Analysis of Global Information Sharing in Hyper-heuristics for 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. Hyper-heuristics show great promise in handling dynamic environment dynamics because hyper-heuristics adapt to their environment. Different classifications of dynamic environments describe change dynamics such as spatial change severity, temporal change severity, homogeneity of peak movement, etc. Previous studies show that different hyper-heuristic selection mechanisms perform differently across different types of dynamic environments. This study investigates three hyper-heuristic selection methods with different selection pressures and shows an inverse correlation with environment change severity.**

I. Introduction

The fitness landscape of a dynamic optimisation problem changes over time. Formally, to solve a dynamic optimisation problem means minimising $f(\vec{x}, \vec{\omega}(t))$ with $\vec{x} = (x_1, ..., x_{n_x})$ and $\vec{\omega}(t) = (\omega_1(t), ..., \omega_{n_\omega}(t))$ subject to 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 goal is to find the optimum at time t namely $\vec{x}^*(t) = \min_{\vec{x}} f(\vec{x}, \vec{\omega}(t))$. Landscape change dynamics can be non-trivial, which makes optimization algorithms that have been developed for static problems ineffective in dynamic environments. Challenges include outdated memory of fitness values, the inability to detect, track and adapt to change, and diversity loss [1] [2] [3]. Specialised methods have been developed to solve dynamic optimisation problems [4]. Well-known algorithms include charged particle swarm optimisation (CPSO) [5], quantum particle swarm optimisation (QSO) [6], and dynamic differential evolution (DynDE) [7], amongst others.

Loosely speaking, hyper-heuristics are 'heuristics to choose heuristics' [8]. A hyper-heuristic selectively applies different heuristics at various stages while solving a problem. A key differentiator of hyper-heuristics over other selection methods is that hyper-heuristics are completely domain agnostic – a hyper-heuristic does not need to be cognisant of the problem it solves. The intuitive expectation is that hyper-heuristics should be well-suited to solve dynamic optimisation problems because different heuristic search methods can be brought to bear at different times during the search. Recently a number of studies have investigated this expectation [9] [10] [11] [12] [13] [14].

Most of the studies above focus on a subset of dynamic environments, i.e. the full spectrum of landscape change capabilities is not fully investigated. Duhain and Engelbrecht identified four key types of dynamic environments based on the environment's change frequency and change severity [3]. This study uses this comprehensive dynamic environment classification to determine if different kinds of hyper-heuristics perform differently across various types of dynamic optimisation problems. Van der Stockt and Engelbrecht found that a random selection hyper-heuristic can outperform the individual constituent algorithms in certain types of dynamic environments [14]. This paper extends the previous study by investigating three different hyper-heuristics across different types of dynamic environments. Hyper-heuristics with and without global information sharing between heuristics are examined. Specifically, the trade-off between increased performance and increased computational complexity when including or excluding global search information is investigated. The results indicate that there is an inverse correlation between the change severity of a dynamic environment and the selection pressure of the hyper-heuristic selection mechanism.

Furthermore, while global information sharing does improve performance, it comes at the cost of significantly increased computational complexity. Section II discusses related work in dynamic environment classifications, hyper-heuristics and meta-heuristics. Section III outlines the mechanism, possible benefits and cost trade-off in sharing global state information across heuristics. Section IV outlines the settings used in experiments. Section V discusses the experimental results across various types of dynamic environments. Section VI concludes the study.

II. Related Work

A. Dynamic environment classifications

Dynamic optimisation problems have been classified in various ways based on their behaviour [15] [16] [17] [18] [19] [20]. A dynamic environment can behave differently w.r.t the trajectory of its optima, frequency and magnitude of changes, homogeneity of optima movement, and length of change cycles. Eberhart *et al.* [18] [19] defined three types of dynamic environments based on the direction of change of the optima: Type I environments, where optima positions change but not their values; Type II environments, where optima values change but not their positions; and Type III where both the optima values and positions change. Angeline showed that optima trajectory changes are either linear, circular or random [15]. Duhain and Engelbrecht combines the classifications of Eberhart *et al.* and Angeline together with new classes based on spatial and temporal change severity [3]. This comprehensive classification comprises of four major types of dynamic environments together with Eberhart *et al.*'s and Angeline's classifications to yield 27 specific types of dynamic environments. The four major environment types (as shown in Figure 1) are:

- **(Quasi-)Static environments** have low temporal and spatial change severities compared to the scale of the problem. The landscape behaves like a static environment.
- **Progressively changing environments** change extremely often, but with small changes.
- **Abruptly changing environments** have severe changes that happen infrequently. In effect the landscape is a series of different static environments for extended periods.
- **Chaotic changing environments** change very frequently with severe landscape changes.

Each type of environment has unique dynamics, which intuitively means that any given heuristic should perform differently in each type of environment. A hyper-heuristic should ideally prefer the best performing heuristic to be the active heuristic without having direct knowledge of the landscape dynamics. The aim is to obtain improved performance when compared to running those same heuristics in isolation, either through better overall accuracy, faster convergence, or both after landscape changes occur. Van der Stockt and Engelbrecht demonstrated that increased performance is possible using hyper-heuristics in dynamic environments [14].

B. Hyper-heuristics

Hyper-heuristics are loosely defined as 'heuristics to choose heuristics' and can include search or learning components for either selecting or generating heuristics to solve a problem [21]. Hyper-heuristics are related to fields such as ensembles, algorithm portfolios, adaptive memetic algorithms, meta-learning, evolutionary computing parameter control and adaptive operator selection [8] [21]. What distinguishes hyper-heuristics from these fields is that selection or generating methods are problem agnostic – hyper-heuristics are completely domain independent with a clear separation between

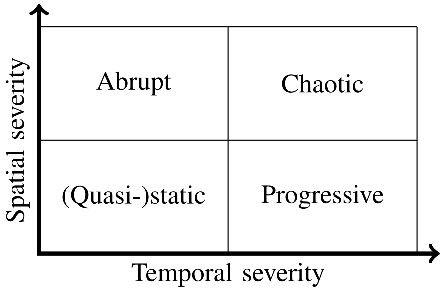

Fig. 1: Four classes of dynamic environments [3]

the problem representation and the hyper-heuristic search space. The ultimate goal of hyper-heuristics is to create domain independent reusable search methods that reduce the need for labour-intensive bespoke systems or costly tuning exercises.

Early selection hyper-heuristics were simple methods to choose which heuristic to apply next [8] [21] [22]. The chosen heuristic would then be applied successively until the next selection interval. Examples of these early hyper-heuristics are: *simple random*, which chooses a heuristic randomly; *random descent*, which selects 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 both as hyper-heuristics (to learn and optimise heuristic selection) and heuristics is becoming an active research topic [21]. A notable method is the *heterogeneous meta-hyper-heuristic* (HMHH) [23] [24]. The HMHH maintains a population of candidate solutions (called *entities*) and a pool of meta-heuristics that serve as heuristics. During a selection interval the HMHH decides how all entities are assigned to specific heuristics. An interval consists of k algorithm iterations of each heuristic. While each entity is assigned to only one heuristic at any time, the HMHH provides the option for heuristics to use global search information, i.e. each heuristic may or may not use entity information from other heuristic's assigned entities. An overview of the HMHH algorithm is given in Algorithm 1.

Hyper-heuristics are intuitively well-suited to solve dynamic optimisation problems – as the problem space changes over time a hyper-heuristic should ideally change the search methods that are applied accordingly. Note that the hyper-heuristic would not need to know when or how the environment changes, but should ideally give more weight to the methods that perform best in any given landscape. Early examples to solve dynamic optimisation problems show hyper-heuristics managing relatively simple heuristics, where the hyper-heuristic is solely responsible for adapting to the problem dynamics by selecting the most suitable heuristic:

- Kiraz *et al.* employed *simple random*, *greedy*, *choice*

Algorithm 1 Heterogeneous Meta-Hyper-Heuristic [23] [24]

```
1: $\chi \leftarrow$ Population of solution entities
2: $\mathbf{A} \leftarrow$ Population of constituent algorithms
3: $t = 0, k = 10$
4: for each entity $e_i \in \chi$ do
5:     $a_{mi}(t) \leftarrow$ assign $e_i$ to random algorithm $a_m$
6: end for
7: while no stopping condition is met do
8:     for each algorithm $a_m \in \mathbf{A}$ do
9:         Apply $a_m$ for $k$ iterations
10:    end for
11:    for each entity $e_i \in \chi$ do
12:        $Q_{\delta m}(t) \leftarrow$ total fitness improvement for entity $i$
13:        $a_{mi}(t+k) \leftarrow$ HYPER-HEURISTIC$(e_i, Q\delta_m(t))$
14:    end for
15:    $t = t + k$
16: end while
```

function and *reinforcement learning* as hyper-heuristics that use simple Gaussian mutation operators (with different standard deviations) as heuristics [10]. The study only considered spatial and temporal severity of changes as outlined in [16]. The moving peaks benchmark was used [25].

- Ozcan *et al.* used a *greedy* selection mechanism, also on simple Gaussian mutational heuristics [11]. The study also considered only spatial and temporal severity of changes. The study did not use the moving peaks benchmark.
- Topcuoglu *et al.* embeded hyper-heuristics selection methods into memory/search algorithms to solve the generalised assignment problem and the moving peaks benchmark [13]. Heuristics are simple Gaussian mutation operators. Only abruptly changing landscapes were considered as per Duhain and Engelbrecht's classification.
- Uludag *et al.* combined off-line and on-line learning techniques to create a hybrid between hyper-heuristic selection and population-based incremental learning (PBIL) [9]. The study focused on binary coded discrete environments, considered both spatial and temporal severities, as well as cycle lengths. Representative examples of different environment types were sampled off-line to learn the probability vectors for an estimation of distribution algorithm (EDA). The EDA was used in the on-line phase to guide the search to promising areas.
- Kiraz *et al.* used an ant-based selection hyper-heuristic that use pheromone intensities between all possible pairings of heuristics to guide the heuristic ordering choice [12]. Simple Gaussian mutation operators were used as heuristics, the moving peaks benchmark was used, and only spatial and temporal changes were considered.
- Van der Stockt and Engelbrecht investigated a *simple random* hyper-heuristic to manage a pool of meta-heuristics that are specialised to solving dynamic optimisation problems [14]. The HMHH (without global information sharing) as outlined above was used. The study showed that hyper-heuristics utilising a pool of heuristics specialised for dynamic environments 1) outperformed hyper-heuristics based on simple methods such as Gaussian mutation operators, and 2) could sometimes outperform the managed heuristics running in isolation. The study showed the difference in performance of the same hyper-heuristic across different types of dynamic environments. Results showed that a *simple random* hyper-heuristic outperformed individual heuristics in progressive environments, outperformed all but one of its own constituent heuristics in abrupt environments, and was on par with other approaches in chaotic environments. The results highlighted the need for a more comprehensive approach to crafting hyper-heuristics for different dynamic environments.

Van der Stockt and Engelbrecht used a *simple random* hyper-heuristic selection mechanism without any global information sharing between heuristics in the pool. This paper expands on these findings by investigating three hyper-heuristics namely *simple random*, *roulette wheel* and *tournament* selection. All three hyper-heuristics are tested with and without global information sharing between heuristics.

III. Fully Informed Hyper-heuristics

A. Sharing Global State Information Between Heuristics

HMHH makes provision to share global state information between heuristics. Van der Stockt and Engelbrecht's initial study did not share global state information [14]. This paper extends Van der Stockt and Engelbrecht's previous study by analysing HMHH performance when each managed heuristic has access to the information of all the other heuristics in the pool. Without sharing, HMHH assigns each of n candidate solutions, called *entities*, across m heuristics at each iteration. Each heuristic h_i thus has its own set of n_i entities which it solely manages, with $\sum n_i = n$. The only way to transfer information between heuristics is when an entity is reassigned to a new heuristic. With sharing, each heuristic h_i has access to its own assigned n_i entities as well as copies of the entities from the other heuristics. Copies are made synchronously, i.e. before each heuristic is run for k iterations. The total number of entities each heuristic can access is n, but each heuristic can only permanently modify the n_i entities assigned to it.

B. Impact and Trade-off

Sharing global information increases computational complexity - each of the m heuristics runs against all n entities, resulting in m times the fitness function evaluations. Without sharing there are $\sum n_i = n$ entities spread over m heuristics and k algorithm iterations which result in $n \times k$ fitness evaluations per hyper-heuristic interval. With global information sharing that becomes $m \times n \times k$ fitness evaluations. The question is whether or not it is worth sharing global information. The hope is that global state information would allow the hyper-heuristic to converge in fewer iterations. Intuitively, if one heuristic finds a good solution, the other heuristics can immediately capitalise on the information, without having to wait for information to permeate through entity reassignment.

On the other hand, since only n_i of the n entities assigned to heuristic h_i can be permanently updated, global information sharing could cause the heuristics to never converge at all. This is not necessarily a bad thing depending on the type of dynamic optimisation problem, i.e. in a chaotic environment this possible non-converging behaviour might actually improve adaptability of the HMHH search. Without sharing, each heuristic can focus on its own search unabated by the results from other heuristics. The only way to share information is when an entity moves between heuristics and brings along different search space information (a new position and fitness). For large numbers of entities and/or a large heuristic pool, it might not be worthwhile to use sharing for the possible gain in performance. This paper investigates the effect that global information sharing has on the performance of three different hyper-heuristic selection mechanisms under the HMHH hyper-heuristic framework.

IV. Experimental Setup

A. Dynamic optimisation problem instances

Duhain and Engelbrecht provided parameters for Branke's *moving peaks benchmark* (MPB) function to create representative examples of each of the four major types of environments outlined in section II-A [25]. All 27 specific types as outlined in section II-A are not considered due to space constraints – each environment is Type III in Eberhart *et al.*'s classification [18] [19] and random as per Angeline's classification [15]. The exact settings for each environment is outlined in Table I. Environment changes are made at regular intervals as per the recommended settings. The MPB generates the desired number of peaks that exude the desired dynamics as outlined by Duhain and Engelbrecht. The function value f is the maximum of the combined peak functions, i.e.

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

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

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

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

$$\mathbf{s}_v(t) = \frac{s}{|\mathbf{p}_r + \mathbf{s}_v(t-1)|}((1-\lambda)\mathbf{p}_r + \lambda\mathbf{s}_v(t-1)) \tag{3}$$

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

TABLE I: 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	∞	10	200	10

B. Hyper-heuristics

The three hyper-heuristics investigated in this study are:

- *Simple random* selection, which assigns each entity to one heuristic with the same probability, $P_i = \frac{1}{|H|}$, where H is the pool of heuristics. *Simple random* selection makes no effort to learn the heuristic search space.
- *Roulette wheel* selection, which assigns each entity to one heuristic $h_i \in H$ relative to each heuristic's overall performance, i.e.
$$P_i = \frac{\Upsilon_i}{\sum_{l=1}^{|H|}\Upsilon_l} \tag{4}$$
where Υ_i is the mean fitness of all the entities, n_i assigned to heuristic h_i.
- *Tournament* selection, which chooses, for each entity $n_i \in n$, a tournament size of $h_T \in H$ heuristics. The entity is assigned to the winning heuristic $h_w \in h_T$, where the winner h_w is the heuristic with the best mean fitness of all entities assigned to the heuristic.

Simple random selection has a very low selection pressure, *roulette wheel* has the highest selection pressure, and *tournament* selection has an intermediate selection pressure compared to *simple random* and *roulette* selection (depending on the tournament size) [1]. Each of the three hyper-heuristics thus have a different '*intensity*' with which they prefer one heuristic over another. Each of the three hyper-heuristics above are ran with and without using global state information, as outlined in section III.

C. Heuristic pool

Four heuristics that specialise in solving dynamic optimisation problems are available in the pool, namely *atomic charged particle swarm optimisation* (CPSO) [5], *quantum particle swarm optimisation* (QSO) [6], a variation of the *random immigrants genetic algorithm* (RIGA) [26], and *rand/1/bin differential evolution* (DE) with an increased rate of mutation when the environment changes [1]. Default parameters as found in the literature are used (see Table II) and heuristics are not tuned to perform better on any type of dynamic environment – the aim is to see how well the hyper-heuristics manage the performance difference between heuristics, and not to conduct a detailed sensitivity analysis of the heuristics. It is

TABLE II: 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	*cross-over*	*Binomial*

also key to stress that any heuristics could have been chosen, as the hyper-heuristics are problem domain agnostic.

Each heuristic always has a minimum of five entities assigned to it to avoid completely restarting the algorithm. Each heuristic is ran in parallel for $k = 10$ algorithm iterations. After k iterations, the active hyper-heuristic reassesses the entity-to-heuristic assignments and possibly makes reassignments as laid out in section IV-B. Entity fitnesses are re-evaluated after environment changes to prevent outdated entity fitnesses.

D. Performance metrics

Since the environment changes over time, different performance measures are needed than in a static environment [2] [3]. Duhain and Engelbrecht recommended evaluating *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 the following performance metrics:

- **Collective mean error** (CME), which records the mean error of the best solution over the entire experiment [27], defined as
$$CME = \frac{\sum_{t=1}^{n_t}(err(t,m))}{n_t} \tag{5}$$
where n_t is the number of iterations per simulation, and $err(t,m)$ measures the difference between the optimum and the best solution found at iteration t in simulation m.
- **Average best error before change** (ABEBC), which records the mean error of the best solution across all time instants preceding an environment change to measure *exploitation capability* [28], defined as
$$ABEBC = \frac{\sum_{c=1}^{n_c} err_{c,r-1}}{n_c} \tag{6}$$
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), which is the mean error of the best solution at the time instants immediately after environment changes and measures *stability* [2], defined as
$$ABEAC = \frac{\sum_{c=1}^{n_c} err_{c,0}}{n_c} \tag{7}$$
where $err_{c,0}$ is the error at the iteration after a change.

E. Experiment Design

Each class of problem had five peaks in five dimensions. The HMHH, as laid out in Algorithm 1, was used as the hyper-heuristic. HMHH was combined with each of the three selection methods in section IV-B and either shared or ignored global state information as outlined in section III. This gave a total of six hyper-heuristics. Each hyper-heuristic was run for 1000 algorithm iterations of the constituent heuristics on 30 random instances of each of the four types of environments with 50 assigned entities. A summary of the settings can be found in Table III.

V. Results and Discussion

For each hyper-heuristic in each environment the applicable error metrics from section IV-D are presented as statistical box-plots. Abbreviations for each hyper-heuristic are NS-R (no sharing-random), NS-Ro (no sharing-roulette), NS-T (no sharing-tournament), S-R (sharing-random), S-Ro (sharing-roulette), and S-T (sharing-tournament). For each combination of error metric, problem instance, and hyper-heuristic a Kruskal-Wallis test was performed to ascertain whether or not there were statistical differences among the performance of any pairs of algorithms [29]. If the results were deemed different, a pairwise Mann-Whitney-Wilcoxon rank sum test with Holm correction was used to assess individual differences [30]. A value of 1 was allocated to an algorithm if it was superior to another, where the inferior algorithm is allocated a score of -1. Table IV shows the win/loss ranks of each algorithm across al environments. Table V shows the pairwise ranks of each hyper-heuristic with and without global information sharing.

TABLE III: Experiment parameters

Parameter	Value
Interval length (k)	10
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*

TABLE IV: All algorithm ranks

Env.	Error	NS-R	NS-Ro	NS-T	S-R	S-Ro	S-T
Static	CME	-3	2	-3	-3	5	2
Progr.	CME	0	0	0	1	1	-2
Abrupt	CME	-2	-5	-2	4	1	4
	ABEBC	-1	-3	1	3	-1	1
	ABEAC	-2	-2	-1	5	-1	1
Chaotic	CME	2	-5	0	5	-3	1
	ABEBC	2	-5	0	5	-3	1
	ABEAC	3	-5	0	5	-3	0
Win/loss totals		-1	-23	-5	25	-4	8

TABLE V: Pairwise algorithm ranks with vs. without sharing

Env.	Error	NS-R	S-R	NS-Ro	S-Ro	NS-T	S-T
Static	CME	0	0	-1	1	-1	1
Progr.	CME	0	0	-1	1	0	0
Abrupt	CME	-1	1	-1	1	-1	1
	ABEBC	-1	1	-1	1	0	0
	ABEAC	-1	1	-1	1	-1	1
Chaotic	CME	-1	1	-1	1	0	0
	ABEBC	-1	1	-1	1	0	0
	ABEAC	-1	1	-1	1	0	0
Win/loss totals		-6	6	-8	8	-3	3

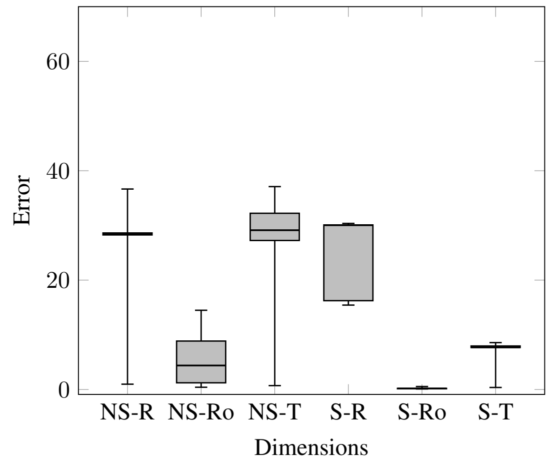

Fig. 2: Static CME

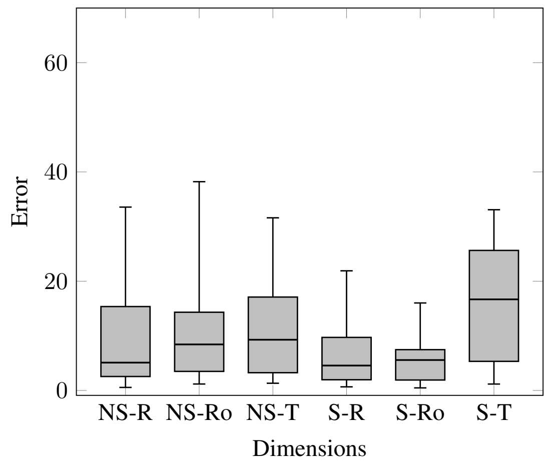

Fig. 3: Progressive CME

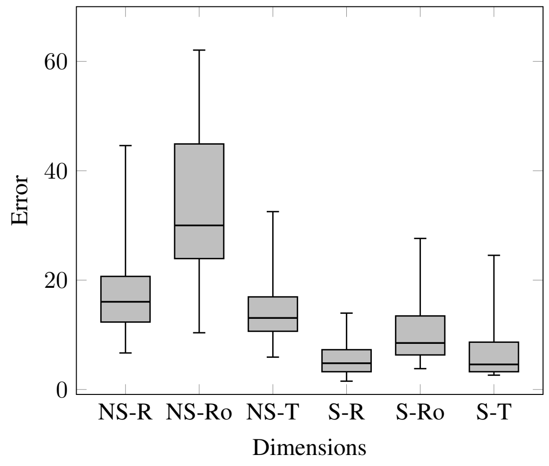

Fig. 4: Abrupt CME

A. Static environments

The CME performance for all six hyper-heuristics is shown in Figure 2. Overall hyper-heuristic performance struggled, which is to be expected since the heuristics are designed for dynamic environments. The S-R, S-Ro and S-T hyper-heuristics consistently achieved better results compared to the NS-R, NS-Ro and NS-T respectively. The best performing hyper-heuristics were NS-Ro and S-Ro (both *roulette selection* variants), suggesting that a hyper-heuristic with a high selection pressure performs well in static environments. Given that a highly selective hyper-heuristic would readily choose the best performing heuristic at each interval, this performance makes sense when the environment never changes. The ABEBC and ABEAC metrics are not applicable in static environments.

B. Progressive environments

The overall CME performance is acceptable in the progressive environment, with all hyper-heuristics managing to reach errors close to zero as shown in Figure 3. Both S-R and S-Ro outperform NS-R and NS-Ro respectively, with much tighter error distributions for the information sharing versions of each hyper-heuristic . As Table IV indicates, S-T performed significantly worse than the other algorithms. The ABEBC and ABEAC metrics are not applicable in progressive environments.

C. Abrupt environments

Figure 4 shows that the CME is significantly better for the information sharing hyper-heuristics (S-R, S-Ro and S-T) compared to the non-sharing variants (NS-R, NS-Ro and NS-T) respectively. It is clear that information sharing greatly improves hyper-heuristic performance in abruptly changing environments. Most notable is the difference in performance between NS-Ro and S-Ro, where information sharing had a massive impact on performance. The ABEBC (see Figure 5) is also similar or better in the information sharing variants of the hyper-heuristics. In abruptly changing environments the landscape changes are infrequent but dramatic, and it is to be expected that the ABEAC should be high. Figure 6 shows that the ABEAC is consistently high for all six hyper-heuristics, with S-R being significatly better than the others. A possible explanation is low diversity across all the included heuristics after environment changes. The information sharing variants (S-R, S-Ro and S-T) have slightly better ABEAC metrics and ranks, which suggests that information sharing does help to maintain higher diversity across heuristics.

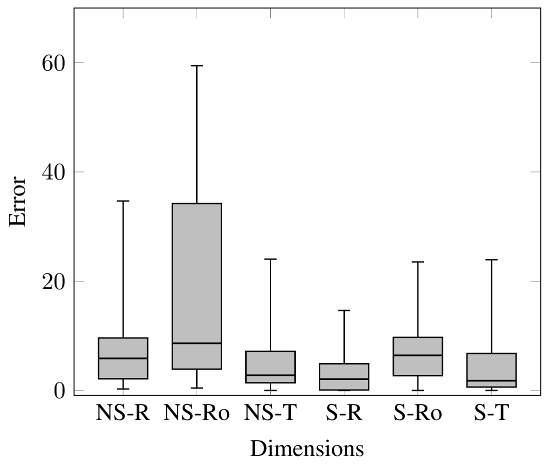

Fig. 5: Abrupt ABEBC

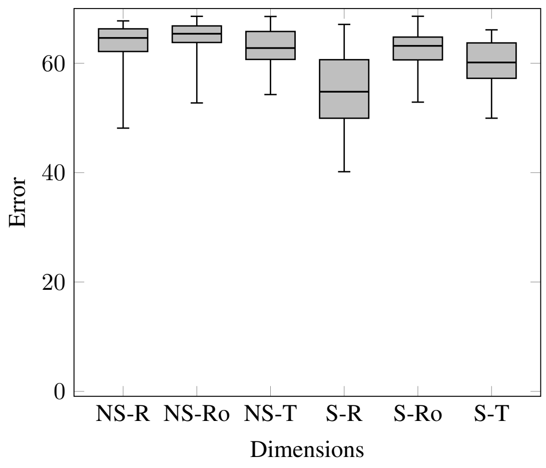

Fig. 6: Abrupt ABEAC

D. Chaotic environments

As shown in Figures 7, 8 and 9, chaotic environments produced vastly different performance outcomes for each hyper-heuristic. The hyper-heuristics with high selection pressure, NS-Ro and S-Ro, performed very badly. This is in stark contrast to static and progressive environments, where both variants of roulette selection were clear winners. Both variants of random selection (NS-R and S-R) performed better than the other methods. This makes intuitive sense given that there are massive landscape changes every 5 iterations and clear environment change patterns are not readily discernible – since random selection has a low selection pressure all heuristics are given an equal chance to address the problem. Supporting this result is the fact that the performance of both medium selection pressure hyper-heuristics (NS-T and S-T) is worse than the low selection pressure hyper-heuristics (NS-R and S-R) and better than the high-selection pressure hyper-heuristics (NS-Ro and S-Ro). The fact that the ABEBC almost mirrors the CME for all six hyper-heuristics indicates that all the methods had difficulty in converging on a good solution before a landscape change occurs. This conclusion is corroborated by

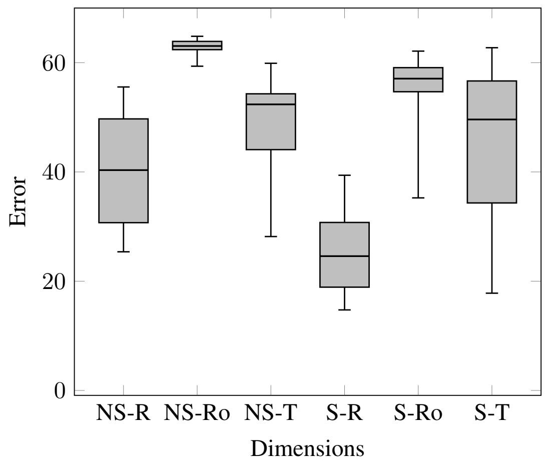

Fig. 7: Chaotic CME

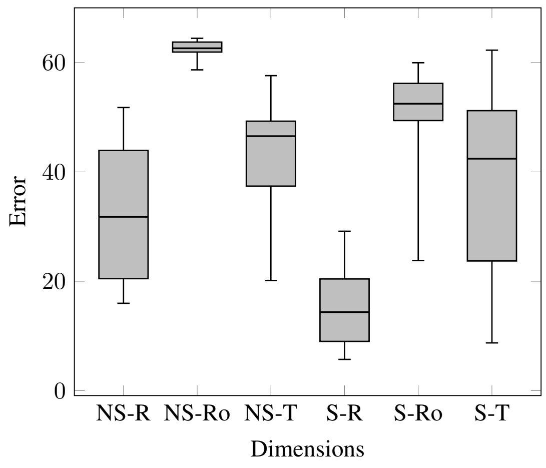

Fig. 8: Chaotic ABEBC

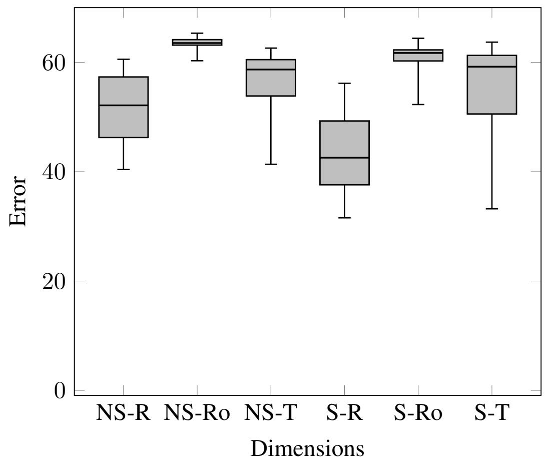

Fig. 9: Chaotic ABEAC

relatively good ABEAC, which seems to suggest that diversity is high just after a change (possibly due to low convergence of candidate solutions in the underlying heuristics).

VI. Conclusion and Future Work

A. Impact of Sharing

It is clear that information sharing between heuristics improve performance significantly. However, there is a trade-off with the increased computational complexity, since information sharing in HMHH results in an m-fold increase in the number of fitness evaluations as per section III. A future study should investigate the performance difference if the hyper-heuristic variants with and without sharing each receive the same number of fitness evaluations i.e. m times more entities are assigned to the non-sharing hyper-heuristic.

B. Impact of Different Hyper-heuristics

The other goal of the paper was to investigate how well hyper-heuristics with different selection pressures perform across different types of dynamic optimisation problems. S-R is by far the overall winner by ranks. Both S-R and NS-R dominate in chaotic environments, which shows that *simple random* selection can adapt better to severe spatial and temporal changes than either *roulette* or *tournament* selection. In abrupt environments NS-R performs on par with NS-T and S-R performs on par or better than S-T. Methods with a medium selection pressure such as *tournament* selection seem to be a good choice for abrupt environments that radically change on an infrequent basis. In progressive environments NS-R perform on par with NS-Ro and NS-T, while S-R similarly performs on par or better than S-Ro and S-T. In contrast, both versions of *simple random* selection are the worst performing method in static environments, while S-Ro and NS-Ro respectively perform the best. This indicates that a strong selection pressure is a good choice for environments that never change.

In conclusion, there seems to be a correlation between the selection pressure of a hyper-heuristic selection methods and the spatial and temporal change severity of the dynamic environment. More investigation is required to confirm and possibly exploit this correlation better.

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] J. Duhain and A. Engelbrecht, "Towards a more complete classification system for dynamically changing environments," in *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, June 2012, pp. 1–8.

[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. M. Blackwell and P. J. Bentley, "Dynamic search with charged swarms," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 19–26. [Online]. Available: http://dl.acm.org/citation.cfm?id=646205.682961

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

[7] R. Mendes and A. S. Mohais, "Dynde: a differential evolution for dynamic optimization problems," in *The 2005 IEEE Congress on Evolutionary Computation*, 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] S. Van der Stockt and A. Engelbrecht, "Analysis of hyper-heuristic performance in different dynamic environments," in *the IEEE Symposium Series on Computational Intelligence – Dynamic and Uncertain Environments*, Orlando/Florida, USA, 2014.

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

[16] J. Branke, *Evolutionary optimization in dynamic environments*. Kluwer academic publishers, 2001.

[17] 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.

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

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

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

[21] 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.

[22] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Practice and Theory of Automated Timetabling III*, ser. Lecture Notes in Computer Science, E. Burke and W. Erben, Eds. Springer Berlin Heidelberg, 2001, vol. 2079, pp. 176–190.

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

[24] J. Grobler, A. Engelbrecht, G. Kendall, and V. Yadavalli, "Multi-method algorithms: Investigating the entity-to-algorithm allocation problem," in *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 570–577.

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

[26] J. J. Grefenstette *et al.*, "Genetic algorithms for changing environments," in *2nd Int. Conf. on Parallel Problem Solving from Nature*, vol. 2, 1992, pp. 137–144.

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

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

[29] 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.

[30] 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.