Computing and Informatics, Vol. 29, 2010, 783–800

ON THE DISRUPTION-LEVEL OF POLYNOMIAL MUTATION FOR EVOLUTIONARY MULTI-OBJECTIVE OPTIMISATION ALGORITHMS

Mohammad Hamdan

Department of Computer Science
Faculty of IT, Yarmouk University
Irbid 21163, Jordan
e-mail: hamdan@yu.edu.jo

Manuscript received 5 March 2009; revised 26 August 2009
Communicated by Vladimír Kvasnička

Abstract. This paper looks at two variants of polynomial mutation used in various evolutionary optimisation algorithms for mutliobjective problems. The first is a non-highly disruptive and the second is a highly disruptive mutation. Both are used for problems with box constraints. A new hybrid polynomial mutation that combines the benefits of both is proposed and implemented. The experiments with three evolutionary multi-objective algorithms on well-known multi-objective optimisation problems show the difference in terms of generational distance, hypervolume, convergence speed and hit rate metrics. The hybrid polynomial mutation in general retains the advantages of both versions in the same algorithm.

Keywords: Multi-objective optimisation, evolutionary algorithms, polynomial mutation

1 INTRODUCTION

There are many objectives to be optimised at the same time when dealing with real world problems. Normally, the objectives are conflicting and make it difficut to solve the problem. Such problems are known as Multiobjective Optimisation Problems (MOPs) [1] and can be formally (for a minimisation problem) defined as

$$\min[f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), \ldots, f_d(\overrightarrow{x})]$$

where $\overrightarrow{x} \in \Omega \subseteq \Re^n$ is a vector of decision variables. Finding a single solution vector $\overrightarrow{x}$ that minimises all objectives at the same time is a difficult task. For such problems, the solution is a set called *non-dominated solutions*. The main property of this set is that no solution dominates other solutions. A dominance relation denoted by "$\prec$" can be defined as follows [1]:

$$\begin{aligned}(\overrightarrow{x_1} \prec \overrightarrow{x_2}) \Leftrightarrow \quad & \forall i \in \{1, \ldots, d\}, f_i(\overrightarrow{x_1}) \leq f_i(\overrightarrow{x_2}) \\ & \text{and } \exists i \in \{1, \ldots, d\}, f_i(\overrightarrow{x_1}) < f_i(\overrightarrow{x_2}).\end{aligned}$$

Using the Pareto dominance relation, the minimal elements are called Pareto-optimal. The set of Pareto-optimal vectors are known as *Pareto-optimal set*. When plotted in the objective space they define the true Pareto-Front (PF_{true}). The goal of algorithms solving MOPs is to generate PF_{known} that is a good approximation of PF_{true}.

One of the well known-methods for optimisation is taken from the concepts of evolutionary systems (ES) [9] called genetic algorithms (GAs) [11]. They simulate the natural selection process in biological systems using three major operators: selection, crossover and mutation.

Due to the success of GAs in solving single objective optimisation problems, they have been extended for solving MOPs. The nature of GAs of being population-based fits well the needs of MOPs since each individual in the population can define a non-dominated solution in the Pareto-optimal set. It is worth noting that most of evolutionary-based algorithms solving MOPs use the simulated binary (SBX) [3] and polynomial [4] as crossover and mutation operators, respectively. Both operators were first introduced and implemented in Non-dominated Sorting Genetic Algorithm (NSGA) and NSGA-II [5].

However, the polynomial mutation for box constraints in the published literature has two different implementations under a single name. The difference is in the amount of disruptions they can make to a decision variable when mutated.

In this study, we need to investigate the effect of both versions of polynomial mutation on the behaviour of selected algorithms when solving well-known test MOPs. To control the disruptions, a new version of polynomial mutation is introduced enabling the user to manage disruption levels. The new version combines the benefits of both implementations.

The rest of the paper is structured as follows. In Section 2, a complete description of both versions of polynomial mutation is given and the proposed hybrid polynomial mutation is presented. The experimental environment is presented in Section 3. In Section 4, the results are shown and, finally, Section 5 concludes the paper and outlines future work.

2 POLYNOMIAL MUTATION

In biological systems, mutation could happen due to errors in copying DNA material during the exchange of chromosomes and is introduced in the optimisation algorithm by perturbing the solution variable using a certain probability.

In traditional GAs, mutation has been introduced in various forms. It depends on the representation used in GA such as binary or real values. For example, the possible mutation implementations could be flip-bit, uniform and non-uniform [12]. For GAs solving MOPs, a more specialised implementation was introduced called polynomial mutation. This operator was first proposed in [3], used in NSGA [15] then later improved in [6].

The original polynomial mutation [4, 2] is presented in Algorithm 1. Mutation probability is set by the variable P_m, n is number of decision variables and η_m is distribution index which can take any non-negative value. For each decision variable x_i, box constraints are defined in $[x_i^{\text{Lower}}, x_i^{\text{Upper}}]$. Each decision variable has a probability P_m to be perturbed. For each decision variable, a random value r is drawn. If $(r < P_m)$ then using the procedure described in Algorithm 1 a mutated variables gets its new value. The procedure first finds the difference between the variable and both (lower and upper) boundaries. The smallest difference divided by $(X_{\text{Upper}} - X_{\text{Lower}})$ is called δ. Another random value is drawn. The procedure samples to the left hand side of the variable (region between X_i and X_{Lower}) if random value is ≤ 0.5; otherwise it samples to the right hand side (region between X_i and X_{Upper}). The procedure calculates the δ_q value to be used in getting the variable its new value. The probability distribution function is polynomial and its shape is controlled by η_m. Large values of η_m give higher probabilities of creating offspring within the vicinity of the parent and small values allow distant solution to be created. The distribution index η_m produces a perturbation of the order $\mathrm{O}(1/\eta_m)$ in the normalised decision variable.

The problem with this version is shown in Figure 1. If the value of the decision variable to be mutated is close to one of the boundaries (i.e. the δ value is very small) then the mutation becomes useless and the algorithm could get trapped in local optima especially in multi-modal problems. This version has been used in NSGA [15], early versions of NSGA-II [5] and in the DEME project [10]. This can be determined by analysing the available source code of various algorithms solving MOPs that use polynomial as mutation operator.

The modified polynomial mutation [6] is shown in Algorithm 2. It is similar to original polynomial mutation described in Algorithm 1. The only difference is in the choice of δ. As illustrated in Figure 1, the mutation can sample the entire search space of the decision variable even though the value to be mutated is close to one of the boundaries. It allows big jumps in the search space of decision variable. In summary, this version is highly disruptive, gives better chances of escaping from local optima and can modify a solution when on the boundary. However, such kind of disruptions might not be good for smooth approximation to Pareto front. For example, if a solution is near the optimal solution then large jumps in the decision

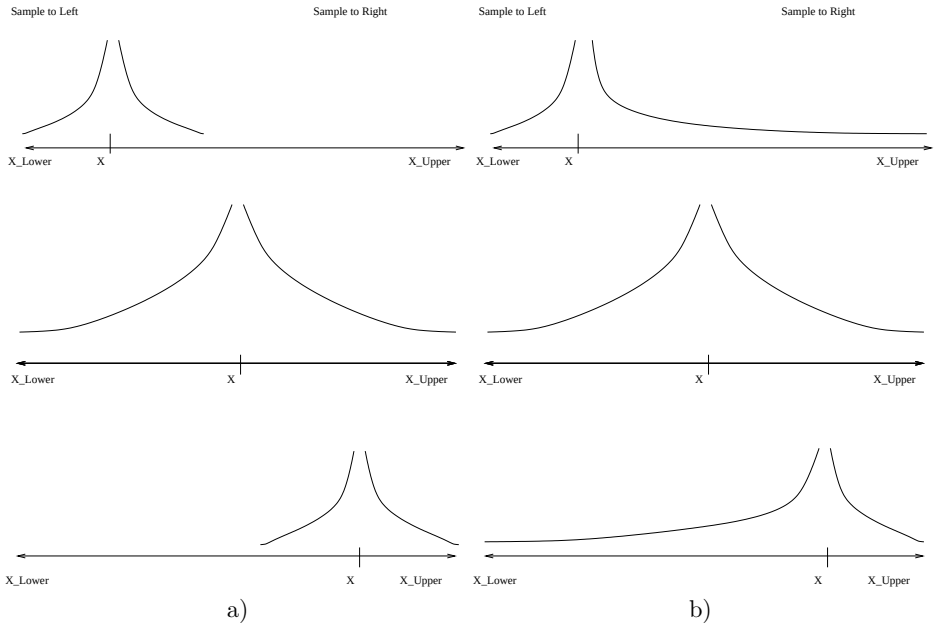

Fig. 1. The sampling space for the non-highly a) and highly disruptive polynomial mutations. The y-axis is the probability distribution (spread)

space might not be very useful. It has been used in the jMetal Toolkit [8] and in the latest version of NSGA-II [5].

i $\leftarrow 0$
repeat
  r $\leftarrow$ U[0,1]
  if $r \leq P_m$ **then**
    $\delta \leftarrow \frac{min\{X_i^{Upper}-X_i, X_i-X_i^{Lower}\}}{X_i^{Upper}-X_i^{Lower}}$
    r $\leftarrow$ U[0,1]
    $\delta_q \leftarrow \begin{cases} [(2r)+(1-2r)* \\ (1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}}-1 & \text{if } r \leq 0.5 \\ 1-[2(1-r)+2(r-0.5)* \\ (1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} & \text{otherwise} \end{cases}$
    $X_i \leftarrow X_i + \delta_q.(X_i^{Upper} - X_i^{Lower})$
until $i++ == n$

Algorithm 1: Non-highly disruptive polynomial mutation

In Algorithm 1, $\delta = min[x_i - x_i^{Lower}, x_i^{Upper} - x_i]/(x_i^{Upper} - x_i^{Lower})$. In Algorithm 2, $\delta = (x_i - x_i^{Lower})/(x_i^{Upper} - x_i^{Lower})$ or $(x_i^{Upper} - x_i)/(x_i^{Upper} - x_i^{Lower})$ which depends on which side to sample (left or right hand sides of the mutated variable).

To decide which side, a random value r is drawn. If $r \leq 0.5$ then sample to the left hand side; otherwise to the right hand side.

i $\leftarrow 0$
repeat
  r $\leftarrow$ U[0,1]
  if $r \leq P_m$ **then**
    $\delta_1 \leftarrow \frac{X_i - X_i^{Lower}}{X_i^{Upper} - X_i^{Lower}}$
    $\delta_2 \leftarrow \frac{X_i^{Upper} - X_i}{X_i^{Upper} - X_i^{Lower}}$
    r $\leftarrow$ U[0,1]
    $\delta_q \leftarrow \begin{cases} [(2r) + (1-2r)* \\ (1-\delta_1)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} - 1 & \text{if } r \leq 0.5 \\ 1 - [2(1-r) + 2.(r-0.5)* \\ (1-\delta_2)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} & \text{otherwise} \end{cases}$
    $X_i \leftarrow X_i + \delta_q.(X_i^{Upper} - X_i^{Lower})$
until $i++ == n$

Algorithm 2: Highly disruptive polynomial mutation

In Algorithm 3, a hybridisation of both versions is proposed. Using different p values, the EMOA can specify the disruptions level of mutation. It works as follows: with probability $1 - p$ use the non-highly disruptive polynomial mutation (illustrated in Algorithm 1) and with probability p use the highly disruptive version (described in Algorithm 2).

In the new hybrid polynomial mutation, the disruptions level can be quantified using different p values. When $p = 0.0$ then only the non-highly disruptive polynomial mutation is used. When setting $p = 1.0$ then only the highly disruptive is used. The best p value can be determined after experimental evaluation of different p values for various algorithms and problems.

3 EXPERIMENTAL ENVIRONMENT

3.1 Parameter Setup

The jMetal [8] toolkit[1] has been used to test the different p values for the proposed polynomial mutation. It is a Java based framework for multiobjective optimisations. Other researchers have also used the toolkit for studying multiobjective metaheuristics [7]. The jMetal toolkit provides implementations for the Nondominated Sorting Genetic Algorithm II (NSGA-II) [5], the Strength Pareto Evolutionary Algorithm (SPEA2) [18] and the MultiObjective Cellular Genetic Algorithm (MOCell) [13].

[1] Can be downloaded from `http://mallba10.lcc.uma.es/wiki/index.php/JMetal`.

p ← User specified value in the range [0,1]
u ← U[0,1]
if $u > p$ **then**
  mut ← 1
else
  mut ← 2
i ← 0
repeat
  r ← U[0,1]
  if $r \leq P_m$ **then**
    $\delta_1 \leftarrow \frac{X_i - X_i^{Lower}}{X_i^{Upper} - X_i^{Lower}}$
    $\delta_2 \leftarrow \frac{X_i^{Upper} - X_i}{X_i^{Upper} - X_i^{Lower}}$
    r ← U[0,1]
    if $u > p$ **then**
      $\delta \leftarrow min\{\delta_1, \delta_2\}$
    else
      $\delta \leftarrow \begin{cases} \delta_1 & \text{if } r \leq 0.5 \\ \delta_2 & \text{otherwise} \end{cases}$
    $\delta_q \leftarrow \begin{cases} [(2r) + (1-2r)* & \\ (1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} - 1 & \text{if } r \leq 0.5 \\ 1 - [2(1-r) + 2.(r-0.5)* & \\ (1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} & \text{otherwise} \end{cases}$
    $X_i \leftarrow X_i + \delta_q.(X_i^{Upper} - X_i^{Lower})$
until $i++ == n$

Algorithm 3: Proposed hybrid polynomial mutation

These algorithms are well known for solving MOPs and are used for the experimental study. All of the genetic algorithms use binary tournament, simulated binary crossover (SBX) [3] and polynomial mutation as selection, crossover and mutation operators, respectively. It is worth noting that polynomial mutation was first introduced and used in NSGA and NSGA-II.

The new hybrid mutation presented in Algorithm 3 has been integrated with the jMetal toolkit and can be used as follows:

```
mutation = MutationFactory.getMutation-
 Operator("HybridPolynomialMutation");
mutation.setParameter("probability",
 1.0/problem.getNumberOfVariables());
mutation.setParameter("p", 0.2);
```

The crossover probability is $P_c = 0.9$ and mutation probability is $P_m = 1/n$, where n is the number of decision variables. The distribution indices for the crossover

and mutation operators are $\eta_c = 20$ and $\eta_m = 20$, respectively. Population size is set to 100, using 25 000 function evaluations with 100 independent runs.

3.2 The Test Problems

The Zitzler-Deb-Theile (ZDT) test suite [17] is widely used for evaluating algorithms solving MOPs. The following five bi-objective MOPs named ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6 were used for evaluating the disruptions level of polynomial mutation. They are well understood and their Pareto front shapes are convex, non-convex, disconnected, multi-modal and non-uniformly spaced. ZDT1, ZDT2 and ZDT3 use 30 decision variables while ZDT4 and ZDT6 use 10 decision variables. The test functions are described in the Appendix.

3.3 The Performance Metrics

The Generational Distance (GD) [16] is a standard metrics for measuring how far the solutions generated by an algorithm (PF_{known}) are from those in the Pareto optimal set (PF_{true}). It is defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{1}$$

where n is the number of solutions in PF_{known} and d_i is the Euclidean distance between each member in PF_{known} and the nearest member in PF_{true}. The aim is to minimise the GD value.

The hypervolume (HV) metrics [19] measures the hypervolume (hyperarea in case of 2 objectives) of dominated space by the set of non-dominated solutions in PF_{known}. Higher values of HV are desirable. Assume solution set S that contains the non-dominated solutions generated by an algorithm. For a given reference point W (selected by the user or chosen from the worst solution vectors), a hypercube v_i is constructed for every non-dominated solutions in S using W and v_i as the diagonal corners of the hypercube. The union of all hypercubes and the corresponding hypervolume (HV) can be calculated as follows:

$$HV = \text{volume}\left(\bigcup_{i=1}^{|S|} v_i\right). \tag{2}$$

Using the HV metrics it is possible to measure the *convergence speed* metrics [7] at run-time. If $HV(PF_{\text{known}}) \geq 98\,\% * HV(PF_{\text{true}})$ then an execution is successful as long as the algorithm did not reach the maximum number of function evaluations. This feature is implemented in jMetal and it checks for the hypervolume of the current Pareto front after each function evaluation/iteration. It returns the number of function evaluations used so far once the hypervolume of current Pareto front

reaches at least 98 % of the hypervolume of the PF_{true}. However, this requires the availability of PF_{true} for the test problem and this is available for the ZDT test problems. Moreover, the *hit rate* metrics [7] can be used to indicate the percentage of successful executions.

4 RESULTS

The results reported in this section address two objectives:

1. the convergence speed and success (i.e. *hit rate*) of obtaining acceptable approximation of PF_{true} using different p values, and
2. the quality of the final obtained solution using the GD and HV metrics for different p values.

To provide results with confidence a non-parametric statistical test was performed on the results to find if the differences are statistically significant. The Wilcoxon rank sum test [14] was performed with significance level of $p^2 < 0.05$.

Tables 1, 2 and 3 show the median $\tilde{x}$ and interquartile range (IQR) for the needed function evaluations ($\tilde{x}_{IQR}$) (i.e. *convergence speed*) to generate an acceptable Pareto front for ZDT test functions using different p values when using NSGA-II, MOCell and SPEA2, respectively. The *hit rate* is shown in Tables 4, 5 and 6 for NSGA-II, MOCell and SPEA2, respectively. In addition, the box plots for the GD and HV metrics are shown in Figures 2, 3 and 4 for NSGA-II, MOCell and SPEA2, respectively.

p	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
0.0	12100_{900}	19050_{1200}	11150_{700}	22200_{5400}	23200_{1200}
0.1	12200_{800}	19300_{1200}	11200_{900}	21050_{5000}	23700_{1100}
0.2	12300_{1000}	19900_{1300}	11400_{900}	22200_{5500}	24350_{900}
0.3	12700_{800}	20350_{1200}	11600_{700}	21800_{4600}	24750_{800}
0.4	12800_{900}	20800_{1100}	11800_{800}	21550_{5000}	25000_{300}
0.5	13100_{800}	21400_{1400}	11900_{900}	21650_{5700}	-
0.6	13300_{1100}	21900_{1400}	12200_{900}	21850_{5900}	-
0.7	13450_{1100}	22600_{1800}	12400_{1000}	23500_{4900}	-
0.8	13800_{900}	23150_{1200}	12450_{800}	23250_{5200}	-
0.9	13800_{1000}	24000_{1200}	12700_{900}	21350_{4700}	-
1.0	14200_{1000}	24350_{1600}	12800_{1200}	22000_{5500}	-

Table 1. Function evaluations ($\tilde{x}_{IQR}$) for NSGA-II solving ZDT MOPs using different values of p

- ZDT1: All algorithms converge faster (minimum number of function evaluations) when the non-highly disruptive polynomial mutation (i.e. $p = 0.0$) is

[2] p is the error in this context.

p	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
0.0	10250_{1100}	14150_{13600}	10800_{1300}	18450_{10500}	15600_{900}
0.1	10600_{900}	10600_{4000}	10800_{1200}	16500_{4700}	15900_{1100}
0.2	11000_{900}	12100_{5100}	11150_{1400}	16200_{4300}	16550_{900}
0.3	11200_{1000}	11100_{5700}	11400_{1400}	16550_{5500}	17000_{1100}
0.4	11300_{900}	12800_{4700}	11650_{1200}	17050_{5000}	17750_{1100}
0.5	11600_{1300}	11400_{6000}	11800_{1300}	17650_{5100}	18400_{900}
0.6	11800_{1000}	12400_{4500}	11800_{1300}	16900_{4600}	18900_{1300}
0.7	12200_{1300}	13500_{5400}	12300_{1200}	16800_{6400}	19600_{1400}
0.8	12400_{1000}	14100_{5800}	12450_{1300}	15900_{3900}	20400_{1400}
0.9	12900_{1300}	14600_{5100}	12600_{1500}	17050_{4600}	20800_{1200}
1.0	13000_{800}	15650_{5500}	13000_{1400}	17650_{5700}	21700_{1400}

Table 2. Function evaluations ($\tilde{x}_{IQR}$) for MOCell solving ZDT MOPs using different values of p

p	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
0.0	13800_{800}	20700_{1400}	13500_{1000}	24050_{4200}	-
0.1	14050_{1100}	20800_{1200}	13650_{900}	23700_{3900}	-
0.2	14100_{900}	21200_{1400}	13950_{1100}	23400_{3400}	-
0.3	14300_{900}	21650_{1500}	13950_{1000}	23950_{3800}	-
0.4	14400_{900}	22100_{1400}	14200_{1000}	23850_{3500}	-
0.5	14700_{900}	22400_{1200}	14500_{1000}	23700_{4000}	-
0.6	15100_{1000}	23050_{1600}	14500_{1200}	23850_{3200}	-
0.7	15300_{700}	23600_{1400}	14800_{800}	24550_{3600}	-
0.8	15500_{1100}	24050_{1200}	15100_{1100}	23700_{3900}	-
0.9	15950_{1000}	24700_{1000}	15200_{1100}	24050_{3400}	-
1.0	16000_{900}	24900_{700}	15500_{1000}	24200_{3200}	-

Table 3. Function evaluations ($\tilde{x}_{IQR}$) for SPEA2 solving ZDT MOPs using different values of p

used. The *convergence speed* progressively decrease (linearly) as p increases towards 1.0. The *hit rate* is identical for all algorithms and p values. The "✓" symbol indicates that the problem was solved in the 100 independent runs executed. Regarding the GD and HV metrics it is clear that the best result is achieved when $p = 0.0$.

- ZDT2: For NSGA-II and SPEA2 the *convergence speed* decreases progressively as p increases. However, MOCell gave different speeds for p values from 0.0 to 0.5. As p increases beyond 0.5 then speed decreases progressively. The *hit rate* for NSGA-II is 100 % success for $p = 0.1 \ldots 0.6$, then it decreases progressively as p increases. However, for $p = 0.0$ it is 0.98 which is almost a 100 % success but observing the GD and HV box plots in Figure 2 shows few outliers. This could be caused by failing in escaping from a local optima or the convergence of the entire population to one point. For $p \geq 0.1$ the GD and HV metrics decrease as p increases. This suggests that the non-highly disruptive polynomial (with

p	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
0.0	✓	0.98	✓	0.76	✓
0.1	✓	✓	✓	0.8	0.97
0.2	✓	✓	✓	0.7	0.85
0.3	✓	✓	✓	0.81	0.69
0.4	✓	✓	✓	0.77	0.42
0.5	✓	✓	✓	0.75	0.23
0.6	✓	✓	✓	0.73	0.04
0.7	✓	0.99	✓	0.65	0.01
0.8	✓	0.95	✓	0.64	-
0.9	✓	0.85	✓	0.82	-
1.0	✓	0.68	✓	0.74	-

Table 4. Hit rate for NSGA-II solving ZDT MOPs using different values of p

p	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
0.0	✓	0.61	✓	0.65	✓
0.1	✓	✓	0.99	0.96	✓
0.2	✓	✓	0.99	0.99	✓
0.3	✓	✓	0.99	0.99	✓
0.4	✓	✓	0.99	0.94	✓
0.5	✓	✓	✓	0.96	✓
0.6	✓	✓	✓	0.95	✓
0.7	✓	✓	✓	0.94	✓
0.8	✓	✓	0.99	✓	✓
0.9	✓	✓	0.97	✓	✓
1.0	✓	✓	0.99	0.95	✓

Table 5. Hit rate for MOCell solving ZDT MOPs using different values of p

p	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
0.0	✓	0.85	0.99	0.57	0.03
0.1	✓	0.99	✓	0.65	0
0.2	✓	✓	0.99	0.63	0
0.3	✓	0.99	✓	0.61	0
0.4	✓	0.96	✓	0.63	0
0.5	✓	0.99	✓	0.59	0
0.6	✓	0.98	✓	0.59	0
0.7	✓	0.9	✓	0.58	0
0.8	✓	0.9	✓	0.63	0
0.9	✓	0.59	✓	0.58	0
1.0	✓	0.54	✓	0.65	0

Table 6. Hit rate for SPEA2 solving ZDT MOPs using different values of p

$p = 0.0$) is neither resilient to local optima nor prone to premature convergence. In fact this is even clearer for MOCell and SPEA2 as their *hit rates* for $p = 0.0$ are 0.61 and 0.85, respectively. It seems that MOCell is very sensitive to the non-highly disruptive mutation and prefers big jumps in the decision variable space which is possible when using $p \geq 0.1$. The box plots for GD and HV metrics shown in Figure 3 confirm this as there are many outlier values when using $p = 0.0$. SPEA2 also has problems with $p = 0.0$ as shown in Figure 4.

- ZDT3: Again as in ZDT1, the non-highly disruptive polynomial mutation ($p = 0.0$) achieves the fastest *convergence speed* and best GD and HV metrics for all algorithms. The *hit rate* is 100 % for NSGA-II and almost 100 % for MOCell and SPEA2 for all p values. In summary, all algorithms for the *convergence speed*, GD and HV metrics progressively decrease as p increases.

- ZDT4: This is a multi-modal problem and large jumps in the decision space are desirable. By studying the *convergence speed* and *hit rate* no major difference can be inferred for $p \geq 0.1$ for all algorithms. However, for $p = 0.0$ MOCell's *speed* is worst and best when $p = 0.2$. Also the *hit rate* is worst when $p = 0.0$. There is a clear difference between $p = 0.0$ and $p \geq 0.1$. This observation can only be understood after studying the box plots for all algorithms. When $p = 0.0$ there are few outlier values which is an indication of falling in local optima and of premature convergence. In other words, all algorithms failed in solving ZDT4 using the non-highly disruptive polynomial mutation in few of the 100 independent executions. Nonetheless, observing the high IQR values for the function evaluations with $p = 0.0$ in Table 2 indicate that MOCell had problems with ZDT4 using the non-highly disruptive mutation.

- ZDT6: The "-" symbol indicates that it was not possible to generate an acceptable front with the given function evaluations. This was the case for ZDT6 only when using $p = 0.4 \ldots 1.0$ for NSGA-II and for all p values for SPEA2. Also regarding the GD and HV metrics both progressively decrease for NSGA2 and SPEA2 as p increases. MOCell managed to have a 100 % success for all p values; it is interesting how function evaluations increase as p values increase. For example the median of needed FEs to generate an acceptable PF_{known} for $p = 0.0$ and 1.0 were 15 600 and 21 700, respectively. This indicates that MOCell needed another 6 000 FEs when it used the highly-disruptive polynomial mutation to be able to generate an acceptable PF_{known}.

In summary, the non-highly disruptive outperformed the highly disruptive polynomial mutation but has a high risk of failed runs as seen in ZDT2 and ZDT4. The difference in performance is substantial. The proposed hybrid polynomial mutation can generate very competitive results to the non-highly disruptive mutation and at the same it is prone to local optima and more resilient to premature convergence.

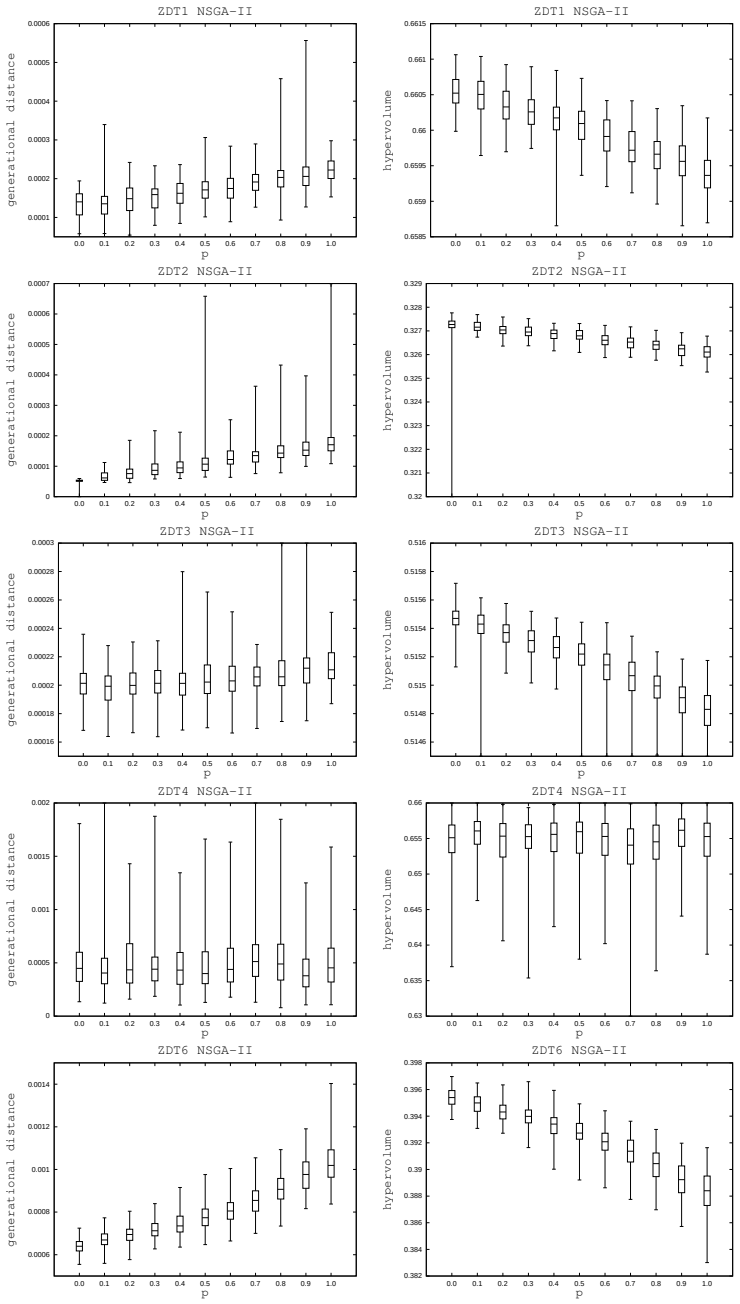

Fig. 2. Box plots for the generational distance (left) and hypervolume metrics using NSGA-II

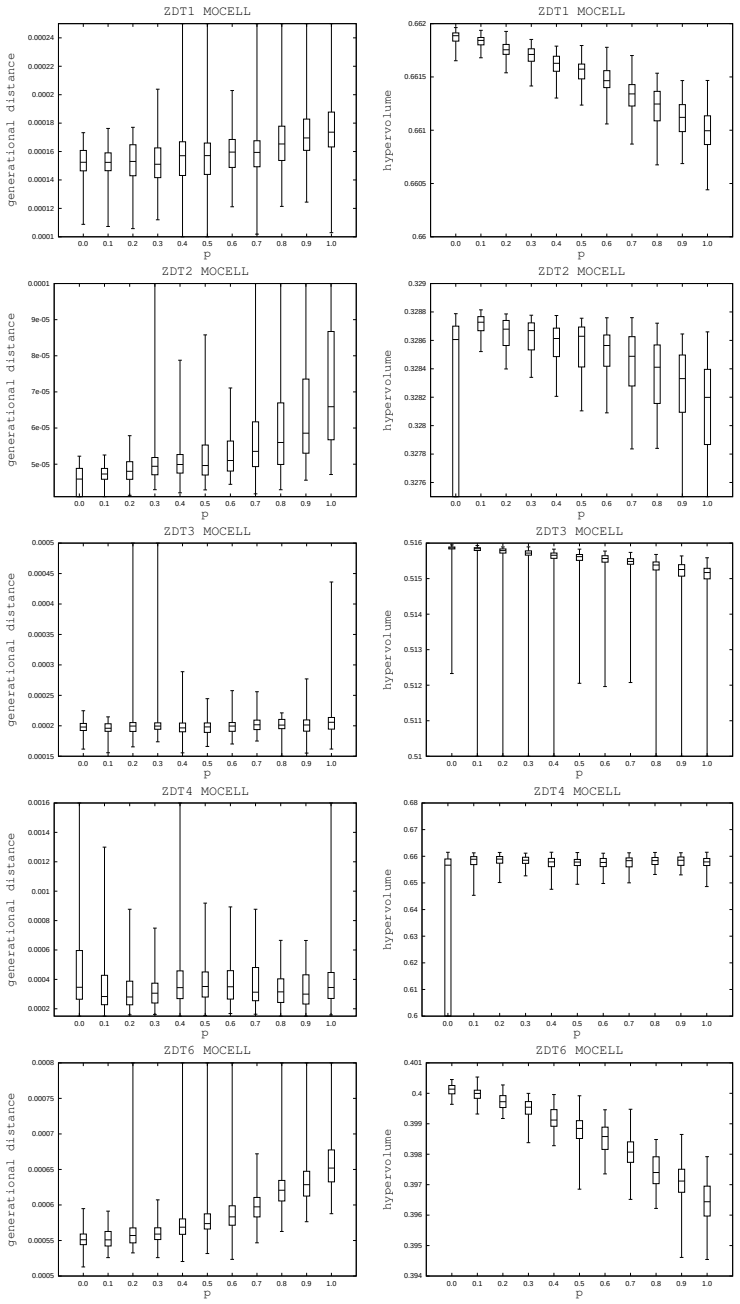

Fig. 3. Box plots for the generational distance (left) and hypervolume metrics using MOCELL

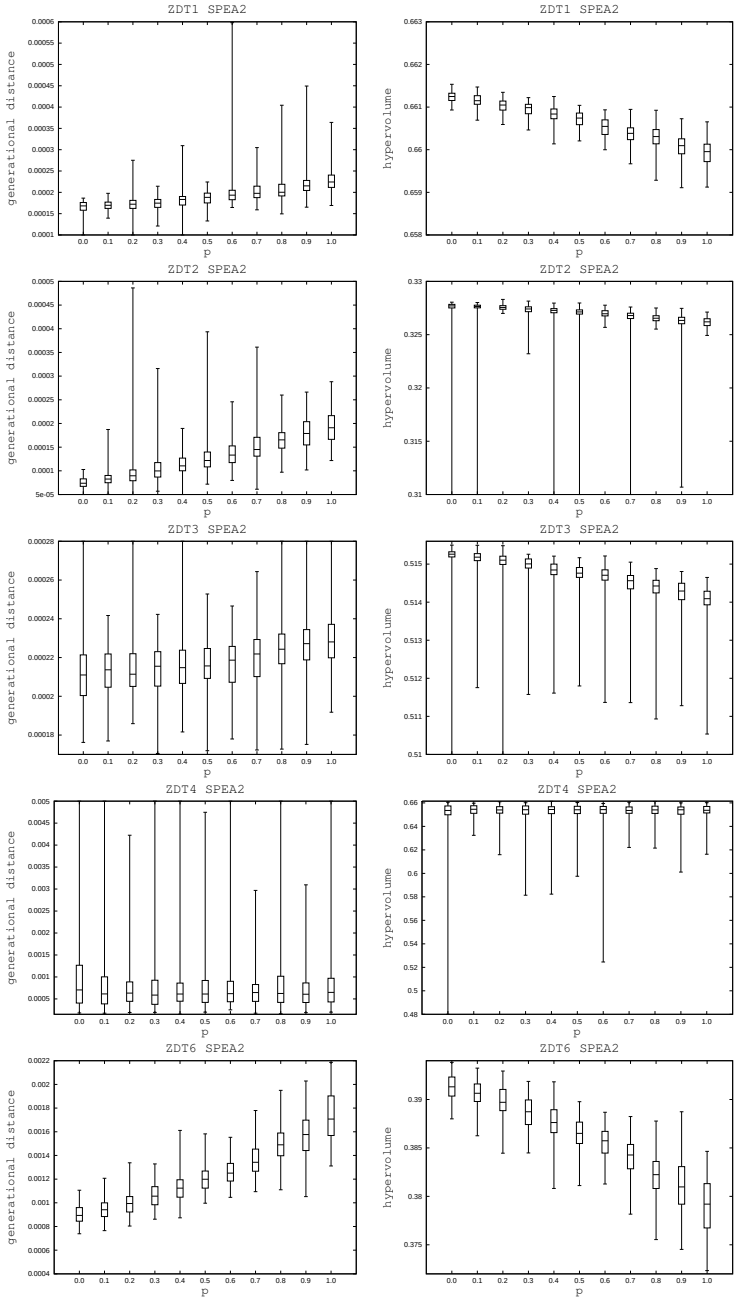

Fig. 4. Box plots for the generational distance (left) and hypervolume metrics using SPEA2

5 CONCLUSIONS

The study in the paper has looked at two well known variants of polynomial mutation used by various researchers interchangeably. The benefits of both variants are proposed in a hybrid mutation where the disruptive-level can be managed by the user. The original implementation of polynomial mutation (non-highly disruptive) can improve all performance metrics of all algorithms used in this study significantly for 4 out of 5 of the test problems used. However, it suffers from premature convergence as seen in ZDT2 and ZDT4. Therefore, it is better to use the new hybrid polynomial mutation with $0.2 \leq p \leq 0.4$. This will give better chances of escaping from local optima and guaranteeing the best possible convergence and hypervolume metrics. This means that the mutation will normally use small jumps but occasionally enables large jumps to be able to escape from local optima and make the algorithm less prone to premature convergence.

In future work, it might be possible to look at how to choose the p value at run-time according to the performance of the system. This requires collecting the population metrics such as hypervolume and checking for improvement every few iterations. If there is no improvement, then the algorithm changes the p value.

6 APPENDIX

- ZDT1:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= g(x)\left[1 - \sqrt{x_1/g(x)}\right] \\ g(x) &= 1 + 9\sum_{i=2}^{n} x_i/(n-1) \end{aligned}$$

$0 \leq x_i \leq 1, i = 1, \ldots, 30$

- ZDT2:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= g(x)\left[1 - (x_1/g(x))^2\right] \\ g(x) &= 1 + 9\sum_{i=2}^{n} x_i/(n-1) \end{aligned}$$

$0 \leq x_i \leq 1, i = 1, \ldots, 30$

- ZDT3:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= g(x)\left[1-\sqrt{x_1/g(x)}-(x_1/g(x))\sin(10\pi x_1)\right] \\ g(x) &= 1+9\sum_{i=2}^{n} x_i/(n-1) \end{aligned}$$

$0 \le x_i \le 1, i = 1, \ldots, 30$

- ZDT4:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= g(x)\left[1-\sqrt{x_1/g(x)}\right] \\ g(x) &= 1+10(n-1)+\sum_{i=2}^{n}[{x_i}^2-10\cos(4\pi x_i)] \end{aligned}$$

$x_1 \in [0, 1], -5 \le x_i \le 5, i = 2, \ldots, 30$

- ZDT6:

$$\begin{aligned} f_1(x) &= 1-\exp(-4\pi x_1)\sin^6(4\pi x_1) \\ f_2(x) &= g(x)\left[1-(x_1/g(x))^2\right] \\ g(x) &= 1+9\left[\sum_{i=2}^{n} x_i/(n-1)\right]^{0.25} \end{aligned}$$

$0 \le x_i \le 1, i = 1, \ldots, 30$

Acknowledgment

The author would like to thank DFG for supporting the research visit.

REFERENCES

[1] Coello, C.—van Veldhuizen, D.—Lamont, G.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[2] Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Chichester, UK, 2001.

[3] Deb, K.—Agrawal, R.: Simulated Binary Crossover for Continuous Search Space. Complex Systems, Vol. 9, 1995, pp. 115–148.

[4] Deb, K.—Goyal, M.: A Combined Genetic Adaptive Search (Geneas) for Engineering Design. Computer Science and Informatics, Vol. 26, 1996, No. 4, pp. 30–45.

[5] Deb, K.—Pratap, A.—Agarwal, S.—Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA, II. IEEE Transactions on Evolutionary Computation, Vol. 6, April 2002, No. 2, pp. 182–197.

[6] Deb, K.—Tiwari, S.: Omni-Optimizer: A Generic Evolutionary Algorithm for Single and Multi-Objective Optimization. European Journal of Operational Research, Vol. 185, 2008.

[7] Durillo, J.—Nebro, A.—Coello, C.—Luna, F.—Alba, E.: A Comparative Study of the Effect of Parameter Scalability in Multi-Objective Metaheuristics. In IEEE Congress on Evolutionary Computing, Hong Kong, June 2008.

[8] Durillo, J.—Nebro, A.—Luna, F.—Dorronsoro, B.—Alba, E.: jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, December 2006.

[9] Eiben, A. E.—Smith, J. E.:. Introduction to Evolutionary Computing. Springer, Natural Computing Series, 2nd edition, 2007.

[10] Nebro, A. et al.: The Deme Project. [URL] `http://neo.lcc.uma.es/Software/deme/html/`. Accessed December 2008.

[11] Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company, Inc., 1989.

[12] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, 3rd Edition, 1996.

[13] Nebro, A.—Durillo, J.—Luna, F.—Dorronsoro, B.—Alba, E.: A Cellular Genetic Algorithm for Multiobjective Optimization. In David A. Pelta and Natalio Krasnogor (Eds.): Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO 2006), pp. 25–36, Granada, Spain 2006.

[14] Sheskin, D.: Handbook of Parametric and Nonparametric Statistical Procedures. CRC Press, 4th edition, 2007.

[15] Srinivas, N.—Deb, K.: Multi-Objective Function Optimization Using Non-dominated Sorting Genetic Algorithms. Evolutionary Computation, Vol. 2, 1995, pp. 221–248.

[16] van Veldhuizen, D.—Lamont, G.: Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1998.

[17] Zitzler, E.—Deb, K.—Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computing, Vol. 8, 2000, No. 2, pp. 173–195.

[18] Zitzler, E.—Laumanns, M.—Thiele, L.: Spea2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich (Switzerland) 2001.

[19] Zitzler, E.—Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation, Vol. 3, November 1999, No. 4, pp. 257–271.

Mohammad Hamdan received his Ph. D. in computer science from Heriot-Watt University/UK in 2000. He also obtained an M. Sc. in knowledge based systems from same university in 1994. Both degrees were supported by a grant from British Council. Was appointed as an Assistant Professor in the Department of Computer Science at Yarmouk University in Jordan in July 2000. In September 2002 he became the Assistant Dean in the Faculty of Information Technology. In September 2005 he became the Chairman of Computer Science Department. He is a senior member in IEEE. Since January 2002 he became the Activity Secretary for Jordan IEEE executive committee and in May 2006 he became the chair for IEEE Computer/Computational Intelligence Chapter in Jordan. Worked as project coordinator for TEMPUS JEP project from September 2007 till December 2009. Also he is the programme coordinator for joint M. Sc. programmes between Yarmouk and Sunderland Universities since October 2002. He became an Associate Professor of Computer Science in August 2010.