Analysis of Hyper-heuristic Performance in Different Dynamic Environments

Stefan van der Stockt
Computational Intelligence
Research Group
University of Pretoria
Gauteng, South Africa
Email: stefan.vanderstockt@gmail.com

Andries P. Engelbrecht
Computational Intelligence
Research Group
University of Pretoria
Gauteng, South Africa
Email: engel@cs.up.ac.za

***Abstract*—Optimisation methods designed for static environments do not perform as well on dynamic optimisation problems as purpose-built methods do. Intuitively, hyper-heuristics show great promise in handling dynamic optimisation problem dynamics because hyper-heuristics can select different search methods to employ at different times during the search based on performance profiles. Related studies use simple heuristics in dynamic environments and do not evaluate heuristics that are purpose-built to solve dynamic optimisation problems. This study analyses the performance of a random-based selection hyper-heuristic that manages meta-heuristics that specialise in solving dynamic optimisation problems. The performance of the hyper-heuristic across different types of dynamic environments is investigated and compared with that of the heuristics running in isolation and the same hyper-heuristic managing simple Gaussian mutation heuristics.**

I. Introduction

The search space of a dynamic optimisation problem changes over time, and the goal is to minimize $f(\vec{x}, \vec{\omega}(t))$ with $\vec{x} = (x_1, ..., x_{n_x})$ and $\vec{\omega}(t) = (\omega_1(t), ..., \omega_{n_\omega}(t))$ subject to $x_j \in dom(x_j)$, inequality constraints $g_m(\vec{x}) \leq 0, m = 1, ..., n_g$, and equality constraints $h_m(\vec{x}) = 0, m = n_g + 1, ..., n_g + n_h$, where $\vec{\omega}(t)$ is a vector of time-dependent objective function control parameters, and n_g and n_h are respectively the number of inequality and equality constraints [1]. The objective is to find $\vec{x}^*(t) = \min_{\vec{x}} f(\vec{x}, \vec{\omega}(t))$ where $\vec{x}^*(t)$ is the optimum at time t.

Optimization algorithms for static environments tend to be ineffective on dynamic optimisation problems because they fail to adapt to the changing landscape [2] [3]. Common reasons for this include diversity loss, outdated memory of fitness values, and the inability of the algorithm to detect environment changes [1]. Algorithms have been specifically developed to solve dynamic optimisation problems [4]. Notable examples include quantum particle swarm optimisation (PSO) [5], charged PSO [6], and DynDE [7], amongst others.

Hyper-heuristics are loosely defined as 'heuristics to choose heuristics' [8]. Hyper-heuristics manage a pool of heuristics that can be selectively applied to a problem at each selection interval (i.e. a number of algorithm iterations). The selection criteria may or may not use heuristic performance to decide which heuristic(s) to apply in the next interval. The intuitive expectation is that hyper-heuristics should perform well in dynamic environments because different search methods can be employed by the hyper-heuristic at different times during the search as needed. Recently, a number of studies have investigated this expectation [9] [10] [11] [12] [13]. These studies use simple heuristics that work well in static environments.

This paper explores the performance of a random-based selection hyper-heuristic to manage meta-heuristics that are purpose-built for dynamic optimisation problems. The results show that hyper-heuristic performance is dependant on the type of dynamic optimisation problem, and that further research is needed into adapting hyper-heuristics to a given dynamic optimisation problem. The results also show that a hyper-heuristic managing a pool of purpose-built meta-heuristics performs better than the same hyper-heuristic that manages generic heuristics suitable for static environments. Section II gives a brief overview of dynamic optimisation problem classifications, performance measures for dynamic optimisation problems, and hyper-heuristics. Section III outlines the goals of the empirical study, explains hyper-heuristic and meta-heuristic choices and discusses the problem environments studied. Section IV reviews the experimental findings and section V concludes the study.

II. Related work

A. Classification of dynamic optimisation problems

Many classifications of dynamic optimisation problems have been developed [14] [15] [16] [17] [18] [19]. These classifications describe the behaviour of the optima and include aspects such as the direction and patterns of change, optima trajectories, frequency and magnitude of changes, cycle length, and homogeneity of optima movement. Eberhart *et al.* [15] [16] describe the direction of change of the optima as either: Type I, where the positions changes but not the values; Type II, where the values change but not the positions; or Type III, where both the positions and values change. Angeline describes the trajectory of optima over time as either linear, circular or random [14]. Duhain *et al.* unites the classifications of Eberhart *et al.*, Angeline and new behavioural classes that are based on the spatial and temporal severity of changes [20]. Figure 1 shows the behavioural classes from Duhain *et al.*

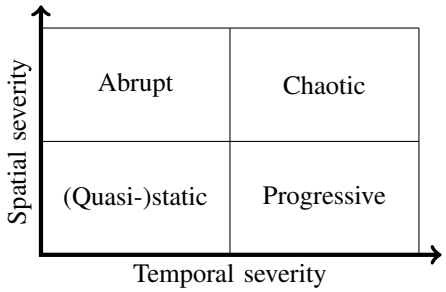

Fig. 1: Spatial vs. temporal severity trade-off [20]

The new classes of Duhain *et al.* comprise of:

- **(Quasi-)Static environments** have both low spatial and temporal changes relative to the problem's scale. Effectively the search landscape does not change.
- **Progressively changing environments** have small but very frequent changes to the search landscape.
- **Abruptly changing environments** have severe but infrequent spatial changes. The search landscape is essentially static for medium to long periods, with severe landscape changes at the end of each period.
- **Chaotic changing environments** are not to be confused with random change, but rather describes a frequently changing environment with severe spatial change.

The classification of Duhain *et al.* thus describes 27 types of dynamic optimisation problems based on the classifications of Eberhart *et al.*, Angeline and the severity of changes. This system does not aim to incorporate every possible characteristic of dynamic optimisation problems – the goal is to provide a more complete classification compared to using other classification systems independently. Duhain *et al.* also describe representative benchmarks for each type of environment in the classification using the moving peaks benchmark [21].

B. Dynamic environment performance measures

Dynamic optimisation problems require different performance metrics than static problems since the environment changes over time. A comprehensive analysis by Duhain [2] recommends using the following measures: *accuracy*, the quality of solutions over time; *stability*, solution quality after a change; and *exploitation capability*, the quality of the best solution between changes. This study uses metrics that assume knowledge of the optima (and thus the error), namely:

- **Collective mean error** (CME) records the mean error of the best solution over the entire experiment [22], defined as

$$CME = \frac{\sum_{t=1}^{n_t}(err(t,m))}{n_t} \tag{1}$$

  where n_t is the number of iterations per simulation, and $err(t,m)$ is the difference between the best solution and the optimum at iteration t in simulation m.
- **Average best error before change** (ABEBC) is the mean error of the best solution across all time instants preceding an environment change to measure *exploitation capability* [23], defined as

$$ABEBC = \frac{\sum_{c=1}^{n_c} err_{c,r-1}}{n_c} \tag{2}$$

  where n_c is the number of changes per simulation, r is the number of iterations per change period, and $err_{c,r-1}$ is the error in change period c just before a change.
- **Average best error after change** (ABEAC) shows the mean error of the best solution at the time instants immediately after environment changes and measures *stability* [2], defined as

$$ABEAC = \frac{\sum_{c=1}^{n_c} err_{c,0}}{n_c} \tag{3}$$

  where $err_{c,0}$ is the error at the iteration after a change.

C. Hyper-heuristics

Hyper-heuristics are search methods or learning mechanisms for selecting or generating heuristics to solve computational search problems [24]. The field of hyper-heuristics is closely related to fields such as algorithm portfolios, ensembles, meta-learning, adaptive memetic algorithms, adaptive operator selection, and evolutionary algorithm parameter control [8] [24]. The discerning characteristic of hyper-heuristics is that the method of choosing heuristics is problem agnostic – there is a clear separation between the problem representation space and hyper-heuristic search space. This makes hyper-heuristics more generally applicable than bespoke methods.

Selection hyper-heuristics comprise of, firstly, the *selection* of suitable heuristics at regular selection intervals, and secondly *acceptance criteria* to decide if a newly proposed solution should be accepted or not [8] [24]. Well-known selection mechanisms include: *simple random*, randomly selecting search methods; *random descent*, selecting a new method only if there is no more improvement in fitness; *reinforcement learning*, which steers heuristic selection based on performance; *greedy*, which applies all heuristics and selects the best performing heuristic; and *choice function*, which uses a learning mechanism to learn which search methods are best to apply at each heuristic selection interval [8]. The use of meta-heuristics as hyper-heuristics to learn and optimise heuristic selection is becoming an active research topic [24]. Corresponding acceptance criteria include: *all moves*, where a newly proposed solution is always accepted; *improving and equal*, where only solutions that are better or equal to the current best solution are accepted; *only improving*, which only accepts improving solutions; and various *stochastic mechanisms*, which use simulated annealing approaches to accept inferior solutions at a decreasing probability over time [8].

Recent studies incorporate meta-heuristics as heuristics [24]. A notable method is the *heterogeneous meta-hyper-heuristic* (HMHH) [25] [26]. An entity is a single candidate solution. The HMHH manages a pool of heuristics that run in parallel. Entities are divided among the heuristics into disjoint subpopulations. Entities' behaviours are modified by assigning

entities to different heuristics. At each selection interval, the HMHH selects the best heuristic to assign each entity to. State information is reinitialised if needed. The length of an interval between selections is k iterations of each heuristic in the pool. The HMHH algorithm is outlined in algorithm 1.

Hyper-heuristic performance on dynamic optimisation problems is a becoming an active research topic. Kiraz *et al.* investigate the performance of *simple random*, *greedy*, *choice function* and *reinforcement learning* selection mechanisms with *improving and equal* acceptance criteria in dynamic environments [10]. The heuristics pool contains different Gaussian mutation operators with different standard deviations and zero mean. The study only considers spatial and temporal severity of changes as outlined in [15]. The moving peaks benchmark is used [21]. In contrast, this paper uses the representative environments of the more comprehensive classification of Duhain *et al.* [20].

Ozcan *et al.* investigate a *greedy* selection mechanism for different dynamic environments [11]. The study also considers only spatial and temporal severity of changes as outlined in [15]. Heuristics are also restricted to simple Gaussian mutation operators with zero mean and different standard deviations on a bitwise representation adapted from genetic algorithms. The study does not use the moving peaks benchmark.

Kiraz *et al.* propose an ant-based selection mechanism for dynamic environments utilising pheromone intensities between all pairs of heuristics [12]. The pheromone values are used to decide the heuristic application ordering. Roulette wheel selection and tournament selection is investigated. Simple Gaussian mutation operators with zero mean and different standard deviations are also used as heuristics. The study also uses the moving peaks benchmark and only considers spatial and temporal severity of changes as outlined in [15].

Uludag *et al.* hybridise hyper-heuristic selection and population-based incremental learning by combining off-line training and on-line learning techniques [9]. The study focuses on binary coded discrete environments. In the off-line learning phase, representative examples of different environment types are sampled to learn probability vectors for an estimation of distribution algorithm. In the on-line learning phase, the probability vectors are used as heuristics to seed the search with probable candidate solutions. The study considers both spatial and temporal severities as well as cycle lengths.

Topcuoglu *et al.* propose a hybrid strategy of embedding hyper-heuristic selection mechanisms into memory/search algorithms to solve the generalised assignment problem and the moving peaks benchmark [13]. The heuristic pool consists of simple Gaussian mutation operators with a zero mean and different standard deviations. The moving peaks benchmark is used with parameter settings that represent a mild abrupt environment when compared to the representative landscape that is described by Duhain *et al.* There are 50 iterations between changes and relatively large landscape modifications.

All the studies above use simple heuristics such as Gaussian mutation or probability estimations. No methods that are specialised for dynamic optimisation problems are considered as heuristics. As such the adaptability of the algorithm in a dynamic environment is solely dependant on the hyper-heuristic selection mechanism. While all the studies consider spatial and temporal severity changes, none use the rigorous classification of Duhain *et al.* [20].

Algorithm 1 Heterogeneous Meta-Hyper-Heuristic [25] [26]

$\chi \leftarrow$ Population of solution entities
$\mathbf{A} \leftarrow$ Population of constituent algorithms
$t = 0, k = 5$
for each entity $e_i \in \chi$ **do**
    $a_{mi}(t) \leftarrow$ assign e_i to random algorithm a_m
end for
while no stopping condition is met **do**
    for each algorithm $a_m \in \mathbf{A}$ **do**
        Apply a_m for k iterations
    end for
    for each entity $e_i \in \chi$ **do**
        $Q_{\delta m}(t) \leftarrow$ total fitness improvement for entity i
        $a_{mi}(t+k) \leftarrow$ HYPER-HEURISTIC$(e_i, Q\delta_m(t))$
    end for
    $t = t + k$
end while

III. Experimental procedure

A. Goals

This paper's goals are, firstly, to show that the same hyper-heuristic managing meta-heuristics that are specifically designed to solve dynamic optimisation problems performs better than when using simple heuristics. Secondly, the study aims to show that a hyper-heuristic managing a pool of meta-heuristics designed for dynamic environments can outperform those same meta-heuristics that are applied in isolation. Thirdly, the study highlights the difference in performance of the same hyper-heuristic over different types of dynamic environment as discussed in section II. The study uses the representative examples of each type of dynamic environment as per Duhain *et al.'s* comprehensive classification [20]. The results highlight the need for a more comprehensive approach to crafting hyper-heuristics for different dynamic environments.

B. Hyper-heuristic and meta-heuristic choices

The *heterogeneous meta-hyper-heuristic* algorithm [25] [26] with random-based selection is used as the hyper-heuristic. *Simple random* selection makes no effort to learn the heuristic search space, thus preventing any possibility of pre-mature hyper-heuristic convergence or outdated memory at the hyper-heuristic level. This eliminates potential bias that may emerge if different types of dynamic environments yield different heuristic search spaces. *Simple random* selection thus provides an even playing field to compare the performance of the same hyper-heuristic across different types of dynamism.

The managed heuristics are meta-heuristics that are specialised to solve dynamic optimisation problems, namely *quantum particle swarm optimisation* (QSO) [5], *atomic*

charged particle swarm optimisation (CPSO) [6], a variation of the *random immigrants genetic algorithm* (RIGA) [27], and *rand/1/bin differential evolution* (DE) with a high mutation rate after an environment change [1]. Each meta-heuristic uses default parameter settings as found in literature (listed in table I). The aim is to investigate how well the hyper-heuristic handles the algorithmic differences between meta-heuristics, and not to conduct a sensitivity analysis of the meta-heuristics.

Each heuristic has a minimum entity count of five to avoid stalling any particular heuristic's execution. Using algorithm 1, each heuristic in the pool is run in parallel for $k = 5$ iterations. After k iterations the entities are randomly reassigned to different heuristics. All environment changes are made at regular intervals as stipulated by the representative examples from Duhain *et al.* (see table II). Upon any environment change, each entity's fitness is re-evaluated to avoid outdated memory of entity fitness. It is key to stress that the technique allows any meta-heuristic tuned with any choice of parameter values to be used. The aim of this study is to highlight the performance at the hyper-heuristic level. The hyper-heuristic uses the same parameters for each heuristic as when the heuristic is run in a stand-alone fashion.

TABLE I: Meta-heuristic parameter values

Parameter	**Value**	**Parameter**	**Value**
Quantum PSO		**Random Immigrants GA**	
charge ratio	0.5	*Immigration ratio*	0.1
r_{cloud}	5	*cross-over*	*Uniform*
Atomic CPSO		*mutation*	*Gaussian*
R_p	5	*selection*	*Elitism*
R_c	2	**Rand/1/bin DE**	
charge	16	*mutation rate*	0.25
charge ratio	0.5		

TABLE II: Moving Peaks benchmark test environments

	Static	**Progressive**	**Abrupt**	**Chaotic**
peaks	$\{5\}$	$\{5\}$	$\{5\}$	$\{5\}$
h_p	$\in [30, 70]$	$\in [30, 70]$	$\in [30, 70]$	$\in [30, 70]$
w_p	$\in [0.8, 7]$	$\in [0.8, 7]$	$\in [0.8, 7]$	$\in [0.8, 7]$
heightSeverity	0	1	10	10
widthSeverity	0	0.05	0.05	0.05
s	0	1	50	50
λ	0	0	0	0
iterations	∞	1	200	5

C. Dynamic optimisation problem instance

Problem instances are created using Branke's moving peaks benchmark (MPB) [21]. The MPB allows any number of peaks to be generated along with the desired dynamics. The function value f is the maximum of the combined peak functions, i.e.

$$f(\mathbf{x}, t) = \max\{B(\mathbf{x}), \max_{p=1..n_p}\{P(\mathbf{x}, h_p(t), w_p(t), \mathbf{l}_p(t))\}\} \tag{4}$$

where f is the MPB function, B is the basis function landscape (zero in this study), n_p is the number of peaks, P defines the peak for each peak p with height h_p, width w_p, and location $\mathbf{l}_p$ at time t. In this study P is defined as

$$P(\mathbf{x}, h_p(t), w_p(t), \mathbf{l}_p(t)) = h_p(t) - w_p(t)\sqrt{\sum_{j=1}^{n_x}(\mathbf{l}_p(t) - x_j)^2} \tag{5}$$

where n_x is the dimension of $\mathbf{x}$. Peak heights and widths are modified as

- height: $h_p(t) = h_p(t-1) + \textit{heightSeverity} \cdot \sigma(t)$
- width: $w_p(t) = w_p(t-1) + \textit{widthSeverity} \cdot \sigma(t)$

where $\sigma(t) \sim N(0, 1)$. The MPB shift vector $\mathbf{s}_v(t)$ is

$$\mathbf{s}_v(t) = \frac{s}{|\mathbf{p}_r + \mathbf{s}_v(t-1)|}((1-\lambda)\mathbf{p}_r + \lambda\mathbf{s}_v(t-1)) \tag{6}$$

where $\mathbf{p}_r$ is a random vector normalised to length s (the spatial severity).

D. Experiment design

The four behavioural classes as presented by Duhain *et al.* are considered, namely *static*, *progressive*, *abrupt*, and *chaotic*. Each environment is Type III in Eberhart *et al.*'s classification [15] [16] and random as per Angeline's classification [14]. For each class of problem, five peaks are considered in 5, 10, 30 and 50 dimensional problems, giving 16 problem instances. The parameters to generate each problem are listed in tables II and III. Six algorithms with 100 entities each are evaluated for 1000 iterations over 30 simulations on each problem instance. Each algorithm thus has the same number of fitness evaluations per problem instance. Four of the algorithms are the individual meta-heuristics discussed in section III-B. This establishes a baseline performance for each meta-heuristic in isolation. The HMHH is used as the hyper-heuristic as laid out in algorithm 1. The selection interval length is taken as $k = 5$ as used in [25] [26]. For comparison purposes the same HMHH using a *simple random* selection mechanism is set up to manage simple Gaussian mutation operators with zero mean and standard deviation, $\sigma \in \{0.5, 2, 7, 15, 20, 25, 30\}$ and is designated as HHG. These settings are representative of that of [10] [11] [12] [13].

IV. Results and discussion

The CME, ABEBC, and ABEAC as defined in section II-B are used to evaluate algorithm performance. For each combination of error metric, problem instance, and dimension a Kruskal-Wallis test is performed to ascertain whether or not there are statistical differences among the performance of any pairs of algorithms [28]. If the results are deemed different, a pairwise Mann-Whitney-Wilcoxon rank sum test with Holm correction is used to assess individual differences [29]. A value of 1 is allocated to an algorithm if it is superior to another, where the inferior algorithm is allocated a score of -1. Table IV shows the win/loss ranks of each algorithm across each dimension and environment. The graphs below show box-plots of the performance of each algorithm in each dimension in each type of environment.

TABLE III: Hyper-heuristic parameters

Parameter	Value
Interval length (k)	5
Total iterations (t_{max})	1000
Entities in population	100
Dimensions (d)	$d \in \{5, 10, 30, 50\}$
Function domain	$R(0, 100)^d$
Boundary constraint	*Unconstrained*

A. Analysis of collective means error

Figures 2, 3, 4 and 5 respectively show the CME ranges for the static, progressive, abrupt and chaotic environments. The majority of the literature reviewed in section II only considers five dimensions. It is notable that every algorithm has difficulty in 10, 30 and 50 dimensions MPB problems and no algorithm achieves a low CME. Since the CME never reaches an acceptable level in dimensions other than five, the rest of this section only considers five dimensions.

With reference to table IV the HMHH performs badly in static environments, with each of the HMHH constituent heuristics performing better in isolation than the HMHH. In the dynamic environments the HMHH is however always the top, or tied for top, performing algorithm when compared to the meta-heuristics in isolation. In progressive environments the HMHH consistently achieves the lowest median and smallest deviation as can be seen in figure 3. Table IV shows that the HMHH is consistently the winner when compared to the meta-heuristics in isolation. For the abrupt environment the HMHH is also the best performing algorithm, with no other meta-heuristic except the CPSO matching its performance, as can be seen in both figure 4 and table IV. All of the algorithms struggle in the chaotic environment where the HMHH achieves comparable performance to the meta-heuristics in isolation.

For all four environment types the HMHH always outperforms the HHG. It is interesting to note that HHG, which is based on simple Gaussian mutation, consistently performs the worst out of all the algorithms. In the dynamic environments the HHG consistently performs worse than the meta-heuristics that are purpose-built to solve dynamic optimisation problems. For the static environment, while the HMHH does not do as well as the meta-heuristics in isolation, the HHG never achieves an acceptable CME while the HMHH consistently achieves a CME less than 20 as can be seen in figure 2. Further study is needed to determine if shorter selection intervals might improve the performance of the HHG.

It is clear that in five dimensions there is an advantage to using a hyper-heuristic with random selection above just the constituent meta-heuristics in isolation. It is also unequivocally clear that a hyper-heuristic operating on meta-heuristics purpose-built for dynamic environments greatly outperforms the same hyper-heuristic operating on simple heuristics fit for static environments. Table IV confirms this by comparing the sum of ranks between the HMHH and HHG for just the CME scores in five dimensions.

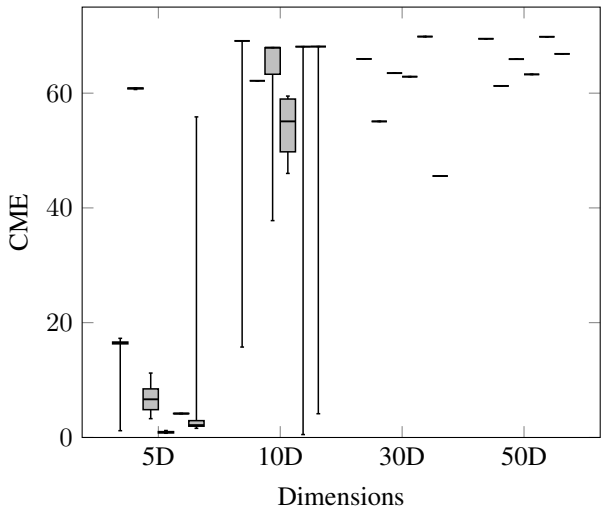

Fig. 2: CME for static environments – box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

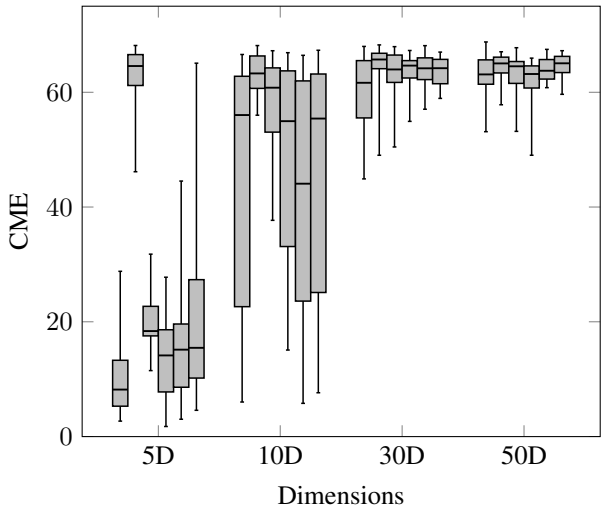

Fig. 3: CME for progressive environments – box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

B. Analysis of average best error before change

Figure 6 indicates that the HMHH is able to achieve an acceptably low ABEBC, and that only the CPSO has similar performance. This indicates that the HMHH is able to exploit the new landscape for abrupt environments. The HMHH always outperforms the HHG. In chaotic environments no algorithm is the clear winner as can be seen in figure 7, although again the HMHH significantly outperforms the HHG.

C. Analysis of average best error after change

For abrupt environments the HMHH achieves a much lower median ABEAC than the other algorithms as seen in figure 8, indicating that the HMHH has better stability than the other approaches. The HMHH performs on par to other algorithms in the chaotic environment. In both environments the HMHH performs better than the HHG.

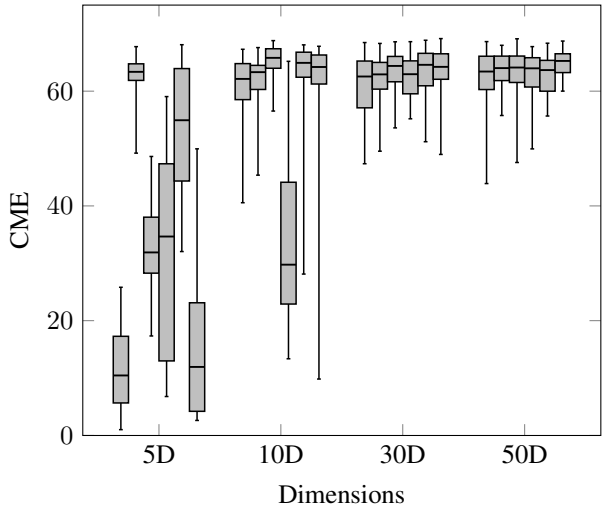

Fig. 4: CME for abrupt environments – box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

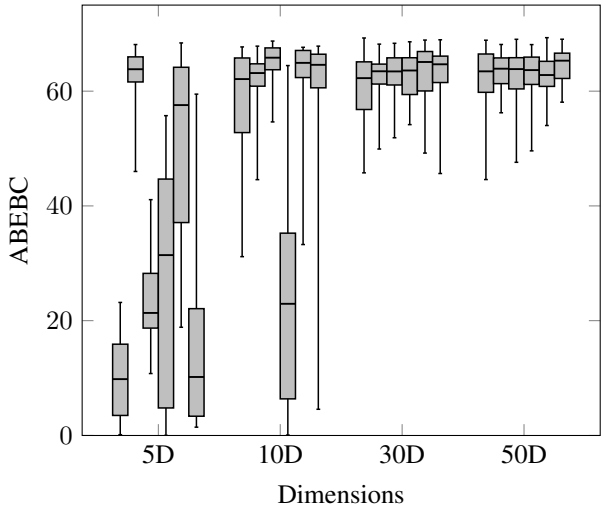

Fig. 6: ABEBC for abrupt environments – box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

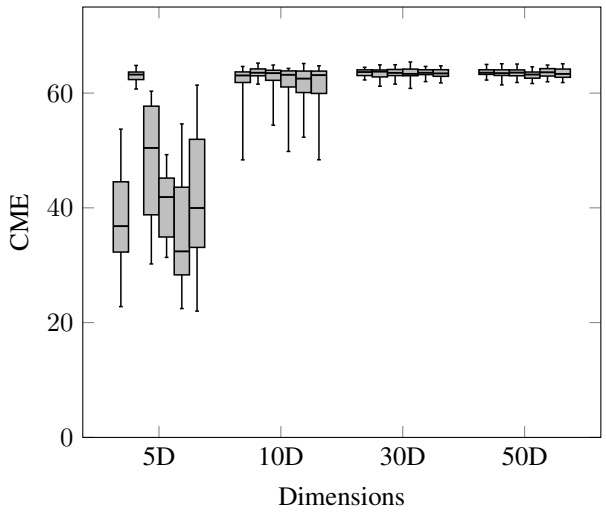

Fig. 5: CME for chaotic environments –box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

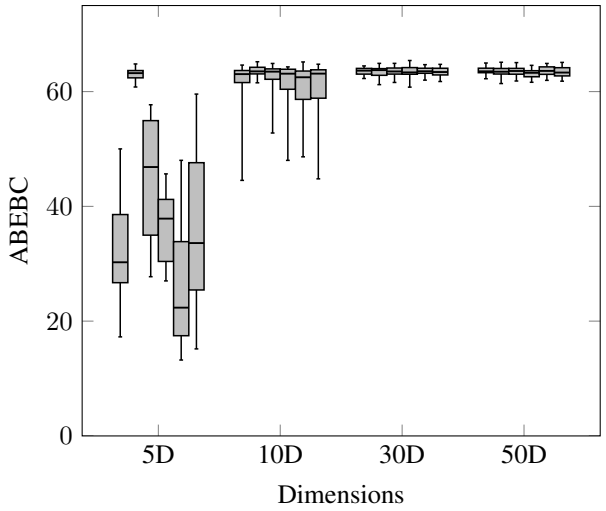

Fig. 7: ABEBC for chaotic environments – box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

V. Conclusion

All algorithms struggle to perform well on the MPB problem in dimensions higher than five with the given parameters (see table II). Further investigation is required to ascertain why this is the case, although a sparse function landscape is suspected, i.e. the overwhelming majority of the landscape has a near zero value in high dimensions.

This study highlights that a random-based selection hyper-heuristic is a viable method to solve dynamic optimisation problems. The results show that a random-based selection hyper-heuristic managing a pool of meta-heuristics purpose-built for dynamic environments can 1) outperform those same meta-heuristics working in isolation, and 2) performs much better than when using simple Gaussian mutation operators as heuristics as is done in existing literature.

Different types of dynamic environments also yield different performances for the HMHH when the heuristic pool consists of purpose-built meta-heuristics: performance was superior to all other approaches in the progressive environment; in the abrupt environment the HMHH is also the best performing algorithm and is matched by only one of the constituent heuristics (CPSO); and performance of the HMHH is on par with the other approaches in the chaotic environment. For the static environment the HMHH does not compare as well to the performance of each of the meta-heuristics in isolation. This performance variation for different types of dynamic environments highlights the need to develop more sophisticated selection mechanisms that can adapt to the type of dynamic environment at hand. In this study, only *simple random* selection is considered as a selection mechanism to remove any possible biases. A future study incorporating different selection mechanisms over a pool of meta-heuristic

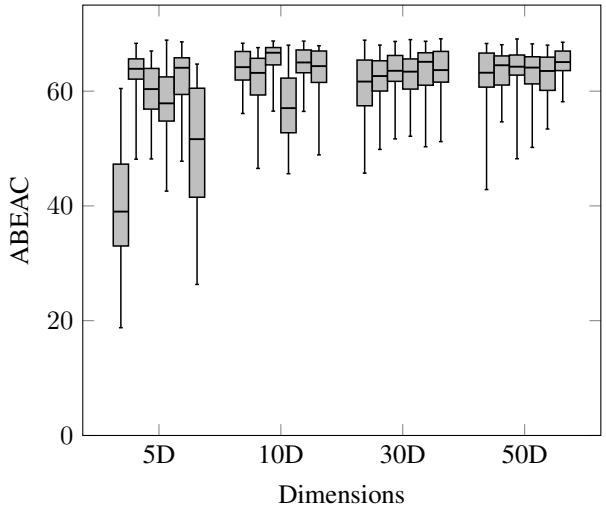

Fig. 8: ABEAC for abrupt environments – box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

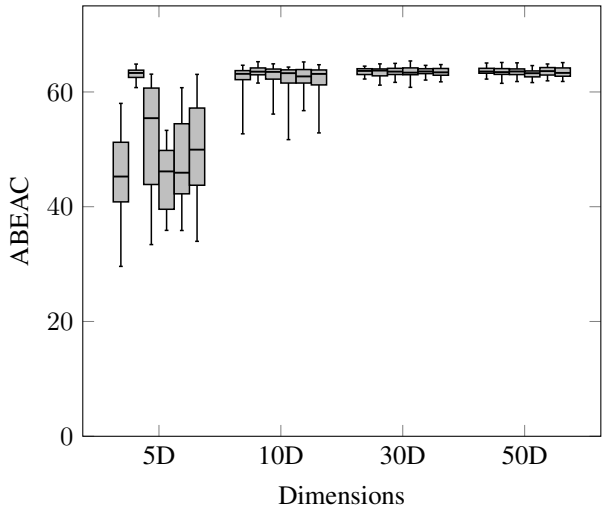

Fig. 9: ABEAC for chaotic environments – box-plot order: HH, HHG, RIGA, DE, QSO, CPSO

specialised to solve dynamic optimisation problems should prove interesting.

REFERENCES

[1] A. P. Engelbrect, *Computational Intelligence: An Introduction*, 2nd ed. Wiley, 2007.

[2] J. G. Duhain, "Particle swarm optimisation in dynamically changing environments," Ph.D. dissertation, University of Pretoria, 2011.

[3] V. Noroozi, A. B. Hashemi, and M. R. Meybodi, "Cellularde: a cellular based differential evolution for dynamic optimization problems," in *Adaptive and natural computing algorithms*. Springer, 2011, pp. 340–349.

[4] C. Cruz, J. R. González, and D. A. Pelta, "Optimization in dynamic environments: a survey on problems, methods and measures," *Soft Computing*, vol. 15, no. 7, pp. 1427–1448, 2011.

[5] T. Blackwell and J. Branke, "Multi-swarm optimization in dynamic environments," in *Applications of Evolutionary Computing*. Springer, 2004, pp. 489–500.

[6] T. M. Blackwell, P. J. Bentley *et al.*, "Dynamic search with charged swarms." in *GECCO*, vol. 2. Citeseer, 2002, pp. 19–26.

[7] R. Mendes and A. S. Mohais, "Dynde: a differential evolution for dynamic optimization problems," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3. Ieee, 2005, pp. 2808–2815.

[8] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art. school of computer science and information technology, university of nottingham," *Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747*, 2010.

[9] G. Uludağ, B. Kiraz, A. Ş. Etaner-Uyar, and E. Özcan, "A hybrid multi-population framework for dynamic environments combining online and offline learning," *Soft Computing*, vol. 17, no. 12, pp. 2327–2348, 2013.

[10] B. Kiraz, A. Ş. Uyar, and E. Özcan, "An investigation of selection hyper-heuristics in dynamic environments," in *Applications of Evolutionary Computation*. Springer, 2011, pp. 314–323.

[11] E. Ozcan, S. E. Uyar, and E. Burke, "A greedy hyper-heuristic in dynamic environments," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. ACM, 2009, pp. 2201–2204.

[12] B. Kiraz, A. Ş. Etaner-Uyar, and E. Özcan, "An ant-based selection hyper-heuristic for dynamic environments," in *Applications of Evolutionary Computation*. Springer, 2013, pp. 626–635.

[13] H. R. Topcuoglu, A. Ucar, and L. Altin, "A hyper-heuristic based framework for dynamic optimization problems," *Applied Soft Computing*, vol. 19, no. 0, pp. 236 – 251, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S156849461400057X

[14] P. J. Angeline, "Tracking extrema in dynamic environments," in *Evolutionary Programming VI*. Springer, 1997, pp. 335–345.

[15] J. Branke, *Evolutionary optimization in dynamic environments*. kluwer academic publishers, 2001.

[16] J. Branke, E. Salihoğlu, and Ş. Uyar, "Towards an analysis of dynamic environments," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, pp. 1433–1440.

[17] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1. IEEE, 2001, pp. 94–100.

[18] X. Hu and R. Eberhart, "Tracking dynamic systems with pso: wheres the cheese," in *Proceedings of the workshop on particle swarm optimization*, 2001, pp. 80–83.

[19] K. De Jong, "Evolving in a changing world," in *Foundations of Intelligent Systems*. Springer, 1999, pp. 512–519.

[20] J. G. Duhain and A. P. Engelbrecht, "Towards a more complete classification system for dynamically changing environments," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE, 2012, pp. 1–8.

[21] J. Branke, "The moving peaks benchmark," *URL: http://www. aifb. uni-karlsruhe. de/˜ jbr/MovPeaks/movpeaks*, 1999.

[22] R. W. Morrison, "Performance measurement in dynamic environments," in *GECCO workshop on evolutionary algorithms for dynamic optimization problems*. Citeseer, 2003, pp. 5–8.

[23] K. Trojanowski and Z. Michalewicz, "Searching for optima in non-stationary environments," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3. IEEE, 1999.

[24] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[25] J. Grobler, A. P. Engelbrecht, G. Kendall, and V. Yadavalli, "Alternative hyper-heuristic strategies for multi-method global optimization," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[26] ——, "Multi-method algorithms: Investigating the entity-to-algorithm allocation problem," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 570–577.

[27] J. J. Grefenstette *et al.*, "Genetic algorithms for changing environments," in *PPSN*, vol. 2, 1992, pp. 137–144.

[28] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[29] H. B. Mann, D. R. Whitney *et al.*, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, vol. 18, no. 1, pp. 50–60, 1947.

TABLE IV: Algorithm ranks

Problem	Error	HMHH				HH-Gaussian				RIGA				DE				QSO				CPSO				Win/loss check
		5	10	30	50	5	10	30	50	5	10	30	50	5	10	30	50	5	10	30	50	5	10	30	50	
Static	CME	-3	-5	-3	-3	-5	3	3	5	-1	1	-1	1	5	5	1	3	1	-1	-5	-5	3	-3	5	-1	0
Progressive	CME	3	1	0	0	-5	-4	0	0	-2	-1	0	0	2	1	0	1	2	2	0	0	0	1	0	-1	0
Abrupt	CME	4	0	0	0	-5	0	0	0	0	-3	0	0	0	5	0	0	-3	-1	0	0	4	-1	0	0	0
	ABEBC	3	3	3	3	-1	-1	-1	-1	1	1	1	1	5	5	5	5	-5	-5	-5	-5	-3	-3	-3	-3	0
	ABEAC	3	3	3	3	-1	-1	-1	-1	1	1	1	1	5	5	5	5	-5	-5	-5	-5	-3	-3	-3	-3	0
	ITEL	3	3	3	3	-1	-1	-1	-1	1	1	1	1	5	5	5	5	-5	-5	-5	-5	-3	-3	-3	-3	0
Chaotic	CME	2	0	0	0	-5	0	0	0	-3	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	0
	ABEBC	3	3	3	3	-1	-1	-1	-1	1	1	1	1	5	5	5	5	-5	-5	-5	-5	-3	-3	-3	-3	0
	ABEAC	3	3	3	3	-1	-1	-1	-1	1	1	1	1	5	5	5	5	-5	-5	-5	-5	-3	-3	-3	-3	0
	ITEL	3	3	3	3	-1	-1	-1	-1	1	1	1	1	5	5	5	5	-5	-5	-5	-5	-3	-3	-3	-3	0
Win/loss totals		24	14	15	15	-26	-7	-3	-1	0	3	5	7	39	41	31	34	-28	-30	-35	-35	-9	-21	-13	-20	