On the Performance of Baseline Evolutionary Algorithms on the Dynamic Knapsack Problem

Vahid Roostapour, Aneta Neumann, and Frank Neumann

Optimisation and Logistics, School of Computer Science, The University of Adelaide, Adelaide, Australia

Abstract. Evolutionary algorithms are bio-inspired algorithms that can easily adapt to changing environments. In this paper, we study single- and multi-objective baseline evolutionary algorithms for the classical knapsack problem where the capacity of the knapsack varies over time. We establish different benchmark scenarios where the capacity changes every τ iterations according to a uniform or Normal distribution. Our experimental investigations analyze the behavior of our algorithms in terms of the magnitude of changes determined by parameters of the chosen distribution, the frequency determined by τ and the class of knapsack instance under consideration. Our results show that the multi-objective approaches using a population that caters for dynamic changes have a clear advantage on many benchmarks scenarios when the frequency of changes is not too high.

1 Introduction

Evolutionary algorithms [1] have been widely applied to a wide range of combinatorial optimization problems. They often provide good solutions to complex problems without a large design effort. Furthermore, evolutionary algorithms and other bio-inspired computing have been widely applied to dynamic and stochastic problems [2,3] as they have the ability to easily adapt to changing environments.

Most studies for dynamic problems so far focus on dynamic fitness functions [4]. However, in real-world applications the optimization goal such as maximizing profit or minimizing costs often does not change. Instead of this, resources to achieve these goal change over time and influence to quality of solutions that can be obtained. In the context of continuous optimization, dynamically changing constraints have been investigated in [2][5]. Theoretical investigations for combinatorial optimization problems with dynamically changing constraints have recently been carried out [6,7] and the goal of this paper is to contribute to this research direction from an experimental perspective.

In this paper, we investigate evolutionary algorithms for the knapsack problem where the capacity of the knapsack changes dynamically. We design a benchmark set for the dynamic knapsack problem. This benchmark set builds on classical static knapsack instances and varies the constraint bound over time. The change in the constraint bound is done randomly every τ iterations where τ is a parameter determining the frequency of changes. The magnitude of a change is either chosen according to a uniform distribution in an interval $[-r, r]$ where r determines the magnitude of changes. Furthermore, we examine changes according to the normal distribution $\mathcal{N}(0, \sigma^2)$ with mean 0 and standard deviation σ. Here σ is used to determine the magnitude range of changes and large values of σ make larger changes more likely. We investigate different approaches analyzed theoretically with respect to their runtime behavior in [7]. The algorithms that we consider are a classical (1+1) EA and multi-objective approaches that are able to store infeasible solutions as part of the population. Furthermore, the range of feasible and infeasible solutions stored in the multi-objective algorithms can be set based on the anticipated change of the constraint bound.

In our experimental investigations, we start by examining the knapsack problem where all weights are set to 1 and vary the constraint bound. This matches the setting of the optimization of a linear function with a dynamic uniform constraint analyzed in [7]. Our experimental results match the theoretical ones obtained in this paper and show that the multi-objective approaches using a population to cater for dynamic changes significantly reduce the offline error that occurred during the run of the algorithms. For the general setting we investigate different classes of knapsack problem such as uniformly chosen weights and profits and bounded strongly correlated instances, we examine the behaviour of the algorithms in dependence of the frequency and magnitude of changes. Our results show that the (1+1) EA has an advantage of the multi-objective algorithms when the frequency of changes is high. In this case the population of the multi-objective approaches is too slow to adapt to the changes that occur. On the other hand, a lower frequency of changes plays in favor of the multi-objective approaches when the weights and profits are not correlated to make the instances particularly difficult to solve.

The outline of the paper is as follows. Section 2 introduces the problem definition and three algorithms we studied. The dynamic knapsack problem and experimental setting is presented in Section 3. In Section 4 we analyze the experimental results in detail followed by a conclusion in Section 5

2 Preliminaries

In this section, we define the Knapsack Problem (KP) and further notations using in the rest of the paper. We present (1+1) EA and two multi-objective algorithms called MOEA and MOEA_D that are considered in this paper.

2.1 Problem Definition

We investigate the performance of different evolutionary algorithms on the knapsack problem under dynamic constraint. There are n items with profits $\{p_1, \cdots, p_n\}$ and weights $\{w_1, \cdots, w_n\}$. A solution x is a bit string of $\{0,1\}^n$ which has the weight $W(x) = \sum_{i=1}^{n} w_i x_i$ and the profit $P(x) = \sum_{i=1}^{n} p_i x_i$. We denote the maximum profit among the items by p_{max}. The constraint is the weight capacity of

Algorithm 1: (1+1) EA

```
1 $x \leftarrow$ previous best solution;
2 while stopping criterion not met do
3     $y \leftarrow$ flip each bit of $x$ independently with probability of $\frac{1}{n}$;
4     if $f_{1+1}(y) \geq f_{1+1}(x)$ then
5         $x \leftarrow y$;
```

the knapsack and denoted by C. The goal is to find a set of items with the total weight less than or equal to C and the maximum profit:

$$\max_{x\in\{0,1\}^n} \sum_{i=1}^{n} p_i x_i$$
$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \leq C.$$

We consider two types of the this problem according to the weights. Firstly, we assume that all the weights are one and we have uniform dynamic constraint. In this case, the limitation is on the number of items chosen by a solution and the optimal solution is to pick C number of items with the highest profits. Next we consider the general case where the profits and weights are linear integers under linear constraint on the weight.

2.2 Algorithms

We investigate the performance of three algorithms in this paper. The initial solution for all these algorithms is a solution with items chosen uniformly at random. After a dynamic change happens, the algorithms update the solution(s) and start the optimization process with the new capacity. This update is to deal with the fact that after a dynamic change, current solutions may become infeasible or it get as far as the new capacity that isn't worth to be kept anymore. (1+1) EA (Algorithm 1) flips each bit of the current solution with the probability of $\frac{1}{n}$ as the mutation step. Afterward, it picks the solution with higher fitness function between the original solution and the mutated one. The fitness function that we used in (1+1) EA is as follows:

$$f_{1+1}(x) = \sum_{i=1}^{n} p(i) \cdot x(i) - (n \cdot p_{max} + 1) \cdot v(x)$$

where $v(x) = \max\{0, (\sum_{i=1}^{n} w(i) \cdot x(i)) - C\}$ is the constraint violation of x. If x is a feasible solution, then $W(x) < C$ and $v(x) = 0$. Otherwise, $v(x)$ is the weight distance of $W(x)$ from C. The algorithm aims to maximize f_{1+1} which consists of two terms. The first term is the total profit of the chosen items and

Algorithm 2: MOEA

```
1  Update $C$;
2  $S^+ \leftarrow \{z \in S^+ \cup S^- | C < w(z) \leq C + \delta\}$;
3  $S^- \leftarrow \{z \in S^+ \cup S^- | C - \delta \leq w(z) \leq C\}$;
4  if $S^+ \cup S^- = \emptyset$ then
5      $q \leftarrow$ best previous solution;
6  if $C < w(q) \leq C + \delta$ then
7      $S^+ \leftarrow \{q\} \cup S^+$;
8  else if $C - \delta \leq w(q) \leq C$ then
9      $S^- \leftarrow \{q\} \cup S^-$;
10 while a change happens do
11     if $(S^+ \cup S^- = \emptyset)$ then
12         Initialize $S^+$ and $S^-$ by Repair($q$,$\delta$,$C$);
13     else
14         choose $x \in S^+ \cup S^-$ uniformly at random;
15         $y \leftarrow$ flip each bit of $x$ independently with probability $\frac{1}{n}$;
16         if $(C < w(y) \leq C + \delta) \wedge (\nexists w \in S^+ : w \succcurlyeq_{MOEA} y)$ then
17             $S^+ \leftarrow (S^+ \cup \{y\}) \setminus \{z \in S^+ | y \succ z\}$;
18         if $(C - \delta \leq w(y) \leq C) \wedge (\nexists w \in S^- : w \succcurlyeq_{MOEA} y)$ then
19             $S^- \leftarrow (S^- \cup \{y\}) \setminus \{z \in S^- | y \succ z\}$;
```

the second term is the applied penalty on the infeasible solutions. The amount of penalty guarantees that a feasible solution always dominates an infeasible solution. Moreover, between two infeasible solutions, the one with closer weight to C dominates the other one.

The other algorithm we consider in this paper is a multi-objective evolutionary algorithm (Algorithm 2) which is inspired from the a theoretical study on the performance of evolutionary algorithms in reoptimization under dynamic uniform constrain[7]. Each solution x in the objective space is a two-dimensional point $f_{MOEA}(x) = (w(x), p(x))$. We say solution y dominates solution x w.r.t. f_{MOEA}, denoted by $y \succcurlyeq_{MOEA} x$, if $|y|_1 = |x|_1 \wedge f_{(1+1)}(y) \geq f_{(1+1)}(x)$.

According to the definition of $\succcurlyeq_{MOEA}$, two solutions are comparable only if they have the same weight. Note that if x and y are infeasible and comparable, then the one with higher profit dominates the other one. MOEA uses a parameter denoted by δ which determines the number of individuals stored around the current constraint bound. For any weight in $[C - \delta, C + \delta]$, MOEA keeps a solution. It helps the algorithm to prepare for the dynamic change by storing even infeasible solutions which may become feasible after the next change. However, a large δ causes a large number of solutions to be kept that decreases the probability of choosing each one. Since the algorithm chooses only one solution to mutate in each iteration, this affects the performance of MOEA in finding the optimal solution. After each dynamic change, MOEA updates the sets of solutions. If the value of a change be in a range that all the previous solutions

Algorithm 3: Repair

```
input  : Initial solution $q$, $\delta$, $C$
output: $S^+$ and $S^-$ such that $|S^+ \cup S^-| = 1$
1 while $|S^+ \cup S^-| = 0$ do
2     $y \leftarrow$ flip each bit of $q$ independently with probability of $\frac{1}{n}$;
3     if $f_{1+1}(y) \geq f_{1+1}(q)$ then
4         $q \leftarrow y$;
5         if $C < w(q) \leq C + \delta$ then
6             $S^+ \leftarrow \{q\} \cup S^+$;
7         else if $C - \delta \leq w(q) \leq C$ then
8             $S^- \leftarrow \{q\} \cup S^-$;
```

become out of $[C - \delta, C + \delta]$, then the algorithm initializes a single solution with the previous best solution and uses Repair function (Algorithm 3), which behaves similar to (1+1) EA, until find a solution with distance δ from C.

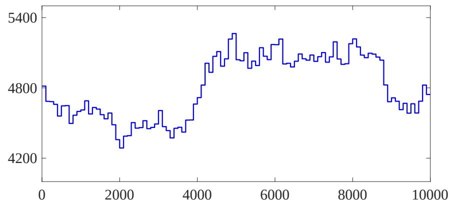
To handle the slow improvement of MOEA caused by a large δ, we defined a new dominance procedure. We say solution y dominates solution x, denoted by $\succcurlyeq_{MOEA\_D}$, if $w(y) \leq w(x) \wedge p(y) \geq p(x)$. This new algorithm called MOEA_D is obtained by replacing lines 14-19 of Algorithm 2 with Algorithm 4. It should be noticed that if y is an infeasible solution then it is only compared with other infeasible solutions and it is the same when y is feasible solution. The new definition of dominance only keeps the solutions which are not dominated by other solutions. Hence, not only MOEA_D keeps fewer solutions than MOEA, but also the quality of kept solutions are higher, since they are non-dominated solutions among all other solutions.

Algorithm 4: MOEA_D (Dominance and Selection)

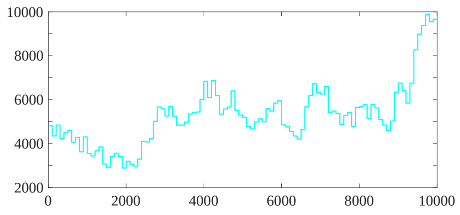
```
14 choose $x \in S^+ \cup S^-$ uniformly at random;
15 $y \leftarrow$ flip each bit of $x$ independently with probability $\frac{1}{n}$;
16 if $(C < w(y) \leq C + \delta) \wedge (\nexists w \in S^+ : w \succcurlyeq_{MOEA\_D} y)$ then
17     $S^+ \leftarrow (S^+ \cup \{y\}) \setminus \{z \in S^+ | y \succ z\}$;
18 if $(C - \delta \leq w(y) \leq C) \wedge (\nexists w \in S^- : w \succcurlyeq_{MOEA\_D} y)$ then
19     $S^- \leftarrow (S^- \cup \{y\}) \setminus \{z \in S^- | y \succ z\}$;
```

3 Benchmarking for the Dynamic Knapsack Problem

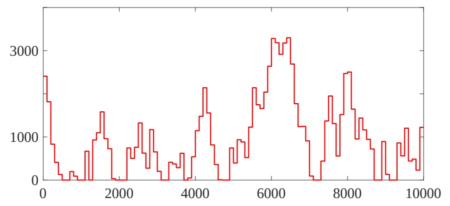
In the following section, the dynamic version of KP used for the experiments is described and we explain how the dynamic changes occur during the optimization process. Moreover, the dynamic benchmarks and the experimental settings are presented.



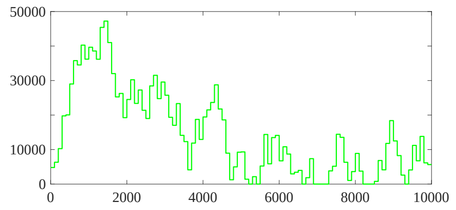
(a) $\mathcal{N}(0, 100^2)$.



(b) $\mathcal{N}(0, 500^2)$.



(c) $\mathcal{U}(-2000, 2000)$.



(d) $\mathcal{U}(-10000, 10000)$.

Fig. 1: Examples for constraint bound C over 10000 generations with $\tau = 100$ using uniform and normal distributions. Initial value $C = 4815$

3.1 The Dynamic Knapsack Problem

In the dynamic version of KP considered in this paper, the capacity dynamically changes during the optimization process according to a preset factor denoted by τ. A change happens every τ generation i.e. the algorithm has τ generations to find the optimum of the current capacity and to prepare for the next change. In the case of uniformly random alterations, the capacity of next interval is achieved by adding a uniformly random value in $[-r, r]$ to C. Moreover, we consider another case in which the amount of changes are chosen under the Gaussian distribution $\mathcal{N}(0, \sigma^2)$. Figure 1 illustrates how dynamic changes from different distributions affect the capacity.Note that the scale of the subfigures are not the same. For example the total change after 100 dynamic changes under $\mathcal{N}(0, 100^2)$ is less than 1000 (Figure 1a) while the capacity reached almost 45000 with dynamic changes under $\mathcal{U}(-10000, 10000)$. This depicts that there are different type of challenges according to the dynamic changes that the algorithms must deal with. The combination of different distributions and frequencies brings interesting challenges for the algorithms. In an environment where the constraint changes with a high frequency, the algorithms has less time to find the optimal solutions. Hence, it is more probable that an algorithm which tries to improve only one solution, perform better than another algorithm that needs to optimize among a bunch of solutions. Furthermore, the uniform distribution guarantees upper and lower bounds on amount of changes. This property could be beneficial for the algorithms which keep a number of solutions in each generation, to get ready and behave faster after a dynamic change. However, if the changes happen under a Normal distribution, there is no strict bound on the value of a change which means it is not easy to predict which algorithms perform better in this type of environments.

3.2 Benchmark and Experimental Setting

In this experiment we used benchmarks eli101, which originally generated for Traveling Thief Problem [8], by ignoring the cities and only using the items. The weights and profits are generated in three different classes. In Uncorrelated (uncorr) instances, the weights and profits are integers chosen uniformly at random within $[1, 1000]$. Uncorrelated Similar Weights (unc-s-w) have uniformly distributed random integers as the weights and profits within $[1000, 1010]$ and $[1, 1000]$,respectively. Finally, we have Bounded Strongly Correlated (bou-s-c) which is the hardest instance and comes from the bounded knapsack problem. The weights of this instance are chosen uniformly at random within $[1, 1000]$ and the profits are set according to the weights within the weights plus 100. In addition, in section 4.1, where the weights are one, we change the weights to one and considered the profits as they are in the benchmarks. The initial capacity in this version is calculated by dividing the original capacity by the average of the original profits. Dynamic changes add a value to C each τ generations. Four different situations in terms of frequencies are considered: high frequent changes with $\tau = 100$, medium frequent changes with $\tau = 1000, 5000$ and low frequent changes with $\tau = 15000$.

In the case that weights are 1, the value of dynamic changes are chosen uniformly at random within the interval $[-r, r]$ where are $r = 1, 10$. In the case of general weights, when changes are uniformly random, we investigate two values for r: $r = 2000, 10000$. Moreover, the situation that changes are from normal distribution is also experimented with $\sigma = 100, 500$.

We used the offline errors to compute the performance of the algorithms. In each generation, we record $e_i = P(x_i^*) - P(x_i)$ where x_i^* and x_i are optimal solution and best achieved feasible solution in generation i, respectively. If the best achieved solution is infeasible, then we have $e_i = C - \sum_{i=1}^{n} w(i) \cdot x(i)$ which is negative. The final error for m generations would be $\sum_{i=1}^{m} e_i/m$.

The benchmarks for dynamic changes are thirty different files. In randomly uniform changes each file consists of 100000 numbers in $[-r, r]$ generated uniformly at random. Similarly, there are thirty files with 100000 numbers generated under the normal distribution $\mathcal{N}(0, \sigma^2)$. Algorithms start from the beginning of each file and picks the amount of changes from the files. Hence, for each setting, we run algorithms for thirty times with different dynamic benchmark and record the total offline error of each run.

In order to establish a statistical comparison of the results among different algorithms we use multiple comparisons test. In particularity, we focus on the method that compare against a set of algorithms. For statistical validation we use the Kruskal-Wallis test with 95% confidence. Afterwards, we apply the Bonferroni post-hoc statistical procedures that are used for multiple comparison of a control algorithm to two or more algorithms. For more detailed description on the statistical tests we refer the reader to [9].

Our results are summarized in the Tables 1, 2 and 3. The columns represent the algorithm (1+1) EA, MOEA, MOEA_D with the corresponding mean value and standard deviation. Note $X^{(+)}$ is equivalent to the statement that algorithm in the column outperformed algorithm X, and $X^{(-)}$ is equivalent to the statement that X outperformed the algorithm given in the column. In the case if the algorithm X not appears this means that no significant difference was determined between algorithms.

4 Experimental Results

In this section we describe the initial setting of the algorithms and analyze their performance according to the statistical tests. The initial solution for all the algorithms is a pack of items which are chosen uniformly at random. Each algorithm runs initially for 10000 generations without any dynamic change. Afterward, the first change happens and algorithms perform one million generations with dynamic changes in every τ generations. In multi-objective algorithms it is needed to initially set δ. These algorithms keep at most δ feasible solutions and δ infeasible solutions which might help them to perform efficiently in dealing with a dynamic change. When the dynamic changes come from $\mathcal{U}(-r, r)$, it is known that the capacity will change at most r. Hence, we set $\delta = r$. In case of changes from $\mathcal{N}(0, \sigma^2)$, δ is set to be 2σ since 95% of values will be in the distance of 2σ from the mean value. Note that larger δ increases the population size of the algorithms and there is a trade-off between the size of population and speed of algorithm in reacting to the next change.

4.1 Dynamic Uniform Constraint

In this section we validate the theoretical results on the performance of (1+1) EA and Multi-Objective Evolutionary Algorithm. Shi et al. state that the multi-objective approach perform better than (1+1) EA in reoptimizing the optimal solution of dynamic KP under uniform constraint [7]. Although MOEA that we used in this experiment is not completely similar to the multi-objective algorithm studied previously and they only considered the reoptimization time, but the experiments show that multi-objective approaches outperform (1+1) EA in case of uniform constraints.

An important reason for this clear remarkable performance is the relation of optimal solutions in different weights. In this type of KP, the difference between the optimal solution of weight w and $w + 1$ is one item. Hence, keeping non-dominated solutions near the constrained bound helps the algorithm to find the current optimal more efficient and behave faster after a dynamic change.

Moreover, according to the results, there is no significance in using MOEA and MOEA_D in this type of KP. A probable reason is that the size of population in MOEA remains small when weights are one. Hence, MOEA_D, which stores

Table 1: Mean, standard deviation values and statistical tests in regards to the offline error for (1+1) EA, MOEA, MOEA_D based on uniform distribution with all the weights are one shown in each column.

	n	r	τ	(1+1) EA (1)			MOEA (2)			MOEA_D (3)		
				mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	5	100	4889.3888	144.4195	$2^{(-)}, 3^{(-)}$	1529.9976	120.7599	$1^{(+)}$	1486.8485	122.9978	$1^{(+)}$
	100	5	1000	1194.2316	86.5164	$2^{(-)}, 3^{(-)}$	44.7524	8.9564	$1^{(+)}$	46.6889	8.5070	$1^{(+)}$
unc-s-w	100	5	100	4990.8004	144.8678	$2^{(-)}, 3^{(-)}$	1545.3579	115.1507	$1^{(+)}$	1500.0710	106.7000	$1^{(+)}$
	100	5	1000	1160.2355	130.3224	$2^{(-)}, 3^{(-)}$	41.9020	6.1326	$1^{(+)}$	43.0584	7.2187	$1^{(+)}$
bou-s-c	100	5	100	13021.9789	780.7601	$2^{(-)}, 3^{(-)}$	4258.5330	580.7689	$1^{(+)}$	4190.5479	573.1300	$1^{(+)}$
	100	5	1000	3874.7641	911.5027	$2^{(-)}, 3^{(-)}$	177.6196	83.1647	$1^{(+)}$	175.1415	80.7294	$1^{(+)}$

fewer items because of its dominance definition, has no advantage in this manner anymore. In addition, constraint is actually on the number of the items. Thus, both definitions for dominance work the same in many cases.

4.2 Dynamic Linear Constraint

In this section, we consider algorithms in more difficult environments where weights are arbitrary under dynamic linear constraint. As it is shown in Section 4.1, the multi-objective approaches outperform (1+1) EA in case of weights are one. Now we try to answer to this question: Does the similar fact holds when weights are arbitrary? The data in Table 2 shows the experimental results in case of dynamic linear constraints under a uniform distribution. It can be seen that (as expected) the mean of errors decreases by increasing τ. Larger τ gives more time to the algorithm to get closer to the optimal. Moreover, starting from a solution which is near to the optimal of the previous capacity, might help the process of finding the new optimal solution in many cases.

We first consider the results in case of dynamic changes under uniform distribution. The interesting observation from Table 2 is that unlike the case of uniform constraint, in almost all the settings, MOEA had the worst performance among all the algorithms. The first reason might be that items selected in optimal solutions with close weights are close to each other in terms of Hamming distance. In other words, when weights are one, we can achieve the optimal of weight w by adding an item to the optimal of weight $w + 1$ or deleting an item from the optimal of $w - 1$. However, in case of arbitrary weights, the optimal solutions of weight w and $w + d$ could have completely different items, even if d is small. The other reason could be the effect of having a large population. A large population may cause the optimization process to be longer and it could get worst according to the definition of $\succcurlyeq_{MOEA}$ which only compares solutions with equal weights. If s be a new solution and there is no solution with $w(s)$ in the set of solutions, MOEA keeps s whether s is a good solution or not i.e. is it really a non-dominated solution or it would be dominated by other solutions in the set? This comparison doesn't consider if s has any good properties to be inherited to the next generation. Moreover, putting s in the set of solutions decreases the probability of choosing all other solutions, even those solutions that

Table 2: Mean, standard deviation values and statistical tests in regards to the offline error for (1+1) EA, MOEA, MOEA_D based on uniform distribution.

	n	r	τ	(1+1) EA (1)			MOEA (2)			MOEA_D (3)		
				mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	2000	100	5564.3676	463.3906	$2^{(+)},3^{(-)}$	11386.3961	769.7717	$1^{(-)},3^{(-)}$	3684.2584	525.4970	$1^{(+)},2^{(+)}$
	100	2000	1000	2365.5625	403.6389	$2^{(+)},3^{(-)}$	7219.1692	587.5044	$1^{(-)},3^{(-)}$	776.1434	334.6927	$1^{(+)},2^{(+)}$
	100	2000	5000	1415.4208	167.0792	$2^{(+)},3^{(-)}$	3598.2916	420.1200	$1^{(-)},3^{(-)}$	270.9025	121.4351	$1^{(+)},2^{(+)}$
	100	2000	15000	914.5508	102.8162	$2^{(+)},3^{(-)}$	2004.1563	368.8171	$1^{(-)},3^{(-)}$	88.8041	43.9791	$1^{(+)},2^{(+)}$
unc-s-w	100	2000	100	3128.4326	188.3569	$2^{(+)},3^{(-)}$	5911.1092	534.2422	$1^{(-)},3^{(-)}$	2106.4485	249.2828	$1^{(+)},2^{(+)}$
	100	2000	1000	606.1364	99.2263	$2^{(+)},3^{(-)}$	1564.2348	619.9709	$1^{(-)},3^{(-)}$	302.3365	24.5988	$1^{(+)},2^{(+)}$
	100	2000	5000	147.5547	31.7991	$3^{(-)}$	174.2289	95.9758	$3^{(-)}$	60.9423	9.12073	$1^{(+)},2^{(+)}$
	100	2000	15000	64.6549	17.1292	$2^{(-)},3^{(-)}$	40.6629	15.5060	$1^{(+)},3^{(-)}$	19.2600	4.0399	$1^{(+)},2^{(+)}$
bou-s-c	100	2000	100	3271.0735	266.5438	$2^{(+)}$	5583.5302	337.8083	$1^{(-)},3^{(-)}$	3036.9726	297.3351	$2^{(+)}$
	100	2000	1000	1483.0084	85.1373	$2^{(+)},3^{(-)}$	2639.1569	106.4706	$1^{(-)},3^{(-)}$	617.9195	186.3456	$1^{(+)},2^{(+)}$
	100	2000	5000	796.7658	89.8043	$2^{(+)},3^{(-)}$	1256.6245	118.2748	$1^{(-)},3^{(-)}$	251.4120	109.5822	$1^{(+)},2^{(+)}$
	100	2000	15000	538.4494	66.9822	$2^{(+)},3^{(-)}$	687.9506	116.9149	$1^{(-)},3^{(-)}$	104.2682	61.0609	$1^{(+)},2^{(+)}$
uncor	100	10000	100	10256.7191	210.5084	$2^{(+)},3^{(+)}$	16278.9718	248.4330	$1^{(-)},3^{(-)}$	11038.0747	236.9141	$1^{(-)},2^{(+)}$
	100	10000	1000	3604.1781	285.7346	$2^{(+)}$	13340.1982	704.3221	$1^{(-)},3^{(-)}$	3508.5130	473.4207	$2^{(+)}$
	100	10000	5000	1607.7785	278.6027	$2^{(+)},3^{(-)}$	10614.4482	1660.3176	$1^{(-)},3^{(-)}$	1183.5216	411.8305	$1^{(+)},2^{(+)}$
	100	10000	15000	987.6461	219.5262	$2^{(+)},3^{(-)}$	8006.3515	1612.2042	$1^{(-)},3^{(-)}$	566.6951	219.537	$1^{(+)},2^{(+)}$
unc-s-w	100	10000	100	7192.8170	153.9257	$2^{(+)},3^{(+)}$	12617.6912	318.2268	$1^{(-)},3^{(-)}$	8057.4438	274.1671	$1^{(-)},2^{(+)}$
	100	10000	1000	1846.4341	115.2344	$2^{(+)}$	6981.8105	768.7823	$1^{(-)},3^{(-)}$	1743.1162	364.3832	$2^{(+)}$
	100	10000	5000	539.3859	65.3926	$2^{(+)}$	3488.2769	819.5125	$1^{(-)},3^{(-)}$	519.6258	175.2236	$2^{(+)}$
	100	10000	15000	208.7307	36.9058	$2^{(+)}$	1525.2297	306.7184	$1^{(-)},3^{(-)}$	201.9746	79.2794	$2^{(+)}$
bou-s-c	100	10000	100	7187.8013	122.5926	$2^{(+)},3^{(+)}$	15111.3818	231.5328	$1^{(-)},3^{(-)}$	12736.5459	229.4770	$1^{(-)},2^{(+)}$
	100	10000	1000	2282.8064	219.2362	$2^{(+)},3^{(+)}$	8301.4260	569.9000	$1^{(-)},3^{(-)}$	3575.2567	550.5409	$1^{(-)},2^{(+)}$
	100	10000	5000	1370.4793	250.5933	$2^{(+)}$	5248.4036	1045.7825	$1^{(-)},3^{(-)}$	1472.1939	493.8786	$2^{(+)}$
	100	10000	15000	955.3767	133.3276	$2^{(+)}$	3852.0701	752.8415	$1^{(-)},3^{(-)}$	977.4152	397.7484	$2^{(+)}$

are very close to the optimal solutions. However, as it can be seen in the Table 2, there is only one case that MOEA beat the (1+1) EA: when the weights are similar, magnitude of changes are small (2000) which means the population size is small and finally τ is at its maximum to let the MOEA to use its large size population to optimize the problem.

Although MOEA does not perform very well in instances with general weights, multi-objective approach with a better defined dominance, MOEA_D, does outperform (1+1) EA in many cases. In the following, we compare the performance of (1+1) EA and MOEA_D. In the case that changes are smaller, it can be seen that the mean of offline errors of MOEA_D is smaller than (1+1) EA. The dominance in MOEA_D is defined such that keeps only the dominant solutions. When a new solution comes, it removes solutions that are dominated by the new solution and keeps the new solution only if it is not dominated by the other ones. This process improves the quality of the solutions i.e. increase the probability of keeping a solution with helpful properties for the next generations. Moreover, it reduces the size of the population significantly. However, large changes on the capacity makes the MOEA_D to keep more individuals and this is where (1+1) EA may overtake the MOEA_D.

When $r = 10000$, MOEA_D doesn't have significantly better results in all cases as the case of $r = 2000$ and in most of the situations, it performs as good as (1+1) EA. In all of high frequent conditions where $\tau = 100$, the (1+1) EA has better performance. It may caused by MOEA_D needing more time to optimize a population with larger size. Moreover, when the magnitude of changes is large,

Table 3: Mean, standard deviation values and statistical tests in regards to the offline error for (1+1) EA, MOEA, MOEA_D based on normal distribution.

	n	σ	τ	(1+1) EA (1)			MOEA (2)			MOEA_D (3)		
				mean	st	stat	mean	st	stat	mean	st	stat
uncor	100	100	100	2714.7237	106.0564	$2^{(+)},3^{(+)}$	9016.8293	2392.4802	$1^{(-)},3^{(-)}$	4271.0873	789.9369	$1^{(-)},2^{(+)}$
	100	100	1000	1386.6581	97.1127	$2^{(+)},3^{(-)}$	3714.8943	737.1095	$1^{(-)},3^{(-)}$	412.8864	27.2491	$1^{(+)},2^{(+)}$
	100	100	5000	801.5359	73.6676	$2^{(+)},3^{(-)}$	1266.3464	119.2522	$1^{(-)},3^{(-)}$	108.2852	14.21993	$1^{(+)},2^{(+)}$
	100	100	15000	549.7143	78.9796	$2^{(+)},3^{(-)}$	749.8565	148.0347	$1^{(-)},3^{(-)}$	61.9285	17.0310	$1^{(+)},2^{(+)}$
unc-s-w	100	100	100	412.2387	111.0699	$2^{(+)},3^{(+)}$	1979.6485	914.3548	$1^{(-)}$	1904.0895	877.5521	$1^{(-)}$
	100	100	1000	85.5528	23.1272	$2^{(+)},3^{(+)}$	1566.5388	409.3200	$1^{(-)}$	1482.3743	391.7501	$1^{(-)}$
	100	100	5000	36.9446	13.6068	$2^{(+)},3^{(+)}$	1414.6565	448.7852	$1^{(-)}$	1322.3538	414.2754	$1^{(-)}$
	100	100	15000	29.1357	19.6958	$2^{(+)},3^{(+)}$	1237.6667	665.2746	$1^{(-)}$	1137.7971	648.7264	$1^{(-)}$
bou-s-c	100	100	100	1491.3596	260.7184	$2^{(+)},3^{(+)}$	4625.4912	1302.5213	$1^{(-)},3^{(-)}$	2903.7694	717.9236	$1^{(-)},2^{(+)}$
	100	100	1000	736.1017	53.9918	$2^{(+)},3^{(-)}$	1748.6106	189.9375	$1^{(-)},3^{(-)}$	312.8776	35.5254	$1^{(+)},2^{(+)}$
	100	100	5000	446.9419	39.3630	$2^{(+)},3^{(-)}$	640.5960	91.2955	$1^{(-)},3^{(-)}$	101.2059	17.4725	$1^{(+)},2^{(+)}$
	100	100	15000	337.8493	40.4405	$2^{(+)},3^{(-)}$	469.1578	93.9935	$1^{(-)},3^{(-)}$	70.1564	22.2599	$1^{(+)},2^{(+)}$
uncor	100	500	100	13400.8775	305.1443	$2^{(+)},3^{(+)}$	46395.4430	4565.6075	$1^{(-)},3^{(-)}$	19218.9393	1035.7244	$1^{(-)},2^{(+)}$
	100	500	1000	6363.1594	194.5869	$2^{(+)},3^{(-)}$	25747.0777	1181.1069	$1^{(-)},3^{(-)}$	2387.6143	151.7328	$1^{(+)},2^{(+)}$
	100	500	5000	3983.0558	254.3827	$2^{(+)},3^{(-)}$	18004.0285	1243.6619	$1^{(-)},3^{(-)}$	1467.5810	152.7721	$1^{(+)},2^{(+)}$
	100	500	15000	3112.7330	315.2934	$2^{(+)},3^{(-)}$	17610.3480	1265.5016	$1^{(-)},3^{(-)}$	1348.2546	194.7150	$1^{(+)},2^{(+)}$
unc-s-w	100	500	100	2845.3063	146.8025	$2^{(+)},3^{(+)}$	11803.9860	1256.9881	$1^{(-)}$	11438.0432	1247.6191	$1^{(-)}$
	100	500	1000	595.7042	86.1936	$2^{(+)},3^{(+)}$	8851.3557	1488.5874	$1^{(-)}$	8478.2106	1313.5508	$1^{(-)}$
	100	500	5000	222.7947	62.2231	$2^{(+)},3^{(+)}$	7025.4493	2639.3898	$1^{(-)}$	6488.4662	2335.9254	$1^{(-)}$
	100	500	15000	171.3333	50.2800	$2^{(+)},3^{(+)}$	7188.6703	4184.8398	$1^{(-)}$	6278.0973	4146.5429	$1^{(-)}$
bou-s-c	100	500	100	7444.2316	289.9988	$2^{(+)},3^{(+)}$	24462.5803	1330.9350	$1^{(-)},3^{(-)}$	15592.6601	791.6970	$1^{(-)},2^{(+)}$
	100	500	1000	4062.6330	210.4886	$2^{(+)},3^{(-)}$	12291.6272	589.1772	$1^{(-)},3^{(-)}$	2781.2022	317.8854	$1^{(+)},2^{(+)}$
	100	500	5000	3013.3515	289.2948	$2^{(+)},3^{(-)}$	9667.9570	571.3437	$1^{(-)},3^{(-)}$	1971.5654	220.6273	$1^{(+)},2^{(+)}$
	100	500	15000	2722.2941	342.3943	$2^{(+)},3^{(-)}$	9308.2817	719.2505	$1^{(-)},3^{(-)}$	1760.5127	251.5120	$1^{(+)},2^{(+)}$

it is more probable that the new change, makes MOEA_D to remove all of the stored individuals and start from the scratch.

Now we study the experimental results came from considering the dynamic changes under the normal distribution (Table 3). The results confirms the better performance of (1+1) EA in high frequent changes. Skipping the instance with uncorrelated similar weights and high frequent changes, the MOEA_D has been always the best algorithm in terms of performance and MOEA has bee the worst. The most noticing behavior is in dealing with instance with uncorrelated similar weights. (1+1) EA outperforms both other algorithms in this instance. This happens because of the value of δ and the weights of the instances. δ is set to 2σ in multi-objective approaches and weights of items are integers in $[1001, 1010]$ in this type of instance. (1+1) EA is able to freely get closer to the optimal solutions from both directions, while multi-objective approaches are only allowed to consider solutions in range of $[C - \delta, C + \delta]$. In other words, it is probable that there is only one solution in that range or even no solution. Hence, multi-objective approaches have no advantage in these type of instances according to the value of δ and weights of the items.

5 Conclusions and Future Work

In this paper we studied the evolutionary algorithms for the knapsack problem where the capacity dynamically changes during the optimization process. In the introduced dynamic setting, the frequency of changes is determined by τ. The magnitude of changes are chosen randomly either under uniform distribution

$\mathcal{U}(-r, r)$ or under Gaussian distribution $\mathcal{N}(0, \sigma^2)$. We compared the performance of (1+1) EA and two multi-objective approaches with different dominance definitions (MOEA, MOEA_D). Our experiments in the case of weights set to 1 verified the previous theoretical studies for (1+1) EA and MOEA. It is shown that multi-objective approach, which use a population in the optimization, outperforms (1+1) EA. In addition, we considered the algorithms in case of general weights for different classes of instances with a variation of frequencies and magnitudes. Our results illustrate that MOEA doesn't perform well in general case due to its dominance procedure. However, MOEA_D which benefits from a population with controlled size and nod-dominated solutions, beats (1+1) EA in most case. On the other hand, in the environments with high frequent changes, (1+1) EA perform better than multi-objective approaches. In such cases, the population slows down MOEA_D in reacting after the dynamic change.

Acknowledgment

This work has been supported through Australian Research Council (ARC) grant DP160102401.

References

1. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. 2nd edn. Springer (2007)
2. Nguyen, T., Yao, X.: Continuous dynamic constrained optimization: The challenges. IEEE Transactions on Evolutionary Computation **16**(6) (2012) 769–786
3. Rakshit, P., Konar, A., Das, S.: Noisy evolutionary optimization algorithms - A comprehensive survey. Swarm and Evolutionary Computation **33** (2017) 18–45
4. Nguyen, T.T., Yang, S., Branke, J.: Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation **6** (2012) 1–24
5. Ameca-Alducin, M.Y., Hasani-Shoreh, M., Neumann, F.: On the use of repair methods in differential evolution for dynamic constrained optimization. In Sim, K., Kaufmann, P., eds.: Applications of Evolutionary Computation - 21st International Conference, EvoApplications 2018, Parma, Italy, April 4-6, 2018, Proceedings. Volume 10784 of Lecture Notes in Computer Science., Springer (2018) 832–847
6. Pourhassan, M., Gao, W., Neumann, F.: Maintaining 2-approximations for the dynamic vertex cover problem using evolutionary algorithms. In Silva, S., Esparcia-Alcázar, A.I., eds.: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, ACM (2015) 903–910
7. Shi, F., Schirneck, M., Friedrich, T., Kötzing, T., Neumann, F.: Reoptimization times of evolutionary algorithms on linear functions under dynamic uniform constraints. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM (2017) 1407–1414
8. Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: A comprehensive benchmark set and heuristics for the traveling thief problem. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM (2014) 477–484
9. Corder, G.W., Foreman, D.I.: Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach. Wiley (2009)