# 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
vahid.roostapour@adelaide.edu.au

**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 $\tau$ 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 $\tau$ 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 applied to many 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, resources to achieve this goal change over time and influence the 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]. 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

A. Auger et al. (Eds.): PPSN 2018, LNCS 11101, pp. 158–169, 2018.
https://doi.org/10.1007/978-3-319-99253-2_13

change in the constraint bound is done randomly every $\tau$ iterations, where $\tau$ 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 $\sigma$. Here $\sigma$ is used to determine the magnitude of changes and large values of $\sigma$ 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 in addition to feasible solutions. 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 one 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 with uniformly chosen weights and profits and bounded strongly correlated instances. We examine the behaviour of the algorithms while varying the frequency and magnitude of changes. Our results show that the (1+1) EA has an advantage over the multi-objective algorithms when the frequency of changes is high. In this case, the population of the multi-objective approaches is slower to adapt to the changes that occur. On the other hand, a lower frequency of changes plays in favor of the multi-objective approaches, if the weights and profits are not correlated to make the instances particularly difficult to solve.

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

## 2 Preliminaries

In this section, we define the Knapsack Problem (KP) and further notations used in the rest of this 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 KP under dynamic constraint. There are $n$ items with profits $\{p_1, \ldots, p_n\}$ and weights $\{w_1, \ldots, w_n\}$. A solution $x$ is a bit string of $\{0, 1\}^n$ which has the overall

**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         $x \leftarrow y$;
```

weight $w(x) = \sum_{i=1}^{n} w_i x_i$ and profit $p(x) = \sum_{i=1}^{n} p_i x_i$. The goal is to compute a solution $x^* = arg\max\{p(x) \mid x \in \{0,1\}^n \wedge w(x) \leq C\}$ of maximal profit which has weight at most $C$.

We consider two types of this problem based on the consideration of the weights. Firstly, we assume that all the weights are one and uniform dynamic constraint is applied. In this case, the limitation is on the number of items chosen for each solution and the optimal solution is to pick $C$ 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 to constraint $C$ happens, all the algorithms update the solution(s) and start the optimization process with the new capacity. This update is addressing the issue that after a dynamic change, current solutions may become infeasible or the distance of its weight from the new capacity become such that it is not 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, the algorithm chooses between the original solution and the mutated one using the value of the fitness function. Let $p_{max} = \max_{i=1}^{n} p_i$ be the maximum profit among all the items. The fitness function that we use in (1+1) EA is as follows:

$$f_{1+1}(x) = p(x) - (n \cdot p_{max} + 1) \cdot \nu(x)$$

where $\nu(x) = \max\{0, w(x) - C\}$ is the constraint violation of $x$. If $x$ is a feasible solution, then $w(x) \leq C$ and $\nu(x) = 0$. Otherwise, $\nu(x)$ 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 the second term is the applied penalty to infeasible solutions. The amount of penalty guarantees that a feasible solution always dominates an infeasible solution. Moreover, between two infeasible solutions, the one with weight closer 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 by a theoretical study

**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 $(C < w(y) \leq C + \delta) \wedge (\nexists p \in S^+ : p \succcurlyeq_{MOEA} y)$ then
17             $S^+ \leftarrow (S^+ \cup \{y\}) \setminus \{z \in S^+ | y \succ_{MOEA} z\}$;
18         if $(C - \delta \leq w(y) \leq C) \wedge (\nexists p \in S^- : p \succcurlyeq_{MOEA} y)$ then
19             $S^- \leftarrow (S^- \cup \{y\}) \setminus \{z \in S^- | y \succ_{MOEA} z\}$;
```

on the performance of evolutionary algorithms in the reoptimization of linear functions under dynamic uniform constraints [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 $w(y) = w(x) \wedge f_{(1+1)}(y) \geq f_{(1+1)}(x)$.

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. MOEA uses a parameter denoted by $\delta$, which determines the maximum number of individuals that the algorithm is allowed to store around the current $C$. For any weight in $[C - \delta, C + \delta]$, MOEA keeps a solution. The algorithm prepares for the dynamic changes by storing nearby solutions, even if they are infeasible as they may become feasible after the next change. A large $\delta$, however, causes a large number of solutions to be kept, which reduces the probability of choosing anyone. Since the algorithm chooses only one solution to mutate in each iteration, this affects the MOEA's performance in finding the optimal solution.

After each dynamic change, MOEA updates the sets of solutions. If a change occurs such that all the current stored solutions are outside of the storing range, namely $[C - \delta, C + \delta]$, then the algorithm consider the previous best solution as the initial solution and uses the Repair function (Algorithm 3), which behaves similar to (1+1) EA, until a solution with weight distance $\delta$ from $C$ is found.

**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              $S^- \leftarrow \{q\} \cup S^-$;
```

**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 $(C < w(y) \leq C + \delta) \wedge (\nexists p \in S^+ : p \succcurlyeq_{MOEA\_D} y)$ then
17     $S^+ \leftarrow (S^+ \cup \{y\}) \setminus \{z \in S^+ | y \succ_{MOEA\_D} z\}$;
18 if $(C - \delta \leq w(y) \leq C) \wedge (\nexists p \in S^- : p \succcurlyeq_{MOEA\_D} y)$ then
19     $S^- \leftarrow (S^- \cup \{y\}) \setminus \{z \in S^- | y \succ_{MOEA\_D} z\}$;
```

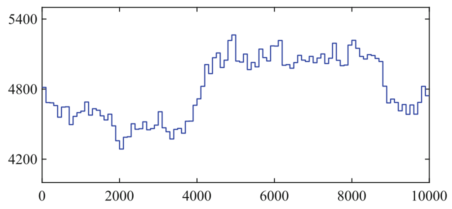
To address the slow rate of improvement of MOEA caused by a large $\delta$, we defined a new dominance procedure. We use the standard definition of dominance in multi-objective optimization and say that solution $y$ dominates solution $x$, denoted by $\succcurlyeq_{MOEA\_D}$, if $w(y) \leq w(x) \wedge p(y) \geq p(x)$. 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 if $y$ is feasible it is only compared with other feasible solutions. MOEA_D keeps fewer solutions than MOEA and overall the quality of the kept solutions is higher, since they are not-dominated by any other solution in the population.

## 3 Benchmarking for the Dynamic Knapsack Problem

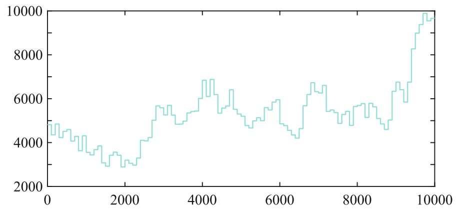
In the following section, the dynamic version of KP used for the experiments is described, and we explain how the dynamic changes occur during the optimization process. In addition, the dynamic benchmarks and the experimental settings are presented.

### 3.1 The Dynamic Knapsack Problem

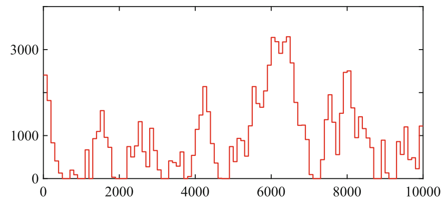
In the dynamic version of KP considered in this paper, the capacity dynamically changes during the optimization with a preset frequency factor denoted by $\tau$.



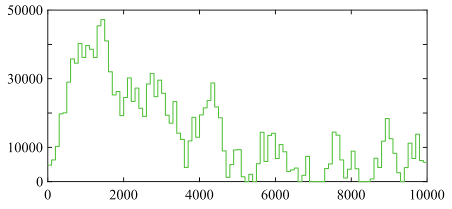
(a) $\mathcal{N}(0, 100^2)$.



(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 $C = 4815$.

A change happens every $\tau$ generations, i.e., the algorithm has $\tau$ 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 the changes is chosen from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$. Figure 1 illustrates how dynamic changes from different distributions affect the capacity. Note that the scales 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 (Fig. 1a) while the capacity reached almost 45000 with dynamic changes under $\mathcal{U}(-10000, 10000)$ (Fig. 1d). This indicates that there are different types of challenges, resulting from the dynamic changes that the algorithms must consider.

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 have less time to find the optimal solution, hence, it is likely that an algorithm which tries to improve only one solution will perform better than another algorithm that needs to optimize among several solutions. Furthermore, the uniform distribution guarantees upper and lower bounds on the magnitude of the changes. This property could be beneficial for the algorithms which keep a number of solutions in each generation, which they do get ready and react faster after a dynamic change. If the changes happen under a normal distribution, however, there is no strict bound on the value of any particular change, which means it is not easy to predict which algorithms will perform better in this type of environment.

### 3.2 Benchmark and Experimental Setting

In this experiment we use eli101 benchmarks, which were originally generated for Traveling Thief Problem [8], 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) instances have uniformly distributed random integers as the weights and profits within $[1000, 1010]$ and $[1, 1000]$, respectively. Finally, there is the Bounded Strongly Correlated (bou-s-c) variations which result in the hardest instances 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 Sect. 4.1, where the weights are one, we set all the weights to one and consider 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 profits. Dynamic changes add a value to $C$ each $\tau$ generations. Four different situations in terms of frequencies are considered: high frequent changes with $\tau = 100$, medium frequent changes with $\tau = 1000$, $\tau = 5000$ 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 $r = 1$ are $r = 10$. In the case of linear weights, when changes are uniformly random, we investigate two values for $r$: $r = 2000, 10000$. Also, changes from normal distribution is experimented for $\sigma = 100$, $\sigma = 500$.

We use the offline errors to compute the performance of the algorithms. In each generation, we record error $e_i = p(x_i^*) - p(x_i)$ where $x_i^*$ and $x_i$ are the optimal solution and the best achieved feasible solution in generation $i$, respectively. If the best achieved solution is infeasible, then we have $e_i = C - w(x)$, 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. Each file consists of 100000 changes, as numbers in $[-r, r]$ generated uniformly at random. Similarly, there are thirty other files with 100000 numbers generated under the normal distribution $\mathcal{N}(0, \sigma^2)$. The algorithms start from the beginning of each file and pick the number of change values from the files. Hence, for each setting, we run the algorithms thirty times with different dynamic change values and record the total offline error of each run.

In order to establish a statistical comparison of the results among different algorithms, we use a multiple comparisons test. In particularity, we focus on the method that compares 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 comparisons of a control algorithm against two or more other algorithms. For more detailed descriptions of the statistical tests we refer the reader to [9].

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

## 4 Experimental Results

In this section we describe the initial settings of the algorithms and analyze their performance using the mentioned statistical tests. The initial solution for all the algorithms is a pack of items which are chosen uniformly at random. Each algorithm initially runs for 10000 generations without any dynamic change. After this, the first change is introduced, and the algorithms run one million further generations with dynamic changes in every $\tau$ generations. For the multi-objective algorithms, it is necessary to initially provide a value for $\delta$. These algorithms keep at most $\delta$ feasible solutions and $\delta$ infeasible solutions, to help them efficiently deal 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 $\mathcal{N}(0, \sigma^2)$, $\delta$ is set to $2\sigma$, since 95% of values will be within $2\sigma$ of the mean value. Note that a larger $\delta$ value increases the population size of the algorithms and there is a trade-off between the size of the population and the speed of algorithm in reacting to the next change.

### 4.1 Dynamic Uniform Constraint

In this section, we validate the theoretical results against the performance of (1+1) EA and Multi-Objective Evolutionary Algorithm. Shi et al. [7] state that the multi-objective approach performs better than (1+1) EA in reoptimizing the optimal solution of dynamic KP under uniform constraint. Although the MOEA that we used in this experiment is not identical to the multi-objective algorithm studied previously by Shi et al. [7] and they only considered the reoptimization time, the experiments show that multi-objective approaches outperform (1+1) EA in the case of uniform constraints (Table 1). An important reason for this remarkable performance is the relation between optimal solutions in different weights. In this type of constraint, the difference between the optimal solution of weight $w$ and $w + 1$ is one item. As a result of this, keeping non-dominated solutions near the constrained bound helps the algorithm to find the current optimum more efficiently and react faster after a dynamic change.

Furthermore, according to the results, there is no significant difference between using MOEA and MOEA_D in this type of KP. Considering the experiments in Sect. 4.2, a possible reason is that the size of population in MOEA remains small when weights are one. Hence, MOEA_D, which stores fewer items because of its dominance definition, has no advantage in this manner anymore. In addition, the constraint is actually on the number of the items. Thus, both definitions for dominance result the same in many cases.

**Table 1.** The mean, standard deviation values and statistical tests of the offline error for (1+1) EA, MOEA, MOEA_D based on the uniform distribution with all the weights as one.

| | n | r | $\tau$ | (1+1) EA (1) | | | MOEA (2) | | | MOEA_D (3) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Mean | St | Stat | Mean | St | Stat | Mean | St | Stat |
| uncor | 100 | 5 | 100 | 4889.39 | 144.42 | $2^{(-)}$, $3^{(-)}$ | 1530.00 | 120.76 | $1^{(+)}$ | 1486.85 | 123.00 | $1^{(+)}$ |
| | 100 | 5 | 1000 | 1194.23 | 86.52 | $2^{(-)}$, $3^{(-)}$ | 44.75 | 8.96 | $1^{(+)}$ | 46.69 | 8.51 | $1^{(+)}$ |
| unc-s-w | 100 | 5 | 100 | 4990.80 | 144.87 | $2^{(-)}$, $3^{(-)}$ | 1545.36 | 115.15 | $1^{(+)}$ | 1500.07 | 106.70 | $1^{(+)}$ |
| | 100 | 5 | 1000 | 1160.23 | 130.32 | $2^{(-)}$, $3^{(-)}$ | 41.90 | 6.13 | $1^{(+)}$ | 43.06 | 7.22 | $1^{(+)}$ |
| bou-s-c | 100 | 5 | 100 | 13021.98 | 780.76 | $2^{(-)}$, $3^{(-)}$ | 4258.53 | 580.77 | $1^{(+)}$ | 4190.55 | 573.13 | $1^{(+)}$ |
| | 100 | 5 | 1000 | 3874.76 | 911.50 | $2^{(-)}$, $3^{(-)}$ | 177.62 | 83.16 | $1^{(+)}$ | 175.14 | 80.73 | $1^{(+)}$ |

### 4.2 Dynamic Linear Constraint

In this section, we consider the same algorithms in more difficult environments where weights are arbitrary under dynamic linear constraint. As it is shown in Sect. 4.1, the multi-objective approaches outperform (1+1) EA in the case that weights are one. Now we try to answer the question: Does the relationship between the algorithms hold when the weights are arbitrary?

The data in Table 2 shows the experimental results in the case of dynamic linear constraints and changes under a uniform distribution. It can be observed that (as expected) the mean of errors decreases as $\tau$ increases. Larger $\tau$ values give more time to the algorithm to get closer to the optimal solution. Moreover, starting from a solution which is near to the optimal for the previous capacity, can help to speed up the process of finding the new optimal solution in many cases.

We first consider the results of dynamic changes under the uniform distribution. We observe in Table 2 that unlike with uniform constraint, in almost all the settings, MOEA has the worst performance of all the algorithms. The first reason for this might be that items selected in optimal solutions with close weights are also close in terms of Hamming distance. In other words, when weights are one, we can achieve the optimal solution for weight $w$ by adding an item to the optimal solution for weight $w-1$ or by deleting an item from the optimal solution for $w+1$. 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. Another reason could be the effect of having a large population. A large population may cause the optimization process to take longer and it could get worse because of the definition of $\succcurlyeq_{MOEA}$, which only compares solutions with equal weights. If $s$ is a new solution and there is no solution with $w(s)$ in the set of existing solutions, MOEA keeps $s$ whether $s$ is a good solution or not, i.e., regardless of whether it is really a non-dominated solution or whether it would be dominated by other solutions in the set. This comparison also does not consider if $s$ has any good properties to be inherited by the next generation. Moreover, putting $s$ in the set of solutions decreases the probability of choosing any other solution, even those solutions that are very close to the optimal solution. As it can be seen in the Table 2, however, there is only one case in which MOEA beat the (1+1) EA: when the weights are similar, and the magnitude of changes are small (2000), which means the population size is also small (in comparison to 10000), and finally $\tau$ is at its maximum to let the MOEA to use its population to optimize the problem.

Although MOEA does not perform very well in instances with general weights, the multi-objective approach with a better defined dominance, MOEA_D, does outperform (1+1) EA in many cases. We compare the performance of (1+1) EA and MOEA_D below.

**Table 2.** The mean, standard deviation values and statistical tests of the offline error for (1+1) EA, MOEA, MOEA_D based on the uniform distribution.

| | n | r | $\tau$ | (1+1) EA (1) | | | MOEA (2) | | | MOEA_D (3) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | mean | st | stat | mean | st | stat | mean | st | stat |
| uncor | 100 | 2000 | 100 | 5564.37 | 463.39 | $2^{(+)},3^{(-)}$ | 11386.40 | 769.77 | $1^{(-)},3^{(-)}$ | 3684.26 | 525.50 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 1000 | 2365.56 | 403.64 | $2^{(+)},3^{(-)}$ | 7219.17 | 587.50 | $1^{(-)},3^{(-)}$ | 776.14 | 334.69 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 5000 | 1415.42 | 167.08 | $2^{(+)},3^{(-)}$ | 3598.29 | 420.12 | $1^{(-)},3^{(-)}$ | 270.90 | 121.43 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 15000 | 914.55 | 102.82 | $2^{(+)},3^{(-)}$ | 2004.16 | 368.82 | $1^{(-)},3^{(-)}$ | 88.80 | 43.98 | $1^{(+)},2^{(+)}$ |
| unc-s-w | 100 | 2000 | 100 | 3128.43 | 188.36 | $2^{(+)},3^{(-)}$ | 5911.11 | 534.24 | $1^{(-)},3^{(-)}$ | 2106.45 | 249.28 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 1000 | 606.14 | 99.23 | $2^{(+)},3^{(-)}$ | 1564.23 | 619.97 | $1^{(-)},3^{(-)}$ | 302.34 | 24.60 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 5000 | 147.55 | 31.80 | $3^{(-)}$ | 174.23 | 95.98 | $3^{(-)}$ | 60.94 | 9.12 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 15000 | 64.65 | 17.13 | $2^{(-)},3^{(-)}$ | 40.66 | 15.51 | $1^{(+)},3^{(-)}$ | 19.26 | 4.04 | $1^{(+)},2^{(+)}$ |
| bou-s-c | 100 | 2000 | 100 | 3271.07 | 266.54 | $2^{(+)}$ | 5583.53 | 337.81 | $1^{(-)},3^{(-)}$ | 3036.97 | 297.33 | $2^{(+)}$ |
| | 100 | 2000 | 1000 | 1483.01 | 85.14 | $2^{(+)},3^{(-)}$ | 2639.16 | 106.47 | $1^{(-)},3^{(-)}$ | 617.92 | 186.35 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 5000 | 796.77 | 89.80 | $2^{(+)},3^{(-)}$ | 1256.62 | 118.27 | $1^{(-)},3^{(-)}$ | 251.41 | 109.58 | $1^{(+)},2^{(+)}$ |
| | 100 | 2000 | 15000 | 538.45 | 66.98 | $2^{(+)},3^{(-)}$ | 687.95 | 116.91 | $1^{(-)},3^{(-)}$ | 104.27 | 61.06 | $1^{(+)},2^{(+)}$ |
| uncor | 100 | 10000 | 100 | 10256.72 | 210.51 | $2^{(+)},3^{(+)}$ | 16278.97 | 248.43 | $1^{(-)},3^{(-)}$ | 11038.07 | 236.91 | $1^{(-)},2^{(+)}$ |
| | 100 | 10000 | 1000 | 3604.18 | 285.73 | $2^{(+)}$ | 13340.20 | 704.32 | $1^{(-)},3^{(-)}$ | 3508.51 | 473.42 | $2^{(+)}$ |
| | 100 | 10000 | 5000 | 1607.78 | 278.60 | $2^{(+)},3^{(-)}$ | 10614.45 | 1660.32 | $1^{(-)},3^{(-)}$ | 1183.52 | 411.83 | $1^{(+)},2^{(+)}$ |
| | 100 | 10000 | 15000 | 987.64 | 219.53 | $2^{(+)},3^{(-)}$ | 8006.35 | 1612.20 | $1^{(-)},3^{(-)}$ | 566.69 | 219.54 | $1^{(+)},2^{(+)}$ |
| unc-s-w | 100 | 10000 | 100 | 7192.82 | 153.93 | $2^{(+)},3^{(+)}$ | 12617.69 | 318.23 | $1^{(-)},3^{(-)}$ | 8057.44 | 274.17 | $1^{(-)},2^{(+)}$ |
| | 100 | 10000 | 1000 | 1846.43 | 115.23 | $2^{(+)}$ | 6981.81 | 768.78 | $1^{(-)},3^{(-)}$ | 1743.12 | 364.38 | $2^{(+)}$ |
| | 100 | 10000 | 5000 | 539.39 | 65.39 | $2^{(+)}$ | 3488.28 | 819.51 | $1^{(-)},3^{(-)}$ | 519.63 | 175.22 | $2^{(+)}$ |
| | 100 | 10000 | 15000 | 208.73 | 36.91 | $2^{(+)}$ | 1525.23 | 306.72 | $1^{(-)},3^{(-)}$ | 201.97 | 79.28 | $2^{(+)}$ |
| bou-s-c | 100 | 10000 | 100 | 7187.80 | 122.59 | $2^{(+)},3^{(+)}$ | 15111.38 | 231.53 | $1^{(-)},3^{(-)}$ | 12736.55 | 229.48 | $1^{(-)},2^{(+)}$ |
| | 100 | 10000 | 1000 | 2282.81 | 219.24 | $2^{(+)},3^{(+)}$ | 8301.43 | 569.90 | $1^{(-)},3^{(-)}$ | 3575.26 | 550.54 | $1^{(-)},2^{(+)}$ |
| | 100 | 10000 | 5000 | 1370.48 | 250.59 | $2^{(+)}$ | 5248.40 | 1045.78 | $1^{(-)},3^{(-)}$ | 1472.19 | 493.88 | $2^{(+)}$ |
| | 100 | 10000 | 15000 | 955.38 | 133.33 | $2^{(+)}$ | 3852.07 | 752.84 | $1^{(-)},3^{(-)}$ | 977.41 | 397.75 | $2^{(+)}$ |

When changes are smaller, it can be seen in Table 2 that the mean of offline errors of MOEA_D is smaller than (1+1) EA. The dominance of MOEA_D is such that only keeps the dominant solutions. When a new solution is found, the algorithm removes solutions that are dominated by it and keeps it only if it is not dominated by the any other one. This process improves the quality of the solutions by increasing the probability of keeping a solution beneficial to future generations. Moreover, it reduces the size of the population significantly. Large changes to the capacity, however, makes the MOEA_D keep more individuals, and it is in this circumstance that (1+1) EA may perform better than MOEA_D.

When $r = 10000$, MOEA_D does not have significantly better results in all cases unlike in the case of $r = 2000$, and in most of the situations it performs as well as (1+1) EA. In all high frequency conditions where $\tau = 100$, the

(1+1) EA has better performance. It may be caused by MOEA_D needing more time to optimize a population with a larger size. Moreover, when the magnitude of changes is large, it is more likely that a new change will force MOEA_D to remove all of its stored individuals and start from scratch.

We now study the experimental results that came from considering the dynamic changes under the normal distribution (Table 3). The results confirm that (1+1) EA is faster with more frequent changes. Skipping the case with uncorrelated similar weights and frequent changes, MOEA_D has always been the best algorithm in terms of performance and MOEA has been the worst.

**Table 3.** The mean, standard deviation values and statistical tests of the offline error for (1+1) EA, MOEA, MOEA_D based on the normal distribution.

| | n | $\sigma$ | $\tau$ | (1+1) EA (1) | | | MOEA (2) | | | MOEA_D (3) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | mean | st | stat | mean | st | stat | mean | st | stat |
| uncor | 100 | 100 | 100 | 2714.72 | 106.06 | $2^{(+)},3^{(+)}$ | 9016.83 | 2392.48 | $1^{(-)},3^{(-)}$ | 4271.09 | 789.94 | $1^{(-)},2^{(+)}$ |
| | 100 | 100 | 1000 | 1386.66 | 97.11 | $2^{(+)},3^{(-)}$ | 3714.89 | 737.11 | $1^{(-)},3^{(-)}$ | 412.89 | 27.25 | $1^{(+)},2^{(+)}$ |
| | 100 | 100 | 5000 | 801.54 | 73.67 | $2^{(+)},3^{(-)}$ | 1266.35 | 119.25 | $1^{(-)},3^{(-)}$ | 108.28 | 14.22 | $1^{(+)},2^{(+)}$ |
| | 100 | 100 | 15000 | 549.71 | 78.98 | $2^{(+)},3^{(-)}$ | 749.86 | 148.03 | $1^{(-)},3^{(-)}$ | 61.93 | 17.03 | $1^{(+)},2^{(+)}$ |
| unc-s-w | 100 | 100 | 100 | 412.24 | 111.07 | $2^{(+)},3^{(+)}$ | 1979.65 | 914.35 | $1^{(-)}$ | 1904.09 | 877.55 | $1^{(-)}$ |
| | 100 | 100 | 1000 | 85.55 | 23.13 | $2^{(+)},3^{(+)}$ | 1566.54 | 409.32 | $1^{(-)}$ | 1482.37 | 391.75 | $1^{(-)}$ |
| | 100 | 100 | 5000 | 36.94 | 13.61 | $2^{(+)},3^{(+)}$ | 1414.66 | 448.78 | $1^{(-)}$ | 1322.35 | 414.27 | $1^{(-)}$ |
| | 100 | 100 | 15000 | 29.14 | 19.70 | $2^{(+)},3^{(+)}$ | 1237.67 | 665.27 | $1^{(-)}$ | 1137.80 | 648.73 | $1^{(-)}$ |
| bou-s-c | 100 | 100 | 100 | 1491.36 | 260.72 | $2^{(+)},3^{(+)}$ | 4625.49 | 1302.52 | $1^{(-)},3^{(-)}$ | 2903.77 | 717.92 | $1^{(-)},2^{(+)}$ |
| | 100 | 100 | 1000 | 736.10 | 53.99 | $2^{(+)},3^{(-)}$ | 1748.61 | 189.94 | $1^{(-)},3^{(-)}$ | 312.88 | 35.52 | $1^{(+)},2^{(+)}$ |
| | 100 | 100 | 5000 | 446.94 | 39.36 | $2^{(+)},3^{(-)}$ | 640.60 | 91.29 | $1^{(-)},3^{(-)}$ | 101.21 | 17.47 | $1^{(+)},2^{(+)}$ |
| | 100 | 100 | 15000 | 337.85 | 40.44 | $2^{(+)},3^{(-)}$ | 469.16 | 93.99 | $1^{(-)},3^{(-)}$ | 70.16 | 22.26 | $1^{(+)},2^{(+)}$ |
| uncor | 100 | 500 | 100 | 13400.88 | 305.14 | $2^{(+)},3^{(+)}$ | 46395.44 | 4565.61 | $1^{(-)},3^{(-)}$ | 19218.94 | 1035.72 | $1^{(-)},2^{(+)}$ |
| | 100 | 500 | 1000 | 6363.16 | 194.59 | $2^{(+)},3^{(-)}$ | 25747.08 | 1181.11 | $1^{(-)},3^{(-)}$ | 2387.61 | 151.73 | $1^{(+)},2^{(+)}$ |
| | 100 | 500 | 5000 | 3983.06 | 254.38 | $2^{(+)},3^{(-)}$ | 18004.03 | 1243.66 | $1^{(-)},3^{(-)}$ | 1467.58 | 152.77 | $1^{(+)},2^{(+)}$ |
| | 100 | 500 | 15000 | 3112.73 | 315.29 | $2^{(+)},3^{(-)}$ | 17610.35 | 1265.50 | $1^{(-)},3^{(-)}$ | 1348.25 | 194.71 | $1^{(+)},2^{(+)}$ |
| unc-s-w | 100 | 500 | 100 | 2845.31 | 146.80 | $2^{(+)},3^{(+)}$ | 11803.99 | 1256.99 | $1^{(-)}$ | 11438.04 | 1247.62 | $1^{(-)}$ |
| | 100 | 500 | 1000 | 595.70 | 86.19 | $2^{(+)},3^{(+)}$ | 8851.36 | 1488.59 | $1^{(-)}$ | 8478.21 | 1313.55 | $1^{(-)}$ |
| | 100 | 500 | 5000 | 222.79 | 62.22 | $2^{(+)},3^{(+)}$ | 7025.45 | 2639.34 | $1^{(-)}$ | 6488.47 | 2335.92 | $1^{(-)}$ |
| | 100 | 500 | 15000 | 171.33 | 50.28 | $2^{(+)},3^{(+)}$ | 7188.67 | 4184.84 | $1^{(-)}$ | 6278.10 | 4146.54 | $1^{(-)}$ |
| bou-s-c | 100 | 500 | 100 | 7444.23 | 290.00 | $2^{(+)},3^{(+)}$ | 24462.58 | 1330.93 | $1^{(-)},3^{(-)}$ | 15592.66 | 791.70 | $1^{(-)},2^{(+)}$ |
| | 100 | 500 | 1000 | 4062.63 | 210.49 | $2^{(+)},3^{(-)}$ | 12291.63 | 589.18 | $1^{(-)},3^{(-)}$ | 2781.20 | 317.88 | $1^{(+)},2^{(+)}$ |
| | 100 | 500 | 5000 | 3013.35 | 289.29 | $2^{(+)},3^{(-)}$ | 9667.96 | 571.34 | $1^{(-)},3^{(-)}$ | 1971.56 | 220.63 | $1^{(+)},2^{(+)}$ |
| | 100 | 500 | 15000 | 2722.29 | 342.39 | $2^{(+)},3^{(-)}$ | 9308.28 | 719.25 | $1^{(-)},3^{(-)}$ | 1760.51 | 251.51 | $1^{(+)},2^{(+)}$ |

The most notable results occur in the case with uncorrelated similar weights. (1+1) EA outperforms both other algorithms in this instance. This happens because of the value of $\delta$ and the weights of the instances. $\delta$ is set to $2\sigma$ in the multi-objective approaches and the 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 the multi-objective approaches are only allowed to consider solutions in range of $[C - \delta, C + \delta]$. In other words, it is possible that there is only one solution in that range or even no solution. Hence, multi-objective approaches have no advantage in this type of instances according to the value of $\delta$ and weights of the items, and in fact, may have a disadvantage.

## 5 Conclusions and Future Work

In this paper we studied the evolutionary algorithms for the KP where the capacity dynamically changes during the optimization process. In the introduced dynamic setting, the frequency of changes is determined by $\tau$. The magnitude of changes is chosen randomly either under the uniform distribution $\mathcal{U}(-r, r)$ or under the normal 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 one verified the previous theoretical studies for (1+1) EA and MOEA [7]. It is shown that the multi-objective approach, which uses a population in the optimization, outperforms (1+1) EA. In addition, we considered the algorithms in the case of general weights for different classes of instances with a variation of frequencies and magnitudes. Our results illustrated that MOEA does not perform well in the general case due to its dominance procedure. However, MOEA_D, which benefits from a population with a smaller size and non-dominated solutions, beats (1+1) EA in most cases. On the other hand, in the environments with highly frequent changes, (1+1) EA performs better than the multi-objective approaches. In such cases, the population slows down MOEA_D in reacting to 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, Heidelberg (2007). https://doi.org/10.1007/978-3-662-44874-8
2. Nguyen, T., Yao, X.: Continuous dynamic constrained optimization: the challenges. IEEE Trans. Evol. Comput. **16**(6), 769–786 (2012)
3. Rakshit, P., Konar, A., Das, S.: Noisy evolutionary optimization algorithms - a comprehensive survey. Swarm Evol. Comput. **33**, 18–45 (2017)
4. Nguyen, T.T., Yang, S., Branke, J.: Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol. Comput. **6**, 1–24 (2012)
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.) EvoApplications 2018. LNCS, vol. 10784, pp. 832–847. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77538-8_55
6. Pourhassan, M., Gao, W., Neumann, F.: Maintaining 2-approximations for the dynamic vertex cover problem using evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 903–910. ACM (2015)
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, pp. 1407–1414. ACM (2017)
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 Conference on Genetic and Evolutionary Computation, pp. 477–484. ACM (2014)
9. Corder, G.W., Foreman, D.I.: Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach. Wiley, Hoboken (2009)