RAIRO-Oper. Res. 50 (2016) 47–66
DOI: 10.1051/ro/2015011

RAIRO Operations Research
www.rairo-ro.org

A PARTHENO-GENETIC ALGORITHM FOR DYNAMIC 0-1 MULTIDIMENSIONAL KNAPSACK PROBLEM *

Ali Nadi Ünal[1] and Gülgün Kayakutlu[2]

Abstract. Multidimensional Knapsack problem (MKP) is a well-known, NP-hard combinatorial optimization problem. Several metaheuristics or exact algorithms have been proposed to solve stationary MKP. This study aims to solve this difficult problem with dynamic conditions, testing a new evolutionary algorithm. In the present study, the Partheno-genetic algorithm (PGA) is tested by evolving parameters in time. Originality of the study is based on comparing the performances in static and dynamic conditions. First the effectiveness of the PGA is tested on both the stationary, and the dynamic MKP. Then, the improvements with different random restarting schemes are observed. The PGA achievements are shown in statistical and graphical analysis.

Mathematics Subject Classification. 90C27, 68T20.

Received April 15, 2014. Accepted April 20, 2015.

1. Introduction

Hard optimization problems are roughly defined as "problems that cannot be solved to optimality, or to any guaranteed bound, by any exact (deterministic) method within a "reasonable" time limit" [7]. Multiple Knapsack Problem (MKP) is a well-known hard and combinatorial optimization problem. It is accepted as NP-hard [11, 13, 14, 16]. Combinatorial optimization problems involve finding optimal solutions from a discrete set of feasible solutions [31]. Evolutionary Algorithm (EA), a sub-group of metaheuristics, is one of the most preferred methods to solve hard optimization problems [6, 31]. EA is a general term for some nature inspired optimization algorithms such as Genetic Algorithms (GAs), Evolution Strategies, Evolutionary Programming and Genetic Programming.

GAs have been widely used in different forms to solve stationary combinatorial optimization problems [10, 11, 14, 22, 32]. However, many real life problems are actually dynamic, for this reason a solution approach is expected to be adaptive to environmental changes [4]. In a changing environment, the problem-specific fitness evaluation function and constraints of the problem, such as design variables and environmental conditions, may change over time [45]. Evolutionary optimization in dynamic environments has become one of the most active

Keywords. Combinatorial optimization, dynamic environments, multidimensional knapsack problem, partheno-genetic algorithm.

* *The authors gratefully thank to the Referee and the Editor for the constructive comments and recommendations which definitely help to improve the readability and quality of the paper.*

[1] Turkish Air Force Academy, Aeronautics and Space Technologies Institute, Industrial Engineering Department, 34149 Yeşilyurt, İstanbul, Türkiye. anunal@hho.edu.tr

[2] İstanbul Technical University, Industrial Engineering Department, 34367 Maçka, İstanbul, Türkiye. kayakutlu@itu.edu.tr

Article published by EDP Sciences

research areas in the field of evolutionary computation [21, 37] and investigating the performance of GAs in dynamic environments has attracted a growing interest from GA's communit [45, 46].

Compared with static extensions, there are relatively far less reported publications about dynamic multiple knapsack problems [4]. This is one of the main motivations of the present study. Secondly, to the best of our knowledge, this is the first study that investigates the PGA for multiple knapsack problem in dynamic environments.

The paper is organized as follows: in Section 2 we give a brief explanation of the Multiple Knapsack Problem and the related work. Standard genetic algorithm and the PGA are described in Section 3. Experimental setup for static and dynamic environments are explained in Section 4. We present the results of the experiments in Section 5. Finally, some concluding remarks and future recommendations are given in Section 6.

2. Problem definition and related work

In this section we outline the Multidimensional Knapsack Problem in dynamic environment and give the related work on GAs in dynamic environments.

2.1. Definition of multidimensional knapsack problem

Before introducing the dynamic version of the problem, it may be helpful to review the static one. The MKP is widely studied combinatorial optimization problem. Both theoreticians and practitioners are interested in the problem; theoreticians think that the simple structured MKPs can be used as sub problems to solve more complicated ones, and practitioners think that these problems can model many industrial opportunities such as cutting stock, cargo loading and the capital budgeting [25].

There are some other names given to this problem in the literature such as *the multi constraint knapsack problem, the multi-knapsack problem, the multiple knapsack problem, m-dimensional knapsack problem*, and some authors also include the term *zero-one* in their name for the problem [11]. Basically, the MKP can be formulated as follows [30]:

$$\text{maximize} \quad f = \sum_{j=1}^{n} p_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} r_{i,j} x_j \leq b_i, i = 1, \ldots, m, \tag{2.2}$$

$$x_j \in \{0, 1\}, j = 1, \ldots, n, \tag{2.3}$$

$$\text{with} \quad p_j > 0, r_{i,j} \geq 0, b_i > 0. \tag{2.4}$$

In this formulation; the MKP consists of m knapsacks of capacities b_1, b_2, …, b_m and n objects, each of which has a profit p_j. If the jth object is included in the solution then the decision variable x_j equals to 1, otherwise it equals to 0. Unlike the simple version of the knapsack problem in which the weights (resource consumptions) of the objects are fixed, the weight of the jth object takes m values [25].

In fact, any 0/1 integer problem with non-negative coefficients can be formulated as a MKP [26]. Several effective algorithms (*i.e.*, exact algorithms, and heuristic or metaheuristic algorithms) have been developed to tackle this stationary type of problem [8,17,18,24,41]. For a good review of these algorithms, we refer the reader to [20].

2.2. Dynamic MKP

According to Simoes [35] "When the environment changes over time, resulting in modifications of the fitness function from one cycle to another, we say that we are in the presence of a dynamic environment". In other words, in a dynamic optimization problem; the objective function, the problem instance, or constraints may change

over time, and thus the optimum of that problem might change as well [21]. Literature includes benchmark generators that translate well known static problems into dynamic versions (refer to [4]).

In [35], on 0/1 dynamic knapsack problem, they used three types of changes in the capacity of knapsack: periodic changes between two values, three values and non-periodic changes between 3 different capacities. In each of the periodic experiments they started a total capacity C1 and after half a cycle the constraint was switched to C2. When using three values, after a complete cycle the capacity was changed to third value C3. Cycle lengths were 30, 100, 200 and 300 generations.

The term "cycle lengths" has the same meaning with "simulation time units" [4]. The less number of simulation time units yield more frequent changes and *vice versa*. In [4], a series of 1000 iterations were adopted as the frequency of changes. A benchmark problem that includes 100 items and 10 dimensions was used as the basis environment. After a change occurs, as in [9], the parameters were updated as stated below [4]:

$$\begin{aligned} p_j &= p_j * (1 + N(0, \sigma_p)), \\ r_{i,j} &= r_{i,j} * (1 + N(0, \sigma_r)), \\ b_i &= b_i * (1 + N(0, \sigma_b)). \end{aligned} \tag{2.5}$$

In (2.5), $r_{i,j}$ denote the weights of the objects and b_i denote the capacities of knapsacks, all standard deviations are assumed to be equal and set to 0.05. While Branke *et al.* in [9] restricted the dynamic changes to a fixed interval, neither lower nor upper bounds for those dynamic parameters were employed in [4]. Additionally, in [4], the best error to optimum was used as the performance measure. According to this measure, the average values of the best solution achieved for each environments on 30 runs are considered. The optimum of each environment was obtained using GAMS CPLEX solver.

Mori and Kita [29] surveyed the methods for applying the adaptation of GA to dynamic environments. They reported the characteristics of dynamic environments, and also surveyed the major kinds of GAs designed to cope with the dynamic environment. They categorized the GAs into two types: (i) the search-based approach and (ii) the memory-based approach.

Yang [45] proposed a hybrid memory and random immigrants scheme for genetic algorithms in dynamic environments. Results of an experimental study, based on systematically constructed dynamic environments, showed that memory-based immigrants scheme efficiently improved the performance of genetic algorithms in dynamic environments.

Branke *et al.* [9] compared and analyzed three different representations (*i.e.*, weight coding, binary representation and permutation representation) on the basis of a dynamic multidimensional knapsack problem. They concluded that the representation could have a tremendous effect on EA's performance in dynamic environments. They stated binary representation with penalty was extremely poor, because of slow improvement and feasibility considerations.

Uyar and Uyar [40] briefly outlined the environmental change methods, discussed their shortcomings and proposed a new method that could generate changes for a given severity level more reliably. Then presented the experimental set up and results for the new method and compared the new method with existing methods.

Yuan and Yang [46] tested a hybrid genetic algorithm by different functional dimensions, update frequencies, and displacement strengths in different types of dynamics. Also they compared with some other existing evolutionary algorithms for dynamic environments. The hybrid genetic algorithm had been illustrated its better capability to track the dynamic optimum, based on the results.

Ünal [39] compared two different approaches using GAs to adapt the changing environments for multiple knapsack problems. These approaches were named random immigrants based approach and memory based approach respectively. It was concluded that the memory based approach was more effective to adapt the changing environments.

Baykasoğlu and Ozsoydan [4] proposed an improved firefly algorithm (FA) to solve dynamic multidimensional knapsack problem. In order to evaluate the performance of the proposed algorithm, the same problem was also modelled and solved by genetic algorithm, differential evolution and original FA. Based on the computational

results and convergence capabilities they concluded that improved FA was a very powerful algorithm for solving the multidimensional knapsack problems for both static and dynamic environments.

3. Methodology

GAs sometimes fail adapt to a dynamic environment [29]. GAs tendency to converge prematurely in stationary problems and their lack of diversity in tracking optima are deficiencies that need do be addressed for dynamic environments [3]. Maintenance of the diversity is an essential requirement to apply the GA to dynamic environments [29].

There are several techniques that have been used to enhance the performance of the standart genetic algorithm in dynamic environments to maintain diversity [3,21]. A new genetic algorithm named partheno-genetic algorithm (PGA) overcomes the premature convergence problem and keeps the diversity of the population [34]. For this reason we tested the PGA, with different reinitializing schemes. Based on this explanation, we give a brief review about standard genetic algorithm, partheno-genetic algorithm and techniques for maintaining the diversity.

3.1. Standard genetic algorithm (SGA)

In 1970s, John Holland and his students, set theoretical foundations of genetic algorithms [1]. GAs are nature inspired optimization techniques based on the evolutionary process of biological organisms. Through generations, the most fitted individuals will survive and less fitted ones will be eliminated. This process is based on the principles of natural selection and "survival of the fittest" [1,11].

In a standard GA, each individual represents a possible solution of the problem to be solved. Randomly generated initial population, which composed of individuals, evolves toward better solutions through generations, using genetic operators (crossover and mutation) and selection procedures. An objective function is used to evaluate the fitness of each individual. The basic steps of a simple genetic algorithm are illustrated in Algorithm 1.

Algorithm 1. A pseudo code for simple GA.

```
Generate initial population;
Evaluate fitness values of individuals in population;
repeat
    Select parents from the population for mating;
    Recombine parents to produce offspring;
    Mutate the resulting offspring;
    Evaluate fitness of the offspring;
    Replace some or all the population by the offspring;
until a termination criterion is satisfied //e.g., a predefined iteration number
Results and visualization.
```

There is a huge amount of GA literature including books, theses and articles. For this reason only necessary explanation (used techniques in this paper) is given about genetic algorithms. We refer the reader to [15,36] for detailed information.

The first step of designing a genetic algorithm is creating an initial population that consists of individuals (chromosomes) [39]. This initial population can be generated randomly. Sometimes, a hybrid application of metaheuristics and Branch and Bound (in particular memetic algorithms), can be used to obtain a good starting solution [38]. To generate an initial population, individuals must be represented in a proper way. Direct or indirect representation schemes can be used. "A representation is called *direct* if it can be interpreted directly as a solution to the problem" [9]. In this representation, an individual consists of zeros and ones. If an item is included in the solution, the value of the corresponding position in the chromosome is 1. A simple example

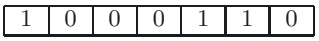

FIGURE 1. A simple example for binary representation, $n = 7$.

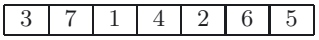

FIGURE 2. A simple example of permutation representation, $n = 7$.

for binary representation for an individual is shown in Figure 1. In this individual; the 1st, the 5th and the 6th items are included in the solution. If any of the knapsack's capacity is not exceeded, this is also a feasible solution.

The main drawback of the binary (direct) representation is the difficulty to maintain feasibility [9]. To maintain feasibility a number of procedures are encouraged to be implemented [11]: (i) to use a representation that ensures all solutions are feasible; (ii) to separate the evaluation of fitness and infeasibility; (iii) to use a heuristic operator to transform any infeasible solution into a feasible one; (iv) to apply a penalty function.

Unlike the direct representation, it is not necessary to design complicated repair mechanisms or to use penalty functions to ensure feasibility in indirect representations [9]. *Priority based encoding technique* [4], *permutation representation,* and *real valued representation with weight coding* [9] can be used as indirect representations. Figure 2 represents a simple permutation representation. In this representation, the 3rd item must be included in the solution firstly and the 7th item must be included secondly, and so on so forth. Inclusions of successive items must be held by considering the knapsack capacities. In this paper we used the permutation representation.

"For selection phase of genetic algorithms, several methods are used such as roulette wheel, stochastic universal sampling, sigma scaling, rank selection, tournament selection and steady state selection" [39]. For a detailed explanation of these methods, reader should refer to [27]. Additionally, a mathematical analysis of tournament selection, truncation selection, linear and exponential ranking selection, and proportional selection is carried out in depth in [5].

We used tournament selection in this study. In tournament selection, a predefined number of individuals are chosen randomly from the population and the best of them is copied from this group into the immediate population [5]. The pseudo code for the tournament selection is shown in Algorithm 2.

Algorithm 2. A pseudo-code for tournament selection.

```
currentpopulation ← P
selectedmatrix ← zeros
tournamentsize ← k
set t := 0;
while t < populationsize do
    t ← t + 1
    Select k individuals from P and evaluate
    row t of selectedmatrix ← the best of k individuals
end while
```

After implementing the selection, the next phase is recombining the population. During recombination process, a new individual solution is created from the information contained within two or more parent solutions. Several recombination operators can be implemented for permutation representations. Eiben and Smith in [12] describe these operators in detail. We use cycle crossover in this paper. In this operator, new offspring is created by dividing the parents into cycles. The procedure for constructing cycles is as follows [12]:

(1) Start with the first unused position and allele of Parent 1.

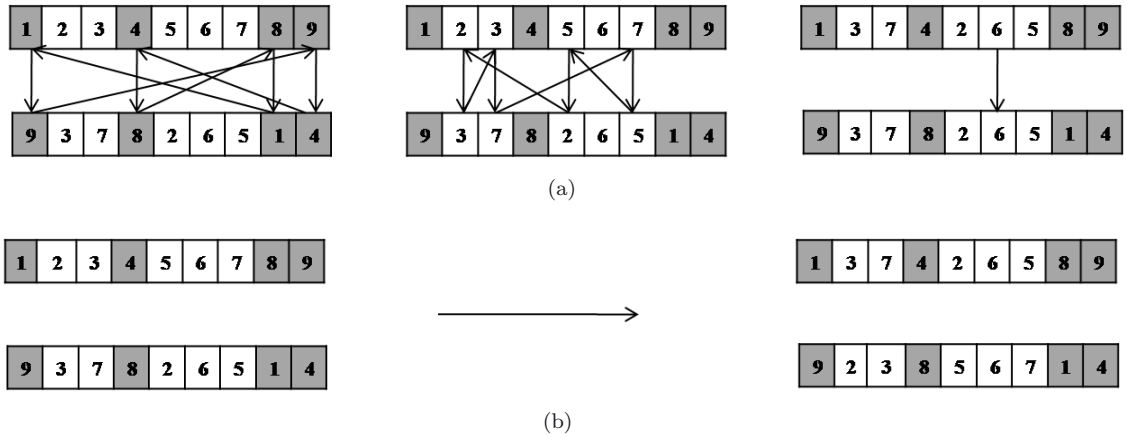

FIGURE 3. Cycle Crossover (a) Step 1: identification of cycles, (b) Step 2: construction of offspring [12].

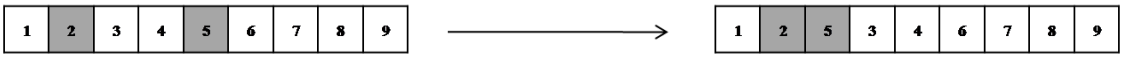

FIGURE 4. Insert mutation [12].

(2) Look at the allele in the same position in Parent 2.
(3) Go to the position with the same allele in Parent 1.
(4) Add this allele to the cycle.
(5) Repeat steps 2 through 4 until you arrive at the first allele of Parent 1.

An example illustration of cycle crossover is shown in Figure 3.

Following the recombination operator, the next step is mutating the population. Swap mutation, insert mutation, scramble mutation or inversion mutation can be implemented as a mutation operator for permutation representations [12]. Mutation operator is often interpreted as a "background operator" supporting the crossover [25] and it is implemented with small probabilities compared to crossover [11]. In this paper, we use insert mutation. It is a simple and effective operator. It works by randomly selecting two positions in the chromosome and moving one next to the other. An example for insert mutation is shown in Figure 4. In this example, the 5th gene is inserted next to the 2nd one, and the others are shifted right.

After the mutation step, a fraction of the best individuals of the current population can be directly inherited to the next generation, if desired. This is called elitism. Using elitism may help maintaining the diversity.

3.2. Partheno-genetic algorithm

Parthenogenesis is a special mode of reproduction that occurs without the male contribution [23, 34]. In this process, an egg cell from female parent produces new individuals without being penetrated by a sperm cell [23]. In a standard genetic algorithm, traditional crossover operators are used in order to produce offspring. This standard process need to use two or more parents to produce children. The PGA is a variant of the GA [42–44] and it simulates the partheno-genetic process of primary species in the nature [23].

Compared with the SGA, the PGA acts on single chromosome [19]. Crossover and mutation operators are replaced by partheno-genetic operators, named "swap", "reverse", and "insert". Several advantages of the PGA are stated in the literature: (i) complicated crossover operators (such as partially mapped crossover, ordered

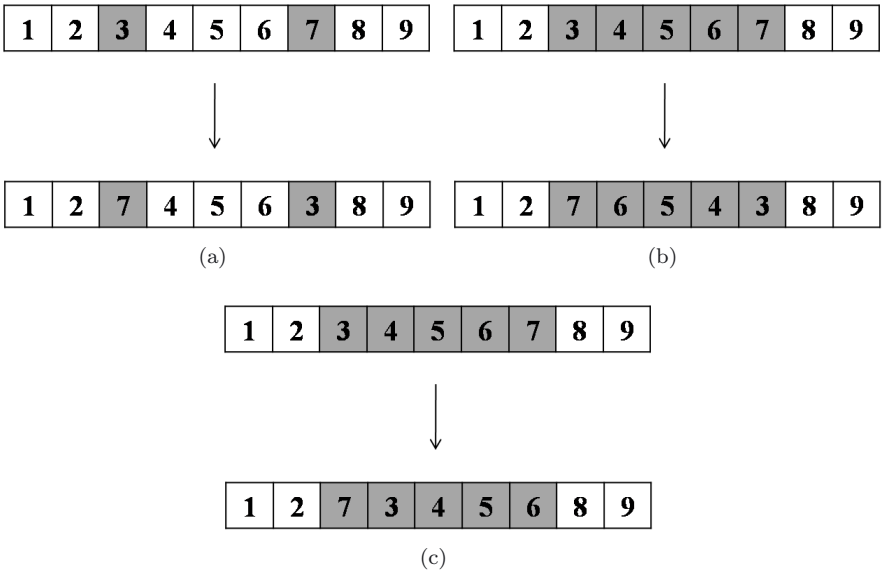

FIGURE 5. Partheno-genetic operators, (a) swap, (b) reverse, (c) insert.

crossover and cycle crossover) are avoided in the optimization process [2, 23, 34]; (ii) the PGA restrains the immature convergence phenomenon [2]; (iii) the population diversity is guaranteed [34]. As stated earlier, maintaining the diversity is a critical issue on adapting the dynamic environments. The procedure of the PGA is summarized in Algorithm 3.

Algorithm 3. A pseudo code for the PGA.

```
Generate initial population;
Evaluate fitness values of individuals in population;
repeat
    apply selection
    apply the swap operator;
    apply the reverse operator;
    apply the insert operator;
until a termination criterion is satisfied //e.g., a predefined iteration number
Results and visualization.
```

Partheno-genetic operators are comparatively easy to implement. In the swap operator, a pair of positions on a chromosome is selected randomly. Then the contents of these two points are swapped with a predefined probability to produce a new chromosome. This operator is especially useful for permutation representations. A simple illustration is shown in Figure 5a.

In the reverse operator, a sub-string is selected from the chromosome randomly, the positions of the genes in this sub-string are reversed to generate a new chromosome with a given probability. An example for this procedure is shown in Figure 5b.

In the insert operator, a substring of a chromosome is selected randomly. Then the last value in this substring is transferred to the first position of the substring, and all the other positions in the substring are moved backwards to generate a new chromosome with a predefined probability. An example illustration is shown in Figure 5c.

Table 1. Parameters for the SGA and the PGA.

Attribute	SGA	PGA
# of generations	3000	3000
Population (randomly generated) size	600	600
Crossover (cycle) rate	0.7	–
Mutation (insert) rate	0.01	–
Elitism rate	0.01	0.01
Insert rate	–	0.3
Reverse rate	–	0.3
Swap rate	–	0.3

3.3. Techniques for adapting the dynamic environments

There are several approaches to enhance the performance of the SGA in dynamic environments (for a detailed survey see [37]). The most straightforward approach to increase the diversity is to restart the algorithm completely by reinitializing the population after each environmental change [3]. But, sometimes the information on hand, gained from the current population, can be useful. This situation is more likely to happen if the environmental change is not considerably severe. In this case, partial random restarting approach can be implemented. In other words, a fraction of the new population is seeded with old solutions, rather than complete reinitialization.

4. Experimental setup

Shuai *et al.* [34] stated that the PGA overcomes the problem of the premature convergence of GAs, and in addition it was guaranteed to keep the diversity of the population in the conditions of the lower diversifying of initial population. This leads us to test the PGA in dynamic environments by solving the MKP. Firstly, we solved the stationary MKP to measure the performance of the PGA. Secondly, we conducted some comparative computational experiments in dynamic environments employing both the SGA and the PGA. Finally, we tested different reinitialization schemes in dynamic environments using the PGA.

4.1. Setup on static environment

In order to test the performance of the PGA in stationary environment, we compared the SGA and the PGA on 12 different benchmark problems available on the OR-LIBRARY web-site[3]. These problems were also previously used by Boussier *et al.* [8] and Hanafi and Wilbaut [18]. Four problem sets include; 10 constraints-250 variables, 5 constraints-500 variables, 10 constraints-500 variables and finally 30 constraints-250 variables. At each set we solved 3 representative problems with different difficulty levels. The parameters for the SGA and the PGA were set as shown in Table 1. For each algorithm and each problem, 5 independent runs with 3000 iterations were executed using Python programming environment, with a personal computer bundled with Intel i5 1.6 GHZ processor and 8 GB RAM. Pseudo-codes of the problem specific SGA and the PGA are shown in Algorithms 4 and 5, respectively.

4.2. Setup on dynamic environment

In a dynamic environment, some factors such as frequency or periodicity of changes (cyclic/periodical/recurrent or not), and components that change (objective functions, constraints, *etc.*) are important issues [37]. In real life problems, changes in parameters of any problem have a stochastic nature. On the other hand, environmental changes typically may not alter the problem completely and may affect only some part of the problem at a time [3].

[3] http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Algorithm 4. The pseudo-code for the SGA used in the stationary problem.

```
begin
maxiter := 3000
t := 0
generate random initial population (size = 600)
while t < maxiter do
    t ← t + 1
    apply cycle crossover (P_c = 0.7)
    apply insert mutation (P_m = 0.01)
    decode the permutation and sort individuals w.r.t fitness values
    apply tournament selection (tournamentsize = populationsize * 0.1)
    apply elitism elitismrate = 0.01
end while
```

Algorithm 5. The pseudo-code for PGA used in the static problem.

```
begin
maxiter := 3000
t := 0
generate random initial population (size = 600)
while t < maxiter do
    t ← t + 1
    swap (P_s = 0.3)
    reverse (P_r = 0.3)
    insert (P_i = 0.3)
    decode the permutation and sort individuals w.r.t fitness values
    apply tournament selection (tournamentsize = populationsize * 0.1)
    apply elitism elitismrate = 0.01
end while
```

Based on this philosophy, we modelled the changing environment as a Poisson Process with parameter (λ), by means of frequency of changes. To generate more frequent changes we set $\lambda = 0.01$ and on the contrary, for rare changes compared with the previous approach, we set $\lambda = 0.005$. In addition to this, we modelled the type of changes as a Markov chain. Again, we used 5 benchmark problems from the OR-LIBRARY. These problems are Weish26 to Weish30, which are previously used by Shih [33]. Within these problems, only the knapsack capacities are different from each other. It means that the problem landscape doesn't change ultimately. These problems constitute the states of an irreducible Markov Chain. At each change point, current problem changes into another problem (or change into itself again) with equal transition probability of 0.2. Different transition probabilities could be used, yet for simplicity we preferred equal transition probabilities.

The best error to the optimum was used as the performance measure. At each iteration, the difference between the optimum value of corresponding environment and the best of generation fitness values were recorded. Then these difference values were averaged over the generation number. Firstly, for each algorithm 15 replicates, each had different environmental conditions, were executed and the performances of the SGA and the PGA were compared. Secondly one representative environment for each change frequencies (number of changes are 8 and 15 for $\lambda = 0.005$ and 0.01, respectively) were selected, and then we tested the PGA's behaviour with respect to the different reinitialization schemes (*i.e.*, 5 different reinitialization rates) on chosen environments through 30 runs. Generation numbers were set to 1000 for each run with the population size of 150. All experiments were conducted using Python. The pseudo-code for the PGA tested in dynamic environment in this study is shown in Algorithm 6.

As mentioned earlier in Section 3, rapid convergence is a problem for GAs in dynamic environments. Once converged, GAs can not adapt well to the changing environment [45]. For dynamic environments, an effective algorithm is expected quickly adapt the new environment. For this reason, diversity must be maintained through

Algorithm 6. The pseudo-code for the PGA used in the dynamic problem.

```
1: begin
2: maxiter := 1000
3: t := 0
4: define regenrate //regeneration rate
5: generate random initial population (size = 150)
6: substitute ← first (regenrate*popsize) elements of initial population
7: define change points using Poisson Process
8: while t < maxiter do
9:    t ← t + 1
10:   swap (P_s = 0.3)
11:   reverse (P_r = 0.3)
12:   inverse (P_i = 0.3)
13:   decode the permutation and sort individuals w.r.t fitness values
14:   apply tournament selection (tournamentsize = populationsize * 0.1)
15:   apply elitism elitismrate = 0.01
16:   record errors from optimum value of current environment
17:   if changepoint = t then
18:      change problem type randomly
19:      insert substitute into current population starting from the first row
20:   end if
21: end while
```

generations. We used the most straightforward approach to increase diversity: to restart the algorithm by reinitializing a predefined fraction of population after each environmental change. For example, if the reinitialization rate is 0.5, then when an environmental change occurs, fifty per cent of the current population is replaced with same proportion of the initial population. This approach can be considered as partial random restarts. We tested the dynamic behaviour of the PGA for 5 different reinitialization schemes (treatments) (*i.e.*, 0, 0.25, 0.5, 0.75, 1) in 2 different change frequencies (0.01 and 0.005). We used analysis of variance (ANOVA) in order to compare the treatments.

5. Results and discussion

Results on stationary environment are shown in Table 2. In this table, the 2nd and the 3rd columns include the best fitness values coupled with the CPU times which are previously achieved by Boussier *et al.* [8]. Fitness values with asterisks are the optimum values. The best fitness values reached with in five runs are given in boldface for the PGA and the SGA in the 5th, and the 7th columns, respectively. Time values in the 6th and the 8th columns were produced using the Python's *time.clock* () function for each run. On Windows®, this function returns wall-clock seconds elapsed since the first call to this function, as a floating point number. The resolution is typically better than one microsecond.

According to the results in Table 2, it is possible to say that the PGA is more effective than the SGA, with given parameters, by means of the solution times and the best fitness values achieved. On the contrary, the PGA is not effective at all compared with the results that are reported by Boussier *et al.*, especially for relatively easy problems (*i.e.*, for 5.500 and 10.250). But, for harder problems (*i.e.*, 10.500 and 30.250) the PGA can achieve acceptable fitness values with reasonable solution times. Moreover, the PGA (and also the SGA) never runs out of memory. In fact, meta-heuristics are strictly dependent to the parameters used. For example, it could be possible to reach better fitness values by increasing the number of iterations. Then, a trade-off between the solution times and the fitness values achieved must be taken into consideration. The convergence characteristics of both the SGA and the PGA on the stationary environment (for 5.500.20 problem instance) are represented in Figure 6.

TABLE 2. Experimental results on stationary environment.

	Boussier *et al.* [8]			SGA		PGA	
Inst.	Best	CPU time	Run	Best	Time (s)	Best	Time (s)
5.500.0	120 148*	40 (s)	1	98 753	1665.73	115 575	1646.58
			2	99 308	1841.8	**117 365**	1654.36
			3	100 164	1758.27	116 656	1650.28
			4	100 896	1793.95	116 732	1656.36
			5	**101 678**	1809.36	116 442	1681.24
5.500.10	218 428*	35 (s)	1	196 502	2926.27	215 078	2918.14
			2	194 942	2951.47	215 973	2867.3
			3	**197 644**	3811.89	215 251	2890.02
			4	195 212	3080.28	215 350	2889.77
			5	193 554	2964.97	**216 079**	2904.08
5.500.20	295 828*	0 (s)	1	279 814	4099.54	293 235	3953.4
			2	280 098	4196.86	292 399	4003.18
			3	279 204	6247.99	292 901	3970.20
			4	**281 950**	4164.70	**293 738**	4028.37
			5	278 849	4134.03	292 355	3996.76
10.250.0	59 187*	0 (s)	1	51 227	1298.34	57 229	2092.4
			2	50 693	1297.29	57 477	1326.41
			3	50 936	1298.82	57 903	1316.14
			4	**51 288**	1264.93	57 362	1278.17
			5	50 543	1250.03	**57 943**	1294.12
10.250.10	110 913*	336 (s)	1	**102 386**	2359.02	108 826	2304.58
			2	98 830	2402.03	108 234	2338.95
			3	100 887	2342.19	106 416	2320.25
			4	99 473	2411.26	108 828	2318.28
			5	101 401	2318.52	**108 977**	2366.28
10.250.20	151 809*	211 (s)	1	144 448	3365.25	**150 728**	3429.24
			2	144 806	3329.11	150 383	5235.56
			3	142 987	3394.98	150 108	3452.85
			4	142 187	3349.07	149 921	3475.20
			5	**146 314**	3597.74	150 608	3483.27
10.500.0	117 821*	24.5 (h)	1	**101 539**	2569.87	114 563	2506.83
			2	99 541	2622.25	114 488	2542.94
			3	99 270	2675.15	**114 842**	2520.3
			4	98 423	2636.77	113 174	2602.56
			5	99 187	2633.77	114 187	2554.77
10.500.7	118 344*	161.7 (h)	1	98 142	2606.73	114 404	2463.17
			2	98 960	2590.43	113 383	2518.62
			3	**102 531**	2681.26	**115 282**	2531.55
			4	98 071	2724.78	114 496	2481.85
			5	100 279	3690.61	114 847	2466.53
10.500.20	304 387*	6.6 (h)	1	287 077	6726.03	301 682	6631.94
			2	285 036	6913.78	**301 994**	6693.94
			3	287 679	6936.82	301 370	7413.24
			4	**289 481**	6831.02	302 344	6770.35
			5	288 520	6965.56	300 422	6670.93
30.250.0	56 842	39.7 (h)	1	49 340	3138.46	55 229	3140.97
			2	48 046	3190.02	**55 374**	3042.05
			3	47 230	3074.92	54 363	3067.34
			4	**50 185**	3135.61	54 612	3051.15
			5	46 128	3111.10	54 557	3130.98

TABLE 2. Continued.

Inst.	Boussier *et al.* [8] Best	Boussier *et al.* [8] CPU time	Run	SGA Best	SGA Time (s)	PGA Best	PGA Time (s)
30.250.13	106 876	86.9 (h)	1	97 895	6100.31	104 536	6076.02
			2	**98 431**	6095.72	104 621	6048.82
			3	95 285	5963.57	**105 642**	5933.81
			4	95 621	6059.43	105 326	5989.37
			5	96 128	6005.22	105 587	6007.77
30.250.28	149 570	16.8 (h)	1	143 942	9068.53	147 653	8994.21
			2	140 364	9051.11	148 333	9057.92
			3	**144 159**	9079.86	**148 549**	9016.86
			4	142 260	9123.31	148 289	9753.12
			5	143 038	9176.41	148 415	9078.3

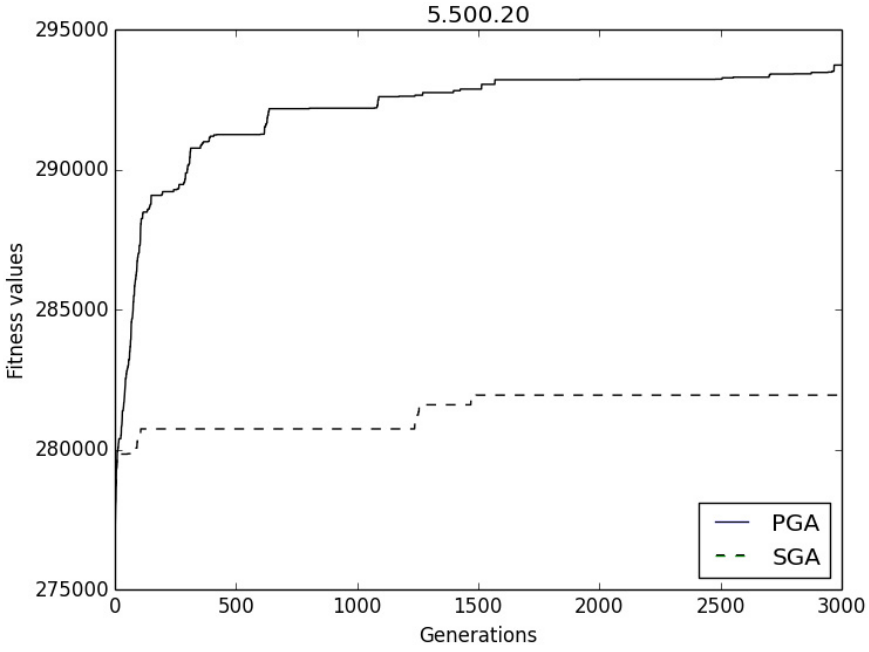

FIGURE 6. Behaviours of the PGA and the SGA in the stationary environment for 5.500.20 instance.

As it can be seen from Figure 6, the PGA rapidly reaches the near optimal value in the initial generations. It has a very strong convergence capability. Using 3 operators (*i.e.*, swap, reverse and insert) the PGA does not stick to the local optimum and it converges the near optimal value rapidly. On the contrary, standard GA sticks to the local optimum.

For dynamic environments, as mentioned in Section 4.2, we firstly implemented 15 replicates of the SGA and the PGA for each of two λ values (*i.e.*, rare changes and frequent changes). Comparisons of the results taken are shown in Table 3 for $\lambda = 0.01$, and in Table 4 for $\lambda = 0.005$.

For the 4th column of Table 3, there are 14 environmental changes through 1000 iterations. The positions of the changes and corresponding problem types between these positions for this environment are depicted in Figure 7. In this Figure, "0" represents Weish26, and "4" represents Weish30 problem type. Until the first point of change, the problem type is Weish26, and following the first point of change (*i.e.*, 47th iteration) the

TABLE 3. Comparison of the PGA with the SGA on dynamic environments (lamda = 0.01).

			Environments (with number of changes in 1000 iterations)														
	R		14	14	13	10	11	11	12	10	9	8	7	15	10	6	9
PGA	0	μ	538.87	428.19	394.99	484.56	388.90	294.52	347.92	365.40	370.22	417.64	203.04	513.22	339.93	204.76	440.75
		t (s)	182.93	190.44	207.70	194.75	70.37	71.74	61.50	65.92	56.78	56.97	55.76	59.58	57.06	55.27	56.14
	0.25	μ	539.68	523.29	358.98	391.51	543.86	343.87	222.95	409.44	373.07	501.07	270.92	488.76	400.51	194.67	534.78
		t (s)	56.73	57.54	57.13	58.81	56.59	65.85	56.02	62.84	58.19	59.58	57.70	61.76	58.19	56.30	56.68
	0.5	μ	555.95	534.68	375.01	461.79	609.14	397.77	230.55	364.17	382.57	576.84	195.45	452.63	470.78	211.11	479.71
		t (s)	56.28	56.55	56.77	58.30	58.33	68.61	55.75	61.44	57.90	58.10	55.33	57.82	57.59	55.00	55.96
	0.75	μ	543.83	566.91	418.79	380.48	507.27	397.31	366.54	419.19	301.13	485.76	312.24	521.81	459.80	168.68	641.91
		t (s)	56.40	57.22	57.15	60.15	57.34	68.16	57.48	61.46	57.75	59.03	57.02	59.53	57.75	54.29	55.65
	1	μ	935.47	847.32	887.52	896.66	927.61	734.19	971.83	865.33	585.74	769.20	765.25	1037.52	867.53	640.39	706.73
		t (s)	54.99	56.24	56.15	56.41	58.03	65.69	52.92	59.75	58.46	60.64	57.58	59.42	56.47	56.51	54.94
SGA	0	μ	3505.69	1564.89	1959.56	3420.28	2660.16	3389.15	2615.73	1694.89	2929.27	2143.27	3381.90	2804.57	2911.22	2887.68	2127.40
		t (s)	161.43	181.69	178.91	164.64	165.94	164.54	162.00	169.38	161.53	170.90	149.78	171.24	180.10	189.81	167.38
	0.25	μ	2441.57	2075.51	1733.25	2698.75	2057.03	1965.09	2739.84	1622.90	2001.25	2143.27	2080.41	2668.88	2466.04	2595.10	2097.22
		t (s)	178.66	190.98	172.80	146.31	55.52	54.74	53.87	56.11	55.16	55.36	55.09	56.84	54.06	53.93	52.72
	0.5	μ	1930.26	2075.51	1733.08	1869.76	1990.13	2831.89	2121.40	2037.50	2207.60	2413.29	1975.57	2053.12	2342.32	2595.10	2075.56
		t (s)	54.83	60.58	55.61	54.46	54.72	54.01	54.87	54.83	53.86	58.08	58.41	56.50	55.28	56.73	50.98
	0.75	μ	2122.94	2075.51	1698.72	2071.45	2050.32	2091.65	1780.97	2056.15	1819.35	2117.82	2027.54	1502.61	2329.97	2291.04	1880.37
		t (s)	55.24	60.49	58.08	57.55	56.26	56.71	58.61	55.29	57.43	54.25	57.09	59.10	54.95	52.46	54.00
	1	μ	1833.99	2038.28	2396.04	1699.71	1973.91	1858.99	1922.09	2141.11	1430.10	1915.52	2045.68	2079.90	1840.37	1804.64	2029.70
		t (s)	53.25	51.68	53.45	54.88	51.96	58.15	54.83	51.30	57.03	53.34	54.42	52.26	55.77	57.70	54.17

R: Random regeneration rate. μ: Mean error value over 1000 iterations. t (s): Execution time in seconds.

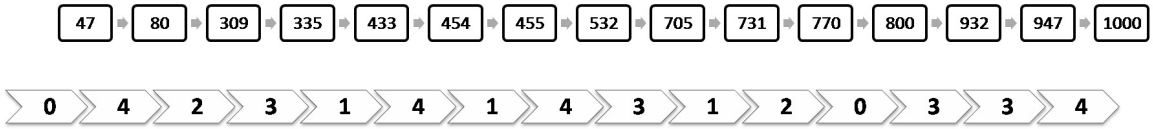

FIGURE 7. Points of change and the problem types for the 4th column of Table 3.

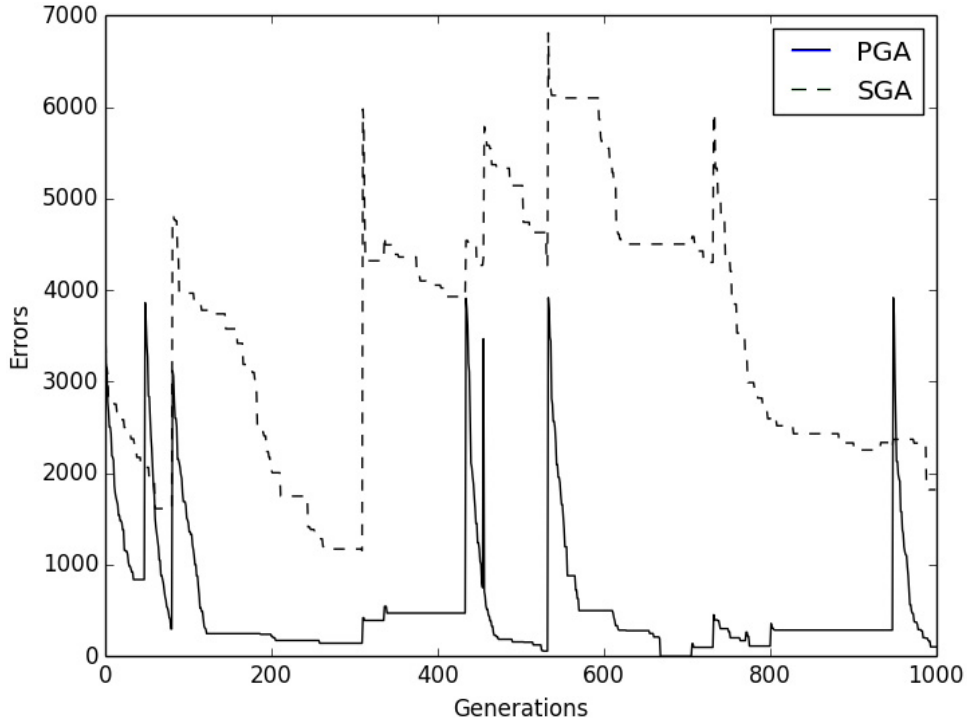

FIGURE 8. Comparison of the PGA and the SGA for a selected dynamic environment.

problem type turns into Weish30, and the process keeps on in a similar fashion. As a representative example, we compared the behaviours of the first entries of the PGA and the SGA graphically in Figure 8 (*i.e.*, 538.87, and 3505.69, respectively).

As shown in Figure 8, the PGA yields lower error values. Moreover, the PGA has lower mean error values (μ) compared with the corresponding SGA counterpart as shown in Tables 3 and 4 for all environmental conditions. Additionally, the mean error values in Table 4 are less than the mean error values in Table 3 on the average. This is an expected result, since the algorithm has more time to decrease the error value when the environment has lower number of changes.

Regarding the comparison of different reinitialization schemes, it is not proper at all to compare the efficiency of treatments with the data only shown in Tables 3 and 4 since the environments are not the same. In order to compare the treatments, we selected one representative environmental condition from each of these tables, namely the 12th environment (with 15 environmental changes) from Table 3, and the 6th (with 8 environmental changes) from Table 4. Then we executed the algorithm 30 times for each treatment. Results are shown in Tables 5 and 6. Each entry within these tables represents the error value averaged over 1000 iterations.

In order to detect whether the treatment schemes differ among each other, analysis of variance (ANOVA) was implemented for the data given in Tables 5 and 6. ANOVA results are given in Tables 7 and 8, respectively.

TABLE 4. Comparison of the PGA with the SGA on dynamic environments (lamda = 0.005).

			Environments (with number of changes in 1000 iterations)														
	R		5	3	4	9	6	8	4	1	3	6	4	4	5	5	5
PGA	0	μ	254.22	120.71	336.87	366.53	558.15	199.65	204.68	219.61	211.10	245.10	296.76	212.62	316.15	174.91	330.01
		t (s)	73.00	55.80	57.23	58.94	59.99	68.70	55.07	55.68	57.67	61.07	60.44	61.95	55.05	53.38	59.79
	0.25	μ	300.86	120.03	358.68	238.47	430.23	238.87	249.35	201.42	268.40	229.67	185.17	356.73	343.50	245.07	298.67
		t (s)	55.81	56.00	57.94	57.77	57.85	67.22	56.28	56.41	57.49	62.29	59.01	62.60	55.86	54.45	59.78
	0.5	μ	254.94	120.03	301.70	343.36	506.19	253.72	147.85	198.00	212.01	220.25	154.31	402.46	337.94	144.42	308.65
		t (s)	54.72	56.06	56.74	58.48	59.88	67.62	55.26	55.67	57.32	61.31	58.65	63.10	54.98	53.94	60.44
	0.75	μ	323.84	122.94	388.27	389.05	381.29	252.02	227.76	283.41	219.92	222.33	200.90	264.30	363.64	267.15	336.39
		t (s)	55.85	56.08	58.37	59.47	58.46	66.95	54.98	56.10	56.46	60.12	58.45	62.38	55.02	54.30	59.12
	1	μ	473.66	303.47	554.96	832.23	618.73	515.39	573.79	264.71	281.14	574.35	489.33	419.04	610.27	644.33	480.13
		t (s)	53.09	54.04	56.84	56.13	58.00	67.64	53.32	53.83	56.86	60.51	58.63	59.94	54.92	52.41	60.07
SGA	0	μ	2032.88	1967.91	1123.74	2760.38	3167.07	3116.16	2504.87	1576.07	2365.10	2635.56	1847.57	2644.68	2097.10	2057.97	2019.03
		t (s)	53.17	51.08	58.05	52.34	48.97	52.28	59.59	56.96	54.35	50.91	55.90	54.83	53.28	49.34	55.89
	0.25	μ	1987.24	1967.91	1123.74	2943.57	1372.03	2038.11	2144.79	1576.07	2200.02	2635.56	1699.79	1408.93	1670.36	2173.00	2019.03
		t (s)	51.58	49.91	57.75	52.77	55.10	55.18	61.83	57.14	55.36	51.12	55.42	58.40	54.64	49.37	55.38
	0.5	μ	2090.66	1967.91	1122.37	2576.80	1033.39	2231.45	2419.43	1576.07	2200.02	2095.78	1679.36	1841.31	1888.66	2173.00	2019.03
		t (s)	51.32	49.56	57.64	49.04	57.23	55.44	59.52	57.93	55.50	52.22	57.30	56.10	53.28	48.92	55.33
	0.75	μ	2090.66	1844.33	1122.37	2018.85	1676.73	1837.14	1986.54	1576.07	2126.78	2059.39	1679.36	1852.58	1728.79	1844.61	2019.03
		t (s)	51.38	51.02	57.54	55.74	52.81	54.15	62.38	57.28	56.00	51.62	56.85	57.19	53.06	49.25	55.99
	1	μ	2103.97	1678.34	1675.96	1685.74	1684.60	1655.59	2103.91	2291.79	1589.00	1848.42	1852.42	1472.55	1373.30	1884.49	1735.73
		t (s)	46.35	52.73	53.17	53.13	53.70	56.14	57.47	50.96	55.82	51.17	55.83	55.20	56.21	55.02	53.37

R: Random regeneration rate. μ: Mean error value over 1000 iterations. t (s): Execution time in seconds.

TABLE 5. Comparison of treatments ($\lambda = 0.01$). Number of changes is equal to 15.

	Treatments				
Replicates	1.0	0.75	0.5	0.25	0.0
1	921.57	500.58	504.64	553.63	469.27
2	908.12	517.69	547.32	456.22	517.77
3	916.64	472.27	411.90	476.93	498.96
4	868.27	584.71	390.88	470.83	431.14
5	1041.53	564.61	631.56	537.99	459.99
6	941.06	480.18	581.09	422.70	527.60
7	843.92	611.40	436.97	441.51	433.14
8	889.08	563.26	483.18	463.73	464.72
9	870.91	489.07	515.57	568.65	458.06
10	842.49	530.95	494.66	471.92	433.47
11	842.81	519.50	550.37	454.08	559.21
12	918.27	567.57	433.24	511.01	487.79
13	844.20	493.77	444.12	461.47	447.42
14	946.54	421.53	501.82	510.72	420.11
15	1017.61	677.17	552.90	509.52	517.67
16	976.55	632.87	407.01	461.19	472.19
17	911.97	494.36	545.99	494.59	474.68
18	844.88	496.40	526.24	482.03	472.96
19	916.82	531.70	506.63	532.91	464.36
20	909.72	560.99	595.15	489.33	442.18
21	845.87	595.95	547.38	476.10	436.15
22	824.58	495.85	592.05	502.88	526.68
23	999.71	447.22	575.80	503.73	525.81
24	1009.21	520.09	526.15	484.49	487.53
25	865.64	536.83	502.91	499.63	500.85
26	829.10	458.23	456.92	561.06	453.54
27	945.44	541.06	475.32	507.05	468.82
28	895.13	489.91	498.34	538.27	489.93
29	915.82	561.52	516.64	446.92	552.10
30	964.53	525.56	542.19	452.04	473.63
Mean	**908.93**	**529.43**	**509.83**	**491.44**	**478.92**

According to the first ANOVA summarized in Table 7, when $\lambda = 0.01$ and the number of changes is 15, since the $P-value = 0.000$, it can be concluded that there is a significant difference between at least one pair of each treatment types by means of their effect on the average solution. A multiple comparison method (*i.e.*, Fisher's Least Significance Difference Method, LSD) was performed to isolate the specific differences [28]. For an α level of 0.05, the LSD value for these data is equal to $LSD = 26.04$. The differences and significance of all pairwise comparisons are shown in Table 9.

According to this comparison there is a significant difference between the first (*i.e.*, 100% reinitialization) and all the others. When the algorithm uses the knowledge on hand, it is easier to adapt the new environment. On the other hand it is interesting that while using the information on the current population has a positive effect on adapting the dynamism, there is not a specific rate for information usage. But it is clear that the complete random restarting is not an effective way of adapting the new environment.

Similar procedures were applied for the data shown in Table 6. For these environments, the number of environmental changes and average errors are lower compared with the situation that $\lambda = 0.01$. Analysis of variance is summarized in Table 8. For an α level of 0.05, the LSD value for these data is equal to $LSD = 14.62$. The differences and significance of all pairwise comparisons are shown in Table 10. According to these results, similar to the previous case, it can be concluded that there is a significant difference between at least one pair of

TABLE 6. Comparison of treatments ($\lambda = 0.005$). Number of changes is equal to 8.

	Treatments				
Replicates	1.0	0.75	0.5	0.25	0.0
1	305.60	305.30	313.62	332.22	243.85
2	293.64	328.93	314.62	253.60	254.53
3	295.42	227.24	257.14	242.42	229.09
4	321.33	228.92	286.49	291.74	232.09
5	275.99	265.26	294.04	279.86	271.37
6	286.15	263.25	286.92	272.45	255.33
7	294.95	312.93	289.59	290.29	213.39
8	306.09	252.64	278.48	304.24	255.48
9	316.23	258.41	276.62	263.06	232.04
10	291.06	257.50	285.84	310.82	324.03
11	324.53	245.19	305.30	298.09	258.23
12	293.57	287.97	264.25	247.37	287.10
13	352.17	263.60	268.23	231.89	262.91
14	329.08	327.87	240.61	242.94	295.49
15	260.70	249.91	292.03	273.87	244.92
16	285.31	315.83	262.76	264.00	256.51
17	282.47	301.91	277.30	279.01	237.54
18	279.38	315.57	250.62	247.68	244.45
19	325.16	292.26	273.19	254.74	296.92
20	302.56	289.12	241.43	283.02	262.18
21	251.17	307.61	315.86	283.44	235.32
22	283.35	263.61	262.75	264.32	258.17
23	264.97	319.63	261.82	270.58	234.24
24	272.11	271.06	293.06	332.89	250.74
25	350.30	266.14	298.97	274.29	231.90
26	321.76	298.52	265.73	284.00	263.83
27	327.20	234.81	290.28	232.42	252.53
28	286.21	281.46	245.93	159.59	297.98
29	380.03	311.37	246.53	256.08	256.33
30	304.08	232.14	331.61	236.61	276.61
Mean	**302.08**	**279.20**	**279.05**	**268.58**	**257.17**

TABLE 7. ANOVA table, # of changes 15 ($\lambda = 0.01$).

Source of variation	Degrees of freedom	Sum of squares	Mean square	F	P-value
Treatments	4	4 010 066.17	1 002 516.54	386.22	0.000
Error	145	376 379.91	2595.72		
Total	149	4 386 446.08			

TABLE 8. ANOVA table, # of changes 8 ($\lambda = 0.005$).

Source of variation	Degrees of freedom	Sum of squares	Mean square	F	P-value
Treatments	4	33 063.95	8265.99	10.12	0.000
Error	145	118 433.19	816.78		
Total	149	151 497.13			

TABLE 9. Pairwise comparisons of treatments ($\lambda = 0.01$). Differences larger than 26.04 are significant at the $\alpha = 0.05$ level and are indicated with an "*".

	Treatments				
	1.00	0.75	0.50	0.25	0.00
1.00	0.00	379.51*	399.10*	417.50*	430.01*
0.75		0.00	19.60	37.99*	50.50*
0.50			0.00	18.40	30.91*
0.25				0.00	12.51
0.00					0.00

TABLE 10. Pairwise comparisons of treatments ($\lambda = 0.005$). Differences larger than 14.61 are significant at the $\alpha = 0.05$ level and are indicated with an "*".

	Treatments				
	1.00	0.75	0.50	0.25	0.00
1.00	0.00	22.89*	23.03*	33.50*	44.91*
0.75		0.00	0.15	10.62	22.03*
0.50			0.00	10.47	21.88*
0.25				0.00	11.41
0.00					0.00

treatments. LSD procedure indicates that the first treatment type results in different means than all the other four types. Parallel with the previous ANOVA, it can be said that using the information on hand increases the efficiency of the algorithm adapting the new environment. There is not a specific more effective information usage rate, however complete random restart is not suggested.

6. Conclusion

Dynamism in the business world encourages the analysis of change. Multidimensional Knapsack Problem (MKP), an NP-hard combinatorial optimization problem, is used in several areas such as cutting stock, cargo loading and the capital budgeting applications. Hence, analysis of MKP in a dynamic environment is one of the motivating necessities of agile business exercises today. One of the analysis methods of MKP, GAs are criticized in dynamic environments. That is why this original study aims to analyze MKP in a dynamic environment using Partheno-Genetic Algorithm (PGA), an enhanced version of genetic algorithms.

First, the performance of PGA is benchmarked with the classical GA on different MKP problems. It is clearly shown that PGA outperforms both by converging fast and finding near optimum. It is also observed that GA easily sticks to the local optimum, which explains its weakness in dynamic environment. Second, we compared the SGA and the PGA in dynamic environments. Again, the PGA outperformed the SGA by means of the mean error values.

Later, a dynamic environment is constructed by using different parameter values for a Poisson initialization to represent rare changes and frequent changes. PGA is used in the dynamic environment for five different reinitialization schemes. The best performances in different environments are tested. ANOVA showed at least one significant difference among the treatments. Fisher's LSD has shown where the significant difference occurred. It is concluded that it is easier to adapt the new environment when old information is used. It is shown that complete random restart is not effective at all.

This study will be enhanced to compare the intelligent methods in the dynamic environment. The achievements will be enriched when the algorithms developed in this study will be implemented in real life exercises.

References

[1] Z. Ahmed and I. Younas, A dynamic programming based ga for 0-1 modified knapsack problem. Published by Foundation of Computer Science. *Int. J. Comput. Appl.* **16** (2011) 1–6.

[2] J.-C. Bai, H.-Y. Chang and Y. Yi, A partheno-genetic algorithm for multidimensional knapsack problem. In Vol. 5, *Proc. of 2005 International Conference on Machine Learning and Cybernetics* (2005), 2962–2965.

[3] O. Basir, Abdunnaser Younes, S. Areibi and P. Calamai, Adapting genetic algorithms for combinatorial optimization problems in dynamic environments. In Chap. 11, *Advances in Evolutionary Algorithms*, edited by Witold Kosinski. I-Tech Education and Publishing, Vienna (2008) 207–230.

[4] A. Baykasoğlu and F. Burcin Ozsoydan, An improved firefly algorithm for solving dynamic multidimensional knapsack problems. *Expert Syst. Appl.* **41** (2014) 3712–3725.

[5] T. Blickle and L. Thiele, *A comparison of selection schemes used in genetic algorithms*. Technical report, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) (1995).

[6] C. Blum and A. Roli, Metaheuristics in combinatorial optimization. *ACM Comput. Surveys* **35** (2003) 268–308.

[7] I. Boussaïd, J. Lepagnot and P. Siarry, A survey on optimization metaheuristics. *Inform. Sci.* **237** (2013) 82–117.

[8] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi and P. Michelon, A multi-level search strategy for the 0–1 Multidimensional Knapsack Problem. *Discrete Appl. Math.* **158** (2010) 97–109.

[9] J. Branke, M. Orbayı and Ş. Uyar, The Role of Representations in Dynamic Knapsack Problems. In *Applications of Evolutionary Computing SE - 74*, edited by F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J.H. Moore, J. Romero, G.D. Smith, G. Squillero and H. Takagi. Vol. 3907 of *Lect. Notes Comput. Sci.* Springer, Berlin, Heidelberg (2006) 764–775.

[10] P.-C. Chang and S.-H. Chen, The development of a sub-population genetic algorithm II (SPGA II) for multi-objective combinatorial problems. *Appl. Soft Comput.* **9** (2009) 173–181.

[11] P.C. Chu and J.E. Beasley, A Genetic Algorithm for the Multidimensional Knapsack Problem. *J. Heuristics* **4** (1998) 63–86.

[12] A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing. SpringerVerlag (2003).

[13] M. Elkihel, V. Boyer and D. El Baz, Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound. *Eur. J. Ind. Eng.* **4** (2010) 434–449.

[14] M. Ersen Berberler, A. Guler and U.G. Nurıyev, A geneti algorithm to solve the multidimensional knapsack problem. *Math. Comput. Appl.* **18** (2013) 486–494.

[15] M. Gen and R. Cheng, Genetic Algorithms and Engineering Optimization. John Wiley and Sons Inc., New York, USA (2000).

[16] M. Haluk Akin, *New heuristics for the 0-1 multi-dimensional knapsack problems*. Ph.D. thesis, University of Central Florida, Ann Arbor (2009).

[17] S. Hanafi and C. Wilbaut, Scatter Search for the 0–1 Multidimensional Knapsack Problem. *J. Math. Model. Algorithms* **7** (2008) 143–159.

[18] S. Hanafi and C. Wilbaut, Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Ann. Oper. Res.* **183** (2011) 125–142.

[19] D. Hu and Z. Yao, Stacker-reclaimer scheduling for raw material yard operation. In *Third International Workshop on Advanced Computational Intelligence (IWACI)* (2010) 432–436.

[20] M. Jalali Varnamkhasti, Overview of the Algorithms for Solving the Multidimensional Knapsack Problems. *Adv. Stud. Biol.* **4** (2012) 37–47.

[21] Y. Jin and J. Branke, Evolutionary Optimization in Uncertain Environments – A Survey. *IEEE Trans. Evol. Comput.* **9** (2005) 303–317.

[22] M. João Alves and M. Almeida, MOTGA: A multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem. *Comput. Oper. Res.* **34** (2007) 3458–3470.

[23] F. Kang, J.-J. Li and Q. Xu, Virus coevolution partheno-genetic algorithms for optimal sensor placement. *Adv. Eng. Inform.* **22** (2008) 362–370.

[24] H. Kellerer, U. Pferschy and D. Pisinger, Multidimensional Knapsack Problems. In *Knapsack Problems SE - 9*. Springer, Berlin, Heidelberg (2004) 235–283.

[25] S. Khuri, T. Bäck and J. Heitkötter, The zero/one multiple knapsack problem and genetic algorithms. In *Proc. of the 1994 ACM Symposium on Applied Computing*. SAC '94. ACM, New York, NY, USA (1994) 188–193.

[26] J. Langeveld and A.P. Engelbrecht, Set-based particle swarm optimization applied to the multidimensional knapsack problem. *Swarm Intelligence* **6** (2012) 297–342.

[27] M. Mitchell, An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, USA (1998).

[28] D.C. Montgomery and G.C. Runger, Applied Statistics and Probability for Engineers. John Wiley and Sons (2003).

[29] N. Mori and H. Kita, Genetic algorithms for adaptation to dynamic environments-a survey. In *the 26th Annual Conference of the IEEE Industrial Electronics Society* (2000) 2947–2952.

[30] G.R. Raidl, An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proc. of the 1998 IEEE International Conference on Evolutionary Computation*. IEEE Press (1998) 207–211.

[31] C.C. Ribeiro, S.L. Martins and I. Rosseti, Metaheuristics for optimization problems in computer communications. *Comput. Commun.* **30** (2007) 656–669.

[32] Masatoshi Sakawa and Kosuke Kato, Genetic algorithms with double strings for 0–1 programming problems. *Eur. J. Oper. Res.* **144** (2003) 581–597.

[33] Wei Shih, A branch and bound method for the multiconstraint zero-one knapsack problem. *J. Oper. Res. Soc.* **30** (1979) 37–47.

[34] Ren Shuai, Wang Jing and Xuejun Zhang, Research on chaos partheno-genetic algorithm for TSP. In vol. 1, *International Conference on Computer Application and System Modeling* (*ICCASM*) (2010) V1–290–V1–293.

[35] A. Simões and E. Costa, Using genetic algorithms to deal with dynamic environments: A comparative study of several approaches based on promoting diversity. In *Proc. of the genetic and evolutionary computation conference GECCO'02*. Morgan Kaufmann Publishers, New York, NY, USA (2002).

[36] S.N. Sivanandam and S.N. Dipa, Introduction to Genetic Algorithms. Springer-Verlag, Berin (2008).

[37] Trung Thanh Nguyen, Shengxiang Yang and Juergen Branke, Evolutionary dynamic optimization: A survey of the state of the art. *Swarm Evol. Comput.* **6** (2012) 1–24.

[38] M. Turkensteen, D. Ghosh, B. Goldengorin and G. Sierksma, Iterative patching and the asymmetric traveling salesman problem. The Traveling Salesman Problem. *Discrete Optimization* **3** (2006) 63–77.

[39] A.N. Ünal, A Genetic Algorithm for the Multiple Knapsack Problem in Dynamic Environment. In *Proc. of the World Congress on Engineering and Computer Science 2013*. IAENG, San Francicso, USA (2013) 1162.

[40] Ş. Uyar and H. Turgut Uyar, A Critical Look at Dynamic Multi-dimensional Knapsack Problem Generation. In Applications of Evolutionary Computing SE - 86. Vol. 5484 of *Lect. Notes Comput. Sci.* Edited by M. Giacobini, A. Brabazon, S. Cagnoni, G.A. Caro, A. Ekárt, A. Esparcia-Alcázar, M. Farooq, A. Fink and P. Machado. Springer, Berlin, Heidelberg (2009) 762–767.

[41] M. Vasquez and Yannick Vimont, Improved results on the 0–1 multidimensional knapsack problem. *Eur. J. Oper. Res.* **165** (2005) 70–81.

[42] J. Wu, A parthenogenetic algorithm for haplotyping a single individual based on WMLF model. In *Eighth International Conference on Natural Computation* (*ICNC*) (2012) 622–626.

[43] J. Wu and H. Wang, A parthenogenetic algorithm for the founder sequence reconstruction problem. *J. Comput.* **8** (2013) 2934–2941.

[44] J. Wu, J. Wang and J. Chen, A parthenogenetic algorithm for single individual {SNP} haplotyping. *Eng. Appl. Artif. Intell.* **22** (2009) 401–406.

[45] Shengxiang Yang, Memory-based immigrants for genetic algorithms in dynamic environments. In *Proc. of the 2005 conference on Genetic and evolutionary computation – GECCO '05*. ACM Press, New York, USA (2005) 1115.

[46] Q. Yuan and Z. Yang, On the performance of a hybrid genetic algorithm in dynamic environments. *Appl. Math. Comput.* **219** (2013) 11408–11413.