

An experimental study of combining Evolutionary Algorithms with KD-Tree to solving dynamic optimisation problems

Trung Thanh Nguyen, Ian Jenkinson, Zaili Yang

School of Engineering, Technology and Maritime Operations, Liverpool John Moores University, L3 3AF, United Kingdom

T.T.Nguyen@ljmu.ac.uk

Abstract. This paper studies the idea of separating the explored and unexplored regions in the search space to improve change detection and optima tracking. When an optimum is found, a simple sampling technique is used to estimate the basin of attraction of that optimum. This estimated basin is marked as an area already explored. Using a special tree-based data structure named KD-Tree to divide the search space, all explored areas can be separated from unexplored areas. Given such a division, the algorithm can focus more on searching for unexplored areas, spending only minimal resource on monitoring explored areas to detect changes in explored regions. The experiments show that the proposed algorithm has competitive performance, especially when change detection is taken into account in the optimisation process. The new algorithm was proved to have less computational complexity in term of identifying the appropriate sub-population/region for each individual. We also carry out investigations to find out why the algorithm performs well. These investigations reveal a positive impact of using the KD-Tree.

1 Introduction and research questions

1.1 Dynamic problems and evolutionary dynamic optimisation

Real-world applications are naturally dynamic. Customer demands change, internet bandwidth fluctuates, policies are being revised, and a changing climate are some examples of real-world dynamic problems. To deal with the inherent time-dependence of the real-world, finding effective ways to solve dynamic problems is very important. If a dynamic problem is solved online when time goes by, it is called dynamic optimisation problem (DOP) [1]. Among many different approaches to solving DOPs, evolutionary algorithms (EAs), is a common approach. The field of applying EAs to solving DOPs is called evolutionary dynamic optimisation (EDO).

1.2 Detecting changes in DOPs

In addition to the need of finding the optimum as quickly as possible (as in static problems), in DOPs the solver also has to react to changes to track the

changing optimum [2]. There are two approaches: the algorithm either react to changes implicitly by some form of self-adaptation, or the algorithm need to react to changes explicitly. This paper focuses on the second approach. For most EAs following this approach, reacting to changes requires the knowledge of when a change occurs [2]. How to know when a change occurs is an important factor and it needs to be taken into consideration when an algorithm is designed.

Regarding the knowledge of the moments of changes, there are two schools of thought. The first school of thought considers that algorithms are well informed of changes or changes can be detected easily by just using one/a few detectors[3–7]. This approach makes sense for solving the current continuous academic benchmark problems, where the whole search space changes at once.

However, in many real-world applications, especially in constrained problems, only a part of the space changes and knowledge of environmental change might not be accessible [1, 8, 9]. In such situations, using just a few detectors to detect changes might not be sufficient because the detectors might not be in the changing region in the search space [2]. The second school of thought considers change detection an important part of the optimisation process rather than just a few detectors. To incorporate change detection in algorithms, some research tried to maintain enough diversity to cover the whole search space [10] or to distribute specific detectors in different search regions [11]. Some studies tried to detect changes by finding the statistical difference between the populations from two consecutive generations [9]. Some detected changes by monitoring the previous best found solutions [12]. The main disadvantages of methods following this school of thought is the additional computational cost spent on detecting/adapting changes in the whole search space. This cause methods following this approach perform generally worse than methods following the first school of thought in solving current benchmark problems.

This difference in performance between the two schools of thoughts raise an important research question of how to improve the efficiency of change detection.

1.3 Tracking multiple peaks in DOPs

One of the most commonly used approaches for EDO is to cover multiple regions of the search space, and separately monitor the movement of optima at each region. This way, multiple optima can be tracked at the same time, and if any of those optima become the global optimum after a change, they would likely be found more quickly. A natural way to track multiple regions is to use multiple populations, one for each region. Multi-population is the most used approach to solve some standard benchmark problems in the field of EDO.

In multi-population/multi-region approach, it is essential that the sub populations / regions are not overlapped so that one area is not searched by two or more sub-populations and an area is not being re-searched multiple times if there is no change. To avoid overlapped sub-populations/regions, existing methods either define each sub-population/region as a hypercube or sphere, then prevent individuals from other sub-populations to enter the cube/sphere [13, 14], or use

distance calculations to estimate the basins of attractions of peaks and use these basins as the separate regions for each sub-population [15].

The above techniques, however, are computationally expensive due to the distance calculations (analysed in Section 3). Finding a more efficient method to separate tracking regions, hence, is an important research question.

This paper describes an attempt to answer the two questions above.

2 Avoiding revisiting explored areas and improving change detection

2.1 Distributing detectors effectively

After having explored a certain part of the search space, if an algorithm remembers the structure of the explored search space, it might be able to use that knowledge to better distribute detectors, e.g. sending more detectors to rugged areas (having more optima) and fewer detectors to smooth areas (having fewer optima). In addition, if it can be assumed that changes in the basin of an optimum might likely change the value and position of the optimum itself, each basin may just need a detector right at the previously found optimum.

Placing detectors at the optima, however, can only detect changes that alter basin's height/position. For other basin changes, it might be necessary to frequently send detectors to the explored basin to check for any newly appearing solution. Such new solutions should only be accepted if they are shown to be promising. Otherwise, they should be discarded and the detectors should be sent to other areas. To implement this idea, it is essential to estimate the basin sizes. Estimating basin sizes also helps maintain just one sub-population per one peak/basin. Although estimating the basin size is a common goal of multi-population approaches, existing methods may not be able to achieve it. Their pre-determined fixed-size search area may not correctly cover the exact basin.

The next subsection proposes a method to estimate basins of attraction.

2.2 Estimating optima's basins of attraction

As mentioned earlier, the problem with many existing methods to estimate optima's basins of attraction is that these methods are both computationally expensive and inaccurate. The procedure below (Algorithm 1) proposes a simple and computationally cheap estimation by taking a number of consecutive samples along each dimensional axis until a slump in fitness is found. This procedure can be applied to all dimensional axes to create a hyper-rectangle, which approximately covers the basin of attraction of a found optimum.

2.3 Separating explored areas from unexplored areas

To separate sub-populations/regions, for every individual many existing algorithms has to calculate individual distances to all sub-populations, then assign

Algorithm 1 BasinEstimation(d)

Note: It is assumed that the problem is maximisation
 d The chosen dimensional axis along which samples are made
 $\mathbf{x}^*(d)$ The d^{th} coordination of optimum \mathbf{x}^*
 d_{\min}, d_{\max} Min and max range of search space in dimension d
 δ Sample step size, $\delta = (d_{\max} - d_{\min}) / 50$
 (l, u) Range of the basin in dimension d

1. Initialisation: $u = l = x^*(d)$
2. Identifying the upper range u of the basin:
 - **while** ($f(u) < f(u + \delta)$) $u = u + \delta$ //continue to go right until out of the basin
 - **else** $u = u + \delta/2$ //approximated upper boundary
3. Identifying the lower range l of the basin:
 - **while** ($f(l) > f(l - \delta)$) $l = l - \delta$ //continue to go left until out of the basin
 - **else** $l = l - \delta/2$ //approximated lower boundary
4. Return (l, u)

each individual to its closest sub-population. This is a computationally expensive task, as mentioned in Subsection 1.3. Another downside is that each sub-population/region has to maintain its own regional information and this information needs to be re-calculated at every generation.

In the previous subsection, an idea has been proposed to estimate the basins of attraction for found optima. This way of estimating basin can be used as a basis for a new idea to separate the sub-regions/populations with low computational cost. The idea is to make use of a special data structure named the K-dimensional tree (KD-tree) [16]. KD-Tree is a special kind of binary tree specialised for representing multi-dimensional spaces into hyper-rectangles. Each non-leaf node of the tree represents a cutting hyperplane perpendicular to one of the k dimensions. This cutting hyperplane will divide the space into two parts, represented by the two subtrees of the node. Figure 1 shows how a KD-Tree can be used to divide a two-dimensional space.

This special property inspires the authors to develop a modified version of the KD-Tree to represent the areas covered by sub-regions/populations and to distinguish explored and unexplored areas (Figure 1). The modified tree still split the space in the same way as that of the original version: at each step the space will be splitted at a chosen plane. However, the newly modified KD-tree has a major structural difference. In the original KD-tree, each node represents (i) a chosen dimension axis that is perpendicular to the splitting hyperplane, and (ii) one point in the space that the splitting hyperplane must go through. On the contrary, in the modified version *there is no point in each node although the nodes still represent the chosen dimensions and cutting splits to divide the space*. In addition, *each leaf of the modified tree represents a hyper-rectangle bounded by the cutting hyperplanes rather than the point the cutting hyperplane goes through*.

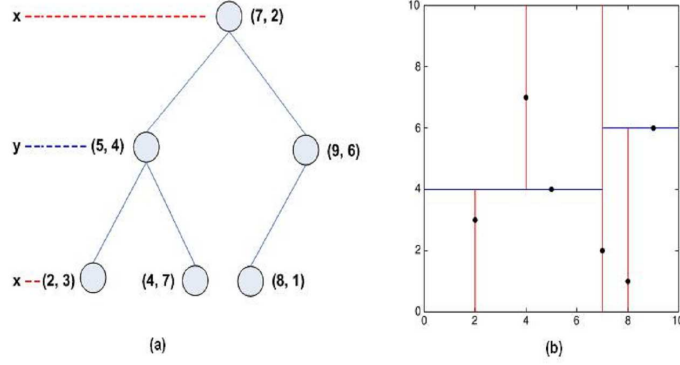


Fig. 1. These figures, reproduced from [17], show how a two-dimensional space is decomposed using a KD-tree. (a): the tree, and (b): the decomposed space.

Algorithm 2 TreeConstruction(\mathbf{x} , $B(\mathbf{x})$)

\mathbf{x} A newly found optimum

$B(\mathbf{x})$ Estimated basin of \mathbf{x}

N_d A hyper-rectangle represented by the tree node at depth d

1. Identify the leaf node (hyper-rectangle) N_d containing optimum \mathbf{x}
 2. **If** $B(\mathbf{x}) \subset N_d$ //check if the basin of \mathbf{x} is within the hyper-rectangle N_d
 - (a) **If** another optimum \mathbf{x}' is in N_d : split N_d in the middle between $B(\mathbf{x})$ and $B(\mathbf{x}')$, at a dimension i where $\text{distance}(B(\mathbf{x}), B(\mathbf{x}'))$ is largest.
 - (b) **Else**: Consider N_d the search area of the sub-region/population that tracks \mathbf{x}
 3. **Else** //go up the tree to find a hyper-rectangle large enough to contain $B(\mathbf{x})$
 - (a) $N_d = N_{d-1}$ //Because N_d is not large enough for $B(\mathbf{x})$, we have to resize N_d . We do so by going up to the parent node N_{d-1} and redo its split.
 - (b) Merge N_d //Merge N_d for resplitting later.
 - (c) Repeat step 2
-

In this modified KD-Tree, each estimated basin of the found optima is represented as a hyper-rectangle in the tree. This hyper-rectangle also indicates the cover area of the corresponding sub-population. Algorithm 2 shows the process of using a modified KD-Tree for separating regions in EDO:

This tree construction procedure help separating the regions covering different peaks automatically. In addition, it takes only $O(\log M)$ (where M is the number of sub-regions/populations) for each individual to identify which sub-region/population the individual belongs to. The procedure also allows the tree to adaptively adjust its structure in response to changes. For example, if a new optimum appears or an existing optimum has moved and the current hyper-rectangle is no longer able to cover the optimum's basin, the size of the hyper-rectangle will be adjusted accordingly. Another benefit is that, since we

need only one KD-tree to memorise all regions/populations in the space, sub-regions/populations no longer have to manage their own regional information.

2.4 Local search

EAs are considered relatively slow to converge. To speed up convergence speed, once a population starts to converge, a local search is applied to the best found solution to find the optimum more quickly and accurately. A population is considered starting to converge when the standard deviation of fitness values in the population becomes smaller than a threshold β . We choose the Brent local search, first used for EA research in [18, 19]. This local search does not require any derivative information, hence can function as a black-box local search. The disadvantage is that it is generally much slower than local searches requiring derivative information such as conjugate gradient or quasi-Newton.

2.5 Tracking the optima movements

Although some existing methods maintain a full sub-population around an optimum to track its potential movement, it might not be necessary. Within the basin of a found optimum, tracking should only be triggered if there is a change that alters the basin. Following this idea, we propose the followings:

1. For changes that alter the existing optimum: simply re-evaluate the value of the optimum at every generation. If the values in two generations are different, a change has occurred and we track the moving optimum by applying the Brent local search to identify its new location.
2. For changes that lead to a new optimum without changing the existing ones, re-evaluating existing optima does not work. To deal with this, we allow individuals to venture into any explored basin, but prevent them from converging to existing optima. To do so, for each found optimum we define a hypercube, which has the optimum at its centre and has a length of $0.8 * l_{\min}$ where l_{\min} is the smallest edge of the hyper-rectangle covering the optimum's basin. Any individual within this hypercube, but with worse value than the optimum's value, will be randomly re-initialised to the unexplored areas.

2.6 The EA-KDTree algorithm

We integrate all the above ideas into a simple Genetic Algorithm (GA). The new EA is called EA-KDTree. The algorithm works as follows. First, a KD-Tree is created with one root node representing the whole search space. Then, whenever a new optimum is found, the algorithm estimates the optimum's basin using `BasinEstimation()` (Algorithm 1). The hyper-rectangle representing this estimated basin is added as a leaf to the KD-Tree, and the space is divided accordingly. This basin is recorded in the tree as an explored area. In addition, its optimum is monitored for changes and the algorithm will be prevented from re-converging to this optimum. A pseudo code is given in Algorithm 3.

Algorithm 3 Pseudo code of EA-KDTree

1. Initialisation:
 - Unexplored area = the whole search space
 - Explored area = null
 2. For each generation, in the unexplored area:
 - (a) Simple GA to search for good basins
 - (b) Once GA starts converging ($\text{stdDev of population fitness} < \beta$), use Brent local search to find the optimum \mathbf{x}^* .
 - (c) $B(\mathbf{x}^*) = \text{BasinEstimation}(\mathbf{x}^*)$ (Algorithm 1)
 - (d) $\text{TreeConstruction}(\mathbf{x}^*, B(\mathbf{x}^*))$ //Add the estimated basin to explored area list
 3. For each generation, in the explored area:
 - (a) Search for any newly appearing optimum
 - i. Allow GA's individuals to enter explored basin
 - ii. If individuals converge to a hypercube length $0.8 * l_{\min}$ around the optimum but with worse values, reinitialise them in unexplored areas
 - iii. Else go to step 2a
 - (b) For each gen., re-evaluate fitness of found optima
 - i. If changes detected, go to 2b //local search to track the moving optimum
 4. Return to step 2
-

3 Complexity analysis

Many existing multi-population methods that track multiple peaks are computationally expensive since they have to do distance calculations. For example, for each generation, methods in [13, 14] and similar studies require distance calculations with a complexity of $O(MNn^2)$ where M is the number of sub-populations, N is the number of individuals and n is the number of variables. The method in [15] requires at least $O(mN^2)$ where m is the number of samples needed to detect the basin of attraction. In comparison, EA-DKTree complexity is significantly less: for each generation it only requires $O(N \log M)$ to identify the correct search region for all individuals (in EA-KDTree M is the number of regions monitored by the algorithm). If we need to restructure the KD-Tree, the cost to restructure is $O(M \log M)$, which is not computationally expensive.

4 Experimental results

4.1 Experimental settings

For this experiment, we choose the classic MovPeaks [20] benchmark problem. This is arguably most tested dynamic academic problem to date. The MovPeaks has multiple peaks whose locations, widths, and heights can change over time. To facilitate cross-comparison among different algorithms, three standard scenarios were proposed, of which scenario 2 was most commonly used. Due to that, in this experiment the algorithms will be tested on Scenario 2 (Table 1).

Table 1. Parameter settings for EA-KDTree and MovPeaks

EA-KDTree		Pop size	25
		Elitism	Yes
		Selection method	Roulette wheel
		Mutation method	Gaussian, $P = 0.15$
		Crossover method	Arithmetic, $P = 0.8$
MovPeaks problem settings	Number of runs	30	
	Number of peaks	10	
	Number of dimensions	5	
	Change frequency	5000 evaluations	
	Peak heights	[30, 70]	
	Peak widths	[1, 12]	
	Change severity s	1.0	

Parameter tuning was not done for EA-KDTree because the purpose is to provide a proof of principle. All parameters of the EA are the default values (Table 1) as used in recent research in the field (see justifications in [8]).

The chosen performance measure is the common *modified offline error* [21].

4.2 Experimental results - comparing with current state-of-the-arts

EA-KDTree is compared with current state-of-the-art population-based methods that follows the aforementioned school of thoughts in change detection to judge the potential of the proposed ideas. The peer algorithms were chosen from: Group 1 include algorithms with complete or semi-complete change detection methods, and Group 2 include algorithms with no change detection or with just one detector, as seen in Tables 2 and 3. EA-KDTree belongs to Group 1. Note that in Group 1, some algorithms offer a full change detection/adaptation mechanism (including EA-KDTree) while some others rely on re-evaluating the current best solution in each sub-population/region only (Cellular DE, mQSO and Sa multi-swarm). The latter are supposed to have better performance than the earlier in the MovPeaks but might not be as robust in detecting changes in some real-world problems.

As seen in Tables 2 and 3, EA-KDTree has the best performance among all Group 1 algorithms (algorithms with (semi) complete change detection). The results in the tables also indicate that due to not having to detect changes comprehensively, most algorithms in Group 1 have worse performance than most in Group 2. EA-KDTree is, however, an exception. It is still better than most algorithms in Group 2 except CDE and CPSO. Overall, EA-KDTree is the second best EA and the third best meta-heuristics of all algorithms. The few better methods are those with no complete change detection. As previously discussed, these methods might become less effective in problems where changes occur in

only a part of the search space. Note that here we do not consider methods that react to changes implicitly (e.g. [22]) or methods that are not population-based.

It is worth noting that EA-KDTree performance, however, has a quite large standard deviation. This suggests that the algorithm might not always be completely reliable. We hypothesize that this might be due to the Brent local search, which is stochastic and hence may need a large number of evaluations in certain situations. This causes a larger standard deviation. This limitation, however, can easily be alleviated by using a more powerful local search.

Table 2. Methods with (semi)-complete change-detection (Group 1).

Algorithm	Offline errors
EA-KDTree	1.50 ± 0.47
Cellular DE [5]	1.64 ± 0.02
mQSO [12]	1.75 ± 0.06
Sa multi-swarm [23]	1.77 ± 0.05
Self-Organizing Scouts [21]	4.01
MOEA DCN [24]	4.60 ± 0.085
Random-immigrant [24]	5.82 ± 0.109
Hyper-mutation [24]	5.88 ± 0.082

4.3 Experimental results - studying algorithmic components

In this section we investigate why and which algorithmic component helps EA-KDTree to have a good performance. We will investigate if the proposed ideas make it possible to (i) correctly approximate the basins of attraction, (ii) divide the space using KD-Tree, (iii) track the moving basins, and (iv) prevent the population from converging to an existing optimum again unless it has changed.

Approximating the basins and dividing the space using KD-Tree:

We investigate the ability of the algorithm in approximating the basin and dividing the space by comparing Simple GA + KD-Tree with Simple GA. The only difference between the two algorithms is the implementation of the KD-Tree and along with it the procedure BasinEstimation (Algorithm 1). To compare, we plot the position of individuals over different generations (for both algorithms) and also plot the division of the space by the KD-Tree (for GA+KD-Tree) to see if the proposed idea can help estimate the basins and divide the space correctly.

Figure 2 shows that after 11 generations GA+KDTree can find all optima, while the original GA is unable to do so after 50 generations. Furthermore the simple GA converges to just one optimum and hence fails to track multiple optima simultaneously. Another interesting observation is that the hyper-rectangles divided by the KD-Tree fits well with optima’s basins. This illustrates the clear

Table 3. Methods with no complete change-detection (Group 2).

Algorithm	Offline errors
CDE [25]	0.92 ± 0.07
CPSO [3]	1.06 ± 0.07
MSO [26]	1.51 ± 0.04
ESCA [4]	1.53 ± 0.02
Cellular DE [5]	1.64 ± 0.02
DynDE [6]	1.75 ± 0.03
MEPSO [7] (5 detectors)	4.02 ± 0.56
jDE ([27], implemented by [25])	5.88 ± 0.31

advantage of estimating the basin and dividing the search space using KDTree. Figure 2c also shows that in the hyper-rectangle on the right half (the explored area), there is almost no individual because they have already been re-initialised to the unexplored area (the left half). This demonstrates that EA-KDTree is able to distinguish between explored and unexplored areas, as well as to prevent individuals from reconverging to an existing optimum.

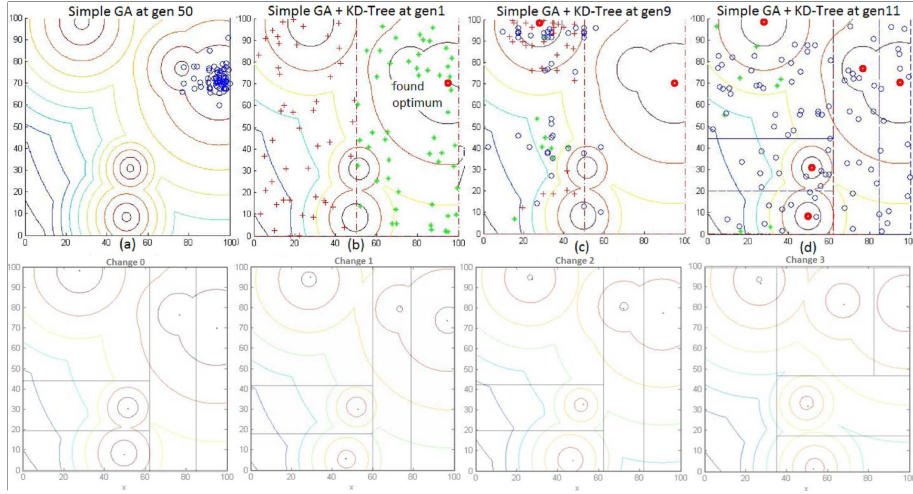


Fig. 2. Top: Simple GA vs Simple GA+KDTree; Bottom: EAKD-Tree adjusts its tree to track the moving optima's basins

Using KD-Tree to track moving optima: We investigate if TreeConstruction (Algorithm 2) can help EA-KDTree to adaptively adjust its tree struc-

ture to track the moving optima by plotting the structure of the KDTree against the search landscape at different moments when changes occur (Figure 2).

Figure 2 shows EAKD-Tree has clearly adjusted the size of its hyper-rectangles to adapt with changes. At change 3, due to the radical level of changes, the KD-Tree even completely changes its structure to better cover the changing basins and optima. The figure confirms that the algorithm is able to resize/relocate its hyper-rectangles to better fit with the changes in both basin sizes and locations of the optima. This ensures that moving optima are tracked successfully.

5 Conclusion and future work

This paper presented a new method to adaptively separate the unexplored and explored areas in search spaces. This method helps improve tracking the moving optima and detecting changes. The resulting algorithm performs competitively against current state-of-the-art, while having the benefits of offering less computational complexity and better change detection, even when being applied to even a not-usually-effective simple GA.

The paper has the following contributions: (a) a novel use of KD-Tree to separate and track explored regions, with low computational cost; (b) a simple method to correctly estimate basins of attraction of optima; (c) a new competitive algorithm; and (d) detailed analyses to provide more insights of the behaviours of the new algorithm.

There are a number of areas for future research. First, we will use a more powerful EA, for example DE or PSO instead of simple GA. Second, we plan to tune the parameters to have better results. Third, we will investigate replacing the current Brent local search with a different local search that is more reliable.

Acknowledgment

This work is supported by a Seed-corn grant award by the Chartered Institute of Logistics and Transport, an EU-funded project named Intelligent Transportation in Dynamic Environments (InTraDE), a research grant from RCUK NEMODE, and a British Council’s UK-ASEAN Knowledge Partnership grant.

References

1. T. T. Nguyen, “Continuous Dynamic Optimisation Using Evolutionary Algorithms,” Ph.D. thesis, Birmingham, 2011, <http://etheses.bham.ac.uk/1296>.
2. T. T. Nguyen, S. Yang, and J. Branke, “Evolutionary dynamic optimization: A survey of the state of the art,” *Swarm and Evolutionary Computation*, 6:1–24, 2012.
3. S. Yang and C. Li, “A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments,” *IEEE Trans. Evo. Comp.*, 14(6):959–974, 2010.
4. R. Lung and D. Dumitrescu, “Evolutionary swarm cooperative optimization in dynamic environments,” *Natural Computing*, 9(1):83–94, 2010.

5. V. Noroozi, A. Hashemi, and M. Meybodi, "Cellularde: A cellular based differential evolution for dynamic optimization problems," in *Adaptive and Natural Computing Algorithms*. Springer, 2011, pp. 340–349.
6. R. Mendes and A. Mohais, "Dynde: a differential evolution for dynamic optimization problems," in *CEC*, 2005, pp. 2808–2815.
7. W. Du and B. Li, "Multi-strategy ensemble particle swarm optimization for dynamic optimization," *Information Sciences*, 178(15):3096 – 3109, 2008.
8. T. T. Nguyen and X. Yao, "Continuous dynamic constrained optimisation - the challenges," *IEEE Trans. on Evolutionary Computation*, 16(6): 769–786, 2012.
9. H. Richter, "Detecting change in dynamic fitness landscapes," in *Congress on Evolutionary Computation*, 2009, pp. 1613–1620.
10. J. J. Grefenstette, "Genetic algorithms for changing environments," in *Parallel Problem Solving from Nature 2*, 1992, pp. 137–144.
11. R. W. Morrison, *Designing Evolutionary Algorithms for Dynamic Environments*, no. ISBN 3-540-21231-0, Springer-Verlag, Berlin, 2004.
12. T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Trans. Evo. Comp.*, 10(4):459–472, 2006.
13. F. Oppacher and M. Wineberg, "The shifting balance genetic algorithm: Improving the ga in a dynamic environment," in *GECCO*, 1999, pp. 504–510.
14. J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," in *Adaptive Computing in Design and Manufacturing*, 2000.
15. R. K. Ursem, "Multinational GA optimization techniques in dynamic environments," in *Genetic and Evolutionary Computation Conference*, 2000, pp. 19–26.
16. J. L. Bentley and J. H. Friedman, "Data structures for range searching," *ACM Computing Surveys*, 11(4):397–409, 1979.
17. Wikipedia, "KD-tree," [accessed Apr-07].
18. T. T. Nguyen and X. Yao, "An experimental study of hybridizing cultural algorithms and local search," *Int. Journal of Neural Systems*, 18(1):1–18, 2008.
19. —, "Hybridizing cultural algorithms and local search," in *Intelligent Data Engineering and Automated Learning*, 2006, pp. 586–594.
20. J. Branke, *Evolutionary Optimization in Dynamic Environments*. Kluwer, 2001.
21. J. Branke and H. Schmeck, "Designing evolutionary algorithms for dynamic optimization problems," in *Theory and Application of Evolutionary Computation: Recent Trends*. Springer, 2003, pp. 239–262.
22. C. Li and S. Yang, "A general framework of multipopulation methods with clustering in undetectable dynamic environments," *IEEE Transactions on Evolutionary Computation*, 16(4):556–577, 2012.
23. T. Blackwell, "Particle swarm optimization in dynamic environment," in *Evolutionary Computation in Dynamic and Uncertain Environments*, 2007, pp. 28–49.
24. L. Bui, M.-H. Nguyen, J. Branke, and H. Abbass, "Tackling dynamic problems with multiobjective evolutionary algorithms," in *Multiobjective Problem Solving from Nature*. Springer, 2008, pp. 77–91.
25. M. C. du Plessis and A. P. Engelbrecht, "Using competitive population evaluation in a differential evolution algorithm for dynamic environments," *European Journal of Operational Research*, vol. 218, no. 1, pp. 7 – 20, 2012.
26. M. Kamosi, A. Hashemi, and M. Meybodi, "A new particle swarm optimization algorithm for dynamic environments," in *Swarm, Evolutionary, and Memetic Computing*. Springer, 2010, pp. 129–138.
27. J. Brest, A. Zamuda, B. Boskovic, M. Maucec, and V. Zumer, "Dynamic optimization using self-adaptive differential evolution," in *CEC*, 2009, pp. 415–422.