Changing or Keeping Solutions in Dynamic Optimization Problems with Switching Costs[*]

Danial Yazdani
Department of Maritime and Mechanical Engineering, Liverpool John Moores University
Liverpool, United Kingdom
d.yazdani@2016.ljmu.ac.uk
danial.yazdani@gmail.com

Jürgen Branke
Warwick Business School, University of Warwick
Coventry, United Kingdom
juergen.branke@wbs.ac.uk

Mohammad Nabi Omidvar
School of Computer Science, University of Birmingham
Birmingham, United Kingdom
m.omidvar@cs.bham.ac.uk

Trung Thanh Nguyen
Department of Maritime and Mechanical Engineering, Liverpool John Moores University
Liverpool, United Kingdom
t.t.nguyen@ljmu.ac.uk

Xin Yao
School of Computer Science, University of Birmingham
Birmingham, United Kingdom
Department of Computer Science and Engineering, SUSTech
Shenzhen, China
x.yao@cs.bham.ac.uk

ABSTRACT

Dynamic optimization problems (DOPs) are problems that change over time. However, most investigations in this domain are focused on tracking moving optima (TMO) without considering the cost of switching from one solution to another when the environment changes. Robust optimization over time (ROOT) tries to address this shortcoming by finding solutions which remain acceptable for several environments. However, ROOT methods change solutions only when they become unacceptable. Indeed, TMO and ROOT are two extreme cases in the sense that in the former, the switching cost is considered zero and in the latter, it is considered very large. In this paper, we propose a new semi ROOT algorithm based on a new approach to switching cost. This algorithm changes solutions when: 1) the current solution is not acceptable and 2) the current solution is still acceptable but algorithm has found a better solution and switching is preferable despite the cost. The main objective of the proposed algorithm is to maximize the performance based on the fitness of solutions and their switching cost. The experiments are done on modified moving peaks benchmark (mMPB) and the performance of the proposed algorithm alongside state-of-the-art ROOT and TMO methods is investigated.

KEYWORDS

Dynamic optimization problems, multi-swarm methods, particle swarm optimization, robust optimization over time, switching cost

ACM Reference Format:
Danial Yazdani, Jürgen Branke, Mohammad Nabi Omidvar, Trung Thanh Nguyen, and Xin Yao. 2020. Changing or Keeping Solutions in Dynamic Optimization Problems with Switching Costs. In *Proceedings of ACM Conference.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[*]Produces the permission block, and copyright information

ACM Conference, July 2018, Kyoto, Japan
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Many real-world problems are dynamic and changing over time. Most previous research on dynamic optimization problems (DOP) focuses on tracking moving optima (TMO) [17]. In TMO, the algorithm assumes that the solution can be changed for each environment without considering any switching cost and/or any resource limitation for changing solutions. In fact, changing solutions in real-world problems is costly. Furthermore, larger changes have more cost and need more resources such as time, human resources and energy. Thus, lack of switching cost consideration in TMO algorithms makes them unsuitable for many of real-world problems. For addressing this issue, another approach called robust optimization over time (ROOT) [30] was proposed in which algorithms search for solutions that are not necessary the best but they are acceptable and can remain acceptable after environmental changes. Therefore, in ROOT, algorithms try to find solutions which are robust to environmental changes and the main objective is maximizing survival time of robust solutions over time [8].

In fact, TMO and ROOT address two extreme cases. TMO is suitable for circumstances in which there is no switching cost or it is very low. On the other hand, ROOT is suitable for situations in which the switching cost is very high so the algorithm tries to keep each solution as long as it remains acceptable after environmental changes.

In this paper, a new adaptive solution chooser (ASC) algorithm is proposed which acts similar to TMO algorithms where switching cost is low and acts as ROOT algorithms when the switching cost is high. However, the main contribution of ASC is where the algorithm can decide about changing or keeping solutions based on their current fitness values, the fitness of other found solutions with better quality and their switching cost from the current solution. Indeed, although changing solutions in real-world problems is costly, there are situations in which the algorithm has found a solution whose quality is so high that the benefit of switching largely outweighs the cost.

The remainder of this paper is structured as follows: in the Section 2, problem definitions are presented. A brief literature review is provided in Section 3. The proposed algorithm is introduced in Section 4. The experimental settings, results and analysis are shown in the Section 5. Finally, this paper is concluded with a summary.

2 DEFINITIONS

DOPs are usually represented as follows:

$$F(\vec{x}) = f(\vec{x}, \theta^{(t)}), \tag{1}$$

where f is the objective function, $\vec{x}$ is a design vector, $\theta^{(t)}$ is environmental parameters which change over time and t is the time index with $t \in [0, T]$ where T is the problem life cycle or number of environments. In this paper, like most previous studies in the DOP domain, we investigate DOPs with $\theta^{(t)}$ that changes discretely. In this type of DOP, the environmental parameters change over time with stationary periods between changes. As a result, for a DOP with T environmental changes, we have a sequence of T static environments that can be described as (2):

$$F(\vec{x}) = \left[f(\vec{x}, \theta^{(1)}), f(\vec{x}, \theta^{(2)}), \ldots, f(\vec{x}, \theta^{(T)})\right], \tag{2}$$

where $\theta^{(i)}$ is the environmental parameters in the i^{th} environment. TMO is the most popular methodology in the domain of DOPs [17]. However, in TMO, the optimizer changes the solution after each environmental change which is not applicable in many of real-world problems due to the expenses and limitations involved with changing solutions [28]. In fact, TMOs are more suitable for problems in which the switching cost is zero or the system can tolerate frequent changes in solution. In TMO, the best case is happening when the best solution according to the fitness value is found for each environment.

In ROOT, the main goal is to minimize the number of times when the chosen solution has to be changed because its performance drops below an acceptable level, or to maximize the average number of environments that a robust solution remains acceptable. Therefore, ROOT algorithms try to keep solutions unchanged as much as they can which means they try to minimize switching cost but without considering any relation between switching cost and fitness values. Thus, in ROOT, the best case is that the first robust solution remains acceptable for all of the N environments and the worst case is that the number of robust solutions is equal to the number of environments.

3 RELATED WORK

Related work to the proposed algorithms can be classified into three groups namely multi-swarm methods for TMO, ROOT methods and methods considering switching cost.

3.1 Multi-swarm methods

Multi-swarm methods are one of the most famous algorithms to tackle DOPs [15, 17]. They consist of at least two sub-swarms handling different tasks or separate regions in the problem space.

In [6], Self Organizing Scouts (SOS) was proposed which utilized a big sub-population for global search and a number of small sub-populations for tracking changes of identified peaks. This strategy has also been proposed with other meta-heuristics such as PSO [13, 21, 24] and artificial fish swarm optimization [23, 26, 29].

In [19], a speciation method was used to split the population into sub-populations. In [2], a regression-based approach (RSPSO) was presented to enhance the convergence rate using speciation-based methods. Every subpopulation was confined to a hypersphere around the best solution.

In [14], a method based on clustering was proposed for developing sub-populations, which was simplified and further improved in [22]. In [7], a method called SPSO was proposed in which every cluster was divided into two. The first cluster was responsible for exploitation and the second one for exploration. Gaussian local search and differential mutation were used to improve diversity in the environment.

In [3], two multi-population methods, called MQSO and MCPSO, were proposed. In MQSO, quantum particles appear at random positions, uniformly distributed around the swarm's global best. In MCPSO, some or all of the particles in each swarm have a 'charge', and charged particles repel each other, leading to larger diversity. The population size is equal for every sub-swarm, and the number of sub-swarms is fixed and pre-determined. An anti-convergence method ensures continued search for possible better peaks. In addition, a mechanism called exclusion is used to avoid several swarms converging to the same peak. A version of MQSO with an adaptive number of sub-populations, called AMQSO, was proposed in [4]. AMQSO starts with one sub-population and a new sub-population is created if all previous sub-populations have converged. This method has significantly improved the performance.

Li et al [12] proposed a method to adapt the number of populations based on statistical data on how many populations have found new peaks. If this number is large, more populations will be introduced and vice versa. Additionally, a new heuristic clustering, a population hibernation scheme, a population exclusion scheme, a peak hiding method and two movement methods (to track peaks and avoid stagnant) were proposed.

A PSO with two types of sub-swarms called finder-tracker multi-swarm PSO was proposed in [25]. The finder swarm finds new uncovered peaks. When it converges to a peak, it creates a new tracker swarm to track the peak. An exclusion mechanism re-initializes the finder swarm if it converges to a peak that already has a tracker swarm on it. In addition, a mechanism to schedule tracker swarms called sleeping-awakening was proposed. It allocates more computational power to more promising swarms. Furthermore, a new method for re-diversification of tracker swarms (after a change) was proposed. The method re-initializes all particles randomly around Gbest [11] and their velocity vector is randomly set based on the peak's shift severity.

3.2 Robust Optimization Over Time

In [30], ROOT was proposed as a new perspective on DOPs. A new framework for ROOT was proposed in [10] with the algorithm searching for robust solutions by means of local fitness approximation and prediction. This method consists of a population-based optimization algorithm, a fitness approximator (to estimate fitness at any point in the search space), a fitness predictor (to predict future fitness values) and a database. In [10], an adapted radial-basis-function network (RBFN) is the local approximator and an autoregressive (AR) model is the predictor. A database was used for storing, in each iteration, all of the individuals' positions alongside their fitness values and the associated time of storage. This database was then used for approximating fitness values of solutions in previous environments which in turn was used for training the predictor.

In [8], authors proposed two new robustness definitions and metrics, namely survival time and average fitness. The survival time is the maximum time interval starting from time t during which the fitness value of the robust solution remains acceptable,

$$\mathcal{S}\left(\vec{x},\theta^{(t)},\delta\right) = \begin{cases} 0 & \text{if } f(\vec{x},\theta^{(t)}) < \delta \\ 1 + \max\{l \mid \forall i \in \{t,\ldots,t+l\} : f\left(\vec{x},\theta^{(i)}\right) \geq \delta\} & \text{otherwise} \end{cases} \tag{3}$$

where δ is a user defined threshold. In Eq. (3), for each environment, $\mathcal{S}$ shows for how many environments the fitness value of the current robust solution has remained above δ.

In [16], a new two-layer multi-objective method was proposed to find robust solutions that can maximize both survival time and average fitness. In [9], another multi-objective method was proposed to minimize switching cost and maximize survival time. A PSO algorithm was used as the optimizer. Additionally, the algorithm used the acceptance threshold for robust solutions similar to [8]. Euclidean distance between two solutions was used as the switching cost.

In [27], for the first time, a new ROOT method was proposed in which the algorithm did not use approximation and prediction like previous ROOT methods. This algorithm tried to track peaks and gather information about fitness variance of peaks after environmental changes. This algorithm choses the most reliable peak based on the current fitness of peaks and their fitness variances as the next solution.

3.3 Methods considering switching cost

There are only a few papers that consider switching cost. In [1, 9, 20, 28], switching cost was considered as an objective in multi-objective problems. However, all of these works considered the switching cost as a separable and independent objective from the optimality objective function and the connection between these two objectives was not considered. In [20], switching cost was investigated as optimization of adaptation. A multi-objective problem was defined which considered the cost of the adaptation and the optimality while the adaptation takes place. In [1], the need for rapid, low-cost changes in a design, in response to changes in performance requirements, within multi-objective problems, was investigated. An algorithm called ROOT/SC [9] was designed for ROOT. ROOT/SC is a multi-objective algorithm in which the first objective is survival time metric [8] and the second one is switching cost. Yazdani et al. [28] considered the time-linkage characteristic [18] of DOPs with switching cost. This work investigated DOPs with previous-solution restriction (PSDR) in which successive solutions should not be much different. However, the proposed algorithm in this paper choses a new solution for each environment even where the switching cost is high. Additionally, this work did not consider any connection and trade-off between the quality of solutions and switching costs.

4 PROPOSED ALGORITHM

In this section a new algorithm named adaptive solution chooser (ASC) is described. ASC tries to maintain a trade-off between TMO and ROOT characteristics based on the switching cost and fitness values of the current solution and other peaks. In this paper, the switching cost is defined based on the Euclidean distance as follows:

$$\mathrm{SC}(\vec{x}) = w \cdot \frac{\|\vec{x} - \vec{x}^*\|}{\sqrt{D}}, \tag{4}$$

where SC is switching cost, $\vec{x}$ is a design variable, $\vec{x}^*$ is the last chosen solution by the algorithm, D is dimension and $w \geq 0$ is a weight which controls the ratio between switching cost and fitness value. Moreover, by setting different values for w, we can simulate higher or lower switching costs. Note that increasing dimension results in increasing Euclidean distance values which leads to increase SC values. Therefore, the ratio between SC and fitness value changes in different dimensions that can be undesirable in experiments. Consequently, this is the reason behind dividing by $\sqrt{D}$ in Eq. (4) which makes SC values independent form dimension of the problem. On the other hand, w can be used for increasing SC values in higher dimensions if it is needed.

In ASC, a multi-swarm optimizer is responsible to find peaks, track them after environmental changes and calculate their fitness variance. This multi-swarm algorithm needs to continuously try to identify new peaks and tracks them after each environmental change. Knowledge about the problem such as number of peaks and their shift severities should not be necessary. Additionally, the algorithm should be able to adapt the number of populations as needed. For example, the proposed multi-swarm algorithms in [4, 26, 27] have such characteristics. Each sub-swarm which is tracking a peak needs to store the Euclidean distance between best found positions (such as Gbest in PSO [11]) at the end of each successive pair of environments. The average of these distances indicates the peak's Shift Severity. Moreover, the differences between fitness values of its best found positions before and after each environmental change. The average of these values indicates the variance of fitness values of the best found position after environmental changes which is denoted fitness variance.

We choose FTmPSO [25] as the multi-swarm method embedded in ASC. The major reasons behind this choice are its simplicity, competitiveness and compatibility with ASC. To make FTmPSO simpler, we do not use the exploiter particle and awakening-sleeping mechanisms proposed in its original paper. Additionally, to make it more realistic, we use the exclusion radius formula proposed in [4] for it and we use learned shift severities instead of the true shift that was used in the original paper.

At each environment, ASC determines the reliable peaks. Reliable peaks are peaks which are expected to remain acceptable after at least one environmental change. For determining reliable peaks, ASC uses the following formula:

$$\begin{cases} \text{if } f(\vec{x}_{\text{best},i}, \theta^{(t)}) - \gamma_i \geq \delta & \text{reliable} \\ \text{else} & \text{unreliable,} \end{cases} \tag{5}$$

where $\vec{x}_{\text{best},i}$ is the best position found by the i^{th} sub-swarm and γ_i is the fitness variance of the peak which is covered by i^{th} sub-swarm.

For each environment, after a predefined computational budget, ASC determines reliable peaks then there are three different possibilities:

(1) If the last chosen solution is not acceptable in the current environment t, i.e. $f(\vec{x}^*, \theta^{(t)}) < \delta$ then a new solution must be chosen from the reliable peaks as follows:

$$j = \text{argmin}_i \text{SC}(\vec{x}_{\text{best},i}), \tag{6}$$

then

$$\vec{x}^*_{\text{new}} = \vec{x}_{\text{best},j}, \tag{7}$$

where $i \in \{ReliablePeaks\}$ which are determined by Eq. (5). If there is no reliable peak, ASC chooses the best found solution.

(2) If the last chosen solution is still acceptable i.e. $f(\vec{x}^*) \geq \delta$ and if among peaks, there is at least one peak which has the following condition:

$$f(\vec{x}^*, \theta^{(t)}) < (f(\vec{x}_{\text{best},i}, \theta^{(t)}) - SC(\vec{x}_{\text{best},i})) \tag{8}$$

Then, the solution will be changed to $\vec{x}_{\text{best},i}$. If there are more than one reliable peak that have the condition in Eq. (8), the one with the lowest $SC(\vec{x}_{\text{best},i})$ will be chosen.

(3) If $f(\vec{x}^*, \theta^{(t)}) \geq \delta$ and there is no peak that has the condition in Eq. (8), then the previous solution will be kept for at least another environment.

5 EXPERIMENTS

5.1 Performance Indicator

For measuring performance, we propose

$$\text{Performance} = \frac{1}{T} \sum_{t=1}^{T} (F_t), \tag{9}$$

where

$$F_t = \begin{cases} f(\vec{x}^*, \theta^{(t)}) & \text{if previous solution is kept} \\ f(\vec{x}^*_{\text{new}}, \theta^{(t)}) - SC(\vec{x}^*_{\text{new}}) & \text{if a new solution is chosen,} \end{cases} \tag{10}$$

where $\vec{x}^*_{\text{new}}$ is a new chosen solution and T is number of environments. Therefore, the value of SC is decreased from the fitness value where the solution is changed. From another point of view, its cost is decreased from profit. Moreover, it can be seen as a penalty value.

Algorithm 1: ASC

```
1  Initialize multi-swarm method;
2  repeat
3      if an environmental change happens then
4          forall sub-swarms do
5              Update database;
6              Calculating shift severity and fitness variance;
7              Introducing diversity;
8              Update memory;
9      if the computational budget has been used up then
10         Determine reliable peaks by Eq. (5);
11         if $f(\vec{x}^*, \theta^{(t)}) < \delta$ then
12             Choose the next solution by Eq. (7);
13         else
14             if there are peaks with condition of Eq. (8) then
15                 Choose the one with minimum $SC(\vec{x}_{\text{best},i})$;
16             else
17                 Keep the current solution;
18     Execute an iteration of the multi-swarm method;
19 until stopping criterion is met;
```

5.2 Benchmark

The Moving Peaks Benchmark (MPB) [5] is the most popular benchmark in the DOP field. In its standard form, all peaks are behaving identical, so no solution is more robust than another and all peaks have almost similar behavior. This is why in ROOT, researchers used various modified versions [8–10, 27]. In this paper, we use the standard baseline function of MPB as follows:

$$f^{(t)}(\vec{x}) = \max_{i=1}^{m} \left\{ h_i^{(t)} - \left(w_i^{(t)} \cdot \left\| \vec{x} - \vec{c}_i^{(t)} \right\| \right) \right\}, \tag{11}$$

where m is the number of peaks, $\vec{x}$ is a solution in the problem space, $h_i^{(t)}$, $w_i^{(t)}$ and $\vec{c}_i^{(t)}$ are the height, width and center of the i^{th} peak in the t^{th} environment, respectively. In the modified version of MPB (mMPB) used in this paper, each peak has its own height, width and shift severities. The reason for having different height, width and shift severities for each peak is to have different behavior among them. The height, width and center of a peak change from one environment to the next is computed as follows:

$$h_i^{(t+1)} = h_i^{(t)} + \alpha_i \cdot \mathcal{N}(0,1), \tag{12}$$

$$w_i^{(t+1)} = w_i^{(t)} + \beta_i \cdot \mathcal{N}(0,1), \tag{13}$$

$$\vec{c}_i^{(t+1)} = \vec{c}_i^{(t)} + \vec{v}_i^{(t+1)}, \tag{14}$$

where

$$\vec{v}_i^{(t+1)} = s_i \cdot \frac{(1-\lambda) \cdot \vec{\mathcal{R}} + \lambda \cdot \vec{v}_i^{(t)}}{\left\| (1-\lambda) \cdot \vec{\mathcal{R}} + \lambda \cdot \vec{v}_i^{(t)} \right\|}, \tag{15}$$

where $\mathcal{N}(0,1)$ represents a random number drawn from a Gaussian distribution with mean 0 and variance 1, α_i is the height severity, β_i is the width severity , s_i is the shift severity of the i^{th} peak, $\mathcal{R}$

is a uniformly generated random vector $\in [-0.5, 0.5]$ and λ is the correlation coefficient.The parameter settings of the mMPBR are shown in Table 1.

Table 1: Parameter settings of mMPBR

Parameter	Value(s)
Number of peaks, m	5,20
Evaluations between changes, f	2500
Shift severity, s	Randomized in [0.5,3]
Height severity, α	Randomized in [1,15]
Width severity, β	Randomized in [0.1,1.5]
Peaks shape	Cone
Correlation coefficient, λ	0
Number of dimensions, D	2,5
Peaks location range, SR	[-50,50]
Peak height, h	[30,70]
Peak width, w	[1,12]
Initial height value	50
Initial width value	6
Number of environments, T	100

5.3 Algorithms and parameter settings

For comparison, we choose FTmPSO as a TMO method (TFTmPSO) which changes solutions to the best found position in each environments. Additionally, we use a ROOT version of FTmPSO (RFTmPSO) proposed in [27] as a ROOT method. Since ASC, TFTmPSO and RFTmPSO methods use the same multi-swarm method as core, the conclusion about performance of their different decision making procedure for choosing the next solution will not be affected by differences in the quality of finding and tracking peaks. Since in all of these three methods, the main task of the FTmPSO is finding and tracking peaks, it seemed appropriate to use the suggested parameter settings as in its original paper [25]. The parameter settings of FTmPSO in all three methods are shown in Table 2. All three algorithms choose solutions (if needed) at the end of each environments meaning the computational budget is f-1. Moreover, we assume that all algorithms will be informed about environmental changes happening. Change detection is another issue that can be dealt with separately, see e.g. [17].

5.4 Experimental results

All experimental results are obtained by performing 31 independent runs and the best results based on Wilcoxon signed-rank test with significance level of 0.05 are set in bold in each table. All experiments are done for three different fitness acceptance threshold $\delta \in \{40, 45, 50\}$ and five different values of $w \in \{0.1, 0.5, 1, 2, 3\}$ in Eq. (4) to simulate the impact of different levels of switching cost on the performance which is measured by Eq. (9). The median, mean and standard error of results are reported in Tables 3 and 4.

Tables 3 and 4 show the obtained results by algorithms on mMPB with 5 and 20 peaks. When w=0.1, the amount of switching cost obtained by Eq.(4) is smaller. Therefore, the problem is more suitable to be solved by TFTmPSO rather than RFTmPSO. As a result, the efficiency of the TFTmPSO which chooses the best solution for each environment is much better than of RFTmPSO. Additionally, obtained results by TFTmPSO are independent of δ. Indeed, RFTmPSO tries to keep solutions as long as they are larger than δ which is not useful for problems with small switching costs. In this situation, with growing δ value, the performance of RFTmPSO improves due to more frequent solution changing to better ones. In this situation ASC acts like a TFTmPSO because the possibility of having solutions with condition in Eq. (8) is high. Therefore, ASC's results are almost the same when we have different δ values.

Table 2: The parameter settings of FTmPSO inside the ASC, TFTmPSO and RFTmPSO

Parameter	Value
$C1, C2$	2.05
χ	0.729843788
$Trackers'\ population\ size$	5
$Finder's\ population\ size$	10
$Exclusion\ fatcor$	0.5
P	1
Q	1
$Convergence\ limit$	1
k	10
Stop criterion	Max fitness evaluation number

According to Tables 3 and 4, increasing w results in decreasing the performance of algorithms because of higher values of switching cost. However, TFTmPSO suffers more than the other two methods in this situation and its performance drops dramatically. The reason is that TFTmPSO changes solution every environment and this is detrimental if cost is large. In problems with higher switching cost, RFTmPSO outperforms TFTmPSO and the gap between their performances become larger as w increases. In fact, in this situation, the problem become more suitable to be solved by ROOT based methods in which solutions are kept as much as they remain acceptable.

ASC obtains the best results in comparison with RFTmPSO and TFTmPSO when switching cost is higher. Indeed, ASC is an adaptive algorithm which with growing switching cost tries to act more similar to ROOT based methods and less to TMO based algorithms. According to these tables, ASC outperforms TFTmPSO and RFTmPSO when $w \geq 0.5$. Surprisingly, ASC keeps its superiority over RFTmPSO even when w=3 in which ASC is expected to act similar to RFTmPSO. The first reason is that even with large w, it is still possible to have solutions with condition in Eq. (8) which can increase the performance of ASC. The second reason is their different strategies for choosing a solution when the current one is not acceptable anymore. Both RFTmPSO and ASC choose solutions from reliable peaks. RFTmPSO chooses the best reliable peak in terms of fitness function. However, ASC chooses the one with lowest switching cost which leads to improved performance of ASC under circumstances with large switching cost.

Different values of δ affect the performance of ASC (except where w=0.1) and RFTmPSO . When w=0.1, ASC acts independent from δ but RFTmPSO obtains better results when δ is higher. The

Table 3: Obtained results by Eq.(9) by TFTmPSO, RFTmPSO and ASC on mMPB with 5 peaks.

			2 Dimensional					5 Dimensional				
δ	Alg.	Stats.	w=0.1	w=0.5	w=1	w=2	w=3	w=0.1	w=0.5	w=1	w=2	w=3
40	TFTmPSO	Median	**61.32**	53.97	46.57	31.11	14.89	**61.30**	54.58	45.67	29.83	14.12
		Mean	**61.04**	54.28	45.83	28.93	12.03	**61.00**	54.41	46.18	29.71	13.25
		StdErr	**0.27**	0.47	0.87	1.73	2.61	**0.31**	0.48	0.80	1.51	2.23
	RFTmPSO	Median	52.45	49.93	47.09	41.40	35.92	51.92	48.77	45.87	38.09	31.05
		Mean	52.63	49.99	46.69	40.09	33.49	52.07	49.09	45.37	37.92	30.47
		StdErr	0.23	0.33	0.52	0.97	1.43	0.20	0.35	0.59	1.10	1.62
	ASC	Median	**61.29**	**57.38**	**54.31**	**48.79**	**45.05**	**60.38**	**56.79**	**54.71**	**50.52**	**45.83**
		Mean	**60.68**	**57.02**	**54.24**	**49.04**	**44.00**	**60.43**	**56.61**	**54.17**	**49.30**	**44.08**
		StdErr	**0.28**	**0.40**	**0.54**	**0.87**	**1.23**	**0.34**	**0.50**	**0.69**	**1.15**	**1.64**
45	TFTmPSO	Median	**61.32**	53.97	46.57	31.11	14.89	**61.30**	54.58	45.67	29.83	14.12
		Mean	**61.04**	54.28	45.83	28.93	12.03	**61.00**	54.41	46.18	29.71	13.25
		StdErr	**0.27**	0.47	0.87	1.73	2.61	**0.31**	0.48	0.80	1.51	2.23
	RFTmPSO	Median	54.92	51.51	47.40	38.95	30.30	54.44	50.76	46.76	38.87	30.10
		Mean	54.89	51.49	47.24	38.74	30.24	54.47	50.73	46.05	36.70	27.35
		StdErr	0.19	0.31	0.56	1.11	1.66	0.18	0.37	0.70	1.40	2.10
	ASC	Median	**60.69**	**57.28**	**54.32**	**47.77**	**41.33**	**60.43**	**56.55**	**52.85**	**46.22**	**39.42**
		Mean	**60.58**	**56.95**	**53.36**	**46.34**	**39.26**	**60.47**	**56.50**	**52.69**	**45.12**	**37.38**
		StdErr	**0.28**	**0.43**	**0.69**	**1.22**	**1.77**	**0.35**	**0.52**	**0.82**	**1.47**	**2.11**
50	TFTmPSO	Median	**61.32**	53.97	46.57	31.11	14.89	**61.30**	54.58	45.67	29.83	14.12
		Mean	**61.04**	54.28	45.83	28.93	12.03	**61.00**	54.41	46.18	29.71	13.25
		StdErr	**0.27**	0.47	0.87	1.73	2.61	**0.31**	0.48	0.80	1.51	2.23
	RFTmPSO	Median	57.52	52.99	46.98	35.65	24.32	57.11	52.23	46.66	35.73	25.33
		Mean	57.25	52.90	47.46	36.59	25.72	57.00	52.39	46.62	35.08	23.54
		StdErr	0.20	0.30	0.55	1.10	1.66	0.22	0.38	0.71	1.41	2.13
	ASC	Median	**61.01**	**56.52**	**52.78**	**42.97**	**33.09**	**60.68**	**56.33**	**51.07**	**40.24**	**29.62**
		Mean	**60.73**	**56.54**	**51.68**	**41.95**	**32.25**	**60.63**	**56.31**	**51.22**	**40.82**	**30.25**
		StdErr	**0.27**	**0.44**	**0.72**	**1.32**	**1.91**	**0.32**	**0.56**	**0.91**	**1.63**	**2.33**

reason is that when δ is higher, solutions become unacceptable more frequently and RFTmPSO needs to change solutions to better ones and since switching cost is low, changing to better solutions improves its performance. On the other hand, when w is larger, by increasing δ the performance of ASC and RFTmPSO get worse because they need to change solutions more frequently which leads to suffer from more higher switching costs.

As can be seen in Tables 3 and 4, the results of all of the algorithms are better in mMPB with 20 peaks in comparison with 5 peaks. When the number of peaks is higher in mMPB, the possibility of having taller peaks is higher which leads to improve the performance when w is smaller, especially for ASC and TFTmPSO . Moreover, when w is higher, the performance of ASC and RFTmPSO are more dependent on the robustness of solutions. Therefore, having more peaks increases the possibility of having more reliable peaks by Eq. (5). In addition, when the number of peaks is low, there are large areas of low fitness because there are few peaks to cover these areas. As a result, the average solution quality is lower, and robust solutions can lose their quality more quickly. By increasing the number of peaks, the average survival time of solutions increases because peaks are likely to overlap and support robust solutions. For ASC, when the number of peaks is higher, the number of reliable peaks is higher as well. Therefore, the density of peaks in the landscape is higher, so the possibility of having reliable peaks closer to the current solution is higher which leads to decrease in switching cost when ASC chooses a solution by Eq. (7).

According to Tables 3 and 4, all of the algorithms obtain better results in 2-dimensional problems than in 5-dimensional ones. In fact, in higher dimensions, the problem becomes more challenging for the optimizer, so the efficiency of finding and tracking peaks is decreased which leads to have worse results.

6 CONCLUSION

Switching cost is an important aspect of dynamic optimization problems (DOPs); however, there are few works considering the switching cost into account during the optimization process. Most of investigations in dynamic optimization literature have been focused on tracking moving optima (TMO), which often pursued irrespective of the switching cost. Robust optimization over time (ROOT) addresses this shortcoming by keeping solutions as long as they remain acceptable. However, ROOT methods are not suitable for problems with smaller switching cost.

In this paper, an adaptive solution chooser (ASC) algorithm for dynamic optimization problems with switching costs was proposed. ASC behaves similar to TMO based algorithms where the switching cost is low and similar to ROOT based algorithms when the

Table 4: Obtained results by Eq.(9) by TFTmPSO , RFTmPSO and ASC on mMPB with 20 peaks.

			2 Dimensional					5 Dimensional				
δ	Alg.	Stats.	w=0.1	w=0.5	w=1	w=2	w=3	w=0.1	w=0.5	w=1	w=2	w=3
		Median	**64.91**	54.03	40.40	13.23	-13.95	**63.49**	53.26	40.10	14.11	-11.85
	TFTmPSO	Mean	**64.96**	53.94	40.17	12.63	-14.91	**63.65**	53.50	40.81	15.44	-9.94
		StdErr	**0.09**	0.31	0.62	1.24	1.85	**0.13**	0.36	0.68	1.35	2.02
		Median	54.36	52.49	50.33	46.01	41.39	53.86	51.75	49.04	43.70	38.53
40	RFTmPSO	Mean	54.63	52.78	50.46	45.83	41.21	53.76	51.59	48.88	43.46	38.05
		StdErr	0.22	0.28	0.42	0.73	1.06	0.17	0.20	0.34	0.68	1.03
		Median	**64.72**	**60.69**	**58.25**	**54.92**	**52.18**	**62.66**	**56.86**	**55.27**	**53.40**	**49.45**
	ASC	Mean	**64.73**	**60.86**	**58.05**	**54.83**	**52.17**	**62.94**	**57.09**	**55.44**	**52.54**	**49.16**
		StdErr	**0.14**	**0.27**	**0.37**	**0.40**	**0.47**	**0.20**	**0.39**	**0.44**	**0.57**	**0.83**
		Median	**64.91**	54.03	40.40	13.23	-13.95	**63.49**	53.26	40.10	14.11	-11.85
	TFTmPSO	Mean	**64.96**	53.94	40.17	12.63	-14.91	**63.65**	53.50	40.81	15.44	-9.94
		StdErr	**0.09**	0.31	0.62	1.24	1.85	**0.13**	0.36	0.68	1.35	2.02
		Median	56.91	54.19	51.42	45.05	38.44	56.23	52.92	49.29	41.55	33.97
45	RFTmPSO	Mean	56.82	54.26	51.06	44.65	38.25	56.13	53.19	49.51	42.15	34.80
		StdErr	0.14	0.22	0.41	0.84	1.28	0.16	0.23	0.41	0.82	1.23
		Median	**64.68**	**61.15**	**58.79**	**55.86**	**52.01**	**62.71**	**58.69**	**56.39**	**52.15**	**48.46**
	ASC	Mean	**64.63**	**61.03**	**58.83**	**55.20**	**51.67**	**63.00**	**58.34**	**56.30**	**52.14**	**47.58**
		StdErr	**0.15**	**0.22**	**0.25**	**0.40**	**0.57**	**0.20**	**0.37**	**0.47**	**0.73**	**1.00**
		Median	**64.91**	54.03	40.40	13.23	-13.95	**63.49**	53.26	40.10	14.11	-11.85
	TFTmPSO	Mean	**64.96**	53.94	40.17	12.63	-14.91	**63.65**	53.50	40.81	15.44	-9.94
		StdErr	**0.09**	0.31	0.62	1.24	1.85	**0.13**	0.36	0.68	1.35	2.02
		Median	59.40	55.70	51.34	42.70	33.60	58.24	54.44	50.02	41.15	31.94
50	RFTmPSO	Mean	59.23	55.51	50.85	41.54	32.23	58.23	54.33	49.46	39.71	29.96
		StdErr	0.13	0.24	0.47	0.95	1.44	0.08	0.26	0.58	1.24	1.89
		Median	**64.76**	**60.85**	**58.42**	**52.47**	**47.88**	**62.97**	**59.29**	**56.16**	**50.44**	**43.56**
	ASC	Mean	**64.69**	**61.00**	**58.19**	**52.76**	**47.26**	**63.08**	**59.27**	**56.10**	**49.53**	**42.60**
		StdErr	**0.12**	**0.25**	**0.43**	**0.79**	**1.15**	**0.19**	**0.36**	**0.60**	**1.09**	**1.61**

switching cost is high. ASC's core is a multi-swarm method which tracks peaks and calculates fitness variance of peaks that is used for determining reliable peaks in terms of robustness of solutions. ASC decided if a new solution is to be chosen or the previous one can be kept based on current solution's fitness values, the fitness of other found solutions with better quality and their switching cost from the current solution. The experimental results by a proposed performance indicator on modified moving peaks benchmark showed that ASC performed significantly better than two state-of-the-art methods for TMO and ROOT in problems with different levels of switching cost.

In fact, by proposing ASC, we try to bridge a gap between academic research and real-world problems in the field of DOPS. In contrary to TMO and ROOT which are addressing two extreme cases i.e. when switching cost is very small or very large, ASC makes decision about changing or keeping solutions according to the switching cost at any range.

ACKNOWLEDGEMENT

This work was supported in part by a Dean's Scholarship by Faculty of Engineering and Technology, LJMU, a Newton Institutional Links grant no. 172734213, funded by the UK BEIS and delivered by the British Council, a NRCP grant no. NRCP1617-6-125 delivered by Royal Academy of Engineering, two EPSRC grants (nos. EP/P005578/1 and EP/J017515/1). Xin Yao was supported by a Royal Society Wolfson Research Merit Award and Honda Research Institute Europe.

REFERENCES

[1] Gideon Avigad, Erella Eisenstadt, and Oliver Schuetze. 2010. Handling changes of performance requirements in multi-objective problems. *Journal of Engineering Design* 23, 8 (2010), 597–617.
[2] Stefan Bird and Xiaodong Li. 2007. Using regression to improve local convergence. In *IEEE Congress on Evolutionary Computation*. IEEE, 592–599.
[3] Tim Blackwell and Juergen Branke. 2006. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation* 10, 4 (2006), 459–472.
[4] Tim Blackwell, Juergen Branke, and Xiaodong Li. 2008. Particle swarms for dynamic optimization problems. In *Swarm Intelligence: Introduction and Applications*, Christian Blum and Daniel Merkle (Eds.). Springer, 193–217.
[5] Juergen Branke. 1999. Memory enhanced evolutionary algorithms for changing optimization problems. In *IEEE Congress on Evolutionary Computation*. IEEE, 1875–1882.
[6] Juergen Branke, Thomas Kaussler, Christian Smidt, and Hartmut Schmeck. 2000. A multi-population approach to dynamic optimization problems. In *Evolutionary Design and Manufacture*. 299–307.
[7] Weilin Du and Bin Li. 2008. Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences* 178, 15 (2008), 3096–3109.
[8] Haobo Fu, Bernhard Sendhoff, Ke Tang, and Xin Yao. 2013. Finding Robust Solutions to Dynamic Optimization Problems. In *Applications of Evolutionary Computation*, Vol. 7835. Lecture Notes in Computer Science, 616–625.

[9] Yuanjun Huang, Yongsheng Ding, Kuangrong Hao, and Yaochu Jin. 2017. A multi-objective approach to robust optimization over time considering switching cost. *Information Sciences* 394-395 (2017), 183–197.

[10] Yaochu Jin, Ke Tang, Xin Yu, Bernhard Sendhoff, and Xin Yao. 2013. A framework for finding robust optimal solutions over time. *Memetic Computing* 5, 01 (2013), 3–18.

[11] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *International Conference on Neural Networks*, Vol. 4. IEEE, 1942–1948.

[12] Changhe Li, Trung Thanh Nguyen, Ming Yang, Michalis Mavrovouniotis, and Shengxiang Yang. 2016. An adaptive multi-population framework for locating and tracking multiple optima. *IEEE Transactions on Evolutionary Computation* 20, 5 (2016), 590–605.

[13] Changhe Li and Shengxiang Yang. 2008. Optimization in dynamic environments utilizing a novel method based on particle swarm optimization. In *4th International Conference on Natural Computation*. IEEE, 624–628.

[14] Changhe Li and Shengxiang Yang. 2009. A clustering particle swarm optimizer for dynamic optimization. In *IEEE Congress on Evolutionary Computation*. IEEE, 439–446.

[15] Michalis Mavrovouniotis, Changhe Li, and Shengxiang Yang. 2017. A survey of swarm intelligence for dynamic optimization: Algorithms and applications. *Swarm and Evolutionary Computation* 33 (2017), 1–17.

[16] Yi nan Guo, Meirong Chen, Haobo Fu, and Yun Liu. 2014. Find robust solutions over time by two-layer multi-objective optimization method. In *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1528–1535.

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

[18] Trung Thanh Nguyen, Zaili Yang, and Stephen Bonsall. 2012. Dynamic Time-Linkage Problems – The Challenges. In *Computing and Communication Technologies, Research, Innovation, and Vision for the Future*. IEEE.

[19] Daniel Parrott and Xiaodong Li. 2006. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation* 10, 4 (2006), 440–458.

[20] Shaul Salomon, Gideon Avigad, Peter J. Fleming, and Robin C. Purshouse. 2013. Optimization of adaptation - a multi-objective approach for optimizing changes to design parameters. In *Evolutionary Multi-Criterion Optimization*, R.C. Purshouse, P.J. Fleming, C.M. Fonseca, and J. Shaw S. Greco (Eds.). Vol. 7811. Springer Lecture Notes in Computer Science, 21–35.

[21] Alireza Sepas-Moghaddam, Alireza Arabshahi, Danial Yazdani, and Mohammad Mahdi Dehshibi. 2012. A novel hybrid algorithm for optimization in multimodal Dynamic environments. In *International Conference on Hybrid Intelligent Systems (HIS)*. IEEE, 143–148. https://doi.org/10.1109/HIS.2012.6421324

[22] Shengxiang Yang and Changhe Li. 2010. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Transactions on Evolutionary Computation* 14, 06 (2010), 959–974.

[23] Danial Yazdani, Mohammad Reza Akbarzadeh-Totonchi, Babak Nasiri, and Mohammad Reza Meybodi. 2012. A new artificial fish swarm algorithm for dynamic optimization problems. In *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[24] Danial Yazdani, Babak Nasiri, Reza Azizi, Alireza Sepas-Moghaddam, and Mohammad Reza Meybodi. 2013. Optimization in dynamic environments utilizing a novel method based on particle swarm optimization. *International Journal of Artificial Intelligence* 11 (2013), 170–192.

[25] Danial Yazdani, Babak Nasiri, Alireza Sepas-Moghaddam, and Mohammad Reza Meybodi. 2013. A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. *Applied Soft Computing* 13, 04 (2013), 2144–2158.

[26] Danial Yazdani, Babak Nasiri, Alireza Sepas-Moghaddam, Mohammad Reza Meybodi, and Mohammadreza Akbarzadeh-Totonchi. 2014. mNAFSA: A novel approach for optimization in dynamic environments with global changes. *Swarm and Evolutionary Computation* 18 (2014), 38–53.

[27] Danial Yazdani, Trung Thanh Nguyen, Juergen Branke, and Jin Wang. 2017. A new multi-swarm particle swarm optimization for robust optimization over time. In *Applications of Evolutionary Computation*, G. Squillero and K. Sim (Eds.). Vol. 10200. Springer Lecture Notes in Computer Science, 99–109.

[28] Danial Yazdani, Trung Thanh Nguyen, Juergen Branke, and Jin Wang. 2018. A Multi-Objective Time-Linkage Approach for Dynamic Optimization Problems with Previous-Solution Displacement Restriction. In *European Conference on the Applications of Evolutionary Computation*. Lecture Notes in Computer Science, Springer.

[29] Danial Yazdani, Alireza Sepas-Moghaddam, Atabak Dehban, and Nuno Horta. 2016. A Novel Approach for Optimization in Dynamic Environments Based on Modified Artificial Fish Swarm Algorithm. *International Journal of Computational Intelligence and Applications* 15, 2 (2016), 1650010–1650034.

[30] Xin Yu, Yaochu Jin, Ke Tang, and Xin Yao. 2010. Robust optimization over time - A new perspective on dynamic optimization problems. In *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–6.