Polibits
ISSN: 1870-9044
polibits@nlp.cic.ipn.mx
Instituto Politécnico Nacional
México

André, Leanderson; Stubs Parpinelli, Rafael
The Multiple Knapsack Problem Approached by a Binary Differential Evolution Algorithm with Adaptive Parameters
Polibits, vol. 51, enero-junio, 2015, pp. 47-54
Instituto Politécnico Nacional
Distrito Federal, México

ISSN 2395-8618

The Multiple Knapsack Problem Approached by a Binary Differential Evolution Algorithm with Adaptive Parameters

Leanderson André and Rafael Stubs Parpinelli

Abstract—**In this paper the well-known 0-1 Multiple Knapsack Problem (MKP) is approached by an adaptive Binary Differential Evolution (aBDE) algorithm. The MKP is a NP-hard optimization problem and the aim is to maximize the total profit subjected to the total weight in each knapsack that must be less than or equal to a given limit. The aBDE self adjusts two parameters, perturbation and mutation rates, using a linear adaptation procedure that changes their probabilities at each generation. Results were obtained using 11 instances of the problem with different degrees of complexity. The results were compared using aBDE, BDE, a standard Genetic Algorithm (GA) and its adaptive version (aGA), and an island-inspired Genetic Algorithm (IGA) and its adaptive version (aIGA). The results show that aBDE obtained better results than the other algorithms. This indicates that the proposed approach is an interesting and a promising strategy to control the parameters and for optimization of complex problems.**

Index Terms—**Adaptive parameter control, binary differential evolution, multiple knapsack problem, evolutionary computation.**

I. Introduction

The 0-1 Multiple Knapsack Problem (MKP) is a binary NP-hard combinatorial optimization problem that consists in given a set of items and a set of knapsacks, each item with a mass and a value, determine which item to include in which knapsack. The aim is to maximize the total profit subjected to the total weight in each knapsack that must be less than or equal to a given limit.

Different variants of the MKP can be easily adapted to real problems, such as, capital budgeting, cargo loading and others [1]. Hence, the optimization of resource allocation is one major concern in several areas of logistics, transportation and production [2]. In this way, the search for efficient methods to achieve such optimization aims to increase profits and reduce the use of raw materials.

According to the size of an instance (number of items and number of knapsacks) of the MKP, the search space can become too large to apply exact methods. Hence, a large number of heuristics and metaheuristics have been applied to the MKP. Some examples are the modified binary particle swarm optimization [3], the binary artificial fish swarm algorithm [4], and the binary fruit fly optimization algorithm [5]. In this work, it is investigated the performance of an adaptive Differential Evolution algorithm designed for binary problems.

The Differential Evolution (DE) algorithm is an Evolutionary Algorithm which is inspired by the laws of Darwin where stronger and adapted individuals have greater chances to survive and evolve [6]. Evolutionary Algorithms simulate the evolution of individuals through the selection, reproduction, crossover and mutation methods, stochastically producing better solutions at each generation [7]. In this analogy, the individuals are candidate solutions to optimize a given problem and the environment is the search space.

It is well documented in the literature that DE has a huge ability to perform well in continuous-valued search spaces [8]. However, for discrete or binary search spaces some adaptations are required [9]. Hence, this paper applies a Binary Differential Evolution (BDE) algorithm that is able to handle binary problems, in particular the 0-1 MKP. The BDE algorithm was first applied in [10] for the 0-1 MKP and the results obtained were promising. BDE consists in applying simple operators (crossover and bit-flip mutation) in candidate solutions represented as binary strings. In this work several different instances are approached.

It is known that the optimum values of the control parameters of an algorithm can change over the optimization process [11], directly influencing the efficiency of the method. As most metaheuristic algorithms, DE also has some control parameters to be adjusted. The parameters of an algorithm can be adjusted using one of two approaches: on-line or off-line. The off-line control, or parameter tuning, is performed prior to the execution of the algorithm. In this approach several tests are performed with different parameter settings in order to find good configurations for the parameters. In the on-line control, or parameter control, the values for the parameters change throughout the execution of the algorithm. The control of parameters during the optimization process has been consistently used by several optimization algorithms and applied in different problem domains [12], [13], [14], [15], [16]. In this way, a method to adapt the control parameters (crossover and mutation rates) of DE is applied. The aim is to explore how effective the on-line control strategy is in solving

Manuscript received on January 20, 2015, accepted for publication on March 8, 2015, published on June 15, 2015.

The authors are with the Graduate Program in Applied Computing, Departament of Computer Science, State University of Santa Catarina, Joinville, Brazil (e-mail: leanderson.andre@gmail.com, rafael.parpinelli@udesc.br).

ISSN 2395-8618

Algorithm 1 Binary Differential Evolution (BDE)

1: Parameters : $DIM, POP, ITER, PR, MUT$
2: Generate initial population randomly: $\overrightarrow{x}_i \in \{0,1\}^{DIM}$
3: Evaluate initial population with the fitness function $f(\overrightarrow{x}_i)$
4: **while** termination criteria not met **do**
5:   **for** i = 1 **to** POP **do**
6:     Select a random individual:
    $k \leftarrow random(1, POP)$, with $k \neq i$
7:     Select a random dimension:
    $j_{rand} \leftarrow random(1, DIM)$
8:     $\overrightarrow{y} \leftarrow \overrightarrow{x}_i$
9:     **for** j = 1 **to** DIM **do**
10:       **if** $(random(0,100) < PR)$ **or** $(j == j_{rand})$ **then**
11:         **if** $(random(0,100) < MUT\ )$ **then**
12:           BitFlip(y_j) {Mutation}
13:         **else**
14:           $y_j \leftarrow x_{kj}$ {Crossover}
15:         **end if**
16:       **end if**
17:     **end for**
18:     Evaluate $f(\overrightarrow{y})$
19:     **if** $(f(\overrightarrow{y}) > f(\overrightarrow{x}_i))$ **then** {Greedy Selection}
20:       $\overrightarrow{x}_i \leftarrow \overrightarrow{y}$
21:     **end if**
22:   **end for**
23:   Find current best solution $\overrightarrow{x}^*$
24: **end while**
25: Report results

the MKP.

In the following, Section II provides an overview of the Multiple Knapsack Problem. The Binary Differential Evolution algorithm is presented in Section III and the adaptive control parameter mechanism is presented in Section III-A. Section IV gives a brief description of Genetic Algorithms and island-inspired Genetic Algorithm both used in the experiments. The experiments and results are presented in Sections V and VI, respectively. Section VII concludes the paper with final remarks and future research.

II. Multiple Knapsack Problem

The 0-1 Multiple Knapsack Problem (MKP) is a well-known NP-hard combinatorial optimisation problem and its goal is to maximize the profit of items chosen to fulfil a set of knapsacks, subjected to constraints of capacity [2]. The MKP consists of m knapsacks of capacities $C_1, C_2, ...C_m$, and a set of n items $I = \{I_1, I_2, ...I_n\}$. The binary variables $X_i (i = 1, ..., n)$ represent selected items to be carried in m knapsacks. The X_i assumes 1 if item i is in the knapsack and 0 otherwise. Each item I_i has an associated profit $P_i \geq 0$ and weight $W_{ij} \geq 0$ for each knapsack j. The goal is to find the best combination of n items by maximizing the sum of profits P_i multiplied by the binary variable X_i, mathematically represented by Equation 1. Their constraints are the capacity $C_j \geq 0$ of each knapsack. Therefore, the sum of the values of X_i multiplied by W_{ij} must be less than or equal to C_j, represented mathematically by Equation 2.

$$\max \left(\sum_{i=1}^{n} (P_i \times X_i) \right) \tag{1}$$

$$\sum_{j=1}^{m} (W_{ij} \times X_i) \leq C_j \tag{2}$$

A binary exponential function with exponent n assembles all possibilities for n items respecting the capacity of each knapsack m. Hence, the MKP search space depends directly on the values of n and m. Therefore, to find the optimal solution should be tested all 2^n possibilities for each knapsack m, i.e., $m \times 2^n$ possibilities. Depending on the instance, the search space can become intractable by exact methods. In such cases, metaheuristic algorithms are indicated. Hence, the Binary Differential Evolution is an interesting algorithm to be applied to solve the MKP. The algorithm was designed for binary optimization and is shown in next section.

III. Binary Differential Evolution

The Binary Differential Evolution (BDE) [10] is a population-based metaheuristic inspired by the canonical Differential Evolution (DE) [6] and is adapted to handle binary problems. Specifically, the BDE approach is a modification of the DE/rand/1/bin variant.

In BDE, a population of binary encoded candidate solutions with size POP interact with each other. Each binary vector $\overrightarrow{x}_i = [x_{i1}, x_{i2}...x_{iDIM},]$ of dimension DIM is a candidate solution of the problem and is evaluated by an objective function $f(\overrightarrow{x}_i)$ with $i = [1, ..., POP]$. As well as the canonical DE, BDE combines each solution of the current population with a randomly chosen solution through the crossover operator. However, the main modification to the canonical DE, besides the binary representation, is the insertion of a bit-flip mutation operator. This modification adds to the algorithm the capacity to improve its global search ability, enabling diversity.

The pseudo-code of BDE is presented in Algorithm 1. The control parameters are the number of dimensions (DIM), the population size (POP), the maximum number of generations or iterations ($ITER$), the perturbation rate (PR) and the mutation rate (MUT)(line 1). The algorithm begins creating a random initial population (line 2) where each individual represents a point in the search space and is a possible solution to the problem. The individuals are binary vectors that are evaluated by a fitness function (line 3). An evolutive loop is performed until a termination criteria is met (line 4). The termination criteria can be to reach the maximum number of iterations $ITER$. The evolutive loop consists in creating new individuals through the processes of perturbation (mutation

ISSN 2395-8618

Algorithm 2 Genetic Algorithm (GA)

```
Parameters : DIM, POP, ITER, CR, MUT, ELI
Generate initial population randomly: x_i ∈ {0,1}^DIM
Evaluate initial population with the fitness function f(x_i)
while termination criteria not met do
    Find current best solution x*
    if (ELI) then
        Copy the best individual x* to next generation
    end if
    for i = 1 to (POP / 2) do
        Select two individuals k and y with tournament selection and k ≠ y
        if (random(0,100) < CR) {Uniform Crossover} then
            for j = 1 to DIM do
                if (random(0,100) < 50) then
                    offspring_a_j ← x_yj
                    offspring_b_j ← x_kj
                else
                    offspring_a_j ← x_kj
                    offspring_b_j ← x_yj
                end if
            end for
        end if
        for j = 1 to DIM do
            if (random(0,100) < MUT) then
                BitFlip(offspring_a_j) {Mutation}
            end if
            if (random(0,100) < MUT) then
                BitFlip(offspring_b_j) {Mutation}
            end if
            Add new individuals to next generation
        end for
    end for
    Evaluate new population with the fitness function f(x_i)
end while
Find current best solution x*
Report results
```

Algorithm 3 Island Inspired Genetic Algorithm (IGA)

```
Parameters : DIM, POP, ITER, CR, MUT
Generate initial population randomly: x_i ∈ {0,1}^DIM
Evaluate initial population with the fitness function f(x_i)
while termination criteria not met do
    for i = 1 to POP do
        Select an individual k, with tournament selection
        y ← x_k
        if (random(0,100) < CR) {Uniform Crossover} then
            for j = 1 to DIM do
                if (random(0,100) < 50) then
                    y_j ← x_ij
                end if
            end for
        end if
        for j = 1 to DIM do
            if (random(0,100) < MUT) then
                BitFlip(y_j) {Mutation}
            end if
        end for
        Evaluate f(y)
        if (f(y) > f(x_k)) then {Greedy Selection}
            x_k ← y
        end if
    end for
    Find current best solution x*
end while
Report results
```

and crossover) (lines 6-17), evaluation of the objective function (line 18), and a greedy selection (lines 19-21).

Inside the evolutive loop, two random indexes k and j_{rand} are selected at each generation. k represents the index of an individual in the population and must be different from the current index of individual i (line 6). j_{rand} represents the index of any dimension of the problem (line 7).

In line 8, the individual $\overrightarrow{x}_i$ is copied to a trial individual $\overrightarrow{y}$. Each dimension of the trial individual is perturbed (or modified) accordingly to the perturbation rate or if the index j is equal to index j_{rand} (line 10). The equality ensures that at least one dimension will be perturbed. The perturbation is carried out by the bit-flip mutation using its probability (line 11-12) or by the crossover operator (line 14).

From the new population of individuals the best solution $\overrightarrow{x}^*$ is found (line 23) and a new generation starts. Algorithm 1 terminates reporting the best solution obtained $\overrightarrow{x}^*$ (line 25).

A. Adaptive Binary Differential Evolution

The Adaptive Binary Differential Evolution (aBDE) algorithm aims to control two parameters: perturbation (PR) and mutation (MUT) rates. To achieve that, a set of discrete values is introduced for each of parameter. Once defined a set of values for each parameter, a single value is chosen at each generation through a roulette wheel selection strategy. The probability of choosing a value is initially defined equally which is subsequently adapted based on a criteria of success. If a selected value for a parameter yielded at least one individual in generation $t+1$ better than the best fitted individual from generation t, then the parameter value has a mark of success. Hence, if at the end of generation $t+1$ the parameter value was successful, its probability is increased with an α value, otherwise, it remains the same. The α is calculated by a linear increase as shows Equation 3:

$$\alpha = min + \left(\frac{max - min}{ITER} \times i \right), \tag{3}$$

ISSN 2395-8618

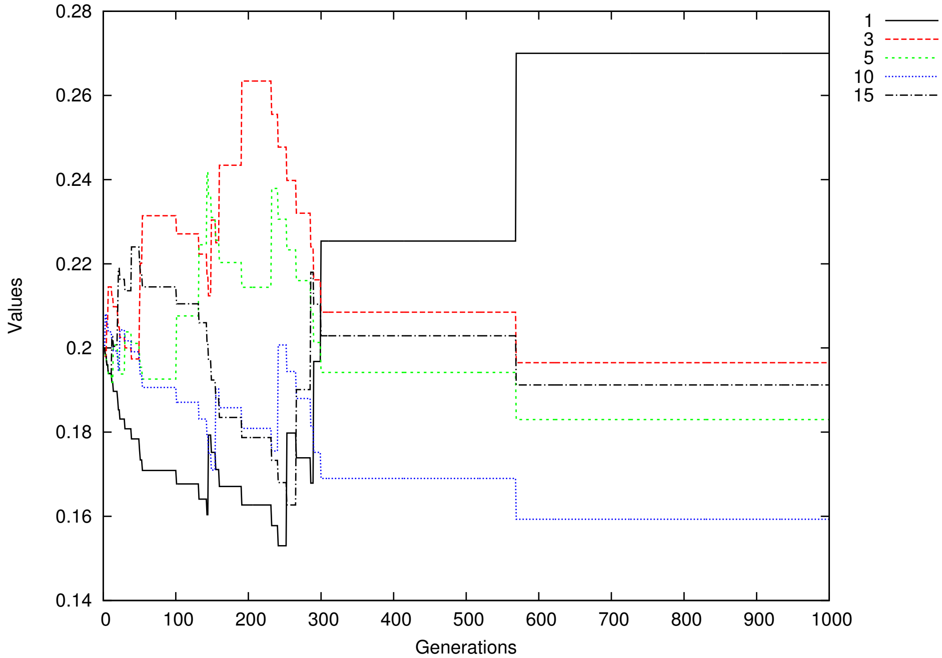

Fig. 1. Adaptive probabilities for mutation rate

where $ITER$ is the number of iterations, i is the current iteration, max is the maximum value of α and min is the minimum value of α. After adjusting the probabilities, the values are normalized between 0 and 1. To ensure a minimum of chance for each value of parameters, a β value is established.

IV. Description of GA and IGA

This section gives a brief description of two algorithms employed in the experiments.

A. Genetic Algorithm

The Genetic Algorithms (GA) are one of the best known and most used algorithms from the Evolutionary Computation field and was proposed by John Holland in 1975 [7]. The inspiration behind GA is based on Darwin's theory of evolution of species. In nature, individuals from different populations compete to survive. According to natural selection, stronger individuals and better adapted to the environment have a greater chance to survive and will continue their species. Thus, GA use the concepts of evolution as an intelligent process for optimization in finding good solutions.

The pseudo-code is presented in Algorithm 2. The control parameters are the number of dimensions (DIM), the population size (POP), the maximum number of generations or iterations ($ITER$), the crossover rate (CR), the mutation rate (MUT) and elitism (ELI) (line 1). The algorithm begins creating a random initial population (line 2) where each individual represents a point in the search space and is a possible solution to the problem. The individuals are binary vectors that are evaluated by a fitness function (line 3). An evolutive loop is performed until a termination criteria is met (line 4). The termination criteria can be to reach the maximum number of iterations $ITER$. From the population of individuals the best solution $\overrightarrow{x}^*$ is found (line 5). If elitism is applied, the best individual is placed in the next generation without any change (line 7). The evolutive loop consists in creating new individuals through the operators of crossover and mutation (lines 10-30), the generation of the new population (line 32) and the evaluation of the objective function (line 33).

Inside the evolutive loop, two individuals k and y are selected at each generation by tournament selection (line 10). The individuals are recombined accordingly to the crossover rate (line 11-21). Finally, it is applied the bit-flip mutation using its probability (line 22-30).

The new population is generated from the temporary population t (line 32), is evaluated by a fitness function (line 33) and a new generation starts. Algorithm 2 terminates reporting the best solution obtained $\overrightarrow{x}^*$ (line 35-36).

B. Island-inspired Genetic Algorithm

The Island-inspired Genetic Algorithm [17] is a meta-heuristic that uses one population of individuals as islands (island-model GA). This approach uses only one population where each individual is considered to be an island itself.

The pseudo-code of IGA is presented in Algorithm 3. The control parameters are the number of dimensions (DIM), the population size (POP), the maximum number of generations or iterations ($ITER$), the crossover rate (CR) and the mutation rate (MUT)(line 1). The algorithm begins creating

ISSN 2395-8618

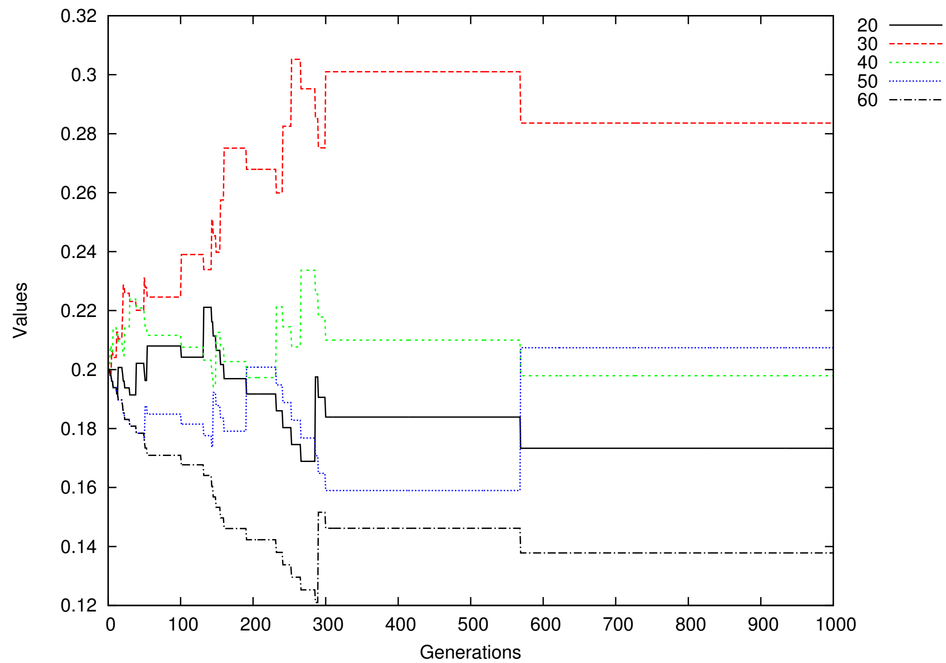

Fig. 2. Adaptive probabilities for perturbation rate

a random initial population (line 2) where each individual represents a point in the search space and is a possible solution to the problem. The individuals are binary vectors that are evaluated by a fitness function (line 3). An evolutive loop is performed until a termination criteria is met (line 4). The termination criteria can be to reach the maximum number of iterations $ITER$. The evolutive loop consists in creating new individuals through the operators of crossover and mutation (lines 6-19), the evaluation of the objective function (line 20), and a greedy selection (lines 21-23).

Inside the evolutive loop, an individual k is selected by tournament selection to where individual i must migrate (line 6). The migration process indicates that individual i will be able to exchange information with individual k. The interaction is made using an uniform crossover that produces one offspring. In line 8, the individual $\overrightarrow{x}_k$ is copied to a trial vector $\overrightarrow{y}$. The trial vector is recombined with individual i accordingly to the crossover rate (line 8-13). Finally, it is applied the bit-flip mutation using its probability (line 15-18).

From the new population of individuals the best solution $\overrightarrow{x}^*$ is found (line 25) and a new generation starts. Algorithm 3 terminates reporting the best solution obtained $\overrightarrow{x}^*$ (line 27).

V. Computational Experiments

For the experiments, 11 instances for the MKP were used[1]. Table I shows the instance reference, the optimum value, the number of knapsacks, and the number of items (or dimensions), respectively. For each instance, 100 independent runs were performed with randomly initialized populations.

[1] Available at: www.cs.nott.ac.uk/˜jqd/mkp/index.html

The algorithms were developed using ANSI C language and the experiments were run on an AMD Phenom II X4 (2.80GHz) with 4GB RAM, under Linux operating system.

TABLE I
Benchmark Instances for the MKP

Instance	Optimum Value	Knapsacks	Items
PB1	3090	4	27
PB2	3186	4	34
PB4	95168	2	29
PB5	2139	10	20
PB6	776	30	40
PB7	1035	30	37
PET7	16537	5	50
SENTO1	7772	30	60
SENTO2	8722	30	60
WEING8	624319	2	105
WEISHI30	11191	5	90

The parameters used for the BDE algorithm are: population size ($POP = 100$), number of iterations ($ITER = 1,000$), perturbation rate ($PR = 50\%$), mutation rate ($MUT = 5\%$).

The Genetic Algorithm (GA) and the Island-inspired Genetic Algorithm (IGA) use tournament selection, uniform crossover and elitism of one individual. For both algorithms the parameters are: population size ($POP = 100$), number of iterations ($ITER = 1,000$), tournament size ($T = 3$), crossover rate ($CR = 80\%$), mutation rate ($MUT = 5\%$), and elitism of one individual.

The strategy to adapt parameters is applied in all algorithms, BDE, GA and IGA, leading to its adaptive versions aBDE, aGA, and aIGA, respectively. The parameters adjusted are PR and MUT for aBDE, and CR and MUT for aGA and aIGA. Thus, the set of values for PR was defined as

ISSN 2395-8618

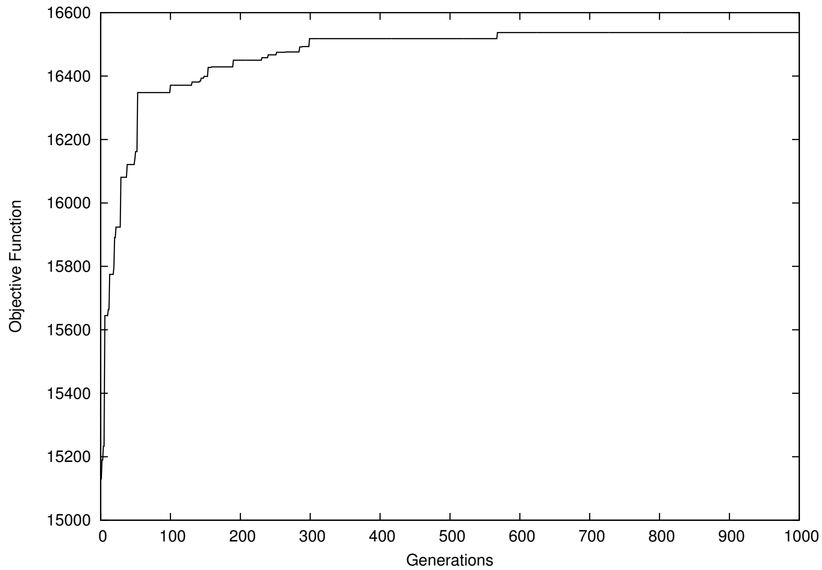

Fig. 3. Convergence graph for instance $PET7$

$\{20, 30, 40, 50, 60\}$ to aBDE, and the set of values for CR was defined as $\{50, 60, 70, 80, 90\}$ to aGA and aIGA. MUT was defined as $\{1, 3, 5, 10, 15\}$ for all algorithms.

A range between $[0.01, 0.1]$ was chosen for α and the β parameter was set to 0.01. The number of function evaluations is the same for all algorithms, resulting in a maximum of 100,000 function evaluations. All choices for the values of parameters were made empirically.

In all approaches, infeasible individuals in the population are fixed by dropping random items from the knapsack until feasibility is obtained. Feasibility of individual is verified inside the objective function as proposed in [18].

VI. Results and Analysis

Table II presents the average and the standard deviation of the best result (*Avg*±*Std*) obtained in all runs for each algorithm, the average number of objective function evaluations (*Eval*) required to achieve the optimum value, the success rate (*Success*) calculated as the percentage that the algorithm reached the optimum value, and the dominance information (P) indicating which algorithms are better than the others concerning both the average best result and the average number of function evaluations. If more than one algorithm is marked in the same benchmark means that they are non-dominated (neither of them are better than the other in both criteria). Also, for each algorithm, the last line (*Average*) shows the average of function evaluations and the average of success rate for all benchmarks. Best results are highlighted in bold.

Analyzing the results obtained by GA and IGA it is possible to notice that IGA achieve better results (success rate) in 8 instances ($PB1$, $PB2$, $PB4$, $PB6$, $PB7$, $PET7$, $SENTO1$, and $SENTO2$), except for $PB5$. This gain can be explained by the model used for exchange information that slows down the premature convergence of the algorithm allowing it to better explore the space of solutions.

Analyzing the results obtained by IGA and BDE we can notice that BDE achieved better results (success rate) in 3 instances ($SENTO2$, $WEING8$, and $WEISH30$), and equivalent results in other 3 instances ($PB4$, $PB6$, and $SENTO1$). In fact, the BDE obtained best results in instances with higher complexity. Also, the average success rate of BDE is better than the average success rate of IGA. This can be explained by the diversification power that BDE employs in its operators.

Comparing the results obtained by BDE and its adaptive version, aBDE, we can notice that the results (success rate) were even better when using the adaptive parameter control strategy for almost all instances except for $SENTO1$ and $WEISHI30$ and equal for $PB4$. Also, the average number of function evaluations decreased when using the parameter control strategy. This improvement can be explained by the adaptive choices for the values of parameters during the optimization process.

Analyzing the effectiveness of the adaptive parameter control strategy, it is possible to notice that aBDE, aIGA, and aGA obtained better success rates for the majority of the instances when compared to its non-adaptive versions. The improvement is boosted in aBDE which has a differentiated diversification mechanism.

Using the dominance information (P) from Table II, it is possible to notice that the Differential Evolution algorithm with adaptive parameter control, aBDE, is present in the non-dominated set in 8 out of 11 instances. This indicates that aBDE is robust concerning both criteria. The aBDE algorithm is dominated in instances $PB2$, $PB5$ and $SENTO1$.

In order to illustrate the behavior of the adaptive control strategy, Figures 1 and 2 show the adaptation of values

ISSN 2395-8618

TABLE II
RESULTS OBTAINED BY ALL ALGORITHMS FOR EACH INSTANCE.

Benchmark	GA				aGA			
	Avg±Std	Eval	Success	P	Avg±Std	Eval	Success	P
PB1	3085.26±10.78	34995.18	82.00%		3086.98±8.17	45491.35	86.00%	
PB2	3131.08±40.44	89051.75	17.00%		3142.10±32.96	91786.79	15.00%	
PB4	95071.01±551.51	9251.30	97.00%		94956.92±769.63	21115.21	91.00%	
PB5	2138.15±3.71	29852.48	**95.00%**	x	2136.62±5.90	33728.52	86.00%	
PB6	769.57±10.49	51759.22	68.00%		770.64±10.04	46877.06	72.00%	
PB7	1026.34±6.92	92079.76	17.00%		1024.34±7.98	92400.93	12.00%	
PET7	16428.88±47.93	100100.00	0.00%		16451.34±50.91	98634.27	6.00%	
SENTO1	7640.90±50.75	100100.00	0.00%		7678.39±80.06	95481.81	14.00%	
SENTO2	8620.05±37.74	100100.00	0.00%		8649.13±50.80	99942.68	1.00%	
WEING8	566282.95±12678.93	100100.00	0.00%		583830.05±20597.21	100100.00	0.00%	
WEISHI30	10824.70±92.10	100100.00	0.00%		10962.33±189.93	99851.97	3.00%	
Average		73408.15	34.18%			75037.32	35.09%	

Benchmark	IGA				aIGA			
	Avg±Std	Eval	Sucess	Pareto	Avg±Std	Eval	Sucess	Pareto
PB1	3090.00±0.00	13912.02	**100.00%**	x	3090.00±0.00	17559.92	**100.00%**	x
PB2	3173.19±17.20	74237.64	51.00%		3173.47±18.83	72674.13	**54.00%**	x
PB4	95168.00±0.00	7231.01	**100.00%**	x	95168.00±0.00	8102.60	**100.00%**	x
PB5	2137.13±5.32	30514.89	89.00%		2136.79±5.72	34976.06	87.00%	
PB6	775.86±1.39	12657.32	99.00%		775.89±1.09	12355.48	99.00%	
PB7	1034.32±2.22	42747.69	**83.00%**	x	1034.12±2.62	43877.91	78.00%	
PET7	16529.44±10.97	80221.17	60.00%		16530.22±10.11	76512.13	64.00%	
SENTO1	7771.64±2.06	47496.00	**97.00%**	x	7770.61±3.98	39808.61	89.00%	
SENTO2	8717.77±5.71	87966.44	49.00%		8718.85±4.54	71824.83	55.00%	
WEING8	612963.36±2750.51	100100.00	0.00%		623388.14±1432.22	72758.08	65.00%	
WEISH30	11159.03±13.71	100100.00	0.00%		11190.72±1.02	51125.45	93.00%	
Average		54289.47	66.18%			45597.74	80.36%	

Benchmark	BDE				aBDE			
	Avg±Std	Eval	Sucess	P	Avg±Std	Eval	Sucess	P
PB1	3089.07±4.96	14104.50	96.00%		3089.54±3.52	13074.74	**98.00%**	x
PB2	3144.55±28.43	91164.94	14.00%		3165.17±24.20	78323.80	40.00%	
PB4	95168.00±0.00	4672.21	**100.00%**	x	95168.00±0.00	5584.56	**100.00%**	x
PB5	2135.60±6.80	32052.98	80.00%		2136.79±5.72	26676.96	87.00%	
PB6	775.86±1.39	7200.84	99.00%		776.00±0.00	6865.16	**100.00%**	x
PB7	1034.12±2.57	35502.17	77.00%		1034.47±1.89	33620.13	**82.00%**	x
PET7	16524.58±19.07	65795.81	56.00%		16529.52±15.30	64335.20	**71.00%**	x
SENTO1	7771.44±3.53	17091.08	**97.00%**	x	7770.66±4.61	25110.83	91.00%	
SENTO2	8720.37±3.49	50493.94	67.00%		8721.17±2.37	42285.83	**78.00%**	x
WEING8	624062.37±770.56	55705.08	86.00%		624241.30±457.11	34517.30	**95.00%**	x
WEISHI30	11191.00±0.00	33645.37	**100.00%**	x	11190.84±0.78	26192.99	**96.00%**	x
Average		37038.99	79.27%			**32417.04**	**85.27%**	

for the mutation and perturbation rates, respectively. Also, a convergence plot is show in Figure 3. All three figures were acquired during a successful run of aBDE algorithm using instance $PET7$. For other instances, the behavior observed was similar.

In the first generation of the algorithm, all possibilities for the values of parameters have the same probabilities to be chosen. Through generations, these probabilities can change according to their success of creating better solutions, as explained in Section III-A. From Figures 1 and 2 one can note that in earlier generations, the probabilities of the values for each parameter change more often than in latter generations.

This can be explained by the diversity loss that occurs during the optimization process, as can be seen in the convergence plot (Figure 3). The adaptive method is able to better explore the values of parameters at the beginning of the optimization process, favoring the best values until its end.

VII. CONCLUSION

In this work, a Binary Differential Evolution algorithm with adaptive parameters was applied to the well-known 0-1 MKP. The Adaptive Binary Differential Evolution (aBDE) algorithm aims to control two parameters: perturbation (PR) and mutation (MUT) rates. To achieve that, a set of discrete values is introduced for each of parameter and it is updated based on a criteria of success. If a selected value for a parameter yielded at least one individual in generation $t+1$ better than the best fitted individual from generation t, then the parameter value has a mark of success. Hence, if at the end of generation $t+1$ the parameter value was successful, its probability is increased, otherwise, it remains the same.

Results obtained using 11 instances of the problem strongly suggest that the adaptive selection strategy has advantages when compared with fixed values. This advantages can be seen

ISSN 2395-8618

in the results (average success rate and average number of function evaluations) when comparing aBDE with the other algorithms. This indicates that the proposed approach is an interesting and promising strategy for optimization of complex problems.

As future work, we intend to apply the adaptive method in other metaheuristics. Also, it is planed to investigate the performance of the aBDE in other real-world problems.

Acknowledgment

Authors would like to thank Fundação de Amparo a Pesquisa e Inovação do Estado de Santa Catarina (FAPESC) by the financial support, as well as to State University of Santa Catarina (UDESC).

References

[1] M. Vasquez, J.-K. Hao *et al.*, "A hybrid approach for the 0-1 multidimensional knapsack problem," in *IJCAI*, 2001, pp. 328–333.

[2] A. Freville, "The multidimensional 0–1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, 2004.

[3] J. C. Bansal and K. Deep, "A modified binary particle swarm optimization for knapsack problems," *Applied Mathematics and Computation*, vol. 218, no. 22, pp. 11 042–11 061, 2012.

[4] M. A. K. Azad, A. M. A. Rocha, and E. M. Fernandes, "Improved binary artificial fish swarm algorithm for the 0–1 multidimensional knapsack problems," *Swarm and Evolutionary Computation*, vol. 14, pp. 66–75, 2014.

[5] L. Wang, X. long Zheng, and S. yao Wang, "A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem," *Knowledge-Based Systems*, vol. 48, no. 0, pp. 17–23, 2013.

[6] R. Storn and K. Price, "Differential evolution : A simple and efficient heuristic for global optimization over continuous spaces," *J. of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[7] K. De Jong, *Evolutionary Computation: A Unified Approach*, ser. Bradford Book. Mit Press, 2006.

[8] X.-S. Yang, "Chapter 6—differential evolution," in *Nature-Inspired Optimization Algorithms*. Oxford: Elsevier, 2014, pp. 89–97.

[9] J. Krause, J. Cordeiro, R. S. Parpinelli, and H. S. Lopes, "A survey of swarm algorithms applied to discrete optimization problems," *Swarm Intelligence and Bio-inspired Computation: Theory and Applications. Elsevier Science & Technology Books*, pp. 169–191, 2013.

[10] J. Krause, R. S. Parpinelli, and H. S. Lopes, "Proposta de um algoritmo inspirado em evolução diferencial aplicado ao problema multidimensional da mochila," *Anais do IX Encontro Nacional de Inteligência Artificial–ENIA. Curitiba, PR: SBC*, 2012.

[11] A. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, Jul 1999.

[12] L. André and R. S. Parpinelli, "Controle de parâmetros em inteligência de enxame e computação evolutiva," *Revista de Informática Teórica e Aplicada*, vol. 21, no. 2, pp. 83–128, 2014.

[13] D. Thierens, "An adaptive pursuit strategy for allocating operator probabilities," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, pp. 1539–1546.

[14] Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag, "Extreme value based adaptive operator selection," in *Parallel Problem Solving from Nature–PPSN X*. Springer, 2008, pp. 175–184.

[15] A. Aleti and I. Moser, "Studying feedback mechanisms for adaptive parameter control in evolutionary algorithms," in *IEEE Congress on Evolutionary Computation (CEC)*, June 2013, pp. 3117–3124.

[16] O. Kramer, "Evolutionary self-adaptation: a survey of operators and strategy parameters," *Evolutionary Intelligence*, vol. 3, no. 2, pp. 51–65, 2010.

[17] L. André and R. S. Parpinelli, "An island-inspired genetic algorithm with adaptive parameters applied to the multiple knapsack problem," in *Proceedings of the 5th International Conference on Metaheuristics and Nature Inspired Computing*, October 2014, pp. 1–2.

[18] A. Hoff, A. Løkketangen, and I. Mittet, "Genetic algorithms for 0/1 multidimensional knapsack problems," in *Proceedings Norsk Informatikk Konferanse*. Citeseer, 1996, pp. 291–301.