Solving dynamic optimisation problems by combining Evolutionary Algorithms with KD-Tree

Trung Thanh Nguyen, Ian Jenkinson, Zaili Yang
School of Engineering, Technology and Maritime Operations
Liverpool John Moores University
Liverpool, United Kingdom
T.T.Nguyen@ljmu.ac.uk

***Abstract*—In this paper we propose a novel evolutionary algorithm that is able to adaptively separate the explored and unexplored areas to facilitate detecting changes and tracking the moving optima. The algorithm divides the search space into multiple regions, each covers one basin of attraction in the search space and tracks the corresponding moving optimum. A simple mechanism was used to estimate the basin of attraction for each found optimum, and a special data structure named KD-Tree was used to memorise the searched areas to speed up the search process. Experimental results show that the algorithm is competitive, especially against those that consider change detection an important task in dynamic optimisation. Compared to existing multi-population algorithms, the new algorithm also offers less computational complexity in term of identifying the appropriate sub-population/region for each individual.**

I. Introduction and research questions

A. Dynamic problems and evolutionary dynamic optimisation

Many real-world problems are dynamic. For example, the stock markets fluctuate, trains are delayed or cancelled, the weather changes from rainy to sunny, and so on. As a result, solving dynamic problems is very important. If the dynamic problem is solved online when time goes by, it is called dynamic optimisation problem (DOP) [1].

There are many different approaches to solving DOPs, of which evolutionary algorithms (EAs) is commonly chosen. This is because EAs simulate the natural evolution process, which in itslef implies adaptation to environmental changes. The study of using EAs to solve DOPs is called evolutionary dynamic optimisation (EDO) and is an active research area.

B. Change detection

Different from stationary optimisation, which focuses on finding the optimum as quickly as possible, in DOPs the solver has to react to changes to track the changing optimum [2]. For most EAs, reacting to changes requires the knowledge of the changing moments [2] and this needs to be taken into consideration when one designs an algorithm.

Regarding this aspect, there are two schools of thought. One considers that algorithms will be informed of changes or changes can be detected easily by just using one/a few detectors[3], [4], [5], [6], [7]. This approach makes sense for solving the current continuous academic benchmark problems, where the whole search space changes at once and hence changes can be detected using any detector.

However, in real-world applications or constrained problems where only a part of the space changes [1], [8], [9], just using a few detectors might not be enough 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; the algorithm is not informed of environmental change; and that change detection is not trivial, so using one or a few detectors might not be enough. Some algorithms following this approach try to maintain enough diversity to cover the whole search space [10]. Some other try to detect changes by finding the statistical difference between the populations from two consecutive generations [9]. Some detect changes by monitoring the previous best found solutions [11]. 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 using fewer detectors in solving current benchmark problems. One important research question would be to improve the efficiency of change detection.

C. Tracking multiple peaks and multi-population approaches

One of the most commonly used approaches for EDO is multi-population: the algorithm maintains more than one sub-populations, each may cover a separate area of the search space. This way, the algorithm can track multiple optima at the same time, and if after a change any of those optima become the global optimum, the algorithm would be able to find it quickly. Multi-population is considered the most flexible approach [2] and it has been used extensively to solve some standard benchmark problems in the field of EDO.

It is necessary to make sure that the sub-populations are not overlapped. The purpose is to (1) avoid one area being searched by two or more sub-populations and (2) to avoid re-searching the same area again if there is no change.

To avoid the sub-populations being overlapped, existing methods either define each sub-population as a hypercube or sphere then prevent individuals from other sub-populations to enter the cube/sphere [12], [13], or use distance calculations to estimate the basins of attractions of peaks and use these basins as the separate regions for each sub-population [14].

The above techniques, however, are computatinally expensive due to the distance calculations (analysed in Section

III). Finding a more efficient method to separate tracking regions, hence, is an important research question to improve the performance of methods that track multiple peaks.

The next section discusses some ideas, implementations and algorithm to answer the two questions above.

II. A NEW EDO APPROACH TO AVOID REVISITING EXPLORED AREAS AND TO IMPROVE CHANGE DETECTION

A. Distributing detectors effectively

Assume that an algorithm knows the structure of the search areas it has explored previously, it might be useful to use that knowledge to distribute change detectors effectively. For example, instead of evenly distributing detectors everywhere in the search space, we can place more detectors in rugged areas (having more optima) and fewer detectors in smooth areas (having fewer optima). Particularly, if we assume that changes in the basin of an optimum might likely change the height and position of the optimum itself, then for each basin we can place the detector right at the peak of the basin.

Placing detectors at the basin peaks helps detect only changes that alter the position or height of the peaks. To discover other changes in the explored areas, we can frequently send individuals to the explored basins to check for any newly appearing solution. However, in the explored basins we should accept a new solution only if it is promising. Otherwise the solution will be discarded and the individual will be sent to other unexplored areas. In order to do so, it is necessary to estimate the size of the basin. Estimating the size of the basin also brings another advantage: it helps each sub-population/region to cover approximately one peak only. Despite that this is the common goal of multi-population approaches, existing methods may not be able to achieve it. Their pre-determined fixed-size search region may not cover the whole basin or may cover more than one basin.

The next subsection will describe our proposed method to estimate basins of attraction for optima in a search space.

B. Estimating optima's basins of attraction

The process of estimating should not be computationally expensive. The following procedure (Algrithm 1) describes a simple and computationally cheap estimation by taking a number of consecutive samples along each dimensional axis until finding a slump in fitness. This can be applied to all dimensional axes, creating a hyper-rectangle approximately covering the basin of attraction of a found optimum.

C. Separating and distinguishing explored areas

In existing methods, in order to separate sub-populations/search regions, for each individual the algorithm has to calculate individual distances to all sub-populations, then assign the individual to the closest sub-population. As mentioned in Subsection I-C, such a task is computationally expensive. In addition, each sub-region/population has to maintain its own regional information, which needs to be re-calculated at every generation.

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 to the right until goes 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 to the left until goes out of the basin
   - **else** $l = l - \delta/2$ //approximated lower boundary
4) Return (l, u)

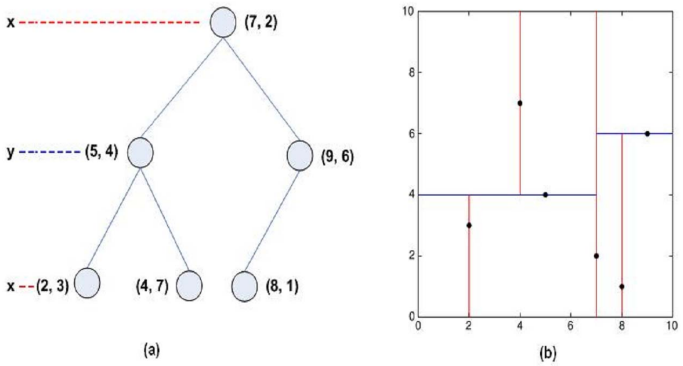

Fig. 1. These figures, reproduced from [16], show how a two-dimensional space is decomposed using a KD-tree. Figure (a) shows us the tree, and figure (b) shows us the decomposed space.

Previously, we have proposed an idea to estimate the basins of attraction for found optima and use them as indications for covering separate peaks. Based on these estimated basins we will proposed a new way to separate the sub-regions/populations with low computational cost. This is done by using a special data structure named the K-dimensional tree (KD-tree) [15]. 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 two-dimensional space is decomposed using a KD-tree.

KD-tree is used in image processing as a compress method to reduce image storage space. It is also widely used as an algorithm to find the nearest neighbour thanks to its low complexity in finding a node in the tree (to be discussed in Section III). This advantage inspires us to develop a modified version of the KD-Tree to represent the covering area of

sub-regions/populations and to distinguish explored and unexplored areas in the search space. The modified KD-tree still has the same way to split the space as that of the original version, i.e. at each step the space will be splitted at a chosen splitting plane. However, our 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*.

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:

Algorithm 2 TreeConstruction(x, B(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 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 that is 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

Given this tree construction procedure, the regions covering different peaks are automatically separated. Futhermore, for every new individual, it takes only $O(\log M)$ (where M is the number of sub-regions/populations) to identify which sub-region/population the individual should belong to. The procedures allow the tree to adaptively adjust its structure in response to changes. For example, if a new optimum is found 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 advantage is that the tree frees sub-regions/populations from managing their own regional information. There is only one KD-tree needed for all regions/populations in the space.

1) Local search: One of the commonly known disadvantages of certain EAs is that they are slow to converge to an optimum. To speed up the process, once a population starts to converge (standard deviation of fitness values in the population becomes smaller than a threshold β), we apply a local search on the best found solution to hopefully reach the optimum faster and more accurately. The local search is the Brent method, first used for EA research in [17], [18].

2) Tracking the moving optima: For each found optimum, it is not necessary to maintain a full sub-population within its basin unless there is a change that alters the basin, making the optimum move or leading to a new optimum. As a result, instead of maintaining a normal sub-population for each explored basin, we propose the followings to detect changes:

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 occured 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.

D. The EA-KDTree algorithm

The ideas mentioned above were integrated into a simple Genetic Algorithm (GA) to create a new evolutionary algorithm called EA-KDTree. Whenever the algorithm found a new optimum, the optimum's basin of attraction is estimated using the BasinEstimation() procedure (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 estimated is recorded in the tree as an explored area, its optimum is monitored for changes, and the algorithm will be prevented from re-converging to this optimum again.

Note that the proposed approach here is different from multi-population approaches: while mult-population approaches have multiple sub-populations tracking multiple peaks, the proposed approach has one one main population. The multiple peaks are still tracked concurrently, but they, or more precisely their basins, are tracked by the combination of BasinEstimation() procedure and the KD-Tree.

The simplified pseudo code is set out in Algorithm 3:

III. Complexity analysis

As mentioned previously, existing methods that track multiple peaks are computationally expensive due to distance calculations. For each generation, methods in [12], [13] and similar studies requires distance calculations with a complexity

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 (stdDev of population fitness $< \beta$), use Brent local search to find the optimum $\mathbf{x}^*$.
   c) $B(\mathbf{x}^*)$ =BasinEstimation($\mathbf{x}^*$) (Algorithm 1)
   d) TreeConstruction($\mathbf{x}^*, B(\mathbf{x}^*)$) //Add the estimated basin to the explored area in the KD-Tree
3) For each generation, in the explored area:
   a) Search for any newly appearing optimum
      i) Allow GA'sindividuals 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 are detected, go to step 2b //use local search to track the moving optimum
4) Return to step 2

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 [14] requires at least $O(mN^2)$ where m is the number of samples needed to detected 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.

IV. Experimental results

A. The MovPeaks benchmark

The MovPeaks [19] is a classic dynamic problem tested by many existing research. It has a number of peaks whose locations p, widths w, and heights h change over time.

$$F(\mathbf{x}, t) = \max\left[B(\mathbf{x}), \max_{i=1,..,m} P(\mathbf{x}, h_i(t), w_i(t), p_i(t))\right]$$

where B is a time-dependent basis landscape, P is the function defining the dynamics of m individual peaks. To facilitate cross-comparison among different algorithms and researchs, three standard scenarios were proposed, of which scenario 2 was most commonly used. For this reason, in this paper we are going to test the algorithms on Scenario 2 (Table I).

B. Parameter settings for EA-KDTree

Because our purpose is to provide a proof of principle, we do not focus on parameter tuning. All the parameters of the EA (simple GA) is the default values (Table I) as used in recent research in the field (see justifications in [8]).

TABLE I
Parameter settings for EA-KDTree and MovPeaks

EA-KDTree	Pop size	25
	Elitism	Yes
	Selection method	Roullette 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

C. Performance measures

The first measure is the common *offline error* [20]. This is measured as the average over, at every evaluation, the error of the best solution found since the last change. Offline error would be zero for a perfect performance.

To investigate how the proposed ideas help the new algorithm improve tracking the moving optima, we study the percentage of peaks covered when time goes by. The more peaks that were covered, the more likely the the global optimum has been found/tracked successfully. In addition, if an algorithm is able to cover more peaks when time goes by despite more changes have occured in the search space, it is likely that it is successful in dealing with changes.

D. Experimental results and peer algorithms

The purpose of this paper is to show that by using the proposed ideas it is 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.

1) Approximating the basins and dividing the space using KD-Tree: 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 and converges to one optimum. In addition, the hyper-rectangles of the KD-Tree fits well with optima's basins, showing the benefit of estimating the basin and storing information 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 shows EA-KDTree is able to prevent individuals from reconverging to an existing optimum.

2) Adjusting the KD-Tree following optima movements: Figure 3 shows how EAKD-Tree adaptively adjusts its tree structure to track the moving optima: the algorithm is able to resize and 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.

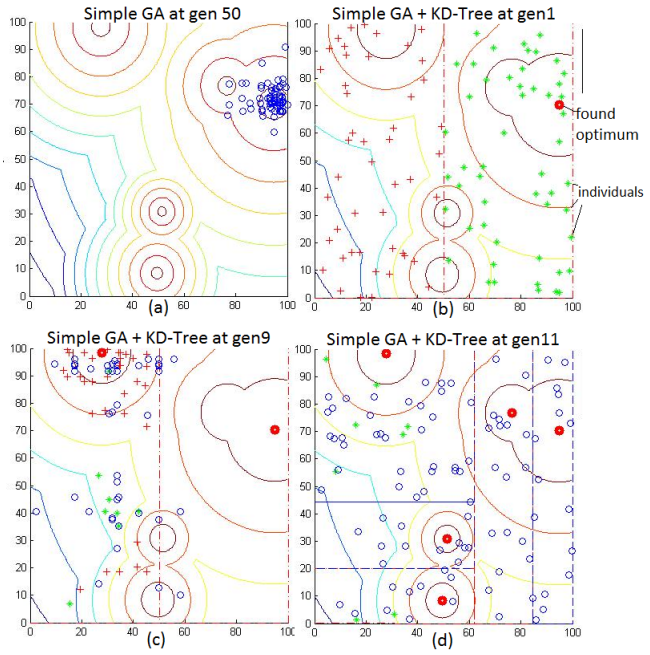

Fig. 2. Simple GA vs Simple GA+the proposed ideas.

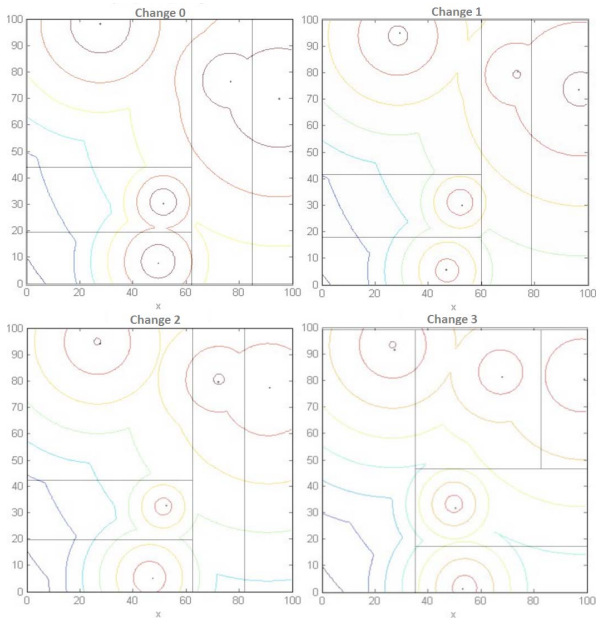

Fig. 3. How EAKD-Tree adjusts its tree to track the moving optima's basins.

3) Peak coverage and the impact of each algorithmic component: The more peaks an algorithm is able to cover, the more chance it can detect and track changes. Since the difference between EA-KDTree and GA is the KD-Tree and the local search, comparing EA-KDTree to GA, Self-Organization Scout (SOS - a GA-based algorithm) [20], restart local search and restart local search + KD-Tree will help us evaluate the impact of each proposed components in EA-KDTree.

Figures 4 shows that EA-KDTree is able to cover significantly more peaks than the peer algorithms for most of the time. The comparison shows that the proposed ideas do help the algorithms to focus more on the unexplored areas, hence improving the peak coverage percentage considerably.

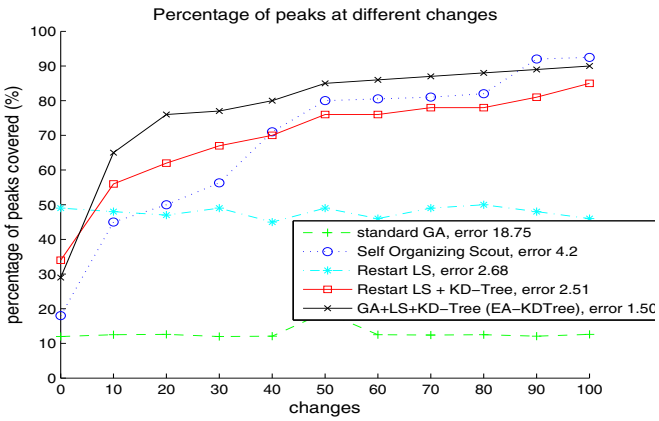

Fig. 4. EA-KDTree (GA+LS+KD-Tree) in comparison to Standard GA, restart local search (restart LS), restart LS+KD-Tree and Self-Organization Scouts (SOS).The figure also shows the offline error performance of each tested algorithm (the smaller the better).

4) Comparing with state-of-the-art evolutionary algorithms: Finally, we compare EA-KDTree with current state-of-the-art population-based methods to evaluate the potential of the proposed ideas. The tested methods are divided into two groups: Group 1 with complete or semi-complete change detection methods, and Group 2 with no change detection or with just one detector. 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.

The results (Table II and III) show that EA-KDTree has the best performance among all algorithms that have (semi) complete change detection (Group 1). The results also show that due not having to detect changes comprehensively, most algorithms in Group 2 are better than most in Group 1. EA-KDTree, however, is still better than most in Group 2 except CDE and CPSO. Overall, EA-KDTree is the second best EA and the third best meta-heuristics.

EA-KDTree's score, however, has a large standard deviation, showing that it might not always be reliable enough. This is due to the Brent local search, which is stochastic and sometimes need a large number of evaluations.

In the literature, there are two other methods that are also better than EA-KDTree, but they do not support complete change detection and are not population-based: one is single-based[21] and the other is multi-agent[1].

[1][22] also had very competitive results but it relies on problem information (population number), making the comparison to EA-KDTree incompatible.

TABLE II
METHODS WITH (SEMI)-COMPLETE CHANGE-DETECTION (GROUP 1).

Algorithm	Offline errors
EA-KDTree	$\mathbf{1.50 \pm 0.47}$
Cellular DE [5]	1.64 ± 0.02
mQSO [11]	1.75 ± 0.06
Sa multi-swarm [23]	1.77 ± 0.05
Self-Organizing Scouts [20]	4.01
MOEA DCN [24]	4.60 ± 0.085
Random-immigrant [24]	5.82 ± 0.109
Hyper-mutation [24]	5.88 ± 0.082

TABLE III
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

The results demonstrate that the proposed ideas are promising. EA-KDTree is the best in Group 1 and third best in all algorithms. The few better methods in the literature are those with no complete change detection. As discussed previously, these methods might become less efficient in problems where only a part of the search space changes.

V. CONCLUSION AND FUTURE WORK

In this paper we have proposed some novel ideas to adaptively separate the explored and unexplored areas in search spaces to facilitate detecting changes and tracking the moving optima. Particularly, a simple method was proposed to estimate the basin of attraction for each found optimum, and a binary KD-Tree was used to separate the estimated basins, as well as distinguish between explored and unexplored areas.

Experimental results show that the ideas not only offer significantly less complexity, but also help to improve algorithm performance in term of offline erros. When being applied to even a not-usually-effective simple GA, the ideas help achieve the best results among current state-of-the-art methods with complete or semi-complete change detection.

Since this is just a proof of principle research, there are a number of areas for further improvements. First, a more powerful underlying EA, for example DE or PSO can be used instead of simple GA. In addition, the parameters can be tuned to have better results. Third, the current Brent local search can be replaced with a different local search that is more reliable.

ACKNOWLEDGMENT

This work is supported by an EU-funded project named "Intelligent Transportation in Dynamic Environments (InTraDE)".

REFERENCES

[1] T. T. Nguyen, "Continuous Dynamic Optimisation Using Evolutionary Algorithms," Ph.D. dissertation, School of Computer Science, University of Birmingham, January 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*, vol. 6, pp. 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. Evolutionary Computation*, vol. 14, no. 6, pp. 959–974, 2010.

[4] R. Lung and D. Dumitrescu, "Evolutionary swarm cooperative optimization in dynamic environments," *Natural Computing*, vol. 9, no. 1, pp. 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 *Congress on Evolutionary Computation*. IEEE, 2005, pp. 2808–2815.

[7] W. Du and B. Li, "Multi-strategy ensemble particle swarm optimization for dynamic optimization," *Information Sciences*, vol. 178, no. 15, pp. 3096 – 3109, 2008.

[8] T. T. Nguyen and X. Yao, "Continuous dynamic constrained optimisation - the challenges," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 6, pp. 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] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments." *IEEE Trans. Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, 2006.

[12] F. Oppacher and M. Wineberg, "The shifting balance genetic algorithm: Improving the ga in a dynamic environment," in *Genetic and Evolutionary Computation Conference*, 1999, pp. 504–510.

[13] 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*. Springer, 2000.

[14] R. K. Ursem, "Multinational GA optimization techniques in dynamic environments," in *Genetic and Evolutionary Computation Conference*, 2000, pp. 19–26.

[15] J. L. Bentley and J. H. Friedman, "Data structures for range searching," *ACM Computing Surveys*, vol. 11, no. 4, pp. 397–409, 1979.

[16] Wikipedia, "KD-tree," [accessed Apr-07].

[17] T. T. Nguyen and X. Yao, "An experimental study of hybridizing cultural algorithms and local search," *International Journal of Neural Systems*, vol. 18, no. 1, pp. 1–18, 2008.

[18] ——, "Hybridizing cultural algorithms and local search," in *Intelligent Data Engineering and Automated Learning*, 2006, pp. 586–594.

[19] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Kluwer, 2001.

[20] 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.

[21] I. Moser and T. Hendtlass, "A simple and efficient multi-component algorithm for solving dynamic function optimisation problems," in *Proceedings of the IEEE Congress on Evolutionary Computation CEC'07.*, 2007, pp. 252–259.

[22] C. Li and S. Yang, "A general framework of multipopulation methods with clustering in undetectable dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 556–577, 2012.

[23] T. Blackwell, "Particle swarm optimization in dynamic environment," in *Evolutionary Computation in Dynamic and Uncertain Environments*. Springer-Verlag, 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 *IEEE Congress on Evolutionary Computation*, 2009, pp. 415–422.