ≤ number_of_neighborhoods
        Find s' ∈ Nj(s,)
        while s' is not locally optimal do
            Find s'' ∈ Nj(s') with f(s'') < f(s');
            s' ← s'';
        end
        if f(s') < f(s) do
                j = 1;
                s ← s';
        else
                j = j + 1;
        end
    end
    return s
end VNS
```

Figure 2: Algorithm VNS

To begin, the following local search and three neighborhoods function considered when developing the algorithm. Note that the order in which they were visited was based on the algorithmic complexity.

- First Neighborhood: This method receives an initial solution denoted as s_o. In such a random item in s_o that has value of 1, it is now have a value of 0. Let the new solution obtained be denoted as n_o. Then, the proposed function (3) is evaluated with n_o and s_o, respectively. If such procedure returns true, the method returns n_o. On the other hand, the method returns s_o.

- Second Neighborhood: This method receives an initial solution denoted as s_o. This method changes two randomly selected items in s_o who have a zero and updates them for a one. The new solution obtained is denoted as n_o. Then, the proposed function (3) is evaluated with n_o and s_o, respectively. If such procedure returns true, the method returns n_o. On the other hand, the method returns s_o.

- Third Neighborhood: This method receives an initial solution denoted as s_o. This method changes two randomly selected items in s_o which have a one and updates them for a zero. The new solution obtained is denoted as n_o. Then, the proposed function (3) is evaluated with n_o and s_o, respectively. If such procedure returns true, the method returns n_o. On the other hand, the method returns s_o.

- Perturbation: This method receives an initial solution denoted as s_o. The local search algorithm deployed is based on the idea of altering a randomly selected item from s_o that has a 0 for a 1, obtaining a new solution denoted n_o. Then, the proposed function (3) is evaluated with n_o and s_o, respectively. If such procedure returns true, the method returns n_o. On the other hand, the method returns s_o.

In regards to the development of the algorithm, lets take into consideration the following premises. The initial solution utilized, shown in Figure 2 as s, was obtained by using a previously developed constructive algorithm. Also, the variable j, which indicates the neighborhood number we are examining is initialized to one, indicating the first neighborhood. Afterwards, a while loop indicates that the preceding code must be repeated based on a condition: j must be less than or equal to the number of neighbors. Then, a termination condition is then established in which the algorithm is going to stop once it has met a time limit of five minutes. Taking into account the value of j, a non-dominated solution, denoted as s', is obtained from the j_{th} neighborhood for s. Afterwards, a local search algorithm denoted as Perturbation is evaluated on s' to obtain a locally optimal solution denoted as s''. Then, s' becomes s''. Then, the

proposed function (3) is evaluated with s' and s, respectively. If the previous procedure returns true, j is assigned a value of 1 and s is updated to s' and $s^{'}$ is added to a list which contains a feasible set of solutions for the Pareto Front. Otherwise, j increases by a unit. The algorithm will then iteratively repeat the process, encapsulated after the while statement shown in Figure 2, until the termination criteria is met. In the previous scenarios presented, the time is updated in order to prevent the termination criteria from being violated. Once the termination criteria is met, the algorithm applies a Pareto dominance relation test to the list of feasible solutions. Hence, a set of solutions conforming the Pareto Frontier are obtained.

4.5 HEA

As previously mentioned, the development of the HEA took into account the genetic algorithm (GA) as the evolutionary search strategy and the iterated local search (ILS) as the methods based on local search. The development of the HEA is based on Figure 3, extracted from Rivera (2020).

```
procedure Genetic_Algorithms()
    for i = 1 to population_size
        P[i] = generate_solution();
    end;
    for i = 1 to generations
        for j = 1 to number_of_children
            (x, y) ← selection(P[]);
            son[j] ← crossover(x, y);
            if random < prob_mutation do
                son[j] ← mutation();
            end;
        end;
        P[] ← update(P[]);
    end;
    return better solution in P[];
end Genetic_Algorithms
```

Figure 3: Genetic Algorithm

The initial population is obtained from the GRASP algorithm described as MC1G with a cardinality of k=10. As described in Rivera (2020), "from this initial situation, a series of iterations is carried out, each of which simulates the creation of a new generation of individuals from the previous generation." In each iteration there is a selection and a crossover operation. The selection operation is based on a technique denoted as selection by tournament. In this case, four solutions are randomly selected and the two that have the greatest distance from each other are chosen. This procedure was done in order to avoid being stuck in local optimums. Afterwards, from the two solutions obtained from the selection operation, the crossover operation is performed. The crossover operation is based on a technique denoted as cross at a point. In this case, as described in Rivera (2020), "two parents are selected and a crossing point is chosen to divide the chromosome. Children result from the exchange of genetic information." Hence, this method will return a solution that contains segments from the two solutions that the method received. Next, a randomly generated value is computed. If the randomly generated value is less than a predefined mutation probability, then the proposed function (2) from Section 4.3 is performed on the solution and the solution is updated. If the randomly generated value is greater than a predefined mutation probability, the solution is not updated. Then a test is effectuated to determine if the number of satisfied restrictions does not decrease when comparing the new solution obtained and those from the population. If the number of satisfied solutions decreases, the new solution is not added to the population. If the number of satisfied solutions does not decrease, the new solution is added to the population and a Pareto dominance relation test is evaluated on the population. The result obtained from the previous operation will be the

updated population. This process is iteratively computed until the 5 minute termination criterion is met. This method will return a set of solutions conforming the Pareto Frontier.

4.6 Pareto Front : Indexes I_1 , I_2

The previous algorithms described return the Pareto Front, this implies that the set of solutions obtained for each method will consists of a set of nondominated solutions. Among the different strategies to compare approximations to the Pareto Frontier, in this work two quality indicators based on cardinality will be used : indexes I_1 and I_2.

This method can be applied when comparing more than one set. However, for simplification processes and to further explain I_1 and I_2, lets consider two sets that approximate the Pareto Frontier, A and B (each found with different methods). In this scenario, the set F can be defined with the set of non-dominated solutions resulting from the union of the sets A and B.

Probability of generating solutions at the Pareto Frontier I_1 : This indicator calculates the relationship between the number of non-dominated solutions generated by each method and the total number of solutions at the Pareto F frontier and is calculated as shown in Eq.(4).

$$I_1 = \frac{|A \cap F|}{|F|}, \tag{4}$$

Probability of generating non-dominated solutions I_2: This indicator calculates the relationship between the non-dominated solutions generated by each method and the total number of non-dominated solutions generated by said method and is calculatated as shown in Eq.(5).

$$I_2 = \frac{|A \cap F|}{|A|}, \tag{5}$$

5 Computational experimentation

The Python programming language on the platform Anaconda-Navigator was used for the implementation of this algorithms. It was developed on a computer with a Core i5 processor with 4 Gigs of RAM memory. The data used was provided from the Heuristics course offered at EAFIT University and is not public. Section 5.1 presents the parameterization of the algorithm. Then, Section 5.2 presents the comparisons of the methods described in Section 4. Lastly, Section 5.3 presents the computing time.

5.1 Algorithm Parameterization

Firstly, a comparison was effectuated in order to decide where the initial population was going to be obtained from. Hence, the hybrid evolutionary algorithm is run by considering as the initial population the results obtained from the GRASP algorithm with a cardinality of k= 6 and k= 10. Table 2 shows respective indexes I_1 and I_2 obtained:

Table 2: HEA with GRASP as initial population: Indexes I_1 and I_2

Iteration	k= 10 - I_1	k= 6 - I_1	k=10 - I_2	k= 6 - I_2
1	0.73	0.27	1	0.6
2	0.55	0.44	1	0.8
3	0.1	0.9	1	1
4	1	0	1	0
5	0	1	0	1
6	0.33	0.66	1	1
7	0.62	0.38	0.83	1
8	0.37	0.63	0.75	1
9	1	0	1	0
10	0.16	0.83	1	1
11	0.27	0.73	1	1
12	0.5	0.5	1	1
13	0.83	0.17	1	1
14	1	0	1	0
15	0.66	0.33	1	1
16	0.66	0.33	1	0
17	0.55	0.44	1	1
18	0.5	0.5	1	1
19	0.66	0.33	1	1
20	0.33	0.66	1	0.8
Total	10.82	9.1	17.58	15.20
Average	0.54	0.45	0.88	0.76

From Table 2 it can be concluded that when the cardinality is of greater magnitude, a better performance will be obtained for the indexes I_1 and I_2. Specifically, when considering the GRASP algorithm with a cardinality of k=10 as the initial population, it is 1.25 % more likely to be in the Pareto Frontier as opposed to when considering a cardinality of k= 6. Additionally, as what can be observed from I_2, when k=10, there is a greater probability that non-dominated solutions are generated. Specifically, it is 1.16 % more likely for a cardinality of k=10 to generate non-dominated solutions as opposed to when considering a cardinality of k = 6. The results obtained make sense because when greater aleatority is considered, such as that of a greater cardinality, the algorithm is prone to evade being stuck in local optimums.

Hence, from the previous conclusion, the initial population is going to be obtained from the GRASP algorithm with a cardinality of k= 10. Afterwards, the permutation probability parameter was parametrized considering the following probabilities: 0.4, 0.6 and 0.8. The results obtained for the indexes I_1 and I_2 can be seen in Table 3:

Table 3: HEA: Indexes I_1 and I_2

Iteration	0.80 - I_1	0.60 - I_1	0.40 - I_1	0.80 - I_2	0.6 - I_2	0.40 - I_2
1	1	0	0	1	0	0
2	0	1	0	0	1	0
3	0.33	0.33	0.33	1	1	1
4	0	1	0	0	1	0
5	0	0.5	0.5	0	1	1
6	0.33	0.33	0.33	1	1	1
7	0.77	0	0.23	0.76	0	0.42
8	0	0.55	0.45	0	0.78	0.53
9	0	1	0	0	1	0
10	0.33	0.33	0.33	1	1	1
11	0.33	0.33	0.33	1	1	1
12	0	1	0	0	1	0
13	1	0	0	1	0	0
14	0	0	1	0	0	1
15	1	0	0	1	0	0
16	0.33	0.33	0.33	1	1	1
17	1	0	0	1	0	0
18	0.18	0.27	0.54	0.66	0.75	1
19	0	0	1	0	0	1
20	0.33	0.33	0.33	1	1	1
Total	6.93	7.3	5.7	11.42	12.53	10.95
Average	0.34	0.36	0.28	0.57	0.62	0.54

From Table 3 it can be concluded that when the permutation probability is 60%, the indexes I_1 and I_2 have a better performance that when considering a permutation probability of 40% and 80%. To begin, Table 2 shows the total and average results obtained for each column. As previously mentioned, the results indicate that the solutions when the percentage permutation probability is equal to 60% are more likely to be members of the Pareto Frontier obtained when joining Pareto Frontiers for each permutation probability evaluated and have a greater probability of generating non-dominated solutions. Specifically, solutions with a permutation probability of 60% are 1.28% and 1.05% more likely to be in the Pareto Frontier as opposed to when considering a permutation probability of 40% and 80%, respectively. Likewise, solutions with a permutation probability of 60% are 1.14% and 1.08% to be non-dominated solutions as opposed to when considering a permutation probability of 40% and 80%, respectively. Additionally, when considering all the iterations, insights obtained from I_1 indicate that overall, 36% of the solutions obtained when considering a permutation probability of 60% are members of the Pareto Frontier. Additionally, it can be concluded that when comparing exclusively the results obtained with a permutation probability of 40% and 80%, according to I_1 and I_2, a permutation probability of 80% shows significantly better performance.

Hence, from the analysis presented, two major findings where obtained. Firstly,the initial population was going to be obtained from a GRASP algorithm with a cardinality of k =10. Secondly, the permutation probability is going to be 60%.

5.2 Comparisons with the randomized and constructive methods developed in previous works

Comparisons with the local search, randomized and constructive methods developed in previous works (thoroughly described in the subsection 4.1, 4.2, 4.3 and 4.4) is also performed. As previously indicated, the constructive methods are represented as MC1 and MC2. While the randomized methods are represented as MC1N, MC1G, MC2N and MC2G. It is worth noting that the first part of the notation

MC1/2 indicates from which constructive algorithm it was obtained from and the last capital letter indicates whether it was obtained using noisy construction(N) or the GRASP algorithm(G). In Table 3 and Table 4, index I_1 and index I_2 are calculated for the algorithms previously described and the hybrid evolutionary algorithm presented.

Table 4: All method comparison: Index I_1

Iteration	VNS	ILS 0.10	MC1	MC2	MC2N	MC2G	MC1N	MC1G	HEA
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0.14	0	0	0	0.85	0	0	0
6	0	0.33	0	0	0	0	0	0.33	0.33
7	1	0	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0
11	0	0.12	0	0	0	0.375	0	0.25	0.25
12	0	0.2	0	0	0	0.4	0	0	0.4
13	0	1	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0.8	0	0	0.2
18	0	0	0	0	0	0.3	0	0.3	0.4
19	0.66	0.33	0	0	0	0	0	0	0
20	0	1	0	0	0	0	0	0	0
Total	10.32	6.45	0	0	0	2.73	0	0.88	1.58
Average	0.51	0.32	0	0	0	0.13	0	0.044	0.079

Table 5: All method comparison: Index I_2

Iteration	VNS	ILS 0.10	MC1	MC2	MC2R	MC2G	MC1R	MC1G	HEA
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0.85	0	0	0
6	0	1	0	0	0	0	0	1	1
7	0	0	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0
11	0	1	0	0	0	0.375	0	0.66	0.66
12	0	0.5	0	0	0	0.66	0	0	1
13	0	1	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	1	0	0	1
18	0	0	0	0	0	1	0	1	1
19	1	1	0	0	0	0	0	0	0
20	0	1	0	0	0	0	0	0	0
Total	9	8.5	0	0	0	3.88	0	2.66	4.66
Average	0.45	0.42	0	0	0	0.19	0	0.13	0.23

Firstly, it is fair to say that the algorithms of MC1, MC2, MC1R and MC2R are the least likely to be members of the resulting Pareto Frontier obtained when merging the Pareto Frontiers solutions for each method evaluated. By analyzing the performance for MC1 and MC2, it can be said that solutions that build up on them (such as VNS, ILS, MC2G, MC1G) are more likely to be members of the resulting Pareto Frontier. However, this is not the case for MC1R and MC2R since it performed equally as poorly. From Table 3, it can be concluded that the VNS and ILS algorithms presented generate solutions that are more likely to be members of the resulting Pareto Frontier and be non-dominated. Specifically, solutions from these algorithms appeared in the result Pareto Frontier 51%, 32% of time respectively. This trait appeared to a lesser extent for other algorithms presented: 13% for MC2G and 7.9% for HEA and 4.4% for MC1G. Hence, a solution is 6.45% more likely to be a member of the Pareto Frontier and 1.95% more likely to be a non-dominated solution when being generated with the VNS algorithm as opposed to the HEA algorithm. Likewise, a solution is 4.05% more likely to be a member of the Pareto Frontier and 2.26% more likely to be a non-dominated solution when being generated with the ILS algorithm as opposed to the HEA algorithm.

5.3 Computing time

As previously mentioned, the computing time for the VNS, ILS and HEA algorithms was 5 minutes per iteration in the data file. This value is much greater than that of the algorithms developed in previous work. To further expand, the average computing time for MC1 and MC2 was 1.216 seconds and 1.157 seconds, respectively. In regards to the randomized methods, the average iteration time for MC1N was 35.78 seconds, MC2N was 35.72 seconds, for MC1G was 87.52 seconds and for MC2G was 76.9 seconds.

6 Conclusions

From the presented work, main conclusions can be obtained. First, it is important to note that according to the analysis effectuated in Section 5, the VNS algorithm provided a better performance, according to the calculated indexes $I1$ and $I2$, that the other algorithms presented. According to the presented indexes, in terms of performance, the ILS and the HEA obtain the second and third place, respectively. Additionally, as what can be seen in Section 5.2, using the solution obtained from the algorithm previously computed has proven to be an effective strategy in creating solutions that are members of the Pareto Frontier. The previous affirmation can be supported by the fact that for the grand majority of the iterations presented in Table 4 and Table 5, the resulting Pareto Frontier was a product of solutions obtained by the VNS, ILS and HEA algorithms. Furthermore, the VNS, ILS and HEA algorithms, when taking into account Indexes $I1$ and $I2$, have proven to obtain better performance than that of the MC1, MC2, MC1R, MC2R and even MC1G and MC2G. Additionally, it is worth noting that even tough calculating the Pareto Frontier of a set of solutions, through the use of the Pareto dominance relation test, is a very computationally expensive procedure to execute, the use of such to obtain the indexes examined in this paper facilitated the analyses and provided an objective comparison of the results presented. In terms of the results obtained, it is worth mentioning that since the KP is NP-Hard, it cannot be guaranteed that the heuristics method will provide an optimal and feasible solution.
For further projects and investigations, two ideas would like to be presented. Firstly, it would be interesting to see if considering the MC2G, as opposed to the MC1G, as the initial solution with a cardinality of k= 10 would improve the performance of the HEA. Secondly, as opposed to considering only one mutation function, it would be interesting if multiple mutation functions (such as the neighborhoods described for the VNS algorithm) would be considered. Also, after this implementation is made, as presented in the referenced literature, it would be interesting to see how and if the order in which the neighborhoods are visited affects the set of non dominated solutions obtained.

Acknowledgment

The author wishes to express her gratitude to the Center for Scientific Computing of the EAFIT University for their collaboration and for facilitating the performance of computational experiments.

References

Aarts, Emile, Emile HL Aarts, and Jan Karel Lenstra (2003). *Local search in combinatorial optimization*. Princeton University Press.

Bercachi, Maroun (2010). "A new hybrid method between VNS and SEA to improve results on the 0-1 multidimensional knapsack problem". In:

Biglar, Abbas (Feb. 2018). *Some applications of Knapsack problem*. DOI: 10.13140/RG.2.2.15115.39209.

Djannaty, Farhad and Saber Doostdar (2008). "A hybrid genetic algorithm for the multidimensional knapsack problem". In: *International Journal of Contemporary Mathematical Sciences* 3.9, pp. 443–456.

Dorigo, Marco and Gianni Di Caro (1999). "Ant colony optimization: a new meta-heuristic". In: *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. Vol. 2. IEEE, pp. 1470–1477.

Drake, John H, Nikolaos Kililis, and Ender Özcan (2013). "Generation of VNS components with grammatical evolution for vehicle routing". In: *European conference on genetic programming*. Springer, pp. 25–36.

Hentenryck, Pascal Van and Laurent Michel (2009). *Constraint-based local search*. The MIT press.

Hoos, Holger and Thomas Stützle (May 2000). "Local Search Algorithms for SAT: An Empirical Evaluation". In: *J. Automated Reasoning* 24, pp. 421–481. DOI: 10.1023/A:1006350622830.

Jakob Puchinger Günther Raidl, Ulrich Pferschy (2010). "The Multidimensional Knapsack Problem: Structure and Algorithms". In:

Kafafy, Ahmed, Ahmed Bounekkar, and Stéphane Bonnevay (2011). "A hybrid evolutionary metaheuristics (HEMH) applied on 0/1 multiobjective knapsack problems". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 497–504.

Knowles, Joshua D (2002). "Local-search and hybrid evolutionary algorithms for Pareto optimization". PhD thesis. University of Reading Reading.

Laabadi, Soukaina et al. (2018). "The 0/1 multidimensional knapsack problem and its variants: a survey of practical models and heuristic approaches". In: *American Journal of Operations Research* 8.05, p. 395.

Labrada-Nueva Enríquez-Urbano, García-Oji (2007). "Un algoritmo de búsqueda local iterada como solución al problema de la mochilla". In: *European conference on genetic programming*. Springer, pp. 25–36.

Lorie J., L.J. Savage (1955). "Three problems in capital rationing. The Journal of Business". In:

Lourenço, Helena R, Olivier C Martin, and Thomas Stützle (2003). "Iterated local search". In: *Handbook of metaheuristics*. Springer, pp. 320–353.

Mallawaarachchi, Vijini (2019). "Introduction to Genetic Algorithms". In:

Manne A.S., H.M. Markowitz (1957). "On the solution of discrete programming problems". In:

Puchinger, Jakob, Günther R Raidl, and Ulrich Pferschy (2006). "The core concept for the multidimensional knapsack problem". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, pp. 195–208.

Rajkumar, M et al. (2011). "A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints". In: *International Journal of Production Research* 49.8, pp. 2409–2423.

Rivera (2020). "Local Search and Genetic Algorithms Presentations". In:

Tan, Raymond R (2007). "Hybrid evolutionary computation for the development of pollution prevention and control strategies". In: *Journal of Cleaner Production* 15.10, pp. 902–906.

Weisstein (1988). "NP-Problem". In: *From MathWorld–A Wolfram Web Resource* 37.1, pp. 79–100.

Yang, Xin-She (2010). *Engineering optimization: an introduction with metaheuristic applications*. John Wiley & Sons.