A New Multi-swarm Particle Swarm Optimization for Robust Optimization Over Time

Danial Yazdani[1(✉)], Trung Thanh Nguyen[1], Juergen Branke[2], and Jin Wang[1]

[1] School of Engineering, Technology and Maritime Operations, Liverpool Logistics, Offshore and Marine Research Institute, Liverpool John Moores University, Liverpool, UK
D.yazdani@2016.ljmu.ac.uk, {T.T.Nguyen,J.Wang}@ljmu.ac.uk
[2] Warwich Business School, University of Warwick, Coventry, UK
Juergen.Branke@wbs.ac.uk

Abstract. Dynamic optimization problems (DOPs) are optimization problems that change over time, and most investigations in this area focus on tracking the moving optimum efficiently. However, continuously tracking a moving optimum is not practical in many real-world problems because changing solutions frequently is not possible or very costly. Recently, another practical way to tackle DOPs has been suggested: robust optimization over time (ROOT). In ROOT, the main goal is to find solutions that can remain acceptable over an extended period of time. In this paper, a new multi-swarm PSO algorithm is proposed in which different swarms track peaks and gather information about their behavior. This information is then used to make decisions about the next robust solution. The main goal of the proposed algorithm is to maximize the average number of environments during which the selected solutions' quality remains acceptable. The experimental results show that our proposed algorithm can perform significantly better than existing work in this aspect.

Keywords: Robust optimization over time · Robust optimization · Dynamic optimization · Benchmark problems · Tracking moving optima · Particle swarm optimization · Multi-swarm algorithm

1 Introduction

Many real-world optimization problems are dynamic and changing over time. Most of previous studies in this domain focus on tracking the moving optimum (TMO) [1]. However, this is not practical in many cases since changing solutions may be very costly, and changing the solution frequently is not possible. As a result, there is a gap between academic research and real-world scenarios in this domain.

Recently, a new approach for solving DOPs was proposed to address the above concern that aims at finding solutions that are robust over the course of time [2]. A robust solution is one that is not necessarily the best solution in the environment, but at least is acceptable. A found robust solution can be utilized until its quality degrades to an unacceptable level in the current environment.

G. Squillero and K. Sim (Eds.): EvoApplications 2017, Part II, LNCS 10200, pp. 99–109, 2017.
DOI: 10.1007/978-3-319-55792-2_7

In case the current robust solution becomes unsatisfactory, a new robust solution must be chosen. Therefore, the task for addressing the DOPs in this approach is not to find the best solutions in each environment but to find robust solutions that can remain acceptable for a large number of environments. The process of finding such a sequence of robust solutions is referred to as robust optimization over time (ROOT) [2, 3].

In [2] ROOT was proposed as a new perspective on DOPs. In [4], a new framework for ROOT algorithms was proposed in which the algorithm searches for robust solutions by means of local fitness approximation and prediction. In this framework, an adapted radial-basis-function was used as the local approximator and an autoregressive model as the predictor. The metric in this framework uses the average of current fitness value of a solution, its p previous fitness values (by approximator) and its q future fitness values (by predictor) to search for robust solutions.

In [5], authors proposed two different robustness definitions and metrics, namely *survival time* and *average fitness*. The *survival time* is the maximum time interval in which the fitness of the robust solution remains acceptable, and the *average fitness* is the fitness value of the robust solution in a pre-defined time window. Then, two metrics and also performance indicators were defined based on these two definitions. In this framework, an autoregressive model is used as the predictor. In [12], a new multi-objective method was proposed to find robust solutions that can maximize both of *survival time* and *average fitness*.

In [6], some problem difficulties of ROOT were analyzed. Also, two different benchmark problems which one of them is specially designed for maximizing survival time and another benchmark is for maximizing average fitness, were proposed in [6].

In this paper, we propose a new algorithm for ROOT based on multi-swarm PSO. The main goal of this algorithm is to maximize the average number of environments that the robust solutions remain acceptable *i.e.,* we focus on the *survival time* definition of ROOT [5, 6]. The procedure of the proposed algorithm in finding robust solutions differs from previous works in several aspects. First, we have a multi-swarm PSO [8] that is responsible for not only finding and tracking optima as usual but also for gathering some information about peaks. Second, different to [4, 5], the fitness function for our proposed algorithm is the normal fitness function of the problem without involving any estimator. The proposed algorithm checks the robust solution at the end of each environment and if its fitness value is not acceptable, then a new robust solution is chosen. Third, for choosing a robust solution, the algorithm uses the gathered information to choose the most reliable *Gbest* [13] among PSO swarms as a position for the next robust solution. The results based on the average number of environments that robust solutions can remain acceptable show that the performance of our algorithm is substantially better than previous works in this domain.

The remainder of this paper is structured as follows. Section 2 presents the proposed algorithm. In Sect. 3, a new generic performance indicator is presented for ROOT and the experimental result, analysis and comparison with previous works are shown in this section. In the final section, we summarize the main findings and suggest directions for future work.

2 A New PSO Algorithm for Robust Optimization Over Time

In this section, a new algorithm based on Multi-swarm PSO [8] is proposed for ROOT. In the proposed algorithm, Multi-swarm PSO behaves in a similar way to previous multi-swarm algorithms proposed to track a moving optimum, *i.e.,* our algorithm tries to find all peaks and track them after each environmental change. However, while tracking peaks, our algorithm gathers information about the behavior of all peaks. Then, our algorithm uses this information to choose a robust solution on peaks that have the most suitable characteristics based on the aim to maximize the number of environments that a robust solution remains acceptable.

From another point of view, our algorithm predicts the robustness of solutions based on this information. In order to highlight the difference of our algorithm with previous works [4, 5], it is worth mentioning that in the previous frameworks, specific estimators were adopted and used in order to search for robust solutions. In addition, the fitness function for algorithms was based on average of these estimated values. But, our algorithm is different in that we use the normal fitness function for dynamic optimization and use information the algorithm has gathered to make decisions about which peak is the best for choosing a robust solution. So, in the proposed algorithm we do not use any specific estimator and the algorithm relies on previous behavior of peaks in order to predict future of candidate robust solutions on them.

In the proposed algorithm, there is a *Finder_swarm* that is responsible for searching for uncovered peaks. Additionally, there are *Tracker_swarms* that have two main tasks, namely tracking peaks and gathering information about the behavior of their covered peak. Each Tracker_swarm stores in its memory the difference between fitness value of the best found position in each environment with its fitness value after environmental change. The average of these values (named *Fit_drop*) shows how much the fitness value of points close to the top of the peak is expected to change after an environmental change. In addition, each *Tracker_swarm* stores the Euclidean distance between its *Gbest* at the end of successive environments, and the average of these distances reflect the *shift_severity* of each peak.

At the beginning, there is only one *Finder_swarm* in the problem space. After it converges to a peak [8], a new Tracker_swarm is created in place of the *Finder_swarm*, and the *Finder_swarm* is re-initialized to continue its global search for finding another possibly uncovered peak. On the other hand, the Tracker_swarm performs a local search for exploiting its peak and aims to reach the top of it.

In the proposed algorithm, exclusion mechanism [7] is used to avoid covering each peak by more than one *Tracker_swarm*. If the *Finder_swarm* converges to a covered peak (if the Euclidean distance between *Gbest* of *Finder_swarm* with *Gbest* of each *Tracker_swarm* is less than r_{excl} [8]), it will be re-initialized. Moreover, if the Euclidean distance between *Gbest* of two *Tracker_swarms* is less than a value r_{excl}, then the older swarm is kept because of its valuable memory and the other one is removed. However, if the *Gbest* fitness value of the newer swarm is better than that of the older one, the better *Gbest* information is copied to the older Tracker_swarm. Moreover, if both *Tracker_swarms* are of the same age, the one with better *Gbest* is kept.

```
Multi_swarm PSO for ROOT
Initialize Finder_ swarm
repeat:
    Re-evaluate Gbest Positions for change detection
    If the environmental change is detected then
        Update Fit_Drop
        If f(Robust_solution )<δ
            Choose the next Robust_Solution based on Eq.1
        End if
        Re-evaluate Pbest positions in Finder_swarm
        Update Shift _severity for each covered peak
        Re_diversify all Tracker_swarms
    End if
    Execute an iteration of PSO on Finder_swarm
    Execute Exclusion mechanism on Finder_swarm
    If the Finder_swarm is converged then
        create a new Tracker_swarm
        Re-initialize Finder_swarm
    End if
    Execute an iteration of PSO on all Trac k-er_swarms
    Execute Exclusion mechanism on each pair of Tracker_swarms
until stopping criterion is met
```

Fig. 1. The pseudocode of the proposed algorithm.

For change detection, the algorithm re-evaluates all *Gbest* positions of all *Tracker_swarms* in each iteration and if any obtained fitness value is different from the saved ones, then a change in the environment is detected. After change detection, first of all, all *Tracker_swarms* store the difference of the new fitness values of their *Gbest* with their saved value to obtain *Fit_drop*. After this, all *Pbest* positions [13] in *Finder_swarm* re-evaluate and a re-diversification mechanism [8] is done by all *Tracker_swarms* based on obtained *Shift_Severity* value for each peak.

In the proposed algorithm, robust solutions are chosen from the best found positions by all *Tracker_swarms* according to a decision making process based on current fitness value as well as *Fit_drop*. For making a decision about the current robust solution, its quality is checked at the end of each environment based on a user-defined lower bound threshold δ [5, 6]. Having this type of threshold is realistic in many real-world problems where there is a specification indicating the acceptability of a solution. If the fitness value of the robust solution is better than δ, then it is deemed acceptable and the robust solution is kept for at least another environment, otherwise the algorithm must choose a new robust solution.

The next robust solution will be a *Gbest* position of one of the *Tracker_swarms*. The best location for the next robust solution is a peak with the highest fitness value and lowest *Fit_drop*. In the proposed algorithm, the next robust solution position is placed on the *Gbest* of a *Tracker_swarm* which is chosen by Eq. 1:

$$C = argmax_{i=1}^{i=S}\left(f\left(Gbest_i\right) - Fit\_Drop_i\right) \tag{1}$$

where S is the number of *Tracker_swarms*, $f(Gbest_i)$ is the *Gbest* fitness value of the i^{th} *Tracker_swarm*, C is the index of a *Tracker_swarm* in which the next robust solution is located. This equation suggests how much the fitness value of a peak solution is dropped in the next environment, based on the difference between it's the current fitness value and the past behavior (*Fit_drop*). The equation allows choosing a robust solution that may likely keep its quality for a greater number of changing environments.

The pseudo code of the proposed algorithm is shown in Fig. 1.

3 Experimental Results and Their Analysis

In this section, the proposed algorithm is tested on a modified version of the Moving Peak Benchmark [9] (mMPB) [5] in which each peak has its own *height_severity* and *width_severity*. The mMPB is described in Eq. 2.

$$F_t\left(\vec{X}\right) = max_{i=1}^{i=m}\left\{H_t^i - W_t^i * \vec{X} - \overrightarrow{C_{t2}^i}\right\} \tag{2}$$

where m is number of peaks, $\vec{X}$ is a solution in the problem space, and H_t^i, W_t^i and $\overrightarrow{C_t^i}$ are the height, width and center of i^{th} peak in the t^{th} environment, respectively. The height, width and center of a peak change in each environment as follows:

$$H^i_{t+1} = H^i_t + height\_severity^i * N(0.1) \tag{3}$$

$$W^i_{t+1} = W^i_t + width\_severity^i * N(0.1) \tag{4}$$

$$\vec{C}^i_{t+1} = \vec{C}^i_t + \vec{V}^i_{t+1} \tag{5}$$

$$\vec{V}^i_{t+1} = Shift\_severity * \frac{(1-\lambda) * \vec{r} + \lambda * \vec{V}^i_t}{(1-\lambda) * \vec{r} + \lambda * \vec{V}^i_t} \tag{6}$$

where $N(0,1)$ represents a random number drawn from Gaussian distribution with zero mean and one variance. The parameter setting of mMPB is shown in Table 1 based on [5].

Table 1. Parameter setting of the modified MPB

Parameter	Value
Number of peaks, M	5
Change frequency	2500
Shift severity, s	1
Height severity	Randomized in range [1.0,10.0]
Width severity	Randomized in range [0.1,1.0]
Peaks shape	Cone
Number of dimensions, D	2
Correlation Coefficient, λ	0
Peaks location range	[0–50]
Peak height	[30.0–70.0]
Peak width	[1–12]
Initial height value	50.0
Initial width value	6.0
Number of Environments	150

In the proposed algorithm, the problem is solved by multi_swarm PSO as a maximization problem. Our algorithm tries to track moving peaks and gathers information about them. Additionally, it uses this information as well as *Gbest* fitness values of *Tracker_swarms* for choosing the next robust solution in its decision making process. The main goal of this process is that the chosen robust solutions keeps their quality above the threshold δ over a larger number of environments. As a result, the average number of environments that each robust solution remains acceptable is used as a performance measure as follows:

$$Average\ Survival\ time = \frac{Number\ of\ Environments}{Number\ of\ Robust\ Solutions} \tag{7}$$

where *number of environments* shows how many times the environment changes, and *number of robust solutions* indicates the number of times that the algorithm changed the robust solution because the existing robust solutions no longer remains acceptable.

Therefore, higher values of *Average Survival time* show better results and the best situation happens when the first robust solution remains acceptable for all environments.

It is worth mentioning that, in [5], Fu *et al*. proposed a performance measure based on average of *survival time* which was calculated by Eq. 8:

$$F^s(x,t,\delta) = \begin{cases} 0, & iff(x,\alpha(t)) < \delta \\ max\{l | t \le i \le t+l : f(x,\alpha(i)) \ge \delta\}, & else \end{cases} \tag{8}$$

where F^s is maximal time interval starting from time t until $t + l$ in which the fitness value of solution x remains above δ. The average of F^s was used as the performance measure which the result is the same with Eq. 7. However, we preferred to use Eq. 7 as performance indicator, because in our proposed algorithm, we do not use the *survival time* metric like in [5]. Furthermore, in [4], Jin *et al.* introduced different performance measures including the robustness rate as Eq. 9:

$$RobustnessRate = 1 - \frac{NumberofRobustSolutions - 1}{NumberofEnvironments - 1} \tag{9}$$

The parameter used in both Eqs. 7 and 9 are the same and the main idea of both of them is measuring *average survival time* based on the length of robust solution sequence. However, the outcome of Eq. 7 is more suited to the main goal of ROOT, *i.e.,* maximizing the number of environments a robust solution can keep its quality above the threshold.

The parameter setting of the proposed algorithm is shown in Table 2. Experiments are done on the mMPB with a parameter setting shown in Table 1 with different values of δ (40, 45 and 50). Results are obtained from 30 executions of the proposed algorithm and each execution continues for 150 environmental changes (375,000 function evaluations).

Table 2. Parameter values of the proposed algorithm

Parameters	Initial value
c1, c2	2.05 [10]
χ	0.729843788 [10]
Trackers' Population Size	5
Finder's Population Size	10
P	1[8]
Q	1[8]
Conv_limit	1[8]
K	10[8]
δ	40,45,50
Environment number	150
Stop criterion	Max number of function evaluations

Table 3 shows the experimental result of the proposed algorithm. In Table 3, *Offline_error* [11] shows the performance of Multi_swarm PSO. Additionally, RS_error

shows the average error of robust solutions in all environments and *RS_Fit* shows the average fitness value of robust solutions in all environments. Also the numbers in parentheses are *standard_error*.

Table 3. Results of the proposed algorithm on mMPB with δ = 40, 45, and 50.

δ	Offline_error (std_error)	RS_error (std_error)	RS_Fit (std_error)	Average Survival time (Eq. 7) (std_error)
40	0.0210 (0.0037)	9.3154 (0.2588)	53.1573 (0.2845)	8.3488 (0.6331)
45	0.0214 (0.0043)	6.9659 (0.2267)	55.8288 (0.2078)	6.8309 (0.6571)
50	0.0218 (0.0055)	5.2021 (0.1488)	58.1354 (0.1426)	4.2483 (0.1849)

The main goal of the proposed algorithm is to increase *Average survival time, i.e.,* to decrease the number of times that the algorithm needs to change the robust solution because of a lack of quality as determined by δ. As expected, a lower δ allows a robust solution to survive longer.

The value of *offline_error* is almost the same for all experiments because the problem is the same from the point of view of DOP and the results show that the accuracy of multi_swarm PSO for finding and tracking peaks is acceptable. This is important because the performance of choosing robust solutions is totally dependent on the performance of multi_swarm PSO in finding and tracking moving peaks. The average error of robust solutions decreases when δ increases because the proposed algorithm tried to keep fitness value of robust solutions above δ and change them if their fitness value came under δ. As a result, with higher values of δ, fitness values of acceptable robust solution increases, and the error decreases. Therefore, when δ increases, the average fitness value of robust solutions increases, at the expense of requiring more changes. Figure 2 illustrates the average fitness of the best found position by multi_swarm PSO in each environment by TMO as well as robust solutions' fitness values.

To compare the result of proposed algorithm with that of previous works in the field of ROOT, in [12], the number of robust solutions for the proposed Multi_objective ROOT algorithm and the ROOT algorithm from [5] are reported. Since the experiments in [12] were done on the same benchmark and for the same number of environments as this paper, the *Average Survival time* of existing works i.e. works in [5, 12] can be calculated by Eq. 7 and compared with our algorithm. It is worth mentioning that since in some papers such as [4, 6], the benchmark problem is different with the one that we used in this paper, so we cannot use their reported results for our comparisons and comparing the result of our algorithm with of them will be done in future works. The best *Average Survival time* with δ = 40, 45 and 50 of Guo's algorithm [12], and the average of reported values in [12] for Fu's Algorithm [5], and our proposed algorithm are shown in Table 4. The results in Table 4 show that our proposed algorithm can perform significantly better than compared works in term of *Average Survival time*. Note

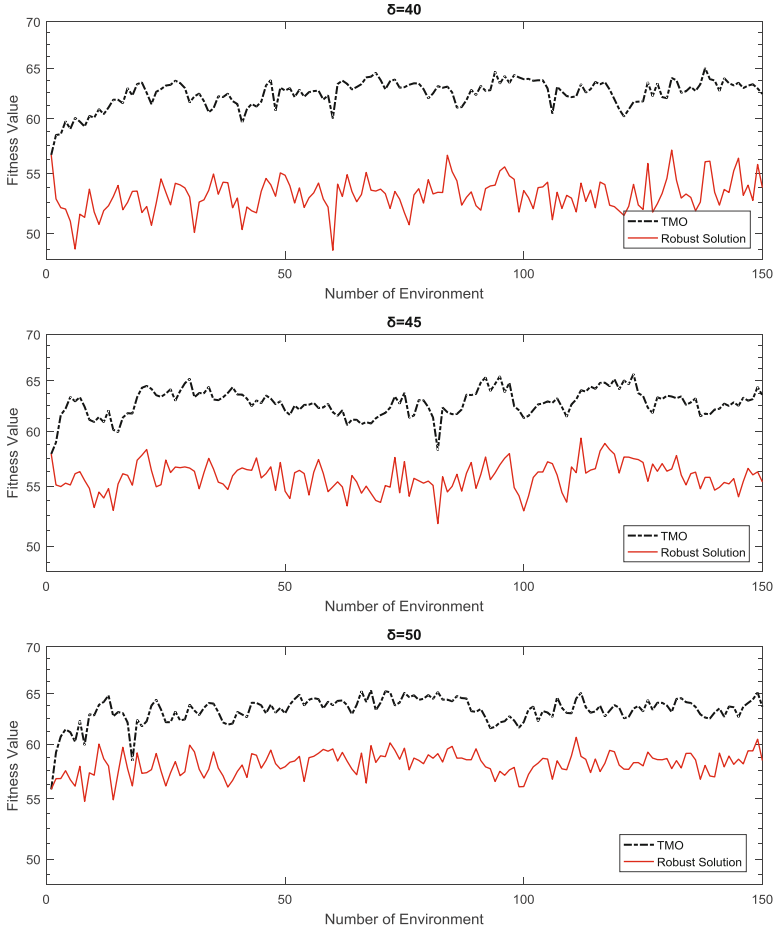

Fig. 2. Average fitness values of the best TMO and robust solutions found by multi_swarm PSO.

that Table 4 does not have comparisons on *standard error* or *standard deviation*, because these figures were not provided in [12] for Guo's and Fu's algorithms.

Table 4. The Average Robustness obtained by the three algorithms on mMPB.

δ	Average Robustness		
	Fu's Algorithm [5]	Guo's Algorithm [5]	Proposed Algorithm
40	2.30	2.67	**8.35**
45	1.91	2.08	**6.83**
50	1.53	1.61	**4.25**

4 Conclusion

In this paper, a new multi-swarm PSO algorithm has been proposed for robust optimization over time (ROOT). The main goal of the proposed algorithm is finding robust solutions that remain acceptable for a longer time, *i.e.,* solutions with fitness quality above an acceptance threshold over a number of environments. The proposed algorithm differs from previous work to find ROOT solutions. We use PSO Tracker_swarms that track peaks and gather information about how they react to changes. Then, the selection of the next robust solution is based on this information. As a performance measure, we use an easy-to-implement and algorithm-independent performance indicator named Average Robustness that reflects the average number of environments that the robust solutions could remain acceptable during optimization. The experimental results show that our proposed algorithm performs better than existing work in terms of Average Robustness. Currently, we are working on finding robust solutions based on different characteristics of peaks for DOPs with higher number of peaks and dimensions, and different specifications. In addition, new dynamic characteristics such as time-linkage [14] and distance constrains between successive solutions should be added to the problem to close the current gaps [15] between academic research and real-world problems in this domain.

Acknowledgements. This work is supported by a Dean Scholarship by the Faculty of Engineering and Technology, Liverpool John Moores University, and is partially supported by a T-TRIG project by the UK Department for Transport, a Newton Institutional Links project by the UK BEIS via the British Council, a Newton Research Collaboration Programme (3) by the UK BEIS via the Royal Academy of Engineering, and a Seed-corn project funded by the Chartered Institute of Logistics and Transport.

References

1. 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)
2. Yu, X., Jin, Y., Tang, K., Yao, X.: Robust optimization over time – a new perspective on dynamic. In: IEEE Congress on Evolutionary Computation, pp. 1–6 (2010)
3. Fu, H., Sendhoff, B., Tang, K., Yao, X.: Characterizing environmental changes in robust optimization over time. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2012)
4. Jin, Y., Tang, K., Yu, X., Sendhoff, B., Yao, X.: A framework for finding robust optimal solutions over time. Memetic Comput. **5**(1), 3–18 (2013)
5. Fu, H., Sendhoff, B., Tang, K., Yao, X.: Finding robust solutions to dynamic optimization problems. In: Esparcia-Alcázar, A.I. (ed.) EvoApplications 2013. LNCS, vol. 7835, pp. 616–625. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37192-9_62
6. Fu, H., Sendhoff, B., Tang, K., Yao, X.: Robust optimization over time: problem difficulties and benchmark problems. IEEE Trans. Evol. Comput. **19**(5), 731–745 (2015)
7. Blackwell, T., Branke, J.: Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Trans. Evol. Comput. **10**(4), 459–472 (2006)

8. Yazdani, D., Nasiri, B., Sepas-Moghaddam, A., Meybodi, M.R.: A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. Appl. Soft Comput. **13**(4), 2144–2158 (2013)
9. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: IEEE Congress on Evolutionary Computation, pp. 1875–1882 (1999)
10. Eberhart, R.C., Shi, Y.: Comparing inertia weights and constriction factors in particle swarm optimization. IEEE Congress Evolut. Comput. **1**, 84–88 (2001)
11. Yang, S., Li, C.: A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans. Evol. Comput. **14**(6), 959–974 (2010)
12. Guo, Y., Chen, M., Fu, H., Liu, Y.: Find robust solutions over time by two-layer multi-objective optimization method. IEEE Congress on Evolutionary Computation, pp. 1528–1535 (2014)
13. Kennedy, J., Eberhart, R.: Particle swarm optimization. IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
14. Nguyen, T.T., Yao, X.: Dynamic time-linkage problems revisited. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, Gianni, A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 735–744. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01129-0_83
15. Nguyen, T.T.: Continuous dynamic optimisation using evolutionary algorithms. Ph.D. thesis, University of Birmingham (2011)