Expert Systems with Applications 41 (2014) 3712–3725

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

An improved firefly algorithm for solving dynamic multidimensional knapsack problems

Adil Baykasoğlu *, Fehmi Burcin Ozsoydan

Dokuz Eylül University, Faculty of Engineering, Department of Industrial Engineering, İzmir, Turkey

ARTICLE INFO

Keywords:
Firefly algorithm
Genetic algorithm
Differential evolution
Dynamic optimization
Multidimensional knapsack problem

ABSTRACT

There is a wide range of publications reported in the literature, considering optimization problems where the entire problem related data remains stationary throughout optimization. However, most of the real-life problems have indeed a dynamic nature arising from the uncertainty of future events. Optimization in dynamic environments is a relatively new and hot research area and has attracted notable attention of the researchers in the past decade. Firefly Algorithm (FA), Genetic Algorithm (GA) and Differential Evolution (DE) have been widely used for static optimization problems, but the applications of those algorithms in dynamic environments are relatively lacking. In the present study, an effective FA introducing diversity with partial random restarts and with an adaptive move procedure is developed and proposed for solving dynamic multidimensional knapsack problems. To the best of our knowledge this paper constitutes the first study on the performance of FA on a dynamic combinatorial problem. In order to evaluate the performance of the proposed algorithm the same problem is also modeled and solved by GA, DE and original FA. Based on the computational results and convergence capabilities we concluded that improved FA is a very powerful algorithm for solving the multidimensional knapsack problems for both static and dynamic environments.

1. Introduction

The area of evolutionary computation widely focuses on static optimization where the entire problem related data or variables remain stationary throughout optimization procedure. However, most of the real-life problems are dynamic in nature which means that a solution approach is expected to be adaptive to environmental changes. Therefore, tracking moving optima or high-quality solutions becomes the main purpose rather than optimizing a precisely predefined system. Evolutionary algorithms which have been widely studied for static optimization problems attracted an increasing attention over the past years on dynamic optimization problems (DOPs) (Branke, 2002; Jin & Branke, 2005; Morrison, 2004; Weicker, 2003). There are reported research theses (Liekens, 2005; Wilke, 1999) and edited volumes (Yang, Ong, & Jin, 2007) devoted to this research area in its infancy.

DOPs can be divided into two major fields as combinatorial and continuous. Because this study focuses on the combinatorial part, continuous DOPs are outside of the scope of this paper. However, a recently published study (Brest et al., 2013) provides a comprehensive research for continuous part. On combinatorial DOP part Rohlfshagen and Yao (2011) presented an extended work of Rohlfshagen and Yao (2008) on the dynamic subset sum problem. The authors used a binary encoding technique and a penalizing procedure to avoid infeasibility. Various stationary extensions of knapsack problems (Kellerer, Pferschy, & Pisinger, 2004) have been commonly studied by the researches. Because the dynamic version of multidimensional knapsack problem (dynMKP) is discussed here, related work is provided in the following.

Branke, Orbayı, and Uyar (2006) studied the effects of solution representation techniques for dynMKP. They used three types of representations which are, *binary encoding*, *permutation encoding*, and *real valued encoding with weight-coding*. The authors generated dynamic environments using a stationary environment as a base, and updated the changing parameters using normal distribution. A penalizing procedure is also presented in their studies in order to handle with infeasibilities.

Karaman and Uyar (2004) introduced a novel method which uses the *environment-quality measuring* technique which was applied to 0/1 knapsack problem for detection of the changes in the environment. Afterwards, Karaman, Uyar, and Eryiğit (2005) classified the dynamic evolutionary algorithms into four categories and they proposed an evolutionary algorithm considering the previous related work for 0/1 knapsack problem.

Kleywegt & Papastavrou, 1998, 2001 reported a stochastic/dynamic knapsack problem where items dynamically arrive with respect to a Poisson distribution. A distinctive feature was that the profit and unit resource consumption values of the items become visible as they arrive. In their problem the aim was to

* Corresponding author. Tel.: +90 232 3017600.
E-mail address: adil.baykasoglu@deu.edu.tr (A. Baykasoğlu).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.11.040

maximize the profit accepting the most appropriate items, rejecting rest of the items, where rejection causes a penalty. The same problem was reported by Hartman and Perry (2006). The authors utilized linear programming and duality for a quick approximation particularly on large scaled problems.

Karve et al. (2006) introduced and evaluated a middleware technology, capable of allocating resources to web applications through dynamic application instance placement which is a related problem of dynMKP. Kimbrel, Steinder, Sviridenko, and Tantawi (2005) reported a similar study.

Farina, Deb, and Amato (2004) proposed several test cases for dynamic multi-objective problems and the authors addressed the dynMKP as an appropriate problem for applications and implementations. For a similar purpose, Li and Yang (2008) proposed a generalized dynamic environment generator which can be instantiated into binary, real-valued and combinatorial problem domains. As reported in their studies, the proposed dynamic environment generator could be implemented to knapsack problems as well.

Adaptation of parameters was widely studied in the field of dynamic evolutionary algorithms. Thierens (2002) proposed adaptive mutation rate schemes for GAs, in order to prevent non-trivial decisions beforehand. Dean, Goemans, and Vondrák (2008) presented a stochastic version of 0/1 knapsack problem where the value of the items are deterministic but the unit resource consumptions are assumed as randomly distributed.

As it can be seen from the previous studies related to the dynamic version of knapsack problems, compared to well known static extensions, there are relatively far less reported publications. This is one of the main motivations behind the present study as dynamic knapsack problems have many practical implications. The second motivation is to investigate the performance of a relatively new and promising optimizer FA and its improved version (as proposed in this paper) on this dynamic optimization problem in comparison to very well known and widely used evolutionary algorithms GA and DE.

The rest of the paper is organized as follows: static and dynamic versions of MKP is presented in Section 2, solution approaches DE, GA and FA are discussed in detail in Section 3. Sections 4–5 represent experimental results and conclusion, respectively.

2. Dynamic multidimensional knapsack problem

As reported in the studies of Branke et al. (2006), MKP is in the NP-complete class and according to Kellerer et al. (2004) knapsack problems have been widely used as combinatorial benchmark problems of evolutionary algorithms. In addition, because MKP has numerous dynamic real-life applications such as cargo loading, selecting project funds, budget management, cutting stock, etc., dynamic version of MKP was used as benchmark environment in this study. Well known stationary version of MKP was formalized as follows in the scientific literature:

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

$$\text{s.t.} \quad \sum_{j=1}^{n} w_{ij} x_j \leqslant c_i \quad \forall i \in I \tag{2}$$

$$x_j \in \{0,1\} \forall j \in J \tag{3}$$

where $J = \{1, 2, 3, \ldots, n\}$ set of items, $I = \{1, 2, 3, \ldots, m\}$ set of dimensions of knapsack, n is the number of items, m is the number of dimensions of the knapsack, x_j is the binary variable denoting whether the jth item is selected or not, p_j is profit of the jth item, c_j is the capacity of the ith dimension of the knapsack and w_{ij} is the unit resource consumption of the jth item for the ith dimension of the knapsack and all parameters are assumed to be positive.

Literature includes well known benchmark generators for DOPs (Morrison, 2004; Rohlfshagen, Lehre, & Yao, 2009; Ursem, Krink, Jensen, & Michalewicz, 2002; Yang, 2003). Those generators briefly translate a well-known static problem into a dynamic version using specialized procedures. Dynamism is adopted here as the changes of the parameters after a predefined simulation time units similar to Branke et al. (2006). Simulation time unit represents number of iterations allocated for each environment. In other words it represents a frequency of dynamic change. The less number of simulation time units yields to more frequent changes and vice versa. In this study, a series of 1000 iterations is adopted as the frequency of changes.

An initial environment is required as a basis in order to dynamically generate other environments. Therefore, as Branke et al. (2006), the first instance of *mknapcb4.txt*[1] was used as the basis environment. This instance includes 100 items and 10 dimensions with a tightness ratio of 0.25. After a change occurs, the parameters are updated as stated below.

$$p_j = p_j * (1 + N(0, \sigma_p)) \tag{4}$$

$$w_{ij} = w_{ij} * (1 + N(0, \sigma_w)) \tag{5}$$

$$c_i = c_i * (1 + N(0, \sigma_c)) \tag{6}$$

The standard deviation of normal distributions of each parameter are assumed to be equal and $\sigma_p = \sigma_w = \sigma_c = 0.05$ which yields to an average 11 out of the 100 items to be assigned or removed (Branke et al., 2006).

Distinctly from the other authors, neither lower nor upper bounds for those dynamic parameters are employed here. It's thought that if a change (whether with respect to a probability distribution or other sources of dynamism) occurs in dynamic environment, bounding it might not exactly reflect real-life applications. Solution approaches for DE, GA and FA are discussed in the following section.

3. Solution approaches

As stated in the aforementioned sections DOPs considerably differ from the traditional stationary problems (Baykasoğlu & Durmuşoğlu, 2011, 2013). In the typical black box optimizations, a population based algorithm starts from randomly chosen points in the search space or mapped representations because of the lack of information. The algorithm is expected to converge through generations. Like many other factors, complexity of problem has an apparent impact on the convergence capability of the algorithm. However, assuming that the changing optimum is around the previous one, intuition suggests that using the information of the previously visited points might make a complex problem to be solved. In other words, once a high-quality solution is reached for a time period of a DOP then the tracking good solutions might be easier unless a severe change occurs. This can also reduce the computational complexity of the problem (Rohlfshagen & Yao, 2011). A basic approach to solve a DOP might be restarting the algorithm with its initially set parameters once a change occurs. In other words DOP can be handled as a sequence of stationary problems. Conversely, the algorithm can be allowed to run in a continuous manner and not take any action to changes. Considering those opposite approaches, it can be seen that, convergence which is a desirable feature for stationary environments might not be enough

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/mknapcb4.txt.

on DOPs. Diversity is required to be introduced and to be maintained on appropriate levels through generations for DOPs.

In the literature, there are published works mainly focusing on the adaptation of traditional evolutionary algorithms to dynamic problems. Rossi, Barrientos, and del Cerro (2007) proposed adaptive mutation operators for tracking of changing optima. The two well-known techniques, hyper-mutation (Cobb, 1990) and random immigrants (Grefenstette, 1992) are mainly based on the adaptation of the population using adaptive parameters. Other recent adaptation of parameters based approaches jDE and SaDE were reported in the studies of Brest, Greiner, Bošković, Mernik, and Zumer (2006) and Qin, Huang, and Suganthan (2009) where scaling factors and crossover rates were used as self-adaptive parameters.

Another widespread technique is exploiting implicit or explicit memories, where memory is embedded in solution representation (Goldberg & Smith, 1987; Hadad & Eick, 1997) or is a record of the previously revealed good solutions (Yang, 2005, 2006).

Additionally, Branke, Kau, Schmidt, and Schmeck (2000) and Yazdani, Nasiri, Sepas-Moghaddam, and Meybodi (2013) proposed a multi population based approach where whole population is divided into smaller sized populations. According to this approach while a subpopulation is tasked with convergence and solution improvements, some other subpopulations introduce and maintain diversity.

A considerably different approach was proposed in van Hemert, van Hoyweghen, Lukschandl, and Verbeeck (2001), Bosman (2005), Bosman and Poutrè (2007). According to their approaches, the data of the problem is collected during the optimization process and an inference is performed in order to predict the future events in a proactive manner.

In the present study an improved version of FA with an adaptation technique applicable on both stationary and dynamic environments is proposed. However, for a fair comparison and evaluation on the base algorithms, any improving procedures such as adaptive memory, specialized crossover or mutation techniques are not allowed.

3.1. Differential evolution

DE is a population based approach which uses the differences of the individuals in the population and like other evolutionary algorithms. It was originally introduced by Storn and Price (1995). DE uses crossover mutation and selection operators. Mutation here is assumed as the difference of randomly chosen individuals. There are different schemes of DE reported in the literature (Das & Suganthan, 2011) but *DE/rand/1/bin* is discussed here. The main steps of DE are given in Fig. 1.

3.1.1. Initialization

Like other population based algorithms, DE starts with a random (optionally initial solutions might be improved) population, comprised of vectors X_i^G. A sample representation of X_i^G is illustrated in Fig. 8.

$$i = 1,\ldots,NP; \quad j = 1,\ldots,D; \quad G = 1,\ldots,G_{max} \tag{7}$$

where NP is the number of the individuals in the population, D is the dimension of an individual, G and G_{max} are the generation index and maximum number of generations, respectively. Finally $x_{i,j}^G$ is the value of the jth dimension of the ith individual at iteration G. Thus, initial population is generated as follows:

$$x_{i,j}^1 = rand_j[0,1] \times (ub_j - lb_j) \quad \forall i \in [1,NP] \quad \forall j \in [1,\mathrm{D}] \tag{8}$$

where ub_j and lb_j are the upper and lower bounds of the corresponding dimensions if they exist.

3.1.2. Mutation

For each target vector X_i^G a mutant vector $V_i^{G+1}$is generated using the equation below:

$$V_i^{G+1} = X_{r_3}^G + F \times \left(X_{r_1}^G - X_{r_2}^G\right) \quad r_1,r_2,r_3 \in [1,NP], \quad r_1 \neq r_2 \neq r_3 \neq i \tag{9}$$

where r_1, r_2, r_3 are random integer numbers, illustrating the index of the randomly chosen individual.

3.1.3. Crossover

The mutant vector is crossed with the target vector in order to produce trial vector U_i^{G+1}

$$u_{i,j}^{G+1} = \begin{cases} v_{i,j}^{G+1} & \text{if } (rand_j \leqslant CR) \\ x_{i,j}^G & \text{otherwise} \end{cases} \tag{10}$$

where $rand_j$ [0, 1] is a random number and $CR \in$ [0, 1] is the crossover rate.

3.1.4. Selection

Finally, fitness of the trial vector U_i^{G+1} is evaluated and if the fitness of U_i^{G+1} is better than the target vector X_i^G, X_i^{G+1} is updated as U_i^{G+1}, otherwise X_i^{G+1} is assigned as X_i^G. Unlike the selection

```
1:  Setting the parameters of DE
2:  Generate initial population of individuals X_i (i = 1,2, ... , n)
3:  Assume that f(X_i^G) is the objective value of X_i^G
4:  while (g<MaxGen)
5:      for i=1:n        /*n is the population size*/
6:          Create the mutant vector V_i^G+1
7:          Apply crossover using V_i^G+1 and X_i^G to produce U_i^G+1
8:          if f(U_i^G+1) is better than f(X_i^G)
9:              X_i^G+1 =: U_i^G+1
10:             f(X_i^G+1) =: f(U_i^G+1)
11:         else
12:             X_i^G+1 =: X_i^G
13:             f(X_i^G+1) =: f(X_i^G)
14:         end if
15:     end for
16:     Find the current best, update the global best if required
17: end while
18: Results and visualization
```

Fig. 1. A pseudo code for DE.

operator as in GAs, selection here gives equal chances to all trial vectors generated from target vectors.

An illustration of operations in DE is given in Fig. 5a. As can be seen from the figure summation/differentiation and scaling operations are applied to r_1, r_2 and r_3. The *mutant* is crossed with *target* (*parent*). Finally, the *trial* is replaced with *target* if it achieves a better fitness.

3.2. Genetic algorithm

GA simulates the behavior of evolution of organisms. It's a widely used stochastic search technique which can find solutions to various NP-hard problems (Goldberg, 1989; Holland, 1975; Lim, 2012; Thengade & Dondal, 2012). GA can also be considered as a basis of evolutionary algorithms. A

```
Setting the parameters of GA
Generate initial population of chromosomes X_i (i = 1,2, ... , n)
Evaluate the fitness of whole population
while (g<MaxGen)
        Perform crossover and mutation operators
        Evaluate the fitness of whole population
        Find the current best, update the global best if required
        Perform selection&copy
end while
Results and visualization
```

Fig. 2. A pseudo code for GA.

```
Setting the parameters of FA
Generate initial population of fireflies X_i (i = 1,2, ... , n)
Assume that f(X) is the objective function of X (x_1, x_2, ... , x_d)
Light intensity L_i is determined by f(X_i)
while (g<MaxGen)
    for i=1:n    all n fireflies
        for j=1:n    all n fireflies
            if (L_j > L_i)
                move firefly i towards j in all d dimensions
            end if
            Attractiveness varies with the distance r via e^(-r^2)
            Evaluate new solution and update new light intensity L_i
        end for
    end for
    Rank the fireflies and find the current best, update the global best if required
end while
Results and visualization
```

Fig. 3. A pseudo code for FA.

```
Setting the parameters of FA
Generate initial population of fireflies X_i (i = 1,2, ... , n)
Assume that f(X) is the objective function of X (x_1, x_2, ... , x_d)
Light intensity L_i is determined by f(X_i)
while (t<MaxGen)
    ς=mod((t-1), MaxGen)/MaxGen    /*MaxGen:=frequency in dyn. env.*/
    for i=1:n    all n fireflies
        for j=1:n    all n fireflies
            ξ=(rank_i)^-ς
            τ=rand ∈ [0,1]
            if (L_j > L_i) and τ ≤ ξ
                move firefly i towards j in all d dimensions
            end if
            Attractiveness varies with the distance r via (1/r)
            Evaluate new solution and update new light intensity L_i
        End
    End
    Rank the fireflies and find the current best, update the global best if required
End
Results and visualization
```

Fig. 4. A pseudo code for FA_2.

pseudo code representing the generic scheme of GA is illustrated in Fig. 2.

In this study a single point crossover is used and one offspring is allowed to be generated. Offspring has a 0.5 chance to take the first segment from the first parent and second segment from the second parent; and 0.5 chances to take the first segment from the second parent and the second segment from the first parent.

Mutation operator assigns a new random number $\varepsilon[0,1]$ to the bits. Finally, roulette wheel selection is used as the selection operator.

A schematic illustration for obtaining new solutions using crossover and mutation for GA is given in Fig. 5.b. According to this figure, ch_1 is crossed with the ch_2 resulting in ch_{temp}. Finally mutation operator is used for generating an offspring.

3.3. Firefly algorithm

FA is a population based metaheuristic algorithm which simulates the flashing and communication behavior of fireflies. It was originally introduced by Yang (2008, 2009). In a typical FA algorithm the brightness of a firefly means how well the objective of a solution is and brighter firefly (solution) attracts other fireflies. As the distance between fireflies increases, the attractiveness decreases exponentially because of the light absorption of medium. A comprehensive survey of this new metaheuristic optimizer was presented in the studies of Fister et al. (2013a), Fister, Yang, Brest, and Fister (2013b) and Yang and He (2013).

The application of the FA for dynamic environment is a new developing research area and there are relatively fewer published papers on this topic. Abshouri, Meybodi, and Bakhtiary (2011) hybridized FA with a learning automata for tuning the parameters of FA. Farahani, Nasiri, and Meybodi (2011) used a multi-population based FA by splitting the whole population into interacting sets of smaller swarms. Additionally each swarm was allowed to interact locally and globally through an anti-convergence operator to prevent an early convergence circumstance. Nasiri and Meybodi (2012) adopted an elitist approach within FA by keeping the best solution through generations. All these FA applications were tested on the well known multi-modal dynamic moving peaks problem (Branke, 1999) and the efficiency of the proposed FAs were shown by the authors.

Another dynamic optimization application of FA was presented by Chai-ead, Aungkulanon, and Luangpaiboon (2011). A comparison of FA and bees algorithm was performed on the various types of noisy continuous mathematical functions with two variables. According to the findings of the study, authors concluded that the FA outperformed Bees algorithm.

Due to the structural properties (i.e. using floating numbers in its bits) of FA, this new optimizer is expected to perform more efficiently particularly on continuous environments and as it can be seen from above, whole literature of dynamic optimization for FA was focused on this domain. However, the performance of FA on dynamic combinatorial problems still remains as a question mark. Additionally the most of real-life problems are combinatorial in nature. To the best of our knowledge this paper constitutes the first study on the performance of FA on a dynamic combinatorial optimization problem.

FA has three main assumptions:

i. All fireflies are unisexual and every firefly attracts/gets attracted to every other firefly.
ii. The attractiveness of a firefly is directly proportional to the brightness of the firefly (The brightness decreases as the distance increases).
iii. They move randomly if they do not find a more attractive firefly in adjacent regions.

Table 1
Adaptive probabilities ξ through iterations ($\xi = (rank_i)^{-\zeta}$).

Rank of fireflies in the swarm (1st rank is the fittest)	Iterations																						
	1	**2**	**3**	**4**	**5**	**6**	**7**	…	**500**	**501**	**502**	**503**	**504**	**505**	**506**	…	**995**	**996**	**997**	**998**	**999**	**1000**
1	1,000	1,000	1,000	1,000	1,000	1,000	1,000	…	1,000	1,000	1,000	1,000	1,000	1,000	1,000	…	1,000	1,000	1,000	1,000	1,000	1,000
2	1,000	0,999	0,999	0,998	0,997	0,997	0,996	…	0,708	0,707	0,707	0,706	0,706	0,705	0,705	…	0,502	0,502	0,501	0,501	0,501	0,500
3	1,000	0,999	0,998	0,997	0,996	0,995	0,993	…	0,578	0,577	0,577	0,576	0,575	0,575	0,574	…	0,336	0,335	0,335	0,334	0,334	0,334
4	1,000	0,999	0,997	0,996	0,994	0,993	0,992	…	0,501	0,500	0,499	0,499	0,498	0,497	0,497	…	0,252	0,252	0,251	0,251	0,251	0,250
5	1,000	0,998	0,997	0,995	0,994	0,992	0,990	…	0,448	0,447	0,446	0,446	0,445	0,444	0,444	…	0,202	0,202	0,201	0,201	0,201	0,200
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
50	1,000	0,996	0,992	0,988	0,984	0,981	0,977	…	0,142	0,141	0,141	0,140	0,140	0,139	0,139	…	0,020	0,020	0,020	0,020	0,020	0,020
51	1,000	0,996	0,992	0,988	0,984	0,981	0,977	…	0,141	0,140	0,139	0,139	0,138	0,138	0,137	…	0,020	0,020	0,020	0,020	0,020	0,020
52	1,000	0,996	0,992	0,988	0,984	0,980	0,977	…	0,139	0,139	0,138	0,138	0,137	0,137	0,136	…	0,020	0,020	0,020	0,019	0,019	0,019
53	1,000	0,996	0,992	0,988	0,984	0,980	0,976	…	0,138	0,137	0,137	0,136	0,136	0,135	0,135	…	0,019	0,019	0,019	0,019	0,019	0,019
54	1,000	0,996	0,992	0,988	0,984	0,980	0,976	…	0,137	0,136	0,136	0,135	0,134	0,134	0,133	…	0,019	0,019	0,019	0,019	0,019	0,019
55	1,000	0,996	0,992	0,988	0,984	0,980	0,976	…	0,135	0,135	0,134	0,134	0,133	0,133	0,132	…	0,019	0,019	0,018	0,018	0,018	0,018
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
95	1,000	0,995	0,991	0,986	0,982	0,977	0,973	…	0,103	0,103	0,102	0,102	0,101	0,101	0,100	…	0,011	0,011	0,011	0,011	0,011	0,011
96	1,000	0,995	0,991	0,986	0,982	0,977	0,973	…	0,103	0,102	0,102	0,101	0,101	0,100	0,100	…	0,011	0,011	0,011	0,011	0,011	0,010
97	1,000	0,995	0,991	0,986	0,982	0,977	0,973	…	0,102	0,102	0,101	0,101	0,100	0,100	0,099	…	0,011	0,011	0,010	0,010	0,010	0,010
98	1,000	0,995	0,991	0,986	0,982	0,977	0,973	…	0,101	0,101	0,101	0,100	0,100	0,099	0,099	…	0,010	0,010	0,010	0,010	0,010	0,010
99	1,000	0,995	0,991	0,986	0,982	0,977	0,973	…	0,101	0,101	0,100	0,100	0,099	0,099	0,098	…	0,010	0,010	0,010	0,010	0,010	0,010
100	1,000	0,995	0,991	0,986	0,982	0,977	0,973	…	0,100	0,100	0,100	0,099	0,099	0,098	0,098	…	0,010	0,010	0,010	0,010	0,010	0,010
Values of ς	**0,000**	**0,001**	**0,002**	**0,003**	**0,004**	**0,005**	**0,006**	…	…	**0,499**	**0,5**	**0,501**	**0,502**	**0,503**	**0,504**	**0,505**	**0,506**	…	**0,994**	**0,995**	**0,996**	**0,997**	**0,998**	**0,999**

The distance between two fireflies (i and j) can simply be evaluated as the Euclidean distance as stated below in Eq. (11), where D is the dimension, x_{ik} is the kth dimension of the ith firefly.

$$r_{ij} = \sqrt{\sum_{k=1}^{D}(x_{ik} - x_{jk})^2} \tag{11}$$

The attractiveness of a firefly decreases exponentially as the distance increases. The attractiveness of a firefly at a distance γ_r is formulated as given in Eq. (12), where γ is the light absorption coefficient, m is number which modifies the distance metric and $\beta_{(0)}$ is the attractiveness at 0 distance usually accepted as 1.

$$\beta_{(r)} = \beta_{(0)} e^{-\gamma r^m} m \geqslant 1 \tag{12}$$

Finally, movement of a firefly, i is attracted to another more attractive (brighter) firefly j, is determined by Eq. (13)., where $\alpha \in [0, 1]$ is a user supplied scaling factor of randomness and $\psi \in [0, 1]$ is a random number.

$$x_{ik} = x_{ik} + \beta_{(0)} e^{-\gamma r^2}(x_{jk} - x_{ik}) + \alpha(\psi - 0.5) \tag{13}$$

A pseudo code for FA is given below in Fig. 3.

Fig. 5c represents a schematic movement of fireflies. f_1, f_2 and f_3 are assumed to have the 1st 2nd and 3rd rank, where the 1st rank represents the fittest firefly. According to this figure, f_2 is only

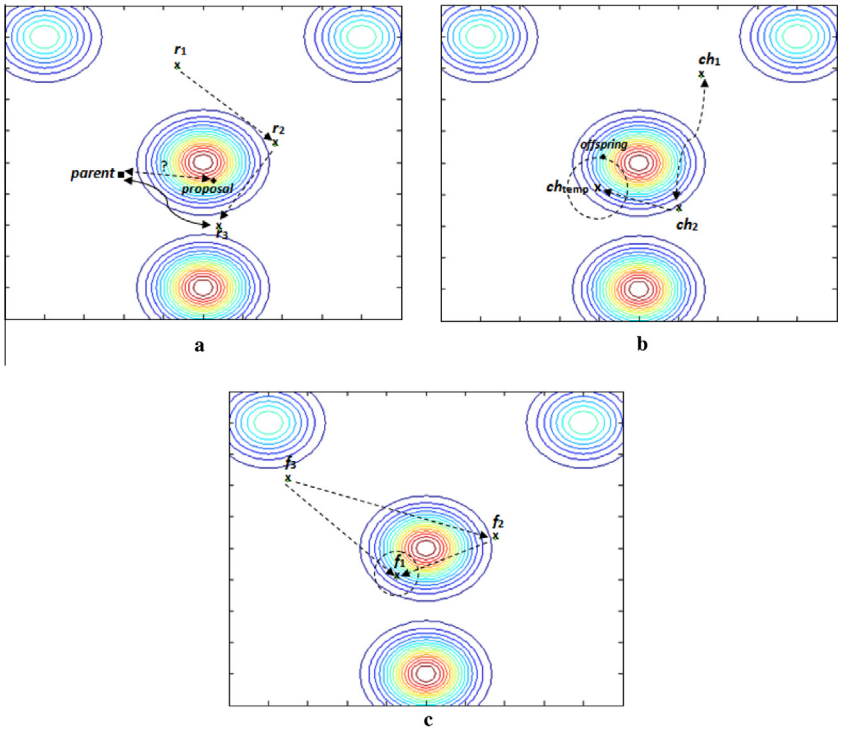

Fig. 5. (a) Obtaining a new proposal in DE (r_i: ith randomly chosen individual) (b) Obtaining a new solution in GA (ch_i: ith chromosome; ch_{temp}: temporary chromosome) (c) A schema of pair-wise comparison and movement in FA (f_i: firefly with the ith rank).

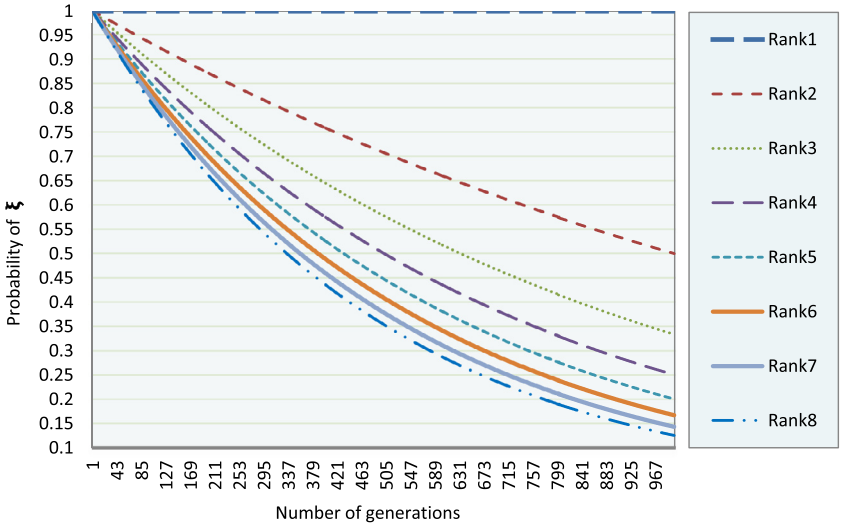

Fig. 6. A visual illustration of ξ for the corresponding rank though iterations.

moved in the direction of f_1, where f_3 is moved to the positions of both f_1 and f_2. The fittest firefly f_1 is allowed to fly randomly.

As it can be seen from the Fig. 3, FA performs a full pair-wise comparison which can be time consuming. As a result the search might suffer from the computations of full pair-wise comparison. Another issue is that the exponential move function might deteriorate the performance of FA unless the parameters are precisely set to their appropriate levels (Eq. (13)). It is widely reported in the literature that γ is assumed to be equal to 1; however, as the distance between fireflies increases, $\beta_{(0)}e^{-\gamma r^2}$ approaches to zero which means that such an assumption might cause fireflies be stucked at a single position through generations. In order to prevent this circumstance, γ can be set to very small values, however, obtaining the appropriate level of γ is not an easy task. Considering these two drawbacks, an improved modification of FA called FA_2 is presented in this study.

```
Let X = (x1, x2, ..., xd) be the individual to be decoded
Each bit of X shows the priority of the corresponding item
for i=1:NP     (NP: population size)
        for j=1:D     (D:dimension)
                k=argmax (X)
                if k^th item satisfies capacity constraints
                        k^th item is included in the knapsack
                end if
                xk=-∞
        end for
end for
```

Fig. 7. A pseudo code to obtain a solution from an encoded individual.

0.421	0.739	0.016	0.294	0.917	0.579	0.371	0.842	0.703	0.428
1	2	3	4	5	6	7	8	9	10

Fig. 8. An example for solution encoding.

Table 2
Parameters of the algorithms.

	GA	DE	FA	FA_2
Maximum generation	10,000	10,000	10,000	10,000
Frequency	1000	1000	1000	1000
Population size	100	100	100	100
Crossover rate	1.0	0.5	*Na*	*Na*
Mutation rate	0.09	*Na*	*Na*	*Na*
F (scaling factor)	*Na*	0.8	*Na*	*Na*
α	*Na*	*Na*	*0.9*	0.9
$\beta_{(0)}$	*Na*	*Na*	1.0/0.35	1.0/0.35
γ	*Na*	*Na*	0.001	*Na*

3.4. Improved firefly algorithm

According to FA_2 a full pair-wise comparison is replaced with a specialized comparison. The parameter ς is changed from zero to approximately one through iterations by using the Eq. (14), where t is the iteration counter and *maxGen* is limit of iterations. It must be noted that *maxGen* is replaced with *frequency* in dynamic environment. Because ς must be re-initialized as the dynamic change is detected and then it adapts itself though the generations where the new environment remains stationary.

$$\varsigma = mod((t-1), maxGen)/maxGen \tag{14}$$

Another parameter ξ represents the probability of a firefly to be moved or not. Differently from the original FA, with the use of parameter ξ the movement of a firefly not only depends on the brightness of a more attractive firefly but also depends on a check that a random number $\tau \in [0,1]$ must satisfy the threshold value of ξ as illustrated in pseudo codes of FA_2 (given in Fig. 4). Thus, movement of a firefly now depends on both the rank of the more attractive firefly and iteration counter of the run. Calculations and values of ξ for 1000 iterations/frequency and adaptation of ξ through iterations for the corresponding rank of firefly for a sample swarm of 8 fireflies are illustrated in Table 1 and Fig. 6, respectively.

Another critical issue here is that the fittest firefly is never missed, because any power of 1 is always 1 which is the upper bound for $\tau \in [0,1]$. As a result, although complexity of the comparison procedure (which is $O(n^2)$) is not reduced and the algorithms FA and FA_2 perform exactly the same number of inner loops, an improvement for the required CPU time is obtained due to lesser calculations performed in FA_2. A simple test on the average of 30 runs show that, move function is called 89.548.283 times in FA and 66.555.939 times in FA_2. Moreover, early convergence of the swarm is attempted to be prevented by using this technique.

In Fig. 6, each individual curve$_i$ with different colors and types correspond to the firefly of rank$_i$ in increasing order from top to bottom. The *y-axis* values of each individual curve at any generation index of *x-axis* represent the probability of a firefly of the corresponding rank, to attract another firefly of greater ranks. For example the firefly of the 1st rank (the uppermost curve) is always allowed to attract any other fireflies of greater ranks regardless of the generation index. However, for the firefly of 2nd rank, the probability of attraction of firefly of this rank is decremented trough generations. Assuming that, although it has a better objective value, at generation 500, that firefly has a chance of approximately 0,708 (see Table 1) to attract any other firefly of greater ranks. This chance (ξ) is systematically decreased through generations for all ranks as given in Fig. 6 and Table 1. A point should be given attention that at the initial generation, any brighter firefly has a chance of 1.00 to attract others with greater ranks.

Table 3
Experimental results on stationary environments.

	GA		DE		$\beta_{(0)}$ = 1.0				$\beta_{(0)}$ = 0.35			
					FA		FA_2		FA		FA_2	
#	*Best*	*CPU* (*s*)	*Best*	*CPU* (*s*)	*Best*	*CPU* (*s*)	*Best*	*CPU* (*s*)	*Best*	*CPU* (*s*)	*Best*	*CPU* (*s*)
Env_1	3087,80	120,19	1776,80	110,37	1127,20	240,92	197,20	112,48	404,60	227,51	**34,20**	115,21
Env_2	3340,10	127,46	1877,05	107,45	1249,53	236,52	298,83	102,32	441,54	242,28	**175,70**	107,34
Env_3	3432,54	125,98	2112,76	115,21	1422,46	193,41	201,55	104,08	535,63	156,37	**154,40**	116,24
Env_4	3208,21	122,54	1859,10	99,64	1172,12	225,63	181,15	111,61	511,33	168,42	**75,61**	120,12
Env_5	3669,24	130,53	2233,41	112,98	1680,68	260,42	376,47	114,82	684,96	198,72	**276,22**	101,84
Env_6	3469,43	138,14	2000,46	107,56	1378,80	224,85	222,48	105,50	598,21	201,29	**192,14**	107,93
Env_7	3272,27	126,73	1858,90	102,64	1554,08	183,12	313,89	103,16	800,57	156,41	**158,61**	114,52
Env_8	3121,18	122,52	1565,68	109,49	1051,17	238,37	366,18	108,19	444,79	134,25	**220,57**	117,51
Env_9	2971,64	119,26	1442,65	103,52	1546,54	281,14	306,67	121,13	679,29	194,21	**190,29**	104,25
Env_{10}	3257,68	125,62	1888,60	101,71	1152,78	215,14	294,53	102,52	608,01	178,97	**163,20**	124,46

Due to the second drawback as mentioned before, the exponential move function is replaced with a move function which is similar to Rahmaniani and Ghaderi (2013) in FA_2. This function is given in Eq. (15):

$$x_{ik} = x_{ik} + \beta_{(0)}(1/(\Omega + r))(x_{jk} - x_{ik}) + \alpha(\psi - 0.5) \quad (15)$$

where Ω is very small number like 0.000001 in order to prevent the division by zero.

3.5. Solution representation

Although GA supports binary or permutation representation, it's obvious that direct use of DE and FA to discrete search spaces yields to illegal solutions. In other words using difference of vectors as in DE or moving operations as in FA is not appropriate in binary, permutation or other similar representation techniques. A mapping procedure is required here in order to show that an item is assigned in the knapsack or not. Therefore in this study, *priority based encoding technique* (Chitra & Subbaraj, 2012; Lotfi & Tavakkoli-Moghaddam, 2012) is used in order to represent a solution.

According to this representation technique, an individual has bits as many as the number of items considered in the problem. The bits of the individuals simply hold unbounded continuous numbers, representing the priority of each item to be included in the knapsack. Additionally, the initial population is comprised of random numbers bounded within the interval [0, 1]. A pseudo code

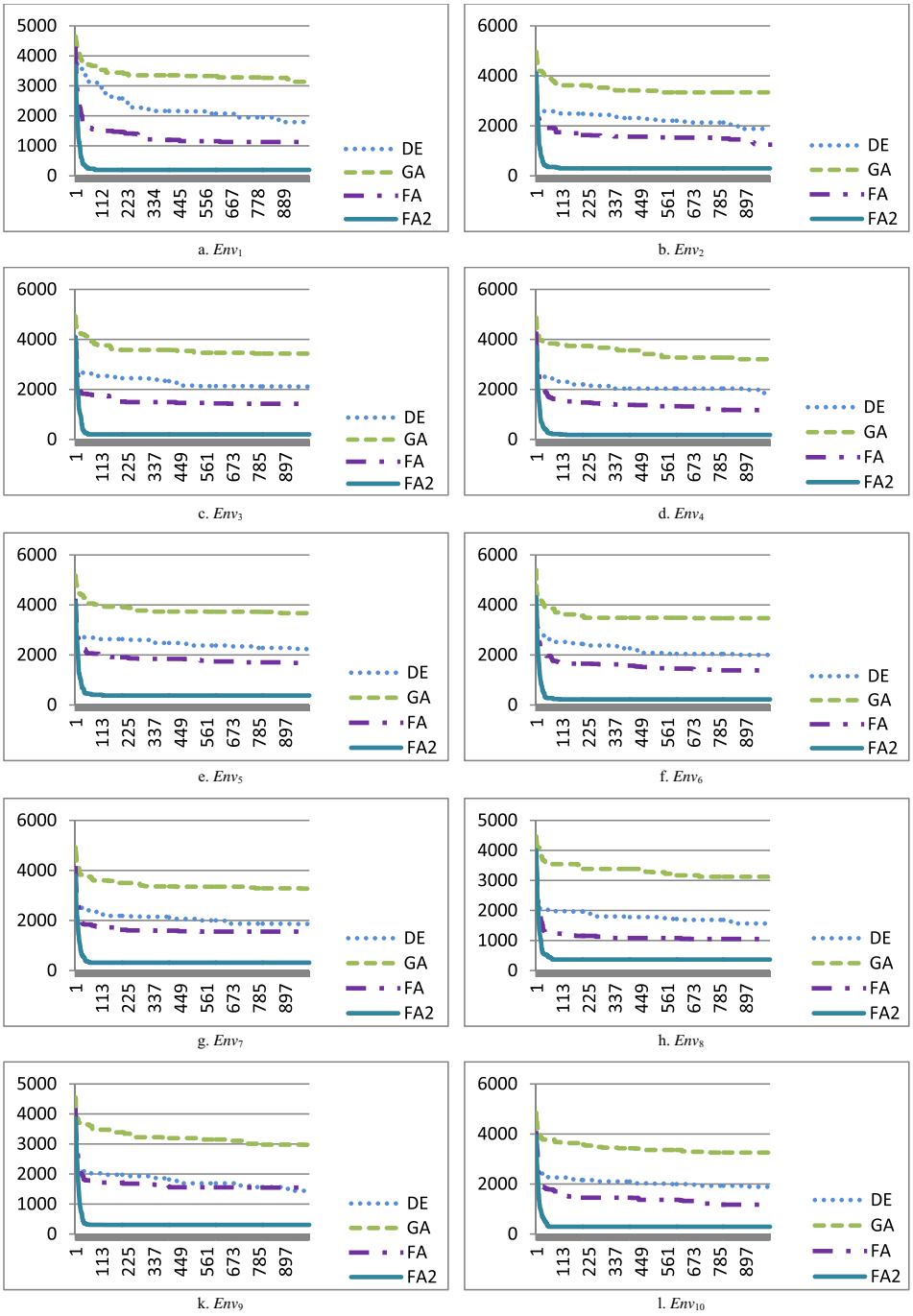

Fig. 9. (a-l) Convergence of GA, DE, FA and FA_2 on each stationary environment ($\beta_{(0)} = 1.0$).

for constructing a solution and a sample encoded solution are presented in Figs. 7 and 8, respectively.

This technique always constructs feasible solutions and prevents infeasibility for both stationary and dynamic environments. Thus, any repairing procedure is not required.

As it can be seen from Fig. 8, there are items indexed in a range from 1 to 10 (2nd row), associated with random priorities (1st row). A higher number in 1st row provides a higher priority to the corresponding item to be assigned unless it violates capacity constraints. According to the priorities, items 5, 8, 2, 9, 6, 10, 1, 7, 4, 3 are attempted to be assigned in sequence. Obviously, assuming that an assignment is performed, remaining capacities of dimensions are updated. For instance, let $item_5$ be assigned. Then the capacities of the knapsack dimensions are decreased as much as the resource requirements of $item_5$. Suppose that remaining capacities are not enough for $item_8$. This yields $item_8$ to be discarded and the same procedure is carried out for the rest of the sequenced items. Finally the fitness of the solution is evaluated by summation of the assigned items.

3.6. Detecting dynamic changes

As mentioned in Yazdani et al. (2013), dynamic changes can occur in two different ways as globally and locally. Keeping any random point (*test_point*) from solution space and comparing its current fitness value with the fitness value of the previous generation detects the global changes. If a difference between two consecutive fitness values of *test_point* is observed, it means

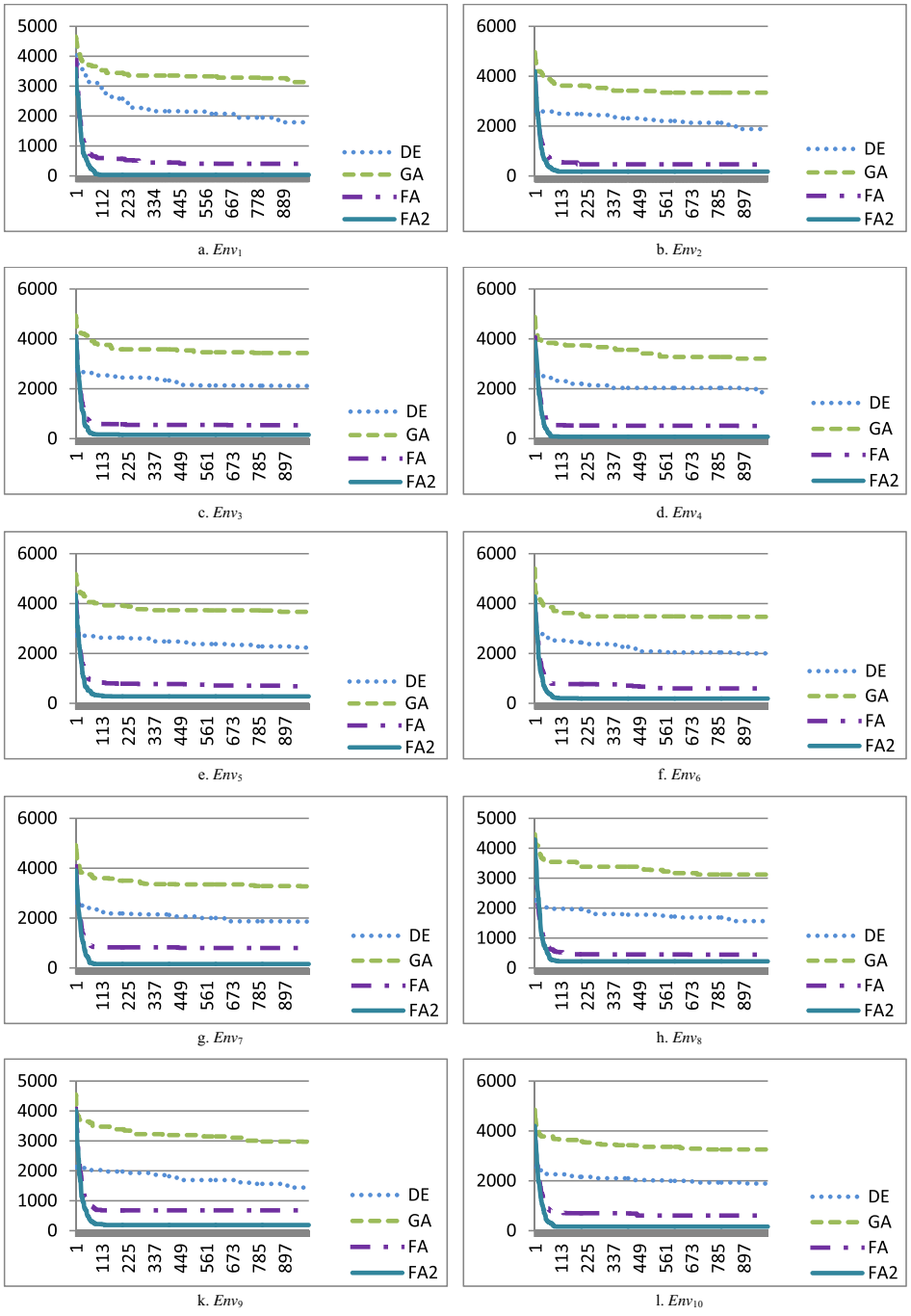

Fig. 10. (a-l) Convergence of GA, DE, FA and FA_2 on each stationary environment ($\beta_{(0)} = 0.35$).

that the environment has changed. It must be noted that such detection test is only appropriate for global changes. However, when a small local change occurs out of the vicinity of the *test_point*, there's a probability that the algorithm cannot detect the change. Assuming that the change has occurred in the neighborhood of the optimum, then optimum solution of the previous generation can be used as the *test_point*. Yazdani et al. (2013) presented a multi-swarm approach to evaluate fitness of the *global_best_solutions* of each swarm at the end of each iteration in order to detect such changes by making use of the advantage of multi-swarm based approaches.

In the present work, dynamic changes are assumed to occur for all of the objective's and constraints' coefficients similar to Branke et al. (2006). Therefore, both solution vector which is constructed by using *test_point* and fitness value of the solution are expected to change subsequent to dynamic event. However, it should be noted here that there is a probability for the solution vector to remain stationary. For this reason, at the end of each iteration, a check on both solution vector and fitness value was performed as a detection test in this study. As a result, if a change is detected between consecutive generations' solution vector or fitness value of the *test_point*, then algorithms react to that change as partial re-initializations to introduce diversity. Additionally, FA_2 adapts the parameter ξ to increase "moving chance" of fireflies in the new environment during the initial generations. It is shown by the results that this approach has detected the changes at each run.

4. Experimental results

4.1. Experimental setup

As mentioned earlier, 10 different environments were generated using the first test problem of *mknapcb4.txt* as a basis. 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 of the environments on 30 runs are considered. The optimum of each environment was obtained using GAMS CPLEX solver. The parameters of DE, GA, FA and FA_2 are set as shown in Table 2. Additionally, as the step size for FA is a critical issue, $\beta_{(0)}$ is assumed to be equal to 1.0 and 0.35 separately, which provides a faster and slower convergence, respectively.

All tests were performed on a PC with hardware of 3.4 GHz processor and 4 GB RAM. Before representing the performance of the algorithms on dynamic environment, experimental results on stationary environments are discussed in the following subsection.

4.2. Results on stationary environment

Each of the dynamic environments is assumed as stationary environments here. In other words, algorithms are assumed to be reinitialized with a random initial population for each of the dynamic environments, and they are not allowed to gather and use the information from the predecessor environment. Results in terms of average of the best error from optimum are illustrated in Table 3.

Columns of Table 3 represent the name of the environment (#), the minimum error from the optimum in terms of average of five runs (*best*) and required CPU time (*CPU*) of the corresponding algorithm, respectively. The best performance among the algorithms on each environment is denoted in bold.

According to the results presented in Table 3, DE, FA and FA_2 shows a better performance than GA. Additionally, except for Env_9, both FA and FA_2 are found to be superior to both DE and GA which demonstrates the success of FA under those parameters. Moreover, the most surprising results are obtained by FA_2. Using a simpler move function, imposing additional restrictions not only reduced the required CPU time but surprisingly increased the solution quality. As it can be seen from Table 3, FA_2 achieves near optimal results. This circumstance can be explained as early convergence phenomenon which may deteriorate the overall performance of any optimizer. Considering the behaviors of fireflies from the viewpoint of real-life, those fireflies might not always precisely and correctly adjust their positions according to the positions of all other brighter fireflies. In other words, like other intelligent swarms, fireflies must be behaving in deterministic, random and adaptive manner simultaneously which is satisfied by FA_2. Thus, considering the improvements in CPU and solution quality, it can be said that the effectiveness of FA_2 is far better than other algorithms under the given parameters.

Another issue is that the parameter $\beta_{(0)}$ might affect the performance of FA crucially. As it can be seen from the corresponding columns of Table 3, compared to FA_2, results of FA are improved more steeply, which means that level of $\beta_{(0)}$ has a more significant role on the performance of the FA than it has on FA_2. In other words, compared to FA, FA_2 is found to be presenting a more robust performance.

Computational results demonstrate the effectiveness of FA_2 on stationary environments of MKP. In addition, convergence graphs of GA, DE, FA and FA_2 are illustrated in Figs. 9 and 10 for $\beta_{(0)} = 1.0$ and $\beta_{(0)} = 0.35$, respectively. Another surprising result is obtained according to those figures.

Compared to GA, DE, and FA; FA_2 has a significantly fast convergence capability on each of the stationary environments. As it can be seen from Figs. 9 and 10, FA_2 achieves near optimal results, approximately in 80 iterations. As stated before, for a fair comparison, none of the local search or solution improvement methods are used during the tests. Therefore, it's clear that, using such techniques might allow FA_2 to achieve optimal results in earlier iterations.

Table 4
Experimental results on dynamic environment.

	#	GA *best*	DE *best*	FA *best*	FA_2 *best*
0.00 Re-initialization	Env_1	2495,10	1296,03	155,67	**64,23**
	Env_2	2749,95	1224,54	**288,59**	300,15
	Env_3	2731,41	1182,60	349,19	**328,01**
	Env_4	2605,79	1119,09	447,14	**411,19**
	Env_5	2797,49	1278,89	630,18	**578,85**
	Env_6	2667,22	1191,80	502,85	**485,89**
	Env_7	2519,34	1100,96	616,87	**503,87**
	Env_8	2734,75	1151,88	504,57	**500,99**
	Env_9	2568,34	1106,50	709,11	**652,72**
	Env_{10}	2553,12	1181,20	643,96	**599,57**
0.30 Re-initialization	Env_1	2499,10	1269,70	134,63	**103,57**
	Env_2	2710,47	1235,21	336,65	**330,56**
	Env_3	2747,88	1225,12	**376,16**	381,11
	Env_4	2646,27	1158,47	456,20	**391,05**
	Env_5	2778,76	1328,24	545,43	**526,82**
	Env_6	2765,77	1288,36	**509,54**	514,92
	Env_7	2607,60	1140,58	549,13	**542,09**
	Env_8	2674,39	1226,11	**456,16**	488,97
	Env_9	2551,32	1215,30	636,89	**620,53**
	Env_{10}	2572,43	1263,64	543,16	**531,78**
0.70 Re-initialization	Env_1	2526,60	1277,60	147,73	**74,83**
	Env_2	2735,42	1317,27	267,84	**264,52**
	Env_3	2706,70	1359,98	270,08	**251,71**
	Env_4	2589,56	1214,60	332,48	**316,83**
	Env_5	2859,46	1438,12	466,20	**405,06**
	Env_6	2813,77	1366,02	**408,60**	440,24
	Env_7	2569,63	1223,63	408,68	**388,43**
	Env_8	2715,51	1295,66	374,35	**365,65**
	Env_9	2552,87	1314,21	**471,37**	514,24
	Env_{10}	2586,08	1285,55	400,16	**357,67**

Following subsection is devoted to the analysis of the performances of GA, DE, FA and FA_2 on dynamic environments.

4.3. Results on dynamic environment

Computational results for stationary environments demonstrate the effectiveness of FA_2. However, these results might be misleading for dynamic environments possibly because of lack of the *capability of tracking the changing optima*. For dynamic environments, an effective algorithm is expected to quickly adapt to new environments and track the changing optima. Moreover, diversity must be introduced or maintained through evaluations for adaptation to dynamic changes. Particularly just after the dynamic event is triggered, an effective algorithm is expected to detect the change and react to it. For this reason, tests on dynamic environments are performed by reinitializing and randomly restarting 0%, 30% and 70% of the population when a dynamic change is detected. This approach can be considered as partial

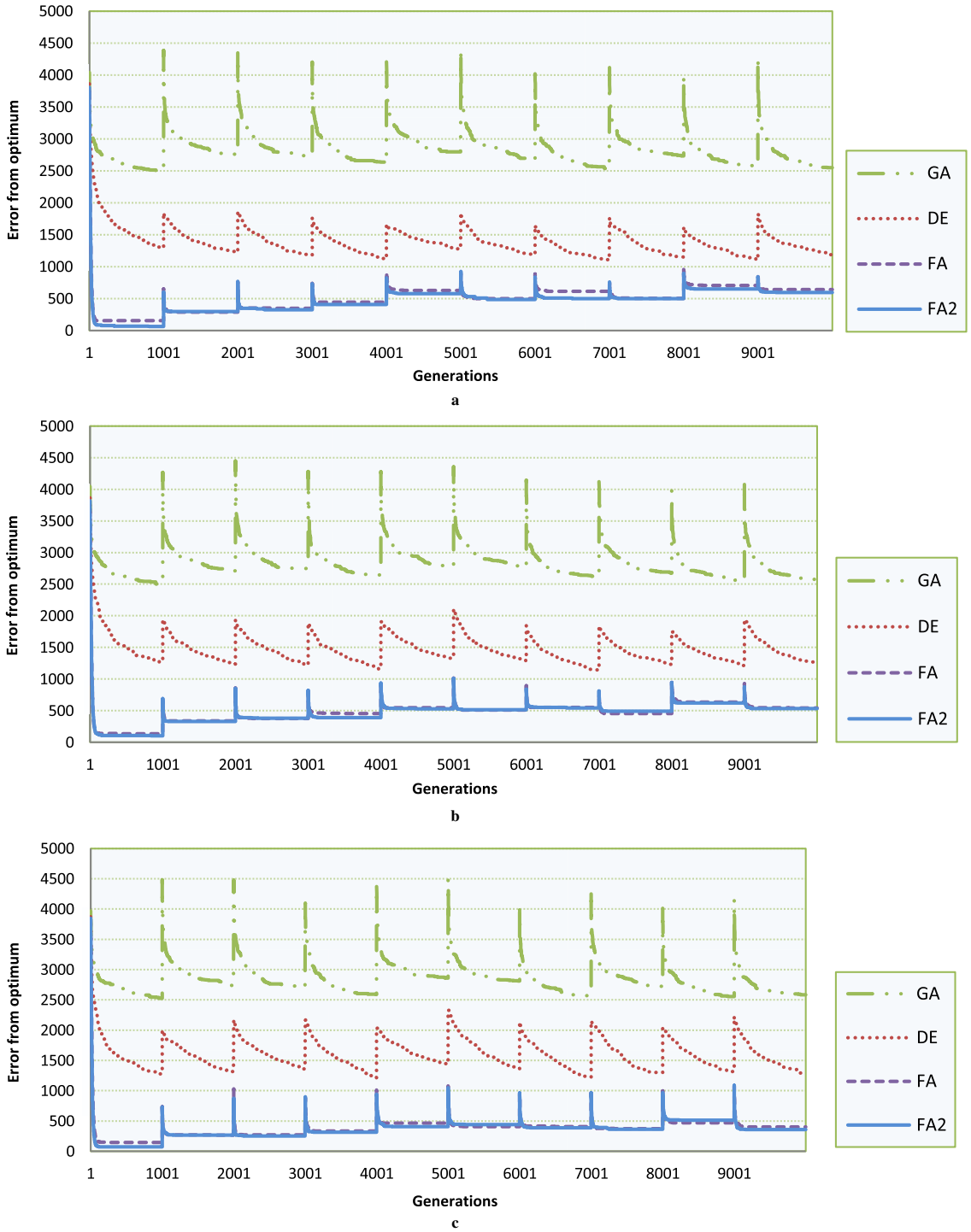

Fig. 11. Convergence of GA, DE, FA and FA_2 on dynamic environments with (a) 0% (b) 30% (c) 70% re-initialization.

random restarts, which is a trade-off between keeping the gathered evolutionary information through generations and employing some random scouts in the swarm to explore the new environment by allowing them to reinitialize their individual memories. Additionally, FA_2 adjusts the parameter ξ when a dynamic change is detected and readapts it through generations of the new environment. Thus, two common concepts of dynamic optimization namely, *introducing* and *maintaining diversity* are simultaneously implemented in FA_2.

As demonstrated in the previous subsection, both FA and FA_2 achieves betters results for $\beta_{(0)} = 0.35$. Therefore tests on dynamic environments are performed using this assumption. Results, in terms of the best average of errors of 30 runs for each algorithm with 0%, 30% and 70% restarts are presented in Table 4.

According to Table 4 where the algorithms are not allowed to be reinitialized, DE, FA and FA_2 are found to be more effective than GA. Additionally, FA_2 achieves the best results although an increment in the error from optimum is apparent compared to the results on stationary environment. However, if the algorithms are allowed to use the 0.30 and 0.70 of the previously gathered information, another surprising result is obtained.

It's obvious that results are improved for both FA and FA_2 which means that the performances of both algorithms increase as the rate of introduced diversity is increased. As a result, FA becomes competitive with FA_2. However, apparently this is not valid for GA and DE. It can be seen that GA performs an insensitive behavior to the rate of the diversity introduced whereas a slight decrement for the performance of DE is observed based on the results represented in Table 4.

Convergence graphs of the GA, DE, FA and FA_2, for the tests with 0.00, 0.30 and 0.70 restarts of the population are illustrated in Fig. 11a–c, respectively. It is apparent from this figure that, as the introduced diversity increases, an obvious increment for the severity of the deviance just after a change is observed for both FA and FA_2.

4.4. Statistical verification

In this subsection, a statistical verification is performed to prove what intuition suggests in the findings of the Sections 4.2 and 4.3. Therefore an initial analysis is scheduled to see whether a meaningful difference among algorithms exists. Friedman Test (Friedman, 1937, 1940) which is a non-parametric test, applicable to multiple classifiers is performed for this purpose. Subsequent to initial results, a paired *t*-test analysis is carried out between each pair of algorithms.

For tests on stationary and dynamic environments data set is comprised of mean errors from optimum on 5 and 30 runs, respectively. Additionally, a data set has 10000 entries due to the maximum number of generations is defined as 10,000. It should be noted that tests can also be applied using the best values of the environments given in Tables 3 and 4. However, because of the convergence capability of an algorithm is an important performance measure for dynamic optimization, all of the 10,000 entries are used here. For both Friedman Test and paired *t*-test, illustrative examples can be found in the studies of (Demsar, 2006).

Using the data set, F_F is calculated as 207600,05 via Excel. With 4 algorithms and 10,000 data entries, F_F distributed according to *F* distribution with $(4-1)(10000-1) = 29997$ degrees of freedom. The critical value of $F_{0.05,3,29997}$, for $\alpha = 0.05$ is 2.6049. As a result, the assumption under null-hypothesis (H_o) imposing equal average ranks is rejected. Therefore we can conclude that a significant difference from mean rank exists which proves that performances of algorithms significantly differ from each other.

Now we have the statistical prove that the performances of GA, DE, FA and FA_2 differ from each other. However, a further analysis is required to prove the superiority of any algorithm. Therefore, a paired *t*-test analysis is performed for each pair of algorithms as given in Table 5.

The paired *t*-test on stationary and dynamic environments is performed for mean of 5 and 30 runs, respectively as in Friedman Test. Each data set here again has 10,000 entries. MATLAB is used to calculate the *p*-values and *t*-values of each paired tests with the assumption of "*a difference does not exist between the mean results of each algorithm*" defined under null-hypothesis. As it can be seen from Table 5, each of the null-hypothesis is rejected according to their *t*-values. Additionally, *p*-values 0.0000 which gives us a probability of observing a test statistic as extreme as, or more extreme than the observed value under the null hypothesis shows the significance of the results.

Finally we can conclude that, due to the experimental results and statistical verification for both dynamic and stationary environments, FA_2 outperforms other three algorithms. These results demonstrate the effectiveness of FA_2 under the given parameters.

An evolutionary optimizer performing on dynamically changing environments needs to have some particular features as given below:

- A successful detection capability of dynamic events.
- Fast convergence capability overcoming the problem of stucking in local optima traps.
- Self adaptive capability when a change is detected.
- Not completely ignoring the gathered evolutionary information.

According to the findings of the present study, FA_2 is found to be meeting these requirements at satisfactory levels. Finally, based

Table 5
Significance (*p*-values), *t*-values and hypothesis check results of paired *t*-tests analysis for each data set ($\alpha = 0.05$, DF = 9999).

		GA–DE	GA–FA	GA–FA_2	DE–FA	DE–FA_2	FA–FA_2
Stationary env.	$\beta_{(0)} = 0.35$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$
		$t = 736.38$	$t = 958.88$	$t = 977.11$	$t = 379.91$	$t = 440.43$	$t = 324.87$
			H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject
	$\beta_{(0)} = 1.00$	H_o = reject	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$
			$t = 951.10$	$t = 272.57$	$t = 227.43$	$t = 482.97$	$t = 497.17$
			H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject
Dynamic env.	0.00 re-init.	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$
		$t = 748.48$	$t = 849.77$	$t = 909.08$	$t = 307.42$	$t = 324.41$	$t = 109.77$
		H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject
	0.30 re-init	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$
		$t = 880.64$	$t = 984.88$	$t = 994.49$	$t = 396.12$	$t = 397.60$	$t = 40.26$
		H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject
	0.70 re-init	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$	$p = 0.0000$
		$t = 846.16$	$t = 1072,99$	$t = 1061,12$	$t = 521.26$	$t = 509.78$	$t = 47.32$
		H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject	H_o = reject

upon the experimental results, it can be concluded that FA_2 can be considered as an effective optimization algorithm on both static and dynamic environments.

5. Conclusions

This paper presents a comparative analysis of the performances of GA, DE and FA on both static and dynamic multidimensional knapsack problems. To the best of our knowledge this paper constitutes the first paper on the combinatorial dynamic optimization of FA.

One of the important contributions of the study is the development of FA_2, which is designed for a more realistic reflection of the behaviors of fireflies. FA_2 requires less computational time; moreover it achieves significantly superior results on both static and dynamic multidimensional knapsack problems. Particularly on stationary environments, FA_2 obtains near optimal results with a significantly faster convergence capability. Thus, it can be said that FA_2 was found to be more effective compared to GA, DE and FA for the problems studied in this paper.

Because it's a relatively new metaheuristic algorithm, the performance of pure FA along with the performances of pure GA and DE is also presented through this paper. In other words, any enhancement procedure is avoided as far as possible for a fair analysis. However, a research for the effects of such procedures might be scheduled as a future work. Additionally, the performance of FA_2 might be analyzed through different dynamic problems. Particularly, continuous dynamic optimization where moving peaks exist might be an appropriate candidate for such an analysis.

As another future work, a comparative study with the sate-of-the-art algorithms such as jDE, SaDE, "HyperMutation GA" and "GA with Random Immigrants" which were designed particularly for dynamic optimization problems can be carried out.

References

Abshouri, A., Meybodi, M., & Bakhtiary, A. (2011). New firefly algorithm based on multi swarm & learning automata in dynamic environments. In *IEEE proceedings* (pp. 73–77).

Baykasoğlu, A., & Durmuşoğlu, Z. D. U. (2011). Dynamic optimization in a dynamic and unpredictable world. In *2011 Proceedings of PICMET '11: Technology management in the energy-smart world (PICMET), July 31–August 4, 2011* (pp. 2312–2319). Portland, OR, USA: Hilton Portland and Executive Tower.

Baykasoğlu, A., & Durmuşoğlu, Z. D. U. (2013). A classification scheme for agent based approaches to dynamic optimization. *Artificial Intelligence Review*. http://dx.doi.org/10.1007/s10462-011-9307-x.

Bosman, P. A. N. (2005). Learning, anticipation and time-deception in evolutionary online dynamic optimization. In *Proceedings of the 2005 workshops on genetic and evolutionary computation* (pp. 39–47).

Bosman, P. A. N., & Poutrè, H. L. (2007). Learning and anticipation in online dynamic optimization with evolutionary algorithms: The stochastic case. In *Proceedings of the 2007 genetic and evolutionary computation conference* (pp. 1165–1172).

Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the congress on evolutionary computation 99* (Vol. 3, pp. 1–8).

Branke, J. (2002). *Evolutionary optimization in dynamic environments*. Dordrecht: Kluwer.

Branke, J., Kau, T., Schmidt, C., & Schmeck, H. (2000). A multi-population approach to dynamic optimization problems. In I. E. Parmee (Ed.), *Evolutionary design and manufacture* (pp. 299–308). London: Springer-Verlag.

Branke, J., Orbayı, M., & Uyar, Ş. (2006). The role of representations in dynamic knapsack problems. In 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, & H. Takagi (Eds.), *Applications of evolutionary computing* (pp. 764–775). Berlin Heidelberg: Springer Verlag.

Brest, J., Korošec, P., Šilc, J., Zamuda, A., Bošković, B., & Maučec, M. S. (2013). Differential evolution and differential ant-stigmergy on dynamic optimisation problems. *International Journal of Systems Science, 44*(4), 663–679.

Brest, J., Greiner, S., Bošković, B., Mernik, M., & Zumer, V. (2006). Self adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation, 10*, 646–657.

Chai-ead, N., Aungkulanon, P., & Luangpaiboon, P. (2011). Bees and firefly algorithms for noisy non-linear optimization problems. In *Proceedings of the international multi conference of engineering and computer scientists* (Vol. 2, pp. 1449–1454).

Chitra, C., & Subbaraj, P. (2012). A nondominated sorting genetic algorithm solution for shortest path routing problem in computer networks. *Expert Systems with Applications, 39*, 1518–1525.

Cobb, H. G. (1990). An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time dependant nonstationary environments. Technical Report, Naval Research Laboratory, WA, USA.

Das, S., & Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation, 15*, 4–31.

Dean, B. C., Goemans, M. X., & Vondrák, J. (2008). Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research, 33*(4), 945–964.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

Farahani, S., Nasiri, B., & Meybodi, M. (2011). A multi swarm based firefly algorithm in dynamic environments. In *The third international conference on signal processing systems (ICSPS2011)* (Vol. 3, pp. 68–72).

Farina, M., Deb, K., & Amato, P. (2004). Dynamic multiobjective optimization problems: Test cases, approximations, and applications. *IEEE Transactions on Evolutionary Computation, 8*(5), 425–442.

Fister, I., Fister, I., Jr., Yang, X.-S., & Brest, J. (2013a). A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation, 13*, 34–46.

Fister, I., Yang, X.-S., Brest, J., & Fister, I. Jr., (2013b). Modified firefly algorithm using quaternion representation. *Expert System with Applications, 40*, 7220–7230.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association, 32*, 675–701.

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics, 11*, 86–92.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, Massachusetts: Addison-Wesley.

Goldberg, D. E., & Smith, R. E. (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette (Ed.), *Second international conference on genetic algorithms* (pp. 59–68). New Jersey: Lawrence Erlbaum Associates.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In R. Manner & B. Manderick (Eds.). *Proceedings of the second international conference on parallel problem solving from nature* (Vol. 2, pp. 137–144). Amsterdam: Elsevier.

Hadad, B. S., & Eick, C. F. (1997). Supporting polyploidy in genetic algorithms using dominance vectors. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, & R. Eberhart (Eds.), *Evolutionary programming VI* (pp. 223–234). Berlin: Springer.

Hartman, J. C., & Perry, T. C. (2006). Approximating the solution of a dynamic, stochastic multiple knapsack problem. *Control and Cybernetics, 35*(3), 535–550.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Michigan Ann Arbor: University of Michigan Press.

Jin, Y., & Branke, J. (2005). Evolutionary optimization in uncertain environments – A survey. *IEEE Transactions on Evolutionary Computation, 9*(3), 303–317.

Karaman, A., & Uyar, A. S. E. (2004). A novel change severity detection mechanism for the dynamic 0/1 knapsack problem. In *Proceedings of mendel 2004: 10th international conference on soft computing*. Zakopane, Poland, June 7–11.

Karaman, A., Uyar, Ş., & Eryiğit, G. (2005). The memory indexing evolutionary algorithm for dynamic environments. In F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, & Y. Jin (Eds.), *Applications of evolutionary computing* (pp. 563–573). Berlin Heidelberg: Springer Verlag.

Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., & Sviridenko, M., et al. (2006). Dynamic placement for clustered web applications. In *WWW2006*. Edinburgh, UK.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer.

Kimbrel, T., Steinder, M., Sviridenko, M., & Tantawi, A. (2005). Dynamic application placement under service and memory constraints. In S. E. Nikoletseas (Ed.), *Experimental and efficient algorithms* (pp. 391–402). Heidelberg: Springer Verlag.

Kleywegt, A. J., & Papastavrou, J. D. (1998). The dynamic and stochastic knapsack problem. *Operations Research, 46*(1), 17–35.

Kleywegt, A. J., & Papastavrou, J. D. (2001). The dynamic and stochastic knapsack problem with random sized items. *Operations Research, 49*(1), 26–41.

Li, C., & Yang, S. (2008). A generalized approach to construct benchmark problems for dynamic optimization. In X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. C. Tan, J. Branke, & Y. Shi (Eds.), *Simulated evolution and learning* (pp. 391–400). Heidelberg Berlin: Springer Verlag.

Liekens, A. M. L. (2005). *Evolution of finite populations in dynamic environments*. Ph.D. thesis, Technische Universitat, Eindhoven.

Lim, T. Y. (2012). Structured population genetic algorithms: A literature survey. *Artificial Intelligence Review*. http://dx.doi.org/10.1007/s10462-012-9314-6 (http://link.springer.com/article/10.1007%2Fs10462-012-9314-6).

Lotfi, M. M., & Tavakkoli-Moghaddam, R. (2012). A genetic algorithm using priority-based encoding with new operators for fixed charge transportation problems. *Applied Soft Computing, 13*, 2711–2726.

Nasiri, B., & Meybodi, M. (2012). Speciation based firefly algorithm for optimization in dynamic environments. *International Journal of Artificial Intelligence, 8*, 118–132.

Morrison, R. W. (2004). *Designing evolutionary algorithms for dynamic environments*. Berlin: Springer.
Qin, A. K., Huang, V. L., & Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 13*, 398–417.
Rahmaniani, R., & Ghaderi, A. (2013). A combined facility location and network design problem with multi-type of capacitated links. *Applied Mathematical Modelling, 37*, 6400–6414.
Rohlfshagen, P., Lehre, P. K., & Yao, X. (2009). Dynamic evolutionary optimisation: An analysis of frequency and magnitude of change. In *Proceedings of the 2009 genetic and evolutionary computation conference, July 8–12* (pp. 1713–172). Montréal Québec, Canada.
Rohlfshagen, P., & Yao, X. (2008). Attributes of dynamic combinatorial optimization. *Lecture Notes in Computer Science, 5361*, 442–451.
Rohlfshagen, P., & Yao, X. (2011). Dynamic combinatorial optimisation problems: An analysis of the subset sum problem. *Soft Computing, 15*, 1723–1734.
Rossi, C., Barrientos, A., & del Cerro, J. (2007). Two adaptive mutation operators for optima tracking in dynamic optimization problems with evolution strategies. In *Proceedings of the 2007 annual conference on genetic and evolutionary computation, July 7–11* (pp. 697–704). London, England, United Kingdom.
Storn, R., & Price, K. (1995). Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, March 1995 (Available via ftp from ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.ps.Z).
Thengade, A., & Dondal, R. (2012). Genetic algorithm – Survey paper. *IJCA proceedings on national conference on recent trends in computing* (Vol. 5, pp. 25–29). NY, USA: Published by Foundation of Computer Science.
Thierens, D. (2002). Adaptive mutation rate control schemes in genetic algorithms. *Evolutionary Computation, 1*, 980–985.
Ursem, R. K., Krink, T., Jensen, M. T., & Michalewicz, Z. (2002). Analysis and modeling of control tasks in dynamic systems. *IEEE Transactions on Evolutionary Computation, 6*(4), 378–389.
van Hemert, J. I., van Hoyweghen, C., Lukschandl, E., & Verbeeck, K. (2001). A "futurist" approach to dynamic environments. In J. Branke & T. Bäck (Eds.), *Proceedings of the workshop on evolutionary algorithms for dynamic optimization problems at the genetic and evolutionary computation conference* (pp. 35–38).
Weicker, K. (2003). *Evolutionary algorithms and dynamic optimization problems*. Der Andere: Verlag.
Wilke, C. O. (1999). *Evolutionary dynamics in time-dependent environments*. Ph.D. thesis, Ruhr-Universität, Bochum.
Yang, S. (2003). Non-stationary problem optimization using the primaldual genetic algorithms. *Evolutionary Computation, 3*, 2246–2253.
Yang, S. (2005). Memory-based immigrants for genetic algorithms in dynamic environments. *Proceedings of the 2005 genetic and evolutionary computation conference* (Vol. 2, pp. 1115–1122). New York: ACM Press.
Yang, S. (2006). Associative memory scheme for genetic algorithms in dynamic environments. In F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, & C. Cotta (Eds.), *Applications of evolutionary computing* (pp. 788–799). Berlin, Heidelberg: Springer-Verlag.
Yang, S., Ong, Y. S., & Jin, Y. (Eds.), (2007). Evolutionary computation in dynamic and uncertain environments. Berlin: Springer. http://www.springer.com/engineering/computational+intelligence+and+complexity/book/978-3-540-49772-1.
Yang, X.-S. (2008). *Nature-inspired metaheuristic algorithm*. Frome, Somerset: Luniver Press.
Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. In Stochastic algorithms: Foundations and applications, SAGA 2009. Lecture notes in computer sciences (Vol. 5792, pp. 169–178).
Yang, X.-S., & He, X. (2013). Firefly algorithm: Recent advances and applications. *International Journal of Swarm Intelligence, 1*, 36–50.
Yazdani, D., Nasiri, B., Sepas-Moghaddam, A., & Meybodi, M. R. (2013). A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. *Applied Soft Computing, 13*, 2144–2158.

Web reference

(http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/mknapcb4.txt) Last access: 02.10.2013.