ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

HYPER-HEURISTICS IN DYNAMIC ENVIRONMENTS

Ph.D. THESIS

Berna KİRAZ

Computer Engineering Department

Computer Engineering Programme

APRIL 2014

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

HYPER-HEURISTICS IN DYNAMIC ENVIRONMENTS

Ph.D. THESIS

Berna KİRAZ
(504082503)

Computer Engineering Department

Computer Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. A. Şima ETANER-UYAR

APRIL 2014

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DİNAMİK ORTAMLARDA ÜST-SEZGİSELLER

DOKTORA TEZİ

Berna KİRAZ
(504082503)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. A. Şima ETANER-UYAR

NİSAN 2014

Berna KİRAZ, a Ph.D. student of ITU Graduate School of Science Engineering and Technology 504082503 successfully defended the thesis entitled **"HYPER-HEURISTICS IN DYNAMIC ENVIRONMENTS"**, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. A. Şima ETANER-UYAR** Istanbul Technical University	....
Co-advisor :	**Assoc. Prof. Dr. Ender ÖZCAN** University of Nottingham	....
Jury Members :	**Prof. Dr. H. Levent AKIN** Boğaziçi University	....
	Prof. Dr. Ahmet Coşkun SÖNMEZ Istanbul Technical University	....
	Assoc. Prof. Dr. Ayşegül GENÇATA YAYIMLI Istanbul Technical University	....
	Asst. Prof. Dr. Ali Fuat ALKAYA Marmara University	....
	Asst. Prof. Dr. Sanem SARIEL-TALAY Istanbul Technical University	....

Date of Submission : 21 March 2014
Date of Defense : 21 April 2014

To Bihter and Emre

FOREWORD

I would like to express my special appreciation and thanks to my advisors, Assoc. Prof. Dr. A. Şima Etaner-Uyar and Assoc. Prof. Dr. Ender ÖZCAN. I appreciate all their encouragement, endless support and the contributions. Their advice on both research as well as on my career is invaluable. The joy and enthusiasm they have for their research was motivational for me, even during tough times in my PhD research. I am especially grateful for weekly meeting with Assoc. Prof. Dr. A. Şima Etaner-Uyar. She was always cheerful, patient and encouraging in these meetings.

I would like to thank my jury members, Prof. Dr. H. Levent Akın and Assoc. Prof. Dr. Ayşegül Gençata Yayımlı for their time, interest, helpful comments and suggestions. I would also like to thank the other three members of my oral defense committee, Prof. Dr. Ahmet Coşkun Sönmez, Asst. Prof. Dr. Ali Fuat Alkaya, and Asst. Prof. Dr. Sanem Sarıel-Talay for accepting to be a member of my defense committee and their time.

I would especially like to thank Gönül Uludağ for her contributions in my thesis. We worked together on the project, namely A Hybrid Multi-population Framework for Dynamic Environments. I appreciated her enthusiasm, intensity, and willingness.

I am indebted to my many colleagues for providing a stimulating and fun filled environment. My thanks go in particular to Asst. Prof. Dr. Fatma Corut Ergin for her friendship and encouragement. I also thank to my officemate, Dr. Işıl Öz, for support and help in Java.

I gratefully acknowledge the Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing scholarship during my PhD research. Our work was also supported by İstanbul Technical University Scientific Research Project (İTÜ / BAP) and supported in part by the EPSRC, grant EP/F033214/1 (The LANCS Initiative Postdoctoral Training Scheme).

Lastly, I would like to thank my whole family for all of the sacrifices that you have made on my behalf. I would like to express appreciation to my husband Alper Kiraz whose love, support and encouragement allowed me to finish this PhD. At the end, a special thanks to my daughter Bihter and my son Emre. I have appreciated their patient and love. Thank you all.

April 2014 Berna KİRAZ

TABLE OF CONTENTS

Page

FOREWORD ..... **ix**
TABLE OF CONTENTS ..... **xi**
ABBREVIATIONS ..... **xiii**
LIST OF TABLES ..... **xv**
LIST OF FIGURES ..... **xix**
SUMMARY ..... **xxi**
ÖZET ..... **xxiii**
1. INTRODUCTION **1**
1.1 Contribution 4
1.2 Outline of the Thesis 5
1.3 Academic Publications 5
2. BACKGROUND AND RELATED WORK **7**
2.1 Dynamic Environments 7
2.1.1 Dynamic optimization problems 9
2.1.1.1 The moving peaks benchmark 9
2.1.1.2 XOR generator 12
2.1.1.3 Dynamic traveling salesman problem 13
2.1.2 Performance evaluation criteria 15
2.2 Hyper-heuristics 17
2.2.1 Selection hyper-heuristics 19
2.2.1.1 Heuristic selection methods 19
2.2.1.2 Move acceptance methods 21
2.2.2 Related literature 22
2.2.3 HyFlex and first cross-domain heuristic search challenge 24
2.2.4 Selection hyper-heuristics in dynamic environments 25
3. SELECTION HYPER-HEURISTICS IN DYNAMIC ENVIRONMENTS .. **27**
3.1 Experimental Setting 28
3.1.1 Approaches used in comparisons 30
3.1.2 Parameter settings of hyper-heuristics 31
3.2 Results 32
3.2.1 Results for EXPSET1 32
3.2.2 Results for EXPSET2 34
3.2.3 Dynamic environment heuristic search challenge 34
3.2.4 Tracking ability of the approaches 37
3.2.5 Scalability results 40
3.3 Discussion 43

4. AN ANT-BASED SELECTION HYPER-HEURISTICS FOR DYNAMIC ENVIRONMENTS.. 45
4.1 Proposed Ant-Based Selection Hyper-heuristic Methods 45
4.1.1 An illustrative example.. 47
4.2 Performance Evaluation of Ant-based Hyper-heuristic................................. 49
4.2.1 Experimental design 49
4.2.2 Results and discussion... 51
4.3 Experiments using a Detection Mechanism 54
4.3.1 Experimental design 55
4.3.2 Results and discussion... 56
4.4 Analysis of the Components of *AbS*.. 57
4.4.1 The behavior of ant-based selection 57
4.4.2 Max-Min ant-based selection hyper-heuristic 62
4.4.3 Re-initialization of pheromone trails with max-min *AbS*...................... 63
4.4.4 The influence of q_0 63
4.4.5 The influence of slow decreasing parameter 65
4.4.6 The influence of evaporation rate 66
5. APPLICATIONS OF THE ANT-BASED SELECTION HYPER-HEURISTICS 69
5.1 Application I: Hyper-heuristics in A Hybrid Multi-population Framework .. 69
5.1.1 A hybrid framework for dynamic environments 70
5.1.2 Computational experiments.. 72
5.1.2.1 Experimental design 72
5.1.2.2 Results.. 75
Comparison of heuristic selection methods 75
Comparisons to selected approaches from literature.............................. 78
5.1.2.3 Discussion... 79
5.2 Application II: An Implementation on HyFlex 81
5.2.1 Experimental design 81
5.2.2 Results and discussion... 82
5.3 Application III: Dynamic Traveling Salesman Problem 83
5.3.1 Comparisons of selection hyper-heuristics.. 83
5.3.1.1 Results.. 87
5.3.2 Comparisons to problem specific approaches 91
5.3.2.1 Results.. 92
6. CONCLUSION AND FUTURE WORK... 97
REFERENCES..101
APPENDICES 111
APPENDIX : Results of Ant-Based Selection with Tournament Selection.........113
CURRICULUM VITAE 116

ABBREVIATIONS

AM	: All Moves
AbS	: Ant-based Selection
AbSrw	: Ant-based Selection with Roulette Wheel
AbSts	: Ant-based Selection with Tournament Selection
ACO	: Ant Colony Optimization
CF	: Choice Function
CMAES	: Covariance Matrix Adaptation Evolution Strategy
DTSP	: Dynamic Traveling Salesman Problem
DUF	: Decomposable Unitation-Based Function
EDA	: Estimation of Distribution Algorithm
EIACO	: Ant Colony Optimization with Elitism-based Immigrants
EMCQ	: Exponential Monte Carlo With Counter
ES	: Evolutionary Strategies
GD	: Great Deluge
GR	: Greedy
HH-EDA2	: Hyper-heuristic based dual population EDA
HM	: Hyper-mutation
IE	: Improving and Equal
MIACO	: Ant Colony Optimization with Memory-based Immigrants
MM AbS	: Max-Min Ant-based Selection with Roulette Wheel
MPB	: Moving Peaks Benchmark
MPBILr	: Memory-based PBIL with restart
MPBILr	: Dual Population Memory-based PBIL with restart
OI	: Only Improving
PBIL	: Population Based Incremental Learning
PBIL2	: A Dual Population PBIL
PBILr	: PBIL with restart
PBIL2r	: PBIL2 with restart
RIACO	: Ant Colony Optimization with Random Immigrants
RL	: Reinforcement Learning
RPD	: Random Permutation Descent
SA	: Simulated Annealing
SA+RH	: Simulated Annealing with Reheating
Sentinel8	: Sentinel-based Genetic Algorithm with 8 sentinels
Sentinel16	: Sentinel-based Genetic Algorithm with 16 sentinels
SR	: Simple Random

LIST OF TABLES

Page

Table 2.1 : Example peak coordinate, height and width values of a 2-dimensional landscape with two peaks. 10
Table 2.2 : The number of low-level heuristics for each heuristic type for each problem domain. 25
Table 3.1 : Parameter settings for the Moving Peaks Benchmark. 28
Table 3.2 : MPB parameter settings for each severity level. 29
Table 3.3 : The offline error generated by each approach during the EXPSET1 experiments for different combinations of change frequency and severity settings. 35
Table 3.4 : The offline error generated by each approach during the EXPSET2 experiments for different combinations of change frequency and severity settings. 36
Table 3.5 : The overall Formula 1 scores for the top fourteen approaches. 38
Table 3.6 : Offline error generated by each approach in the experiments for analyzing the effect of number of dimensions for EXPSET2 for different frequency and severity combinations. 41
Table 3.7 : Offline error generated by each approach in the experiments for analyzing the effect of number of peaks for EXPSET2 for different frequency and severity combinations. 42
Table 4.1 : Final offline error results of various q_0 settings for *AbSrw* and *sAbSrw* under the tested change frequency-severity pairs. 51
Table 4.2 : Final offline error results of various q_0 settings for *AbSts* and *sAbSts* using random tournament size under the tested change frequency-severity pairs. 52
Table 4.3 : Final offline error results for the proposed heuristic selection schemes. Here, for both *AbSrw* and *sAbSrw* $q_0 = 0.5$, for both *AbSts* and *sAbSts* $q_0 = 0.5$ with random tournament size settings. 53
Table 4.4 : Summary of statistical significance comparisons for *AbS variants*. 53
Table 4.5 : Final offline error results for the proposed heuristic selection schemes and RL, CF and ICF. 53
Table 4.6 : Pair-wise comparison of algorithms for each dynamic environment type determined by a given change frequency and severity. Given A vs B, $s+$ ($s-$) denote that A (B) is performing statistically better than B (A), while $\approx$ denotes that there is no statistically significant performance variation between A and B. 54
Table 4.7 : The offline errors generated by each approach for different combinations of change frequency and severity settings. 56
Table 4.8 : Summary of statistical significance comparisons between *AbS*-IE and *AbS*-IEd. 57

Table 4.9 : Summary of statistical significance comparisons between *AbS*, CF and RL combined with IEd. 57
Table 4.10 : The offline errors generated by each individual low-level heuristic for different combinations of change frequency and severity settings. 59
Table 4.11 : The offline errors generated by *AbS* with 7 and 4 low-level heuristics for different combinations of change frequency and severity settings. 59
Table 4.12 : Summary of statistical significance comparisons between *AbS* with 7 and 4 low-level heuristics. 62
Table 4.13 : The offline errors generated by *AbS* and *Max-Min AbS* for different combinations of change frequency and severity settings. 62
Table 4.14 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for *AbS* and *Max-Min AbS*. 62
Table 4.15 : The offline errors generated by *MM AbS* and *MM AbS-R* for different combinations of change frequency and severity settings. 63
Table 4.16 : Final offline error results of various q_0 settings for *MM AbS* under the tested change frequency-severity pairs. 64
Table 4.17 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts various q_0 settings for *MM AbS*. 64
Table 4.18 : Final offline error results of various *sd* settings for *MM AbS* under the tested change frequency-severity pairs. 65
Table 4.19 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts various *sd* settings for *MM AbS*. 66
Table 4.20 : Final offline error results of various ρ settings for *MM AbS* under the tested change frequency-severity pairs. 66
Table 5.1 : Parameter settings for PBILs. 72
Table 5.2 : The value of the change periods. 73
Table 5.3 : Offline errors generated by different heuristic selection methods averaged over 100 runs, on all DUFs for different change severity and frequency settings in randomly changing environments. 76
Table 5.4 : Offline errors generated by different approaches averaged over 100 runs, on the DUF1 for different cycle length and change frequency settings in different cyclic dynamic environments. 77
Table 5.5 : Offline errors generated by different approaches averaged over 100 runs, on the DUF2 for different cycle length and change frequency settings in different cyclic dynamic environments. 78
Table 5.6 : Offline errors generated by different approaches averaged over 100 runs, on the DUF3 for different cycle length and change frequency settings in different cyclic dynamic environments. 79
Table 5.7 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for the different heuristic selection schemes. 80
Table 5.8 : The overall score according to the Formula 1 ranking based on median, best and average offline error values for the different heuristic selection schemes. 80
Table 5.9 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for the algorithms used. 80
Table 5.10 : The overall score according to the Formula 1 ranking based on median, best and average offline error values for the algorithms used. 81
Table 5.11 : The overall Formula 1 scores of our approaches compared to competing hyper-heuristics in CHeSC2011 82

Table 5.12 : The overall Formula 1 scores of *AbS* variants for six problem domains. 83
Table 5.13 : Offline performance generated by different approaches averaged over 31 runs, on the kroA100 for random DTSP. 89
Table 5.14 : Offline performance generated by different approaches averaged over 31 runs, on the kroA150 for random DTSP. 89
Table 5.15 : Offline performance generated by different approaches averaged over 31 runs, on the kroA200 for random DTSP. 90
Table 5.16 : Offline performance generated by different approaches averaged over 31 runs, on the u2152 for random DTSP. 90
Table 5.17 : Parameter settings for ACO with immigrants. 92
Table 5.18 : Offline performance generated by different approaches averaged over 31 runs, on the kroA100 for random and cyclic DTSP. 94
Table 5.19 : Offline performance generated by different approaches averaged over 31 runs, on the kroA150 for random and cyclic DTSP. 95
Table 5.20 : Offline performance generated by different approaches averaged over 31 runs, on the kroA200 for random and cyclic DTSP. 96
Table A.1 : Final offline error results of various q_0 settings for *AbSts* under the tested change frequency-severity pairs. 113
Table A.2 : Final offline error results of various q_0 settings for *sAbSts* under the tested change frequency-severity pairs. 114

LIST OF FIGURES

Page

Figure 2.1 : A 2-dimensional fitness-landscape with two peaks is given in (a). The following changes are applied on this landscape: (b) the peaks are shifted, i.e. their locations are changed, but their heights and widths remain fixed, (c) the widths of the peaks are changed, but their locations and heights remain fixed, (d) the heights of the peaks are changed, but their locations and widths remain fixed, (e) the heights, widths and locations of the peaks are changed. 11

Figure 2.2 : The heights of all the peaks given for each stationary environment over 20 changes. 12

Figure 2.3 : Selection hyper-heuristic framework [1]. 18

Figure 3.1 : Average convergence plot generated by the best solution versus fitness evaluation counts for Simple Random and Improving and Equal. 29

Figure 3.2 : Histograms of Formula 1 scores for (a) CF–OI based on the median, (b) CF–IE based on the best, and (c) CF-OI based on average over 18 dynamic environment cases. 38

Figure 3.3 : Comparison of approaches (CF-IE, HM-AM, ES, and CMAES) for the combinations of (a) Low, (b) Medium, (c) High frequencies and severities of change based on the error values of the best candidate solution versus evaluation counts for EXPSET2. 39

Figure 3.4 : A sample plot of the error values of the best candidate solution versus evaluation counts based on medium change frequency and medium severity combination for EXPSET2. The left and right plots show the results for Choice Function–Improving and Equal and Hypermutation–All Moves, respectively. 39

Figure 3.5 : Box-plots of offline error values for a statistical comparison of the approaches (CF-IE, HM-AM, ES, and CMAES) for the combinations of (a) Low, (b) Medium, (c) High frequencies and severities of change using EXPSET2. 40

Figure 4.1 : A sample plot of the error values of the best candidate solutions versus the number of evaluations for the combinations of (a) Low, (b) Medium, (c) High frequencies of change for *AbS*. 58

Figure 4.2 : A sample semilogarithmic plot for the pheromone trail values versus fitness evaluations for each heuristic pair based on high frequency and medium severity combination for *AbS*. 60

Figure 4.3 : A sample semilogarithmic plot for the pheromone trail values versus fitness evaluations for each heuristic pair based on high frequency and medium severity combination for *AbS* with handling ties. 61

Figure 4.4 : Final offline error versus of different q_0 values for *MM AbS* for different combination of change frequency and severity settings....... 64
Figure 4.5 : Final offline error versus of different *sd* values for *MM AbS* for different combination of change frequency and severity settings....... 65
Figure 4.6 : Final offline error versus of different ρ values for *MM AbS* for different combination of change frequency and severity settings....... 67
Figure 5.1 : The framework of HH-EDA2... 71
Figure 5.2 : A sample plot of the fitness values of the best candidate solutions versus time for (a) AdapHH and (b) *AbS*.. 88

HYPER-HEURISTICS IN DYNAMIC ENVIRONMENTS

SUMMARY

Current state-of-the-art methodologies are mostly developed for stationary optimization problems. However, many real world problems are dynamic in nature. To handle the complexity of dealing with the changes in the environment, an optimization algorithm needs to be adaptive and hence capable of following the change dynamics. From the point of view of an optimization algorithm, the problem environment consists of the instance, the objectives and the constraints. The dynamism may arise due to a change in any of the components of the problem environment. Existing search methodologies have been modified suitably with respect to the change properties, in order to tackle dynamic environment problems. Population based approaches, such as evolutionary algorithms are frequently used for solving dynamic environment problem.

Hyper-heuristics are high-level methodologies that perform search over the space of heuristics rather than solutions for solving computationally difficult problems. They operate at a higher level, communicating with the problem domain through a domain barrier. Any type of problem specific information is filtered through the domain barrier. Due to this feature, a hyper-heuristic can be directly employed in various problem domains without requiring any change, of course, through the use of appropriate domain specific low-level heuristics.

Selection hyper-heuristics are highly adaptive search methodologies that aim to raise the level of generality by providing solutions to a diverse set of problems having different characteristics. In this thesis, we investigate single point search based selection hyper-heuristics in dynamic environments. We first work on the applicability of selection hyper-heuristics proposed in literature for dynamic environments. Then, we propose a novel learning hyper-heuristic for dynamic environments and investigate the performance of the proposed hyper-heuristic and its variants.

In the first phase, the performances of thirty-five single point search based selection hyper-heuristics are investigated on continuous dynamic environments exhibiting various change dynamics, produced by the Moving Peaks Benchmark generator. Even though there are many successful applications of selection hyper-heuristics to discrete optimization problems, to the best of our knowledge, this study is one of the initial applications of selection hyper-heuristics to real-valued optimization as well as being among the very few which address dynamic optimization issues using these techniques. The empirical results indicate that learning selection hyper-heuristics which incorporate compatible components can react to different types of changes in the environment and are capable of tracking them. This study shows the suitability of selection hyper-heuristics as solvers in dynamic environments.

In the second phase, we propose a new learning hyper-heuristic, called the *Ant-based Selection* (*AbS*), for dynamic environments which is inspired from the ant colony optimization algorithm components. The proposed hyper-heuristic maintains a matrix of pheromone intensities (utility values) between all pairs of low-level heuristics. A heuristic is selected based on the utility values between the previously invoked heuristic and each heuristic from the set of low-level heuristics. For this study, we employ the generic Improving and Equal acceptance scheme. We explore the performance of the proposed hyper-heuristic and its variants using Moving Peaks Benchmark (MPB) generator. The empirical results indicate that the proposed heuristic selection scheme provides slightly better performance than the heuristic selection scheme that was previously reported to be the best in dynamic environments.

The proposed approach does not require any special actions whenever a change occurs in the environment. However, the first candidate solution generated after each change is accepted regardless of its quality. Therefore, the move acceptance needs to detect the change. In this study, we use a simple detection mechanism in which the current solution is re-evaluated at each step. If there is a change in the fitness of the current solution, a change is considered to be detected. We consider *Ant-based selection*, Choice Function and Reinforcement Learning as the heuristic selection methods. The results show that the re-evaluation process slightly deteriorates the performance of approaches for especially high frequency changes, however, the approach is suitable for cases where changes cannot be made known to the optimization algorithm. We then investigate the effect of the parameters of the proposed algorithm on overall performance. The results show that the settings of the parameters are not very sensitive and similar results are obtained for a wide range of parameter values.

In the third phase, we explore the performance of the proposed hyper-heuristic through three different applications. As the first application, the selection hyper-heuristics are used in a hybrid multi-population framework. We use a hybridization of the Estimation of Distribution Algorithm (EDA) with hyper-heuristics in the form of a two-phase framework. We investigate the influence of different heuristic selection methods. The empirical results show that a heuristic selection method that relies on a fixed permutation of the underlying low-level heuristics is more successful than the learning approaches across different dynamic environments produced by a well-known benchmark generator. The proposed approach also outperforms some of the top approaches in literature for dynamic environment problems. Ant-based selection is proposed for dynamic environments. However, to see its performance in a stationary environment, Ant-based Selection is applied to six stationary optimization problems provided in HyFlex as the second application. The results are compared with the results of participants in CHeSC2011 competition. Finally, we present the performance of Ant-based Selection on a real-world optimization problem referred to as the Dynamic Traveling Salesman Problem. The overall results show that the proposed approach delivers good performance on the tested optimization problems. These last set of experiments also emphasize the general nature of hyper-heuristics. For all optimization problems in this study, all hyper-heuristics are applied without requiring any modifications or parameter tuning.

DİNAMİK ORTAMLARDA ÜST-SEZGİSELLER

ÖZET

Son zamanlarda önerilen metotlar daha çok statik eniyileme problemleri için geliştirilmişlerdir. Fakat gerçek hayatta karşılaşılan eniyileme problemlerinin pek çoğu dinamik bir yapı göstermektedir. Dinamik bir ortamda, eniyileme yönteminin üzerinde çalışmaya başladığı ortamda zaman içinde değişimler olabilir. Ancak bu problemlerin çözümünde genelde bu dinamiklik göz ardı edilerek klasik eniyileme yaklaşımları uygulanmaktadır. Halbuki bu dinamikliği de göz önüne alarak çalışan bir eniyileme yaklaşımı, ortamdaki değişimleri hızlı bir şekilde izleyebilmeli ve bunlara uyum sağlayabilmek için adaptif olmalıdır. Eniyileme algoritması açısından bakıldığında problem ortamı, problemin tanımlı değerleri, eniyilemede kullanılan amaç fonksiyonları ve kısıtlardan oluşur. Ortamdaki dinamiklik, problem ortamını oluşturan bu parçalardan herhangi birisinde veya birkaçında meydana gelen tekil ya da eş zamanlı değişimlerden kaynaklanabilir. Farklı problemlerde bu değişimler de farklı özellikler göstermektedir. Bu özellikler genelde değişimlerin şiddetine, sıklığına, periyodik olup olmamasına göre sınıflandırılırlar. Ortamdaki dinamizmin özelliklerine göre farklı durumlarda farklı yaklaşımlar başarılı olmaktadır. Bu ise eniyileme yaklaşımını seçerken ortamdaki değişimlerin özelliklerinin bilinmesi anlamına gelir. Halbuki gerçek hayatta bu her zaman mümkün olmayabilir. Ayrıca ortamın gösterdiği değişimin özellikleri de zaman içinde değişebilir. Bu durumda başta seçilen yaklaşım, eniyilemenin ilerleyen aşamalarında başarılı olmayabilir.

Üst-sezgiseller problem uzayında problem ile etkileşim halinde olan ve aday çözümü güncelleyen alt seviyedeki sezgiseller aracılığı ile arama yapar. Alt seviyede kullanılan, probleme özel sezgiseller ise problemin çözüm uzayında arama yaparlar. Bu nedenle alt seviyedeki sezgiseller, üst-sezgiseller ile problemin çözüm uzayı arasında bir ara katman olarak düşünülebilir. Böylece problem uzayında aramayı alt sezgiseller yapmış olur. Bu özellik sayesinde bir üst-sezgisel, uygun alt sezgisellerin kullanılmasıyla, değiştirilmeden çeşitli problemlere uygulanabilir.

Sezgisel seçen üst-sezgiseller konusunda yapılan araştırmaların temel hedefi, eniyilemenin genelleştirme seviyesini yükselterek pek çok farklı problem domeninde ve farklı özellikler gösteren ortamlarda uygulanabilir bir yaklaşım geliştirmektir. Bu nedenle üst-sezgiseller, doğaları gereği adaptif yapıdadırlar. Bu özellikleri sayesinde dinamik ortamlardaki değişimlere, herhangi bir dış müdahale gerektirmeden hızla uyum gösterip, etkin çözümler üretebilirler. Bu tezde öncelikle literatürde var olan üst-sezgisellerin dinamik ortamlar için uygunluğu üzerinde çalışılmıştır. Elde edilen bilgiler ışığında dinamik ortamlarda başarılı çözümler üretecek yeni üst-sezgisel yaklaşım geliştirilmiş ve başarımı ölçülmüştür.

Tezin ilk aşamasında, otuz beş tek çözüm üreten sezgisel seçen üst-sezgisellerin başarımını, farklı değişim dinamikleri sergileyen sürekli dinamik eniyileme problemleri için değerlendirdik. Deneylerde üzerinde çalışmak için yapay oluşturulmuş test problemi (Moving Peaks Benchmark) kullanılmıştır. Ayrık eniyileme problemleri için sezgisel seçen üst-sezgisellerin birçok başarılı uygulamaları olmasına rağmen, bilgimiz dahilinde, bu çalışma reel değerli (sürekli) eniyileme problemleri için sezgisel seçen üst-sezgisellerin ilk uygulamalarından biridir. Bunun yanı sıra bu çalışma, bu teknikleri kullanarak dinamik eniyileme problemlerini ele alan çok az çalışma arasında yer almaktadır. Deneysel sonuçlar göstermiştir ki; uygun bileşenli öğrenme tabanlı üst-sezgiseller ortamdaki farklı tipteki değişimlere hızlı bir şekilde tepki gösterebilmekte ve onları takip edebilmektedir. Bu çalışma üst-sezgisellerin dinamik eniyileme problemlerini çözmek için uygun olduğunu göstermektedir.

İkinci aşamada, karınca kolonisi algoritmasından esinlenerek yeni öğrenme tabanlı üst-sezgisel yaklaşım, karınca tabanlı seçim, geliştirilmiştir. Önerilen üst-sezgisel düşük seviyeli bütün sezgisel çiftleri arasındaki feromon yoğunluklarının bir matrisini tutar. Her adımda bir sezgisel, önceden çağırılan sezgisel ile düşük seviyeli sezgisel kümesinden her bir eleman arasındaki feromon değerlerine göre seçilir. Bu çalışmada iyileştiren ve eşit hareket kabul yöntemi kullanılmıştır. Önerdiğimiz üst-sezgisel yönteminin başarımı yapay oluşturulmuş test problemi (Moving Peaks Benchmark) kullanılarak değerlendirilmiştir. Test sonuçlarına göre, önerilen yaklaşım daha önceden dinamik ortamlar için en iyi olarak belirlenen sezgisel seçme yöntemleri ile benzer sonuçlar vermiştir.

Önerilen yaklaşım ortam değiştiğinde herhangi bir özel eyleme gerek duymamaktadır. Fakat hareket kabul yönteminin doğası gereği, her bir değişimden sonra üretilen ilk çözüm adayı niteliğine bakılmaksızın kabul edilmektedir. Bundan dolayı hareket kabul yöntemi ortamdaki değişikliği algılamak zorundadır. Bu çalışmada ortamdaki değişimleri algılamak için basit bir yöntem kullanılmıştır. Bu yöntemde şu anki çözümün başarım değeri her adımda tekrardan hesaplanmaktadır. Eğer şu anki çözümün başarım değerinde bir değişiklik varsa ortam değişmiş demektir. Sezgisel seçme yöntemi olarak seçin fonksiyonu, destekli öğrenme ve karınca tabanlı seçim kullanılmıştır. Test sonuçlarına göre yeniden değerlendirme yöntemi bütün yaklaşımların başarımını azaltmıştır.

Bu çalışmada ayrıca önerilen yaklaşımın kapsamlı bir analizi yapılmıştır. Bu amaçla önerilen yaklaşımın adaptasyon yeteneği ve algoritmaların parametrelerinin başarıma etkisi incelenmiştir. Deneysel sonuçlara göre, önerilen yaklaşım hızlı bir şekilde değişimlere uyum sağlayabilmektedir. Önerilen yaklaşım parametre atamalarından çok fazla etkilenmemekte ve geniş aralıklı parametre değerleri için benzer sonuçlar vermektedir.

Tezin son aşamasında, önerilen yaklaşımın başarımı üç farklı uygulamada değerlendirilmiştir. Öncelikle, sezgisel seçen üst-sezgiseller çok popülasyonlu hibrid bir çerçeve içinde kullanılmışlardır. Bu çerçeve çevrimiçi ve çevrimdışı öğrenme mekanizmalarına dayanan üst-sezgiseller ile dağılım tahmini algoritmasının hibridleştirilmesine olanak sağlamaktadır. İyi çözümler üretmek için olasılık vektörlerinin listesi ilk aşamada çevrimdışı olarak öğrenilir. İkinci aşamada iki ayrı popülasyon ve her popülasyonun kendi olasılık vektörleri vardır. Bir alt popülasyon dağılım tahmini algoritması kullanarak örneklendirilirken, diğer alt popülasyon çevrimiçi olarak uygun olasılık vektörünü çevrimiçi aşamada öğrenilen olasılık vektörleri listesinden örneklemek için üst-sezgiselleri kullanır. Önerilen hidrid yöntemin başarımı farklı sezgisel seçme yöntemleri kullanılarak denenmiştir ve Rastgele Permütasyon metodunun daha başarılı olduğu gözlemlenmiştir. Ayrıca bu

hibrid yapı literatürde iyi bilinen benzer yaklaşımlarla karşılaştırılmış ve bunlara göre daha iyi sonuç verdiği gözlemlenmiştir.

Önerilen yöntem dinamik ortamlar için önerilmiştir. Bununla birlikte, yöntemin statik ortamlardaki başarımını gözlemlemek için, ikinci uygulama olarak, önerilen metot HyFlex arayüzü üzerinde uygulanmıştır. HyFlex'in Java uygulaması CHeSC2011 yarışmasında kullanılmıştır. Bu uygulama altı statik problem domeni sağlamaktadır. Önerilen yaklaşımın başarımı yarışmadaki katılımcılarla karşılaştırılmıştır. Son uygulama olarak önerilen yaklaşımın başarısı gerçek dünya problemi kullanılarak değerlendirilmiştir. Yapay oluşturulmuş test problemleri problem örneklerini yaratmak için kullanılan önemli araştırma araçları olup verilen domende bu örneklerin özelliklerini kontrol etmemizi sağlar. Bu problem örnekleri farklı algoritmaların başarımını karşılaştırmak için çoğunlukla kullanılmaktadırlar. Öte yandan, gerçek dünya problemleri yapay olarak oluşturulan örneklerden farklı olabilir. Yapay örnekleri kullanarak yapılan algoritmaların test edilmesi verilen algoritmanın gerçek dünya problemi üzerindeki asıl performansını yansıtmayabilir. Dolayısıyla, bu çalışmada, Dinamik Gezgin Satıcı Problemi olarak bilinen gerçek dünya problemi ele alınmış ve önerilen yaklaşımın başarımı değerlendirilmiştir. Dinamik Gezgin Satıcı Problemi örneklerini oluşturmak için literatürde çokça kullanılan Gezgin Satıcı Problemi' nin örneklerine trafik faktörü eklenmiştir. Genel olarak, test edilen problemler üzerinde önerilen metodun iyi sonuç verdiği gözlemlenmiştir. En son yapılan testler üst-sezgisellerin genel bir yapı olduğunu vurgulamıştır. Üst-sezgiseller hiçbir değişikliğe ya da parametre ayarlarına gerek duymadan bu çalışmada kullanılan tüm eniyileme problemlerine uygulanmıştır.

1. INTRODUCTION

A *hyper-heuristic* is a methodology which explores the space of heuristics for solving complex computational problems [1–4]. Although the term *hyper-heuristic* was introduced recently [5, 6], the initial ideas can be traced back to the 60s [7, 8]. There has been a growing interest in this field since then. A hyper-heuristic is an alternative method to meta-heuristics, which operate on the problem directly by using problem-specific information. For meta-heuristic methods, parameters must be fine-tuned for different problems. On the other hand, hyper-heuristics can operate on a problem indirectly by way of heuristics, which interact with the problem and modify the solutions [9]. There are two main types of hyper-heuristics in literature [10]: methodologies that *select*, or *generate* heuristics. This study focuses on the former type of hyper-heuristics based on a single point search framework termed as a selection hyper-heuristic. A selection hyper-heuristic controls a set of low-level heuristics and adaptively chooses the most appropriate one to invoke at each step. This type of hyper-heuristics has been successfully applied to many combinatorial optimization problems ranging from timetabling to vehicle routing [11].

One of the challenges in combinatorial optimization is to develop a solution method which is capable of solving different types of instances having different characteristics for a given problem domain. There is a variety of heuristic search methodologies, such as tabu search and evolutionary algorithms to choose from to solve static combinatorial optimization problems [12]. If the environment changes over time during the optimization/search process for a given problem, then this task becomes even more challenging. Such problems are referred to as *dynamic optimization problems*. When performing a search for the best solution in such environments, the dynamism is often ignored and generic methodologies are utilized. However, the key to success for a search algorithm in dynamic environments is its adaptation ability and speed to react whenever a change occurs. There is a range of approaches in literature proposed for solving dynamic environment problems [13–15]. Often, a given approach performs better than some others for handling a particular type of dynamism in the environment. This implies that the properties of the dynamism need to be known beforehand, if the most appropriate approach is to be chosen. However, even this may be impossible depending on the relevant dynamism associated with the problem.

A key goal in hyper-heuristic research is raising the level of generality. To this end, approaches which generalize well and are applicable across a wide range of problem domains or different problems with different characteristics, have been investigated. Considering the adaptive nature of hyper-heuristics, they are expected to respond to the changes in a dynamic environment rapidly and hence be effective solvers in such environments regardless of the change properties. In this thesis, we study the applicability of selection hyper-heuristics in dynamic environments.

In the first phase of this thesis, we investigate the performance of a set of selection hyper-heuristics proposed in literature for dynamic environments to determine their strengths and weaknesses and to analyze their behavior. In this study, selection hyper-heuristics are applied to a set of real-valued optimization problems generated using the Moving Peaks Benchmark (MPB) generator [16]. This benchmark generator is preferred as a testbed for our investigations mainly because it is one of the most commonly used benchmark generators in literature for creating dynamic optimization environments in the continuous domain [14]. Based on the empirical results, the learning selection hyper-heuristics with appropriate acceptance methods are applicable approaches to solve dynamic optimization problems. They can react rapidly whenever a change occurs and are capable of tracking the changing optima closely.

In the second phase of this thesis, we describe a new learning hyper-heuristic for dynamic environments, which is designed based on the ant colony optimization algorithm components. The proposed hyper-heuristic maintains a matrix of pheromone intensities (utility values) between all pairs of low-level heuristics. A heuristic is selected based on the utility values between the previously invoked heuristic and each heuristic from the set of low-level heuristics. We investigate the performance of the proposed hyper-heuristic controlling a set of parameterized mutation operators for solving the dynamic environment problems produced by the Moving Peaks Benchmark (MPB) generator. The empirical results show that the proposed heuristic selection scheme provides slightly better performance than the heuristic selection scheme previously reported to be the best in dynamic environments. The Ant-based selection hyper-heuristic does not require any special actions when the environment changes. However, due to the nature of the acceptance mechanism, the first solution candidate generated after each environment change is accepted regardless of its solution quality. This means that the algorithm needs to know when a change occurs in the environment. In this thesis, we consider a simple change detection mechanism. To detect a change in the environment, the current solution is re-evaluated at each step. A change occurs in the environments when the fitness value of the current solution is changed. The empirical results show that the re-evaluation slightly degrades the performance of the algorithms. However, the approach is suitable for cases where changes cannot be made known to the optimization algorithm. We further perform exhaustive tests to empirically analyze and explain the behavior of our approach. The results indicate that the parameter settings of the proposed approach are not very sensitive and similar results are obtained for a wide range of parameter values.

In the final phase of this thesis, we present three applications of the proposed hyper-heuristic, namely Ant-based selection hyper-heuristic. Firstly, the hyper-heuristics are used in a hybrid multi-population framework. The framework hybridizes selection hyper-heuristic and Estimation of Distribution Algorithm (EDA) combining offline and online learning mechanisms. A list of probability vectors for generating good solutions is learned in an offline manner in the first phase. In the second phase, *two* sub-populations are maintained. A sub-population is sampled using an Estimation of Distribution Algorithm, while the other one uses a hyper-heuristic for sampling appropriate probability vectors from the previously learned list in an online manner. The empirical results show that the proposed approach using a particular hyper-heuristic outperforms some of the best known approaches in literature on the dynamic environment problems dealt with. Even though Ant-based selection is proposed for dynamic environments, to assess its performance in a stationary environment, we implement the proposed approach on HyFlex [17] which provides six stationary optimization problems. We then compare the performance of proposed method with that of competitors in Cross-domain Heuristic Search Challenge (CHeSC2011). Finally, we investigate the performance of the proposed approach, on a real-world dynamic optimization problem referred to as the Dynamic Traveling

Salesman Problem (DTSP). The instances for the Dynamic Traveling Salesman Problem are generated from the classic Traveling Salesman Problem instances by introducing the traffic factor proposed in [18]. We compare the experimental results with those obtained from well-known approaches in literature. Overall, the proposed methods provide good performance on the tested problems.

1.1 Contribution

The contributions of this work can be stated as follows:

As the first contribution of this thesis, this study is the first study investigating single point search based hyper-heuristics in dynamic environments. In dynamic environments, different approaches are proposed to deal with different change properties. However, hyper-heuristics do not depend on the change dynamics and therefore hyper-heuristics can be directly employed in various dynamic optimization problems without requiring any modifications.

As the second contribution of this thesis, this study provides a complete empirical analysis of different hyper-heuristics coupling well-known heuristic selection and move acceptance methods in dynamic environments. There is no such previous study investigating a single point based search hyper-heuristic framework for solving dynamic environment problems. Moreover, to the best of our knowledge, this is one of the first studies which investigates the application of hyper-heuristics to a real-valued optimization as well as being among the very few which address dynamic optimization issues with these techniques. This study shows that learning selection hyper-heuristics are sufficiently general. This yields them to be viable approaches in solving not only dynamic problems regardless of the change dynamics in the environment, but also continuous optimization problems.

The third contribution is the Ant-based selection hyper-heuristics. We propose a new learning heuristic selection scheme for selection hyper-heuristics, especially for use in dynamic environments. Although the existing hyper-heuristics are appropriate for solving dynamic environment problems, they have some weaknesses. In these methods, it is assumed that they are aware of the time when the environment changes and they act on this. However, the proposed heuristic selection approaches do not require any special actions when the environment changes. The experimental results show that the proposed methods perform well on the tested problems.

1.2 Outline of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 provides background information on hyper-heuristics and dynamic environments as well as a related literature survey on the topic. Chapter 3 presents the empirical analysis of a set of hyper-heuristics in dynamic environments exhibiting different change characteristics. Chapter 4 describes the proposed Ant-based hyper-heuristics for dynamic environment. The empirical analysis of the proposed methods are also provided in Chapter 4. Then, Chapter 5 presents three applications of the proposed hyper-heuristics as well as the experimental study for each application. Finally, Chapter 6 discusses the conclusion and future work.

1.3 Academic Publications

The list of the publications produced during the PhD research are the following:

Journal publications

- Gönül Uludağ, Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan, "A Hybrid Multi-population Framework for Dynamic Environments Combining Online and Offline Learning", Soft Computing, Volume 17, Issue 12, pp. 2327-2348, 2013.

- Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan, "Selection Hyper-heuristics in Dynamic Environments", Journal of the Operational Research Society, 64 (12), pp. 1753-1769, DOI: 10.1057/jors.2013.24, 2013.

International conference publications

- Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan,"An Ant-based Selection Hyper-heuristic for Dynamic Environments", EvoApplications 2013, LNCS vol. 7835, pp. 626-635, Springer, 2013.

- Gönül Uludağ, Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan, "Heuristic Selection in a Multi-phase Hybrid Approach for Dynamic Environments", 12th Annual Workshop on Computational Intelligence (UKCI 2012), pp. 1-8, 2012.

- Gönül Uludağ, Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan, "A Framework to Hybridise PBIL and a Hyper-heuristic for Dynamic Environments", PPSN 2012: 12th International Conference on Parallel Problem Solving from Nature, LNCS vol. 7492, pp. 358-367, Springer, 2012.

- Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan, "An Investigation of Selection Hyper-heuristics in Dynamic Environments", EvoApplications 2011, Part I, LNCS vol. 6624, pp. 314-323, Springer, 2011

2. BACKGROUND AND RELATED WORK

2.1 Dynamic Environments

A dynamic environment is made up of components, such as, the problem instance, the objectives and the constraints, each of which may change in time individually or simultaneously. A change in a component can be categorized based on its characteristics as given in [13]: (i) *Frequency of change* defines how often the environment changes. (ii) *Severity of change* defines the magnitude of the change in the environment. (iii) *Predictability of change* is a measure of correlation between changes. (iv) *Cycle length/cycle accuracy* is a property that defines whether the optima return exactly to previous locations or close to them.

When designing an optimization algorithm for dynamic environments, one of the main issues for the algorithm to deal with is tracking the moving optima as closely as possible after a change occurs. Another one is being able to react to a change in the environment quickly and adapting to the new environment as fast as possible. Several strategies have been proposed to be used as a part of existing search methodologies for dynamic environments depending on the change properties. These strategies can be grouped into four main categories [19]: (i) maintain diversity at all times, (ii) increase diversity after a change, (iii) use memory, (iv) work with multiple populations.

For the approaches which maintain diversity at all times, e.g., as in the *random immigrants* approach [20], achieving and preserving the right level of diversity is crucial. In this method, a subset of population are replaced randomly generated solutions at each step during the search. A high level of diversity which is more than needed for a given problem may be detrimental to the search process during the stationary periods. These approaches are generally more successful in environments where the changes are severe and the change frequency is relatively high.

Approaches, such as *hypermutation* [21] and *variable local search* [22] increase diversity by increasing the mutation rate when the environment changes. It has been observed that too much diversity disrupts the search process, while too little may not be sufficient to prevent premature convergence. These approaches are more suitable for environments where changes are not too severe.

Some approaches make use of memory, as in [23–26], where the evolutionary algorithm remembers solutions which have been successful in the previous environments. These approaches are particularly more useful if a change occurs periodically and a previous environment is re-encountered during the search process at a later stage.

There are also other approaches with a good performance in dynamic environments, which make use of multiple populations, such as [13, 27]. In these approaches, the population is divided into subpopulations, where each subpopulation explores a different part of the search space. Often, the focus of such an algorithm is tracking several optima simultaneously in different regions of the search space.

The sentinel-based genetic algorithm (GA) [28] is another multi-population approach to dynamic environments which makes use of solutions referred to as *sentinels*, uniformly distributed over the search space for maintaining diversity. Sentinels are fixed at the beginning of the search and in general, are not mutated or replaced during the search. Sentinels can be selected for mating and used during crossover. Due to having the sentinels distributed uniformly over the search space, the algorithm can recover quickly when the environment changes and the optimum move to another location in the search space. Sentinels are reported to be effective in detecting and following the changes in the environment.

There is a growing interest in Statistical model-based optimization algorithms which are adaptive and, thus, have the potential to react quickly to changes in the environment and track them. For example, Estimation of Distribution Algorithms (EDAs), such as, Univariate marginal distribution algorithm [29], Bayesian optimization algorithm [30], and Population Based Incremental Learning (PBIL) [31], are among the most common Statistical model-based optimization algorithms used in dynamic environments. There are also some studies based on Statistical model-based optimization algorithms for dynamic environments to estimate both time and direction (pattern) of changes [32–35].

The standard PBIL algorithm is first introduced by [36]. PBIL builds a probability distribution model based on a *probability vector*, $\overrightarrow{P}$ using a selected set of promising solutions to estimate a new set of candidate solutions. Learning and sampling are the key steps in PBIL. Several PBIL variants are presented in literature for dynamic environment. One of them is a dual population PBIL (PBIL2) introduced in [31]. In PBIL2, the population is divided into two sub-populations. Each sub-population has its own probability vector. Both vectors are maintained in parallel.

2.1.1 Dynamic optimization problems

There are different benchmark generators in literature for dynamic environments. The Moving Peaks Benchmark generator [16] is commonly used in continuous domains, while in discrete domains the XOR dynamic problem generator [37, 38] is preferred. In the case of permutation-encoded problems, such as Traveling Salesman Problems (TSP) and Vehicle Routing Problems (VRP), different dynamic versions [18, 39–41] and benchmark generators [41, 42] are proposed in literature.

2.1.1.1 The moving peaks benchmark

The *Moving Peaks Benchmark* (MPB) generator introduced by Branke [16], is used in this study for analyzing and comparing the performance of different approaches. MPB is a dynamic benchmark function generator which is not as simplified as most of the toy problems in literature. Moreover, MPB exhibits similar properties to real world problems, e.g. through the application of the measures proposed in [43], it has been shown in [44] that the change dynamics generated by the MPB show a similar behavior to those observed in a dynamic multi-dimensional knapsack problem.

The MPB generator provides multidimensional and multi-modal landscapes with a variety of different peak shapes. In MPB, the most commonly used peak shape is the cone. The height, width and the location of each peak is altered whenever a change in the environment occurs. A dynamic benchmark function generated using MPB with cone shaped peaks is formulated as follows:

$$F(\vec{x},t) = \max_{i=1..m} \{H_i(t) - W_i(t) * \sqrt{\sum_{j=1}^{d} (x_j - X_{ij}(t))^2}\} \tag{2.1}$$

where m is the number of peaks, d is the number of dimensions, X_{ij} are the coordinates of the peaks in each dimension, H_i and W_i are the heights and widths of the peaks respectively. For example, assume that the current peak coordinates, height and width values of two peaks in a 2-dimensional landscape at the given time t_c are as given in Table 2.1. The function value of a real-valued vector (candidate solution) located at $\vec{x} = (x_1, x_2) = (10.0, 3.0)$ is calculated as follows:

$$\begin{aligned} F((10.0,3.0),t_c) &= \max\{50.0 - 0.1 * \sqrt{((10.0-2.0)^2 + (3.0-2.0)^2)}, \\ &\quad 70.0 - 0.5 * \sqrt{((10.0-20.0)^2 + (3.0-20.0)^2)}\} \\ F((10.0,3.0),t_c) &= \max\{49.19, 60.14\} \\ F((10.0,3.0),t_c) &= 60.14 \end{aligned}$$

Table 2.1 : Example peak coordinate, height and width values of a 2-dimensional landscape with two peaks.

Peak i	$X_{i1}(t_c)$	$X_{i2}(t_c)$	$W_i(t_c)$	$H_i(t_c)$
1	2.0	2.0	0.1	50.0
2	20.0	20.0	0.5	70.0

In some applications, a time-invariant base function $B(\vec{x})$ is used as part of the benchmark function. In this case, the new MPB function, denoted as $G(\vec{x},t)$ becomes $G(\vec{x},t) = \max\{B(\vec{x}), F(\vec{x},t)\}$.

When working with the MPB, firstly, the coordinates, heights and widths of the peaks are initialized. Then, every Δe iterations, the heights and the widths of the peaks are changed by adding a normally distributed random variable, while the location of the peaks are also shifted by a vector $\vec{v}$ of fixed length *vlength* in a random direction. During the search, the height, width and location of each peak are changed according to the following equations:

$$\rho \in N(\mu, \sigma^2) \quad \textbf{(2.2)}$$

$$H_i(t) = H_i(t-1) + height\_severity \cdot \rho \quad \textbf{(2.3)}$$

$$W_i(t) = W_i(t-1) + width\_severity \cdot \rho \quad \textbf{(2.4)}$$

$$\vec{X}_i(t) = \vec{X}_i(t-1) + \vec{v}_i(t) \quad \textbf{(2.5)}$$

where ρ is a random value drawn from a Gaussian distribution $N(\mu, \sigma^2)$, where μ and σ^2 denote its mean and variance set to 0 and 1, respectively and $v_i(t)$ is the shift vector which is the linear combination of the previous shift vector $v_i(t-1)$ and a random vector $\vec{r}$ normalized to *vlength*. The *height_severity*, the *width_severity* and *vlength* parameters determine the severity of the change in the heights, widths and locations of the peaks respectively. Δe determines the frequency of changes in the environment. The shift vector at time t is calculated as:

$$\vec{v}_i(t) \quad = \quad \frac{vlength}{|\vec{r}+\vec{v}_i(t-1)|}((1-\phi)\vec{r}+\phi\vec{v}_i(t-1)) \qquad \textbf{(2.6)}$$

where the random vector $\vec{r}$ is created by drawing uniformly distributed random numbers for each dimension and normalizing its length to *vlength*, and ϕ is the *correlation coefficient*. Higher values of ϕ indicate a higher correlation between the current and previous shift vectors.

Figure 2.1 gives an example of an initial fitness landscape on which various types of changes are applied. The fitness landscapes in the figure are generated using MPB with a basis function of $B(\vec{x}) = 0$. Figure 2.1(a) shows the initial 2-dimensional fitness landscape with 2 peaks ($m = 2$). Each of the rest of the sub-figures shows a specific type of change applied on this initial fitness landscape.

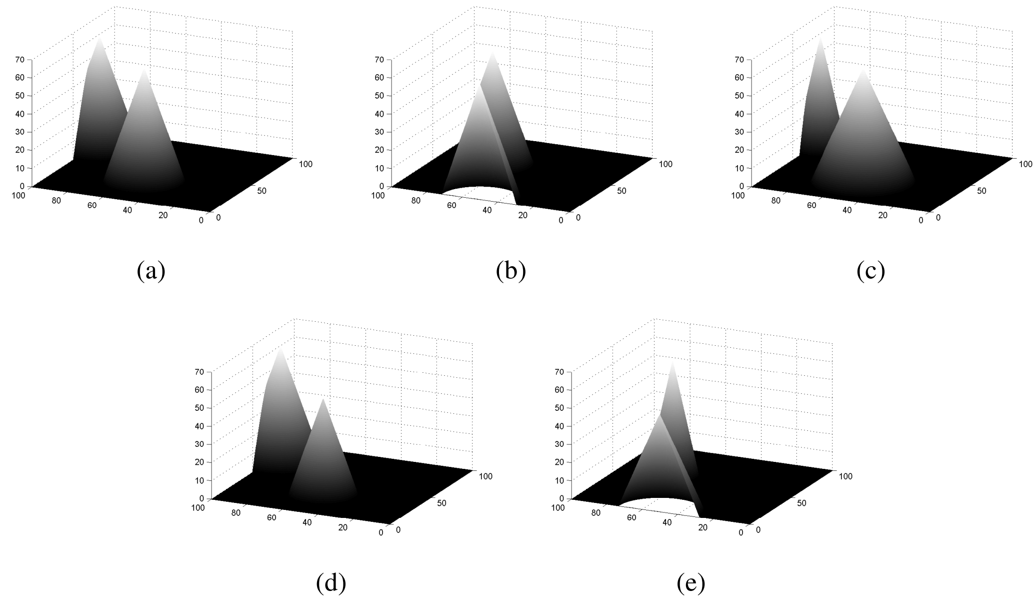


Figure 2.1 : A 2-dimensional fitness-landscape with two peaks is given in (a). The following changes are applied on this landscape: (b) the peaks are shifted, i.e. their locations are changed, but their heights and widths remain fixed, (c) the widths of the peaks are changed, but their locations and heights remain fixed, (d) the heights of the peaks are changed, but their locations and widths remain fixed, (e) the heights, widths and locations of the peaks are changed.

An initial landscape with five peaks is generated to demonstrate the effect of the changes on the landscape further. 20 consecutive changes are applied to this initial landscape. For simplicity, only the heights of the peaks are modified as a change, but their locations and widths are fixed. Figure 2.2 gives the height of each peak including the optimum after each change.

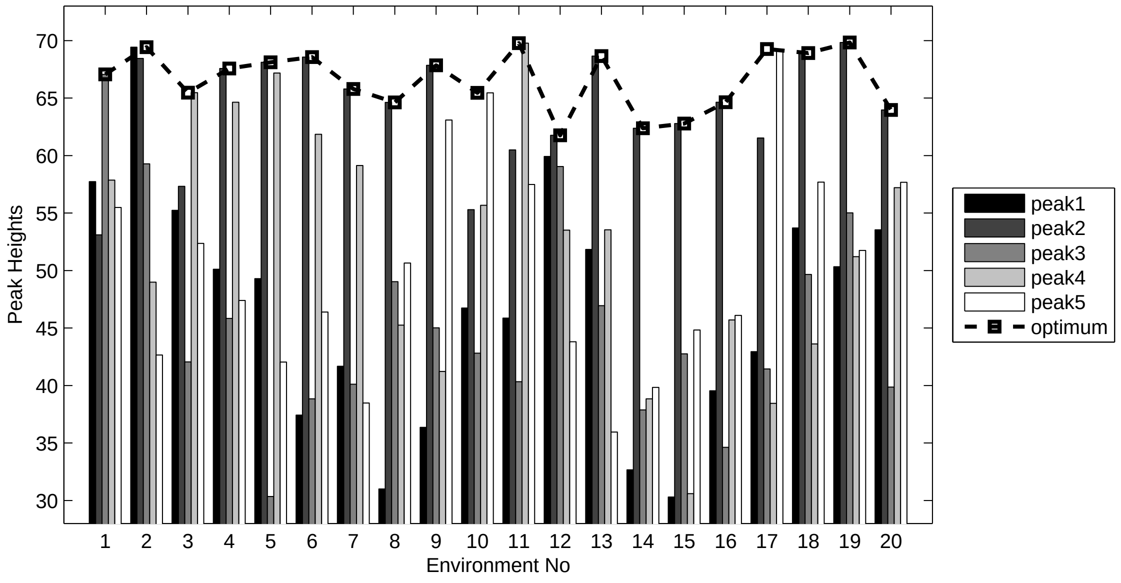


Figure 2.2 : The heights of all the peaks given for each stationary environment over 20 changes.

2.1.1.2 XOR generator

XOR dynamic problem generator [37, 38] creates dynamic environment problems with various degrees of difficulty from any binary-encoded stationary problem using a bitwise exclusive-or (XOR) operator. Given a function $f(\vec{x})$ in a stationary environment and $\vec{x} \in \{0,1\}^l$, the fitness value of the $\vec{x}$ at a given generation g is calculated as the following:

$$f(\vec{x},g) = f(\vec{x} \oplus m_k) \tag{2.7}$$

where m_k is a binary mask for k^{th} stationary environment and $\oplus$ is the XOR operator. Firstly, the mask m is initialized with a zero vector. Then, every τ generations, the mask m_k is changed as

$$m_k = m_{k-1} \oplus t_k \tag{2.8}$$

where t_k is a binary template.

In literature, Decomposable Unitation-Based Functions (DUFs) [25] are used within the XOR generator. All Decomposable Unitation-Based Functions are composed of 25 copies of 4-bit building blocks. Each building block is denoted as a unitation-based function $u(x)$ which gives the number of ones in the corresponding building block. Its maximum value is 4. The fitness of a bit string is calculated as the sum of the $u(x)$ values of the building blocks. The optimum fitness value for all Decomposable Unitation-Based Functions is 100. DUF1 is the *OneMax* problem whose objective is to maximize the number of ones in a bit string. DUF2 has a unique optimal solution surrounded by four local optima and a wide plateau with eleven points having a fitness of zero. DUF2 is more difficult than DUF1. DUF3 is fully deceptive. The mathematical formulations of the Decomposable Unitation-Based Functions, as given in [25], can be seen below.

$$f_{DUF1} = u(x) \tag{2.9}$$

$$f_{DUF2} = \begin{cases} 4 & \text{, if } u(x) = 4 \\ 2 & \text{, if } u(x) = 3 \\ 0 & \text{, if } u(x) < 3 \end{cases} \tag{2.10}$$

$$f_{DUF3} = \begin{cases} 4 & \text{, if } u(x) = 4 \\ 3 - u(x) & \text{, if } u(x) < 4 \end{cases} \tag{2.11}$$

2.1.1.3 Dynamic traveling salesman problem

The Traveling Salesman Problem (TSP) is an NP-complete combinatorial optimization problem and defined as the problem of finding the shortest path that visits each city exactly once and then returns to the starting city. The problem can be represented by a fully connected weighted graph, such that the cities are the vertices of the graph, the connections between cities are the edges of the graph, the distance between two cities is the length of the corresponding edge.

TSP is defined as follows:

$$f(x) = min \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} x_{ij} \tag{2.12}$$

subject to

$$x_{ij} = \begin{cases} 1 & \text{, if (i,j) is used in the path} \\ 0 & \text{, otherwise} \end{cases} \tag{2.13}$$

where n is the number of cities, d_{ij} is the distance between city i and city j.

The Dynamic Traveling Salesman (DTSP) is more close to the real world than the classic TSP. The dynamism can be introduced by changing the location of cities, adding/deleting cities or changing the distance between the cities. In real world, for example, traffic jam may change in time. In this case, the salesman needs to re-plan his route with the minimum cost.

There are different variations of the DTSP in literature. Guntsch and Middendorf [39] present a DTSP solved using Ant Colony Optimization. The dynamic environment is constructed by exchanging a number of cities between the actual problem and a spare pool of cities. This benchmark is adapted in [45, 46]. Eyckelhof and Snoek [40] propose the DTSP where the travel times between the cities are changed. They apply a new Ant System approach to the DTSP. Mavrovouniotis and Yang [47] propose a similar benchmark which is solved using Ant Colony Optimization with Memory-based Immigrants. Younes et al. [41] present a benchmark generator to produce DTSP with three different modes and several Genetic Algorithms are compared on the new benchmark. Mavrovouniotis and Yang [42] propose a benchmark generator for dynamic permutation-encoded problems. They use the benchmark generator to generate several dynamic instances from Traveling Salesman Problem and Vehicle Routing Problem. Mavrovouniotis and Yang [18] propose two novel types of Dynamic Traveling Salesman Problem with traffic factor in random and cyclic environments. The change dynamics generated by DTSP represents a real world problem called the traffic jam.

In this thesis, we consider the DTSP with traffic factor proposed in [18]. In DTSP, the cost of the edge between two cities i and j is changed as follows:

$$c_{ij} = d_{ij} * t_{ij} \tag{2.14}$$

where d_{ij} is the traveled distance and t_{ij} is the traffic factor between two cities i and j. In randomly changing environment (random DTSP), every Δe iterations, each traffic factor between two cities is changed by adding a random number R with a probability of m ($t_{ij} = 1 + R$). Otherwise, the traffic factor is set to $t_{ij} = 1$, which means there is no traffic. For each edge, a different random number $R \in [R_L, R_U]$ is generated to reflect the traffic jam, where R_L and R_U are the lower and upper bound of the traffic factor, respectively. In cyclic environment (cyclic DTSP), on the other hand, the previous environments reappear in the future. To construct cyclic environments, a predetermined number of base states are generated as in randomly changing environments. Then, these base states repeat in a cycle. It should be note that m and Δe determine the severity of change and the frequency of change, respectively.

2.1.2 Performance evaluation criteria

Online and offline performance can be used to compare the performance of the algorithms [13]. Online performance is calculated as the cumulative average of all evaluations, as given below.

$$online\_performance = \frac{1}{T_{eval}} \sum_{t=1}^{T_{eval}} e_t \tag{2.15}$$

where T_{eval} is the total number of evaluations.

Offline performance is calculated as the cumulative average of the best values found so far since the last change until a given time t, as provided in Equation 2.17

$$offline\_performance = \frac{1}{T_{eval}} \sum_{t=1}^{T_{eval}} {e_t}^* \tag{2.16}$$

$${e_t}^* = \max\{e_\tau, e_{\tau+1}, \ldots, e_t\} \tag{2.17}$$

where T_{eval} is the total number of evaluations, τ is the last time step $(\tau < t)$ when change occurred, and e is the best solution found so far until the time step t since the last change at time τ.

The overall offline performance [18, 38] is also used to compare the performance of the algorithms. Given the best of generation fitness of generation i of run j ($F_{BOG_{ij}}$), the overall offline performance is calculated as given in Equation 2.18.

$$\overline{F}_{BOG} = \frac{1}{G} \sum_{i=1}^{G} (\frac{1}{N} \sum_{j=1}^{N} F_{BOG_{ij}}) \tag{2.18}$$

where G is the total number generations and N is the total number of runs.

If the optimum value is known at any point in time, offline error metric [13] can be used to compare the performance of the algorithms. The *error value* of a candidate solution

$\vec{x}$ at time t represents its distance to the optimum in terms of the objective/functional value at a given time as given in Equation 2.19.

$$err(\vec{x},t) = |optimum(t) - F(\vec{x},t)| \tag{2.19}$$

Here $optimum(t)$ and $F(\vec{x},t)$ are the function values of the global optimum solution and a given candidate solution $\vec{x}$ at time t, respectively (MPB provides the location and the function value of the current global optimum). The offline error is calculated as a cumulative average of $err(\vec{x}_b,t)^*$ which denote the error values of the best candidate solutions ($\vec{x}_b$) found so far since the last change until a given time t, as provided in Equation 2.21. An algorithm solving a dynamic environment problem aims to achieve the least overall offline error value obtained at the end of a run.

$$offline\_error = \frac{1}{T_{eval}} \sum_{t=1}^{T_{eval}} (err(\vec{x}_b,t)^*) \tag{2.20}$$

$$err(\vec{x}_b,t)^* = \min\{err(\vec{x}_b,\tau), err(\vec{x}_b,\tau+1), \ldots, err(\vec{x}_b,t)\} \tag{2.21}$$

Here T_{eval} is the total number of evaluations, τ is the last time step $(\tau < t)$ when change occurred, and x_b is the best solution found so far until the time step t since the last change at time τ.

Moreover, the population diversity [18] can be used to evaluate the performance of the algorithms. The total population diversity is calculated as provided in Equation 2.22.

$$\overline{T}_{DIV} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N} DIV_{ij}) \tag{2.22}$$

where G is the total number generations, N is the total number of runs and DIV_{ij} is the diversity of the population of generation i of run j. DIV_{ij} can be calculated as given in Equation 2.23.

$$DIV_{ij} = \frac{1}{\mu(\mu-1)} \sum_{p=1}^{\mu} \sum_{q \neq p}^{\mu} M(p,q) \tag{2.23}$$

where μ is the population size, $M(p,q)$ is the similarity metric between the solution p ans solution q.

Statistical significance tests can also be performed to compare the performance of the approaches. The comparison based on these tests show whether the observed pairwise performance variations are statistically significant or not. Some of the statistical tests include t-test, Wilcoxon sum-rank test and One-way ANOVA and Tukey HSD test. In this thesis, we perform One-way ANOVA and Tukey HSD tests at a confidence level of 95%. To provide a summary of the statistical comparison results, we count the number of times an approach obtains a significance state over the others for different change severity and frequency settings. In the tables providing the summary of statistical comparisons, the values under $s+$ shows the total number of times the corresponding approach performs statistically better than the others and $s-$ shows the vice versa; $\geq$ shows the total number of times the corresponding approach performs slightly better than the others, however, the performance difference is not statistically significant and $\leq$ shows the vice versa.

2.2 Hyper-heuristics

Heuristic and many meta-heuristic approaches operate directly on the solution space and utilize problem domain specific information. *Hyper-heuristics* [48], on the other hand, are described as more *general* methodologies as compared to such approaches, since they are designed for solving a range of computationally difficult problems without requiring any modification. They conduct search over the space formed by a set of low-level heuristics which perturb or construct a (set of) candidate solution(s) [6, 49]. Hyper-heuristics operate at a higher level, communicating with the problem domain through a domain barrier as they perform search over the heuristics space. Any type of problem specific information is filtered through the domain barrier. Due to this feature, a hyper-heuristic can be directly employed in various problem domains without requiring any change, of course, through the use of appropriate domain specific low-level heuristics. This gives hyper-heuristics an increased level of generality. Figure 2.3 shows the framework of the hyper-heuristics.

There are different categorizations of hyper-heuristics in literature. In [9], hyper-heuristics are classified into two categories: (1) without learning and (2) with learning. Hyper-heuristics without learning choose several low-level heuristics randomly or a predetermined order. On the other hand, hyper-heuristics with learning incorporate learning mechanism based on the historical performance of the low-level heuristics.

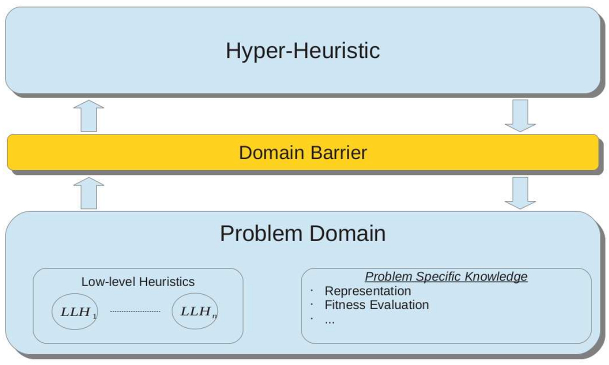


Figure 2.3 : Selection hyper-heuristic framework [1].

In [50], hyper-heuristics are classified into two categories: constructive and perturbative. A constructive hyper-heuristic approach starts with an empty solution; then, it incrementally chooses and applies an appropriate constructive heuristic until a complete solution has been obtained. A perturbative hyper-heuristic, on the other hand, starts from a complete solution which are generated randomly or using a procedure. It iteratively chooses and applies an appropriate perturbative heuristic to improve the current solution. When a stopping condition defined by the user is met, the hyper-heuristic outputs the best solution found during the search.

In [3], hyper-heuristics are classified into four groups: (1) hyper-heuristics based on the random choice of low-level heuristics, (2) greedy and peckish hyper-heuristics, which apply the all or a subset of heuristics and choose the best performing one. (3) meta-heuristic based hyper-heuristics, and (4) hyper-heuristics employing learning mechanisms.

In [10], hyper-heuristics are classified with respect to two dimensions: (1) nature of heuristic search space, and (2) the source of feedback during learning. According to the nature of heuristic search space, there are two main types of hyper-heuristics in literature [10]: methodologies that *select*, or *generate* heuristics. A selection hyper-heuristic controls a set of low-level heuristics and adaptively chooses the most appropriate heuristic to invoke at each step. A generation hyper-heuristic generates new heuristics using basic components of heuristics. Both selection and generation heuristics use the constructions and perturbation low-level heuristics. According to the source of feedback, on the other hand, hyper-heuristics are classified into three groups:

online learning hyper-heuristics, offline learning hyper-heuristics, and hyper-heuristics without learning. Online learning hyper-heuristics get the feedback/guidance during the search process while a problem instance is being solved. Offline learning hyper-heuristics make use of a training session using a set of test instances to learn how to deal with unseen instances.

2.2.1 Selection hyper-heuristics

A selection hyper-heuristic is a high-level heuristic that adaptively controls a set of simple, low-level heuristics [1, 6, 9, 51]. Basically, at any given point during the execution of a problem, a hyper-heuristic will decide the specific low-level heuristic to apply.

In a selection hyper-heuristic framework, an initial candidate solution is iteratively improved through two successive stages: *heuristic selection* and *move acceptance* [49]. Almost all selection hyper-heuristics in literature perform a single point based search [11]. In the first stage, a heuristic is selected from a fixed set of low-level perturbative heuristics and applied to the solution in hand, generating a new one. The heuristic selection method does not use any problem domain specific knowledge while making this decision. Then, the new solution is either accepted or rejected based on an acceptance method. This process is repeated until the termination criteria are satisfied and the best solution is returned. The general view of an selection hyper-heuristics in pseudocode is given in Algorithm 1.

Algorithm 1 A Selection Hyper-heuristic Framework - Single-point Search.

```
generate initial candidate solution $p$
while (termination criteria not satisfied) do
    select a heuristic $h$ from $H_1, \dots, H_n$
    generate a new solution $s = h(p)$ by applying $h$ to $p$
    decide whether to accept $s$ or not
    if ($s$ is accepted) then
        $p = s$
    end if
end while
```

2.2.1.1 Heuristic selection methods

Heuristic selection methods without learning include Simple Random (SR) and Random Permutation (RP) [6, 9]. Simple Random heuristic selection chooses a low-level heuristic at random, whereas Random Permutation uses all low-level heuristics and chooses the one at the head of a queue in which heuristics are randomly ordered.

On the other hand, an *online learning* hyper-heuristic gets feedback during the search process in order to improve its performance. Some of these methods include Random Descent (RD), Random Permutation Descent (RPD), Greedy (GR), Choice Function (CF) [6, 9], and Reinforcement Learning (RL) [52].

Random Descent applies a randomly selected heuristic to the current solution repeatedly as long as the solution improves, then another heuristic is selected randomly. Random Permutation Descent selects a heuristic in the same way as Random Permutation, but it applies the selected heuristic repeatedly as long as the solution improves. The gradient heuristic selection operators can be considered as learning hyper-heuristic components with a short term memory, since the same heuristic is used as long as there is improvement which requires objective value of the solution from the previous step. Greedy applies all low-level heuristics to the current solution and selects the one which generates the largest improvement.

Choice Function maintains a utility score for each low-level heuristic H_i (Equation 2.24), measuring how well it has performed individually ($u_1(H_i)$ in Equation 2.25) and as a successor of the previously selected heuristic ($u_2(H_i, H_{\text{selected}})$ in Equation 2.26), and the elapsed time since its last call ($u_3(H_i)$ in Equation 2.27). The heuristic with the maximum score is selected at each iteration (H_{selected}). The score of each heuristic denoted as $score(H_i)$ gets updated after the heuristic selection process. Given that $\Delta f_n(y)$ ($\Delta f_n(x,y)$) denotes the change in the solution quality and $Time_n(y)$ ($Time_n(x,y)$) denotes time spent, when the n^{th} last time heuristic y was selected and applied to the current solution (before the application of heuristic x):

$$\forall i,\ score(H_i) = \alpha u_1(H_i) + \beta u_2(H_i, H_{\text{selected}}) + \delta u_3(H_i) \quad \textbf{(2.24)}$$

$$\forall i,\ u_1(H_i) = \sum_n \alpha^{n-1} \frac{\Delta f_n(H_i)}{Time_n(H_i)} \quad \textbf{(2.25)}$$

$$\forall i,\ u_2(H_i, H_{\text{selected}}) = \sum_n \beta^{n-1} \frac{\Delta f_n(H_i, H_{\text{selected}})}{Time_n(H_i, H_{\text{selected}})} \quad \textbf{(2.26)}$$

$$\forall i,\ u_3(H_i) = elapsedTime(H_i) \quad \textbf{(2.27)}$$

Cowling et al. [51] provide a mechanism showing how the parameters α, $\beta \in (0,1]$ and δ can be adjusted dynamically.

Reinforcement Learning [52] maintains a utility score (weight) for each low-level heuristic. Initially, all scores are the same for all heuristics, e.g., 0. If the selected heuristic improves the solution, its score is increased; otherwise it is decreased, e.g. by one. A heuristic is selected with the highest utility value (or based on some other criteria) at each step. The scores for the low-level heuristics are restricted to vary between certain lower and upper bounds.

2.2.1.2 Move acceptance methods

Move acceptance methods can be deterministic or non-deterministic. Several move acceptance criteria are proposed in literature. *All Moves* (AM), *Only Improving* (OI), and *Improving and Equal* (IE) are some examples for the deterministic acceptance criteria in literature [6,53]. There are other more sophisticated acceptance mechanisms, such as *Great Deluge* (GD) [54], *Exponential Monte Carlo With Counter* (EMCQ) [55], *Simulated Annealing* (SA) [56], and *Simulated Annealing with Reheating* (SA+RH) [57].

All Moves accepts a solution in any case

Only Improving accepts a solution only if it is better than the previous solution

Improving and Equal accepts improving and equal moves.

Great Deluge accepts improving and equal moves. In addition, a worsening move is accepted, if it is better than a dynamically changing threshold value which depends on the current time and overall duration of the experiment. Linearly decreasing the threshold value at each step is a common practice as illustrated in Equation 2.28 to determine an acceptance range for a given worsening solution.

$$threshold_t = f_{final} + \Delta F \cdot \left(1 - \frac{t}{maxIterations}\right) \qquad \textbf{(2.28)}$$

where f_{final} is the expected objective value, $maxIterations$ is the maximum number of steps (or total time), t denotes the current step (time), ΔF is an expected range for the maximum solution quality (fitness/cost) change.

Exponential Monte Carlo With Counter accepts all improving moves and a worsening move with a probability p given in Equation 2.29.

$$p = e^{-\frac{\Delta f \cdot m}{Q}}, \tag{2.29}$$

where Q is a counter for successive worsening moves and m is the unit time in minutes that measures the duration of the heuristic execution, Δf is the difference in the quality between new and current solutions. Q is reset if the quality of the solution improves, otherwise it is incremented.

Simulated Annealing accepts all improving moves and a worsening move with a probability p given in Equation 2.30.

$$p = e^{-\frac{\Delta f}{\Delta F(1 - \frac{t}{maxIterations})}}, \tag{2.30}$$

Simulated Annealing with Reheating accepts all improving moves. Additionally, the following formula $e^{-\frac{\Delta f}{T}}$ is used while deciding whether or not to accept a worsening move. The temperature (T) is reduced using the nonlinear formula, $T = \frac{T}{1+\gamma T}$ [58], where

$$\gamma = \frac{(t_0 - t_{final}) iter_{temp}}{maxIterations \cdot t_0 \cdot t_{final}}, \tag{2.31}$$

$iter_{temp}$ is the number of iterations at a temperature. During the reheating phase, the temperature is increased using the formula $T = \frac{T}{1-\gamma T}$ and the system reenters the annealing phase.

2.2.2 Related literature

Cowling et al. [6] define hyper-heuristics as "heuristics to choose heuristics" and investigate the performance of different heuristic selection methods on a real-world scheduling problem. These methods include *Simple Random*, *Random Descent*, *Random Permutation*, *Random Permutation Descent*, *Greedy* and a more elaborate learning heuristic selection method, namely *Choice Function*. In [6, 59], the authors combine all the above heuristic selection methods with the following deterministic acceptance methods: *All Move* and *Only Improving*. The computational experiments result with the success of the Choice Function–All Moves hyper-heuristic.

Nareyek [52] applies *Reinforcement Learning* (RL) heuristic selection to Orc Quest and modified Logistics Domain problems. The author investigates different negative and positive adaptation strategies as well as heuristic selection methods based on the scores. *All Moves* is the acceptance method used in this study. The results show that high negative and low positive adaptation rates are preferable and maximum strategy performs better than soft max for choosing a low-level heuristic based on their scores.

Kendall and Mohamad [54] apply a *Great Deluge* move acceptance based hyper-heuristic to a mobile telecommunications network problem.

Ayob and Kendall [55] propose a set of Monte Carlo move acceptance methods inspired from the well-known simulated annealing meta-heuristic. The results show that Simple Random heuristic selection combined with *Exponential Monte Carlo With Counter* move acceptance (EMCQ) performs well.

Bai et al. [56] show that *Simulated Annealing* (SA) as a move acceptance is promising. Bilgin et al. [53] compare the performances of many heuristic selection and move acceptance combinations in hyper-heuristics. The results show that a standard simulated annealing move acceptance performs the best, especially combined with Choice Function.

Bai et al. [57] investigate the performance of a Reinforcement Learning – Simulated Annealing with Reheating (SA+RH) hyper-heuristic on nurse rostering, university course timetabling and one-dimensional bin packing problems. This hyper-heuristic generates a better performance when compared to the other meta-heuristic solutions in each problem domain. The same acceptance is also used by Dowsland et al. [60] as a part of a hyper-heuristic which hybridized Tabu Search with Reinforcement Learning as a heuristic selection method. This hyper-heuristic performs well on a shipper rationalization problem.

Burke et al. [61] compare the performance of different Monte Carlo move acceptance methods over a set of benchmark examination timetabling problems. Exponential Monte Carlo with Counter as a move acceptance delivers a poor performance as compared to Simulated Annealing based methods. Simulated Annealing with Reheating turns out to be very promising as a move acceptance component of a hyper-heuristic.

Özcan et al. [62] experiment with Great Deluge based hyper-heuristics on examination timetabling. It is observed that Reinforcement Learning–Great Deluge delivers a promising performance, when an additive/subtractive adaptation rate is used for rewarding/punishing. Similarly, Gibbs et al. [63] report the success of Reinforcement Learning–Great Deluge and Reinforcement Learning–Simulated Annealing for solving sports scheduling problems.

Drake and Özcan [64] propose a modified version of Choice Function improving its performance (ICF) in which weights dynamically change, enforcing the search process to go into diversification faster than usual, when the successive moves are non-improving.

2.2.3 HyFlex and first cross-domain heuristic search challenge

Hyper-heuristics are highly adaptive search methodologies that aim to raise the level of generality by providing solutions to a diverse set of problems having different characteristics. Hyper-heuristics Flexible framework (HyFlex) [110] is an interface designed to develop, test and compare the hyper-heuristics. The interface is referred as the domain barrier between low-level heuristics and a hyper-heuristic in the hyper-heuristics. HyFlex consists of two parts: a general-purpose and the problem-specific. The problem-specific part provided by the framework contains a number of problem domain modules. The general-purpose part contains the hyper-heuristics which need to be implemented by the user.

In [17], HyFlex is implemented as a modular framework in Java and used at the CHeSC2011 – Cross-domain Heuristic Search Challenge [111], a competition on hyper-heuristics held in 2011. In this competition, different hyper-heuristics compete for solving problem instances from different problem domains. In the current version of HyFlex, it provides six problem domains: maximum satisfiability (MAX-SAT), one-dimensional bin packing (BP), personnel scheduling (PS), permutation flow shop (FS), the traveling salesman problem (TSP) and the vehicle routing problem (VRP). For each problem domain, four main heuristic types, namely mutational heuristics (MU), crossover (OX), ruin-recreate heuristic (RC) and hill-climbing heuristics (HC), are implemented. Table 2.2 summarizes the number of low-level heuristics for each heuristic type for each problem domain. Further details about these problems and their instances can be found in [65–68].

Table 2.2 : The number of low-level heuristics for each heuristic type for each problem domain.

Problem Domain	MU	RC	OX	HC	Total
MAX-SAT	6	1	2	2	11
BP	3	2	1	2	8
PS	1	3	3	5	12
FS	5	2	4	4	15
TSP	5	1	4	3	13
VRP	3	2	2	3	10

In the current implementation of HyFlex, all low-level heuristics are perturbative heuristics and all crossover operators generate a single offspring. The parameters of low-level heuristics, namely mutation density and depth of hill-climbing, can be adjusted. The mutation density indicates the degree of the changes that the mutation operators generate a solution. The depth of hill-climbing determines the number of step completed by the hill-climbing heuristics.

In CHeSC2011, the algorithms are allowed to run with 4 test domains and 2 hidden domains. Moreover, 5 instances are used for each problem domain. Each run is repeated 31 times and is executed 600 seconds running time. As comparison and ranking, the organizers adopted the Formula 1 scoring system used before 2010. The top eight approaches are given a score of 10, 8, 6, 5, 4, 3, 2 and 1 points for each problem instance from the best to the worst, successively. The rest of the approaches receive a score of 0. The comparison and the ranking of the approaches are based on the median result generated by each approach over a given number of runs for an instance. The sum of scores over all problem instances determine the final ranking of an approach.

2.2.4 Selection hyper-heuristics in dynamic environments

Özcan et al. [69] is the first study which proposed a hyper-heuristic for solving dynamic environment problems to the best of our knowledge. The authors apply a Greedy hyper-heuristic to five well known benchmark functions. The Greedy heuristic selection method is chosen as a hyper-heuristic component with the hope that it would respond to the changes in the environment quickly. The results indeed show that this selection hyper-heuristic is capable of adapting itself to the changes.

In [70], the authors compare the performance of different heuristic selection mechanisms within the selection hyper-heuristic framework. The hyper-heuristics combine the Improving and Equal acceptance with five heuristic selection methods controlling a set of mutational low-level heuristics in a very simple dynamic environment. The landscape is only allowed to shift in this environment, and its general features remained the same. The Moving Peaks Benchmark is used during the experiments. Choice Function–Improving and Equal delivers the best average performance.

Kiraz and Topcuoglu [71] propose a population based search framework embedding a variety of hyper-heuristics which combine {Simple Random, Random Descent, Random Permutation, Random Permutation Descent, Choice Function} with {All Moves, Only Improving}. The behavior of these hyper-heuristics is investigated over a set of dynamic generalized assignment problem instances. The authors use an evolutionary algorithm operating on two subpopulations: *search* and *memory*. The individuals in the search subpopulation are perturbed using a heuristic selected by a hyper-heuristic and the other one is evolved using a standard evolutionary algorithm updating the memory periodically. The results show that the Random Permutation Descent–All Moves and Choice Function–All Moves hyper-heuristics performed well in general.

3. SELECTION HYPER-HEURISTICS IN DYNAMIC ENVIRONMENTS

In this thesis, we explore the performance of a set of hyper-heuristics in dynamic environments exhibiting different change characteristics, which are generated using the MPB generator.

We experiment with thirty five hyper-heuristics composed of five heuristic selection methods {Simple Random, Greedy, Choice Function, Reinforcement Learning, Random Permutation Descent} combined with seven move acceptance methods {All Moves, Only Improving, Improving and Equal, Exponential Monte Carlo with Counter, Great Deluge, Simulated Annealing, Simulated Annealing with Reheating}. All these hyper-heuristic components have different properties. Simple Random uses no feedback. Greedy selects the best solution at each step. Choice Function and Reinforcement Learning incorporate an online learning mechanism. Random Permutation Descent makes a random choice, but converts the framework into a hill climber, since the same heuristic is invoked repetitively as long as the solution improves. Great Deluge, Exponential Monte Carlo with Counter, Simulated Annealing and Simulated Annealing with Reheating are non-deterministic acceptance methods for which the acceptance decision depends on a given step. On the other hand, All Moves, Only Improving, Improving and Equal acceptance methods are deterministic.

The experiments consist of four parts. In the first part, a simple dynamic environment scenario is investigated, where only the locations of the peaks are changed but their heights and widths remain the same. We will refer to these set of experiments as EXPSET1. In the second part, denoted as EXPSET2, the approaches are compared in environments of different change frequencies and change severities, where peak locations as well as peak heights and widths are changed. In the third part, we explore the tracking ability of the approaches. In the last part their scalability is investigated through experiments where the number of peaks and the number of dimensions are increased.

3.1 Experimental Setting

The hyper-heuristics used in this study are applied to a set of real-valued dynamic function optimization instances produced by the Moving Peaks Benchmark (MPB) generator. A candidate solution is a real-valued vector representing the coordinates of a point in the multidimensional search space for a given instance, for which the length of the vector is the number of dimensions. In order to perturb a given candidate solution, a parameterized Gaussian mutation, $N(0,\sigma^2)$, where σ denotes the standard deviation, is implemented. Seven mutation operators based on seven different standard deviations; {0.5, 2, 7, 15, 20, 25, 30} are used as low-level heuristics within the hyper-heuristic framework during the experiments. A low-level heuristic draws a random value from the relevant Gaussian distribution for each dimension separately and this random value is added to the corresponding dimension of a candidate solution to generate a new one.

Table 3.1 lists the fixed parameters of the Moving Peaks Benchmark used during the experiments. These parameter settings are taken from [13, 16]. In the scalability experiments (subsection 3.2.5), dimension and peak counts are changed while the rest of the settings are kept the same.

Table 3.1 : Parameter settings for the Moving Peaks Benchmark.

Parameter	Setting	Parameter	Setting
Number of peaks p	5	Number of dimensions d	5
Peak heights	$\in [30,70]$	Peak widths	$\in [0.8,7.0]$
Peak function	cone	Basis function	not used
Range in each dimension	$\in [0.0,100.0]$	Correlation coefficient ϕ	0

In this study, we experiment with combinations of two change characteristics, namely the frequency and the severity of the changes. We performed some initial experiments to determine the settings for various change frequencies and severities.

First, we utilize the Simple Random heuristic selection as a basis to determine change frequency settings. We allow a Simple Random–Improving and Equal hyper-heuristic to run for long periods without any change in the environment. Based on the resultant convergence behavior given in Figure 3.1, we determine the change periods[1] as 6006 fitness evaluations for low frequency (LF), 1001 for medium frequency (MF) and 126

[1] Since we have 7 low-level heuristics and the Greedy heuristic selection method evaluates all at each step, these values are determined as multiples of 7 to give each method an equal number of evaluations during each stationary period.

for high frequency (HF). In the convergence plot, 6006 fitness evaluations correspond to a stage where the algorithm has been converged for some time, 1001 corresponds to a time where the approach has not yet fully converged and 126 is very early on in the search.

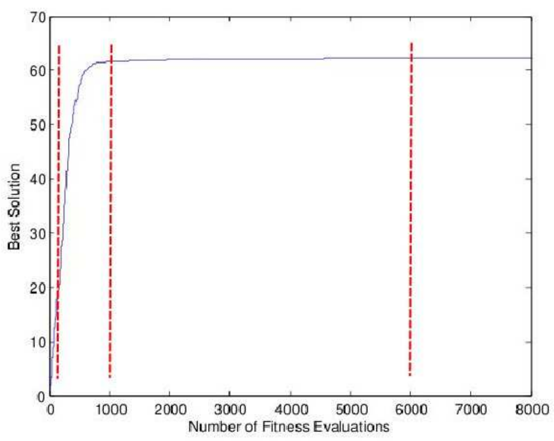


Figure 3.1 : Average convergence plot generated by the best solution versus fitness evaluation counts for Simple Random and Improving and Equal.

In MPB, the severity of the changes in the locations of the peaks, their heights and widths are controlled by three parameters, namely *vlength*, *height_severity* and *width_severity*, respectively. We determine low severity (LS), medium severity (MS) and high severity (HS) change settings based on the Moving Peaks Benchmark formulation given in Equation 2.1. The parameter settings used in the experiments for different levels of severity are provided in Table 3.2.

Table 3.2 : MPB parameter settings for each severity level.

Setting	LS	MS	HS
vlength	1.0	5.0	10.0
height_severity	1.0	5.0	10.0
width_severity	0.1	0.5	1.0

Each run is repeated 100 times for a given setting. Each problem instance contains 20 changes in a given environment, i.e. there are 21 consecutive stationary periods. The total number of iterations per run (*maxIterations*) is determined based on the change period as given in Equation 3.1,

$$maxIterations = (NoOfChanges + 1) * ChangePeriod \tag{3.1}$$

where there are $(NoOfChanges+1)$ stationary periods with a length of $ChangePeriod$, including the initial environment before the first change. The performance of the approaches is compared based on the offline error metric (see Equation 2.17).

3.1.1 Approaches used in comparisons

The performances of different hyper-heuristics are compared to well known techniques from literature including a Hypermutation [21] based approach (HM), (1,λ)-Evolutionary Strategies (ES) [72] and (μ,λ)-Covariance Matrix Adaptation Evolution Strategy (CMAES) [73–75]. These techniques are chosen since they are well known approaches to real-valued optimization and all use a different mutation adaptation scheme to deal with the dynamics in the environment. Hypermutation adapts the mutation rate whenever the environment changes. ES adapts the mutation rate based on the success or failure of the ongoing search. In CMAES, adaptation is based on the adaptation of the covariance matrix.

The parameter settings of HM, ES and CMAES are determined empirically as a result of a series of preliminary experiments so that they achieve a good performance.

Hypermutation performs a Gaussian mutation with a fixed standard deviation of 2 during the stationary periods. When a change occurs, the standard deviation is increased to 7 for 70 consecutive *fitness evaluations*. Afterwards, the standard deviation is reset to 2.

In (1,λ)-ES, λ *offspring* (new candidate solutions) are generated from one *parent* (current solution in hand) by a Gaussian mutation with zero mean and a standard deviation of σ. The initial value for σ is set to 2. Whenever the environment changes, σ is reset to this initial value. During the stationary period of the search, σ is adapted according to the classical $1/5$ success rule [72] as shown in Equation 3.2 at every k iterations. If the percentage of successful mutations, denoted as p_s is greater than $1/5$, σ is increased, otherwise it is decreased. After λ offspring are obtained, a solution is selected from them to replace the parent. The value of k is set to 7. This evolutionary process repeats until a maximum number of iterations is completed.

$$\sigma = \begin{cases} \sigma/c & \text{if } p_s > 1/5 \\ \sigma.c & \text{if } p_s < 1/5 \\ \sigma & \text{if } p_s = 1/5 \end{cases} \qquad \textbf{(3.2)}$$

During the experiments, the value of the parameter c is set to $0.9 \in [0.85, 1)$ as suggested in [72].

CMAES is the state-of-the-art algorithm for global optimization. It is based on the adaptation of the covariance matrix. In CMAES, offspring at generation $g+1$ are generated by sampling the multivariate normal distribution [73], i.e. $k = 1, \dots, \lambda$

$$x_k^{(g+1)} = \langle x \rangle_w^{(g)} + \sigma^{(g)} \sim N(0, C^{(g)}) \tag{3.3}$$

where $\langle x \rangle_w^{(g)}$ is the weighted mean of the μ best individuals at generation g, σ is the mutation step size, $C^{(g)}$ is the covariance matrix at generation g. The covariance matrix C is adapted via the evolution path. The step size σ is initialized to $\sigma = 0.3$ and is then updated using a cumulative step-size adaptation (CSA) approach, in which a conjugate evolution path is constructed [73]. Further details on CMAES can be found in [73–75].

The initial value of μ is set to 1 for CMAES for a fair comparison with the other single point search methods [73], while the value of λ for ES and CMAES is set to 7 for a fair comparison with the Greedy hyper-heuristic which makes 7 evaluations at each step.

3.1.2 Parameter settings of hyper-heuristics

Some of the heuristic selection and acceptance methods have parameters which require initial settings.

- In Reinforcement Learning, the initial scores of all heuristics are set to 15. Their lower and upper bounds are set to 0 and 30, respectively as suggested in [62]. If the current heuristic produces a better solution than the previous one, its score is increased by 1, otherwise it is decreased by 1.

- In Choice Function, α, β, and δ are set to 0.5 and updated by ± 0.01 at each iteration.

- In Exponential Monte Carlo with Counter, the value of B is set to 60, 10, 2 for LF, MF and HF changes, respectively.

- In Great Deluge, Simulated Annealing and Simulated Annealing with Reheating, the expected range is calculated as $\Delta F = initialError - optimumError$, where $initialError$ is the error value of initial candidate solution and $optimumError = 0$.

Also in Simulated Annealing with Reheating, the starting and final temperatures are set to $t_0 = -\Delta F/log(0.1)$ and $t_{final} = -\Delta F/log(0.005)$, respectively.

It is assumed that all programs are aware of the time when a change occurs during the experiments. As soon as the environment changes,

- the current solution is re-evaluated.
- the Exponential Monte Carlo with Counter parameters m and Q are reset to 1.
- the expected range (ΔF) is recalculated for Great Deluge and Simulated Annealing.
- the system enters the reheating phase for Simulated Annealing with Reheating.

On the other hand, the parameters of the heuristic selection methods Choice Function and Reinforcement Learning are not updated at all when the environment changes.

3.2 Results

All trials are repeated for 100 times using each approach for each test case. The results are provided in terms of average offline error values in the tables. The performances of the approaches are compared under a variety of change frequency-severity pair settings where each setting generates a different dynamic environment. In the rows of the tables, we can see the performance of each approach. Each column shows the performance of all the approaches for the corresponding change frequency-severity pair settings. In addition, the best performing approach is marked in bold in the result tables. The comparisons based on One-way ANOVA and Tukey HSD tests at a 95% confidence level are performed to show whether the observed pairwise performance variations are statistically significant or not. We illustrate the tracking ability of the approaches as well as their scalability, only using EXPSET2 in this section, since we have observed the same behavior for EXPSET1 and EXPSET2.

3.2.1 Results for EXPSET1

Table 3.3 summarizes the results of EXPSET1 using MPB in which only the peak locations change in time. This table shows the offline error generated by each approach for different change frequency-severity combinations. The performance of all methods degrades as the change frequency increases. Moreover, the offline error becomes particularly high when the change frequency is high. Performance also degrades for almost all methods as the severity of change increases. These observations are somewhat expected, based on the fact that the methods are provided with a very limited time to respond to the changes in the environment.

We performed statistical significance tests to determine the overall best heuristic selection and best move acceptance methods. Considering all hyper-heuristic runs where a different heuristic selection method is used, Only Improving and Improving and Equal acceptance consistently perform the better over all frequency-severity settings. However, when considering all hyper-heuristic runs where a different move acceptance method is used, there is more variation among the best performing heuristic selection methods for different frequency-severity settings:

- Greedy performs the best when combined with the All Moves acceptance.
- Choice Function is the best as a heuristic selection method to be combined with the Improving and Equal, Only Improving and EMCQ acceptance methods.
- Greedy seems to perform the best for low frequency changes, while the heuristic selection methods that rely on randomness, i.e., RPD and Simple Random perform better for higher frequency changes when combined with Simulated Annealing and Simulated Annealing with Reheating.
- Great Deluge based hyper-heuristics perform similarly regardless of the heuristic selection.

Overall, considering the average offline error results given in Table 3.3 and the statistical significance tests, Choice Function is the best performing hyper-heuristic when combined with Only Improving and Improving and Equal for EXPSET1. Hypermutation performs the best when combined with the Improving and Equal and Only Improving acceptance methods. However, overall it is one of the heuristic selection methods which delivers very poor performance. Evolutionary Strategy performs well in the cases for which the change frequency is low. Its performance deteriorates as the frequency increases. CMAES performs the best only when both the change frequency and severity are low. For this particular case, Evolutionary Strategy is the second best performing approach. For the remaining severity settings with low frequency, Evolutionary Strategy performs best. For all the remaining frequency-severity settings, Choice Function–Improving and Equal and Choice Function–Only Improving give the better performance.

3.2.2 Results for EXPSET2

Table 3.4 summarizes the results of EXPSET2 using MPB in which peak locations, their heights and widths are changed. This table shows the offline error generated by each approach for different combinations of frequency and severity of change. Similar phenomena as in the previous part (EXPSET1) are observed during this set of experiments. The methods deteriorate in performance as the change frequency increases.

We again performed statistical significance tests to determine the overall best heuristic selection and best move acceptance methods. In this set of experiments, the Improving and Equal, Only Improving and EMCQ acceptance methods all perform well. In most cases, there is no statistically significant difference between them when applied in combination with different heuristic selection method. Considering all hyper-heuristic experiments for which a different move acceptance method is used, the Choice function, Reinforcement Learning and Random Permutation Descent perform well. For all cases, there is no statistically significant difference between them when combined with Improving and Equal, Only Improving and EMCQ.

Hypermutation is again among the worst performing heuristic selection methods. Evolutionary Strategy performs the best only when both the change frequency and severity are low. Unlike in the previous experiments, in EXPSET2, CMAES does not perform the best in any of the change frequency-severity settings. For most cases, Evolutionary Strategy and CMAES are outperformed by the Choice Function, Reinforcement Learning and Random Permutation Descent in combination with either Improving and Equal, Only Improving or EMCQ acceptance methods.

3.2.3 Dynamic environment heuristic search challenge

In order to evaluate the performance of hyper-heuristics across different dynamic environments and see their relative performance as compared to the state-of-the-art

Table 3.3 : The offline error generated by each approach during the EXPSET1 experiments for different combinations of change frequency and severity settings.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
GR-AM	24.92	24.69	24.77	38.08	37.69	37.90	63.92	63.03	63.51
GR-OI	1.24	2.22	3.38	3.15	7.42	12.15	13.54	22.58	31.56
GR-IE	1.26	2.23	3.39	3.06	7.36	12.18	13.85	22.95	31.52
GR-GD	2.07	4.10	5.99	4.02	8.34	13.59	14.73	24.19	31.96
GR-EMCQ	2.69	3.67	4.76	4.89	8.20	12.96	14.16	22.72	31.72
GR-SA	3.52	7.28	13.20	11.47	18.50	23.71	38.49	43.33	46.48
GR-SA+RH	6.18	6.74	8.00	15.05	16.30	18.60	55.97	55.86	56.11
CF-AM	123.36	122.02	121.82	155.43	155.19	149.75	190.32	186.44	183.42
CF-OI	0.66	0.72	0.81	**1.43**	1.73	**2.27**	5.21	**7.25**	**11.74**
CF-IE	0.66	0.71	0.81	**1.43**	**1.65**	**2.27**	5.56	7.59	12.04
CF-GD	2.95	4.47	7.12	4.02	6.19	10.57	8.93	12.78	19.43
CF-EMCQ	0.86	0.91	1.03	1.57	1.92	2.58	5.99	8.12	12.21
CF-SA	5.88	11.39	19.77	30.97	32.45	54.03	131.49	134.67	130.75
CF-SA+RH	13.53	13.58	13.96	24.86	26.87	27.50	61.42	67.82	75.62
SR-AM	35.04	34.93	35.23	52.96	53.09	52.76	86.68	86.45	85.54
SR-OI	0.97	1.19	1.37	1.82	2.99	4.21	5.44	11.29	18.41
SR-IE	0.97	1.18	1.38	1.87	3.01	4.23	5.25	11.47	18.06
SR-GD	2.06	4.02	6.62	3.33	6.34	10.29	6.95	13.07	20.76
SR-EMCQ	1.68	2.08	2.31	2.77	4.06	5.19	6.47	12.13	18.51
SR-SA	3.70	9.72	15.96	6.79	15.01	24.01	40.63	42.19	48.23
SR-SA+RH	8.79	8.93	8.87	14.45	15.18	16.04	31.26	32.62	35.66
RL-AM	37.72	37.23	37.76	61.31	60.54	60.93	96.23	94.37	97.85
RL-OI	0.96	1.12	1.25	1.82	2.63	3.40	5.22	9.96	15.51
RL-IE	0.96	1.11	1.25	1.84	2.62	3.47	5.41	10.16	15.50
RL-GD	2.26	4.01	6.28	3.48	6.28	10.10	6.96	12.22	19.27
RL-EMCQ	1.43	1.66	1.81	2.41	3.24	4.13	6.55	10.22	15.46
RL-SA	3.63	8.95	15.65	7.43	15.33	25.40	55.90	65.70	68.59
RL-SA+RH	8.56	8.38	8.64	15.81	16.47	16.14	33.39	36.73	41.01
HM-AM	60.44	59.60	59.58	88.57	87.00	87.15	113.11	111.65	112.23
HM-OI	2.22	2.51	2.56	3.47	4.66	5.23	8.17	14.50	18.09
HM-IE	2.22	2.50	2.57	3.46	4.71	5.19	8.66	14.60	18.68
HM-GD	3.74	4.60	6.11	5.61	7.36	9.64	9.43	15.81	19.72
HM-EMCQ	2.57	2.78	2.86	3.92	4.95	5.50	9.37	14.76	18.49
HM-SA	5.14	9.14	14.87	9.79	15.51	23.90	56.10	65.38	70.23
HM-SA+RH	7.83	8.06	8.45	14.75	15.33	14.91	31.68	32.93	33.53
RPD-AM	36.60	36.90	36.43	54.86	54.28	54.40	88.81	88.96	89.45
RPD-OI	0.97	1.13	1.28	1.78	2.68	3.63	5.16	10.41	16.24
RPD-IE	0.96	1.13	1.28	1.78	2.68	3.70	**5.09**	10.27	16.36
RPD-GD	2.09	3.93	6.42	3.24	6.13	9.87	6.64	12.24	19.28
RPD-EMCQ	1.52	1.80	1.96	2.47	3.48	4.43	6.02	10.73	16.65
RPD-SA	3.56	9.19	15.04	6.27	14.16	23.00	39.40	42.66	48.91
RPD-SA+RH	8.14	8.07	8.50	13.83	14.19	14.96	30.75	32.66	35.28
ES	0.53	**0.65**	**0.79**	2.87	3.44	4.19	11.08	12.67	15.88
CMAES	**0.42**	1.59	3.14	1.96	5.60	10.06	9.66	13.57	19.97

Table 3.4 : The offline error generated by each approach during the EXPSET2 experiments for different combinations of change frequency and severity settings.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
GR-AM	26.47	22.85	24.86	39.36	32.98	35.00	63.41	52.63	56.06
GR-OI	4.35	8.82	11.48	6.19	14.06	19.14	17.08	28.16	36.08
GR-IE	4.63	8.96	11.69	6.33	15.03	19.38	17.06	27.61	35.58
GR-GD	5.11	9.96	13.09	7.05	15.04	20.34	18.70	27.81	36.31
GR-EMCQ	5.35	8.52	11.47	8.34	13.37	19.50	17.60	28.80	36.47
GR-SA	5.52	11.10	17.17	15.88	20.18	25.86	43.31	40.30	45.14
GR-SA+RH	8.88	11.35	13.81	16.93	19.29	22.84	56.71	47.78	50.02
CF-AM	120.80	100.11	103.07	157.44	127.87	128.45	191.67	147.74	149.18
CF-OI	4.38	9.07	12.19	**4.36**	9.60	13.66	8.70	16.11	24.34
CF-IE	3.81	9.46	11.04	4.68	10.75	13.59	8.58	15.97	24.82
CF-GD	5.94	12.65	17.62	7.25	14.59	21.97	11.54	19.94	31.22
CF-EMCQ	4.16	9.76	11.72	4.79	**9.54**	13.22	9.27	**15.85**	24.60
CF-SA	9.79	16.80	30.05	33.59	50.68	76.72	127.10	120.53	120.31
CF-SA+RH	17.17	21.65	24.68	29.76	34.78	39.77	68.99	73.81	85.31
SR-AM	37.03	33.09	35.36	54.60	47.71	50.13	88.27	75.67	78.21
SR-OI	3.89	8.19	10.24	5.46	9.65	13.27	8.83	18.65	26.90
SR-IE	4.04	7.54	9.84	4.96	10.76	13.60	8.87	18.46	27.24
SR-GD	5.22	9.63	12.96	6.39	13.25	17.59	9.87	20.10	29.26
SR-EMCQ	4.78	7.84	10.23	5.79	10.04	14.04	10.03	18.83	28.16
SR-SA	5.36	12.79	19.78	9.06	18.05	27.41	44.63	41.90	50.21
SR-SA+RH	10.74	12.97	14.06	17.19	19.58	21.70	35.85	35.60	39.64
RL-AM	39.82	35.35	37.06	62.72	53.69	56.58	99.50	83.29	85.32
RL-OI	4.04	7.75	9.81	4.97	9.55	13.28	8.45	18.14	25.26
RL-IE	4.10	7.96	**9.32**	5.24	9.93	13.23	9.04	18.63	**24.05**
RL-GD	5.64	10.11	13.64	6.65	13.52	17.79	9.98	19.51	28.55
RL-EMCQ	4.37	7.58	9.96	5.46	10.03	13.22	9.36	18.51	26.18
RL-SA	5.26	12.85	20.49	9.79	21.40	34.07	65.18	65.29	73.03
RL-SA+RH	10.72	12.94	15.10	18.75	21.92	24.78	37.70	43.07	48.33
HM-AM	62.52	56.36	59.70	90.72	78.52	82.27	115.32	98.13	101.77
HM-OI	5.59	10.63	13.00	6.88	13.51	15.73	11.41	22.21	29.32
HM-IE	5.44	11.37	13.48	6.72	13.09	15.81	11.26	23.53	29.63
HM-GD	6.66	11.92	15.94	8.91	15.81	19.79	12.46	23.95	30.43
HM-EMCQ	5.80	9.90	12.49	7.09	12.95	15.48	12.59	22.39	29.49
HM-SA	7.50	13.82	22.14	12.10	20.13	32.12	62.87	72.03	81.57
HM-SA+RH	11.16	14.46	16.58	17.69	22.23	24.34	35.04	38.31	42.72
RPD-AM	38.80	33.99	36.77	56.75	48.90	51.39	90.22	76.93	80.70
RPD-OI	4.26	7.54	10.17	5.01	10.19	12.61	**8.12**	17.73	25.56
RPD-IE	4.14	8.12	10.28	5.00	9.67	**12.54**	8.31	16.65	26.20
RPD-GD	5.20	8.87	14.15	6.71	12.44	17.27	10.12	18.98	28.36
RPD-EMCQ	4.28	**7.42**	9.34	5.80	10.01	13.89	8.99	17.51	26.59
RPD-SA	5.16	12.32	19.30	8.51	17.44	26.67	42.81	42.63	51.06
RPD-SA+RH	10.26	12.30	13.78	16.50	19.09	21.29	33.40	34.82	39.29
ES	**3.69**	9.19	12.74	6.18	12.21	15.68	14.58	21.37	27.40
CMAES	6.20	13.78	17.01	7.86	17.25	21.28	15.53	24.17	31.73

techniques, all approaches are scored in the same way as in CHeSC (See Section 2.1.2). The scoring system in CheSC are based on the median result generated by each approach over the number of runs. In addition to that of the competition, best and average values over all runs are used for the comparison and the ranking of the approaches in this study. Considering both EXPSET1 and EXPSET2 with all change frequency-severity combinations, there are 18 different problems. Therefore, 180 is the maximum overall score an approach can get.

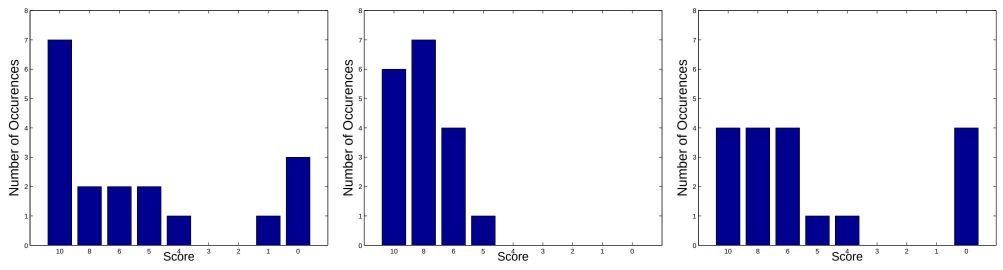
The results are summarized in Table 3.5, where the overall scores of the best fourteen approaches are included. Based on the median and average, Choice Function–Only Improving is the winner which is followed by the Choice Function–Improving and Equal. However, Choice Function–Improving and Equal is the winner based on the best. For all metrics, the top three hyper-heuristics use Choice Function as the heuristic selection component. All hyper-heuristics using All Moves, Great Deluge, Simulated Annealing or Simulated Annealing with Reheating as an acceptance component perform poorly with an overall score of 0 regardless of the heuristic selection component. Only when the best value is considered, Choice Function–Great Deluge receives 7 points. Based on the median, ES ranks eighth with a score of 40, CMAES ranks thirteenth with a score of 10, while all the Hypermutation based methods receive a score of 0 in all cases. Histograms of Formula 1 scores for Choice Function–Only Improving based on the median, Choice Function–Improving and Equal based on the best, and Choice Function–Only Improving based on average are given in Figure 3.2. Choice Function–Improving and Equal ranks the first, second and third among all approaches in all cases based on the best. Choice Function–Only Improving ranks the first in a total of seven out of the eighteen cases based on median and in a total of four out of the eighteen cases based on average, respectively.

3.2.4 Tracking ability of the approaches

The error values of the best candidate solutions calculated using Equation 2.19 versus the number of evaluations based on different change frequency and severity combinations are plotted in Figure 3.3 for Choice Function–Improving and Equal, Hypermutation–All Moves, ES and CMAES to illustrate and compare their tracking ability when the environment changes. Choice Function–Improving and Equal is

Table 3.5 : The overall Formula 1 scores for the top fourteen approaches.

Approach	*Median*	*Best*	*Average*
Choice Function–Only Improving	113	125	105
Choice Function–Improving and Equal	98	145	100
Choice Function–EMCQ	85	87	83
Reinforcement Learning–Only Improving	71	65	74
Reinforcement Learning–Improving and Equal	66	48	62
Random Permutation Descent–Only Improving	64	25	59
Random Permutation Descent–Improving and Equal	60	36	62
Evolutionary Strategies	40	50	40
Simple Random–Improving and Equal	27	23	28
Random Permutation Descent–EMCQ	23	12	25
Simple Random–Only Improving	22	11	23
Reinforcement Learning–EMCQ	21	19	26
CMAES	10	47	10
Simple Random–EMCQ	2	0	5

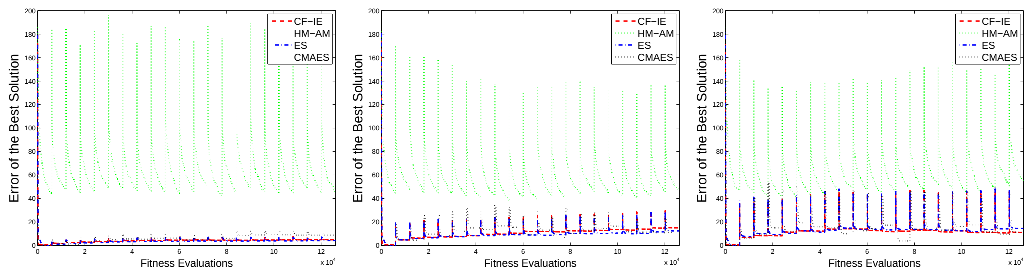


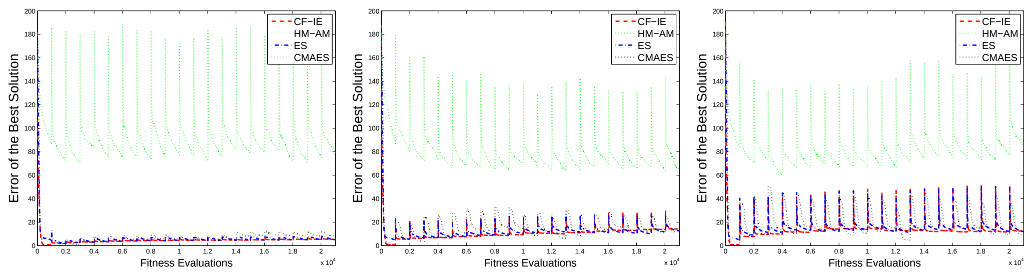
(a) CF–OI based on median (b) CF–IE based on best (c) CF–OI based on average

Figure 3.2 : Histograms of Formula 1 scores for (a) CF–OI based on the median, (b) CF–IE based on the best, and (c) CF-OI based on average over 18 dynamic environment cases.

chosen as the best performing hyper-heuristic, while Hypermutation–All Moves is chosen as a poor approach. ES and CMAES are included as they are known to be among the best real-valued optimization approaches. Figure 3.5 shows the boxplots for the final offline error values of the corresponding approaches. In the boxplot, the minimum and maximum values obtained (excluding the outliers), the lower and upper quartiles and the median are shown. The outlier points are also marked.

To be able to demonstrate the tracking behavior of the approaches more clearly, we isolated the plots for a medium frequency and a medium severity change scenario from Figure 3.3, and plotted them in Figure 3.4. From Figures 3.3 and 3.4, it can be observed that when the environment changes, the error values of the best candidate solutions produced by Choice Function–Improving and Equal, ES and CMAES increase much less than that of Hypermutation–All Moves. Moreover, these approaches are able to recover much more quickly, following the optimum. This indicates that Choice Function–Improving and Equal, ES and CMAES display


(a) LF (-LS, MS and HS)


(b) MF (-LS, MS and HS)

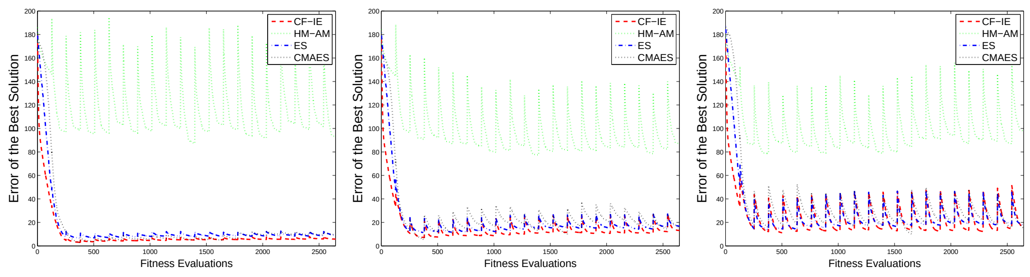

(c) HF (-LS, MS and HS)

Figure 3.3 : Comparison of approaches (CF-IE, HM-AM, ES, and CMAES) for the combinations of (a) Low, (b) Medium, (c) High frequencies and severities of change based on the error values of the best candidate solution versus evaluation counts for EXPSET2.

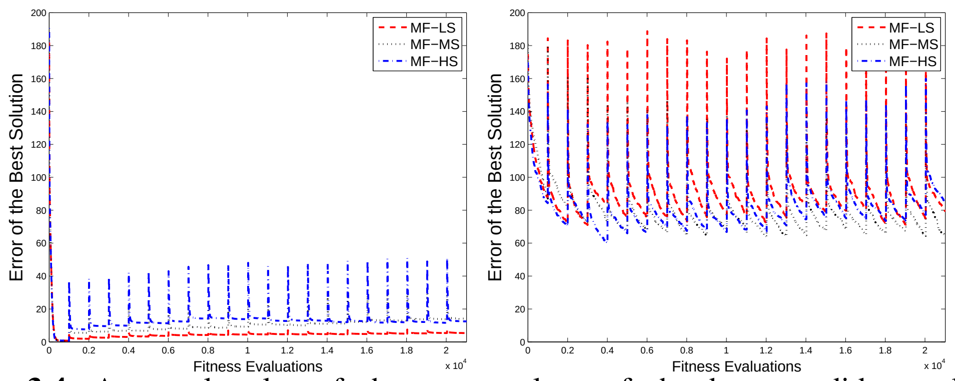
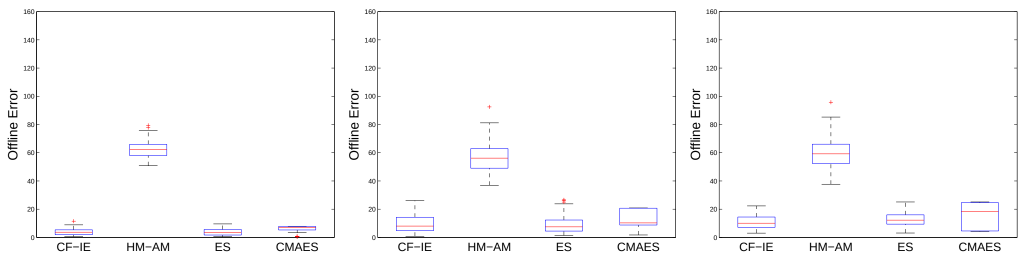
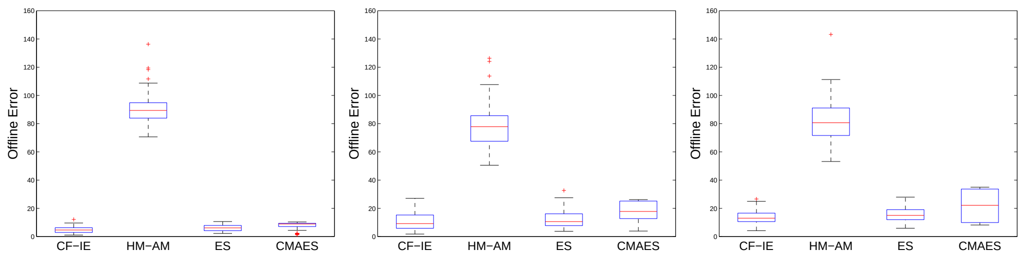


Figure 3.4 : A sample plot of the error values of the best candidate solution versus evaluation counts based on medium change frequency and medium severity combination for EXPSET2. The left and right plots show the results for Choice Function–Improving and Equal and Hypermutation–All Moves, respectively.

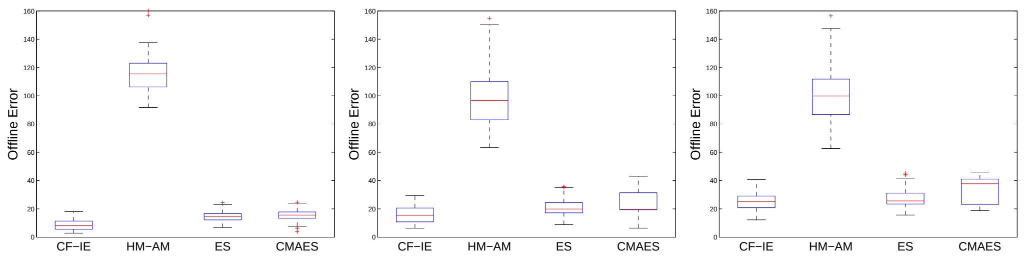
a good tracking behavior. However, the tracking behavior of Hypermutation–All Moves is poor. Choice Function-Improving and Equal performs significantly better



(a) LF (-LS, MS and HS)



(b) MF (-LS, MS and HS)



(c) HF (-LS, MS and HS)

Figure 3.5 : Box-plots of offline error values for a statistical comparison of the approaches (CF-IE, HM-AM, ES, and CMAES) for the combinations of (a) Low, (b) Medium, (c) High frequencies and severities of change using EXPSET2.

than the Hypermutation-All Moves, ES and CMAES on average during most of the environment changes as illustrated in Figure 3.5. The average performance of an approach reflects upon its tracking behavior as well.

3.2.5 Scalability results

In this part, we investigate the scalability of the approaches for different frequency-severity settings. We perform experiments with different number of peaks and dimensions. Table 3.6 summarizes the results for analyzing the effect of the number of dimensions on performance using EXPSET2 for different change frequency and severity combinations. In these experiments, only the best hyper-heuristics {Choice Function–Improving and Equal, Choice Function–EMCQ} are considered along with Hypermutation–Improving and Equal, Hypermutation–EMCQ, ES and CMAES. As expected, the performance of the hyper-heuristics and ES worsens as the number of dimensions increases. ES seems to be less affected from the dimensionality increase for lower frequency and severity settings. CMAES improves its performance when the number of dimensions is increased to 10. However, Choice Function–Improving and Equal scales better and is the best performing approach for most change frequency and severity settings.

Table 3.6 : Offline error generated by each approach in the experiments for analyzing the effect of number of dimensions for EXPSET2 for different frequency and severity combinations.

	# of dimensions	*CF-IE*	*CF-EMCQ*	*HM-IE*	*HM-EMCQ*	*ES*	*CMAES*
LF							
LS	5	**4.13**	4.18	5.39	5.74	4.14	5.98
	10	5.07	5.69	9.22	11.03	4.60	**2.30**
	20	8.65	10.94	17.76	24.09	5.65	**4.75**
MS	5	**8.52**	9.61	12.40	10.06	9.73	12.05
	10	11.03	13.23	15.91	15.03	9.88	**7.08**
	20	14.76	16.24	26.22	25.76	13.73	**11.94**
HS	5	**11.65**	12.64	12.42	12.43	12.16	12.48
	10	14.21	14.67	19.99	17.68	**12.67**	14.15
	20	18.41	20.89	32.57	29.62	**16.56**	21.84
MF							
LS	5	**4.64**	4.96	6.66	7.20	6.35	7.94
	10	7.04	8.59	11.33	13.92	9.22	**5.35**
	20	13.94	16.24	21.99	28.51	17.66	**12.46**
MS	5	11.29	**11.01**	13.11	13.07	13.10	17.94
	10	13.75	14.43	19.89	21.26	17.68	**12.31**
	20	22.83	23.03	33.63	36.96	30.54	**22.83**
HS	5	**12.90**	13.22	15.69	15.11	15.67	24.23
	10	18.12	**18.05**	25.47	24.78	23.07	21.80
	20	**25.98**	27.04	41.16	44.28	37.12	33.39
HF							
LS	5	**8.70**	8.90	11.68	13.03	14.04	14.70
	10	**21.52**	22.10	23.22	26.32	31.88	27.20
	20	56.65	60.20	**48.42**	53.73	68.98	73.81
MS	5	**16.34**	16.36	22.62	23.33	21.26	23.18
	10	**29.43**	32.85	36.73	36.70	40.03	34.87
	20	66.53	70.05	64.35	**63.72**	78.68	86.49
HS	5	**24.58**	24.98	29.81	29.55	28.32	42.36
	10	**41.31**	44.66	50.78	49.52	46.62	50.22
	20	79.43	86.44	78.59	**75.79**	85.02	86.55

Table 3.7 provides the results for analyzing the effect of the number of peaks in the environment on performance using EXPSET2 for different change frequency and severity combinations. The same hyper-heuristics {Choice Function–Improving and Equal, Choice Function–EMCQ, Hypermutation–Improving and Equal, Hypermutation–EMCQ}, ES and CMAES are included in the experiments. Again, the performance of the hyper-heuristics worsens as the number of peaks

increases. This time, ES performs similar to the hyper-heuristics. However, the effect of the increase in the number of peaks is less than the effect of the increase in dimensionality for ES. CMAES improves its performance as the number of peaks increases for all frequencies combined with low and medium severities. However, in almost all cases, it is no longer the best performing approach. All methods seem to scale well with respect to the increase in the number of peaks in the environment.

Table 3.7 : Offline error generated by each approach in the experiments for analyzing the effect of number of peaks for EXPSET2 for different frequency and severity combinations.

	# of peaks	*CF-IE*	*CF-EMCQ*	*HM-IE*	*HM-EMCQ*	*ES*	*CMAES*
LF							
LS	5	**3.78**	4.30	5.40	5.79	3.95	5.85
	10	5.07	5.23	6.27	6.32	**4.65**	5.11
	15	5.07	5.53	6.95	6.83	4.87	**3.38**
MS	5	**8.73**	8.74	9.90	10.16	9.07	13.88
	10	11.03	11.11	12.72	**10.53**	10.95	13.14
	15	10.69	11.56	12.26	**9.73**	11.36	11.06
HS	5	11.60	11.88	13.56	12.41	**11.33**	14.06
	10	13.87	14.04	14.55	**13.22**	14.34	16.06
	15	13.90	13.52	15.08	**12.28**	13.72	17.48
MF							
LS	5	**4.64**	4.83	6.48	7.17	6.23	7.28
	10	**5.15**	5.16	7.36	7.95	6.82	6.88
	15	6.13	5.84	7.41	8.33	7.60	**5.20**
MS	5	**10.59**	10.84	12.31	11.14	11.46	17.53
	10	12.08	**11.63**	14.45	12.57	13.62	17.36
	15	13.16	**11.50**	14.31	12.31	13.50	16.28
HS	5	**12.89**	13.55	15.82	15.48	15.69	22.58
	10	**15.31**	16.20	17.27	16.47	17.62	23.97
	15	**15.29**	16.26	16.20	16.38	17.36	26.29
HF							
LS	5	**8.42**	8.72	11.06	12.08	14.54	15.97
	10	8.60	**8.46**	11.82	12.92	13.77	13.79
	15	**8.69**	9.13	11.81	12.55	13.52	10.37
MS	5	**16.11**	16.77	21.93	23.49	21.77	28.37
	10	**16.87**	17.52	23.62	22.38	21.63	23.64
	15	**17.12**	17.29	21.97	21.89	21.82	23.35
HS	5	**24.13**	25.70	29.06	29.67	27.34	33.46
	10	**25.77**	25.83	29.79	28.81	28.67	30.28
	15	**25.35**	25.48	27.59	27.50	28.13	35.18

3.3 Discussion

The empirical results show that learning selection hyper-heuristics perform well in dynamic environments, especially when combined with the proper acceptance method. They can react rapidly to different types of changes in the environment and are capable of tracking them closely. The acceptance criteria which rely on some algorithmic parameter settings, such as Simulated Annealing, do not perform well as part of a hyper-heuristic in dynamic environments. This is possibly because the relevant parameters of such non-deterministic or stochastic acceptance methods often require a search for tuning. In dynamic environments, as a result of the changes in the environment, another level of complexity is added on top of the search process.

The overall results also show that accepting all moves is the worst strategy regardless of the heuristic selection method for solving dynamic environment problems. As an online learning approach which receives feedback during the search process, the Choice Function–Only Improving hyper-heuristic ranks performance-wise the first among all others based on the median and average values over all runs. Choice Function–Improving and Equal ranks the first among all approaches based on the best value over all runs. Evolutionary Strategies, Covariance Matrix Adaptation Evolution Strategy and Hypermutation perform mostly worse than the learning selection hyper-heuristics when compared across a range of dynamic environments exhibiting a variety of change properties.

In this study, it is assumed that the learning heuristic selection methods are aware of the time when the environment change occurs and acts on this. To this end, we focus on the investigation of learning heuristic selection methods which are more suitable for dynamic environments as selection hyper-heuristic components.

4. AN ANT-BASED SELECTION HYPER-HEURISTICS FOR DYNAMIC ENVIRONMENTS

Dynamic environment problems require adaptive solution methodologies which can deal with the changes in the environment during the solution process for a given problem. A selection hyper-heuristic manages a set of low-level heuristics and decides which one to apply at each iterative step. Recent studies [70, 71, 76–79] show that selection hyper-heuristic methodologies are suitable for solving dynamic environment problems with their ability of tracking the change dynamics in a given environment. Among the tested selection hyper-heuristics, learning selection hyper-heuristics are reported to perform especially well in dynamic environments. In this thesis, we propose a novel learning selection hyper-heuristic for dynamic environments, which is inspired from the ant colony optimization algorithm components. In this chapter, we describe the proposed hyper-heuristic and its variants. We investigate the performance of the proposed hyper-heuristic controlling a set of parameterised mutation operators for solving dynamic environment problems produced by the Moving Peaks Benchmark (MPB) generator. Then, we perform a comprehensive analysis of our approach.

4.1 Proposed Ant-Based Selection Hyper-heuristic Methods

In this thesis, we propose a selection hyper-heuristic incorporating a novel heuristic selection method, called the *Ant-based Selection* (*AbS*), which is based on simple ant colony optimization (ACO) algorithm components [80]. Most of the mechanisms used in ACO are adapted within *AbS*. A distinct feature of *AbS* is that, unlike ACO, *AbS* is based on a single point based search framework.

Ant Colony Optimization (ACO) [80] is a swarm intelligence technique for solving optimization problems. Basic ACO consists of solution construction and pheromone update stages. Each ant constructs a complete solution at each step. Each ant starts from a random solution component and adds the next component to the solution. The next component is determined through a stochastic local decision policy based on the pheromone trail values and the heuristic information. Pheromone trail represents the long-term memory about the search. Heuristic information, on the other hand, represents the information on the problem instances. After all ants construct a complete solution, pheromone trail values are modified. Firstly, pheromone values are decreased by a constant factor (evaporation) for all pairs of components. Then, pheromone values are increased by the amount of pheromone deposited by each ant.

Similar to Choice Function and Reinforcement Learning heuristic selection schemes, *AbS* also incorporates an online learning mechanism using a matrix of utility values. In *AbS*, each low-level heuristic pair is associated with a pheromone trail value (τ_{h_i,h_j}) which shows the desirability of selecting the j^{th} (h_j) low-level heuristic after the application of the i^{th} (h_i) low-level heuristic. All pheromone trail values are initialized to a small value τ_0. *AbS* selects a random low-level heuristic at the first step. In the following steps, the most appropriate low-level heuristic is selected based on the pheromone trail value and is applied to the solution in hand .

AbS consists of heuristic selection and pheromone update stages. For the first stage, we consider two variants of heuristic selection schemes. In both variants, the low-level heuristic h_s with the highest pheromone trail value ($h_s = \text{argmax}_{i=1..N}\, \tau_{h_c,h_j}$) is selected with a probability of q_0 where h_c is the previously selected low-level heuristic and N is the number of low-level heuristics. Otherwise, methods inspired by two of the mate selection techniques most commonly used in Evolutionary Algorithms [81] are employed to determine the next low-level heuristic to select. In the first variant, like in ACO, the next low-level heuristic is determined based on probabilities proportional to the pheromone levels of each low-level heuristic pair. This is similar to the roulette wheel mate selection in Evolutionary Algorithms. This method termed as *AbSrw* selects the next low-level heuristic h_s with a probability which is proportional to the pheromone trail value of τ_{h_c,h_s} as given in Eq 4.1.

$$p_{h_c,h_s} = \frac{\tau_{h_c,h_s}}{\sum_{l=1...N} \tau_{h_c,h_l}} \qquad \textbf{(4.1)}$$

where N is the number of low-level heuristics. In the second variant (*AbSts*), the choice of the next low-level heuristic is based on tournament selection. *AbSts* chooses the next low-level heuristic h_s with the highest pheromone trail ($h_s = \text{argmax}_{i=1..k}\, \tau_{h_c,h_j}$).

After selecting a low-level heuristic, pheromone trails are updated. Unlike in ACO, only the pheromone value between the previously selected heuristic (h_c) and the last selected heuristic (h_s) is decreased by a constant value (evaporation) and then increased by the amount of pheromone. If evaporation acted on all heuristic pairs (like in ACO), this would have caused the pheromone values of unused heuristics to drop to very low levels over time. This approach we have used has a similar effect to the one used in Choice Function where selection probabilities of heuristics not used for a long time are increased.

In the proposed method, the pheromone values are modified as follows: Firstly, only pheromone value on the pheromone matrix is decreased by a constant factor (evaporation) between h_c and h_s as given in Equation 4.2.

$$\tau_{h_c,h_s} = (1-\rho)\tau_{h_c,h_s} \quad \textbf{(4.2)}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate.

After evaporation, only the pheromone value between h_c and h_s (τ_{h_c,h_s}) is increased using Equation 4.3.

$$\tau_{h_c,h_s} = \tau_{h_c,h_s} + \Delta\tau \quad \textbf{(4.3)}$$

where h_c is the previously selected low-level heuristic and h_s is the last selected low-level heuristic. $\Delta\tau$ is the amount of pheromone added and is defined as in Equation 4.4.

$$\Delta\tau = 1/(sd * f_c) \quad \textbf{(4.4)}$$

where f_c is the fitness value of the new solution generated by applying the selected low-level heuristic h_s and *sd* is the slow decreasing parameter which controls the step size. The pseudocode of the proposed method is shown in Algorithm 2.

4.1.1 An illustrative example

Let us illustrate the working of *AbS* on the MPB. Assume that we have four low-level heuristics. So, we have a 4-by-4 matrix of pheromone trail information. All pheromone trail values are initialized to a small value $\tau_0 = 1/f_0 = 1/214.13 = 0.004670$ where $f_0 = 214.13$ is the fitness value of the initial solution. As a result, the initial pheromone values are as follows:

Algorithm 2 Pseudocode of the proposed approach.

```
1: initialize $\tau_0 = 1/f_c$
2: initialize $\tau_{h_i,h_j} = \tau_0, \forall i, j$
3: while (termination criteria not fulfilled) do
4:     if ($rand[0.0, 1.0] < q_0$) then
5:         Select $h_s = \text{argmax}_{i=1..k}\, \tau_{h_c,h_j}$
6:     else
7:         if ($AbSrw$ is selected) then
8:             the next low-level heuristic $h_s$ is determined based on roulette wheel
9:         end if
10:        if ($AbSts$ is selected) then
11:            the next low-level heuristic $h_s$ is determined based on tournament selection
12:        end if
13:    end if
14:    $\tau_{h_c,h_s} = (1-\rho)\tau_{h_c,h_s}$ (evaporation)
15:    $\tau_{h_c,h_s} = \tau_{h_c,h_s} + \Delta\tau$
16: end while
```

$$\begin{bmatrix} 0.004670 & 0.004670 & 0.004670 & 0.004670 \\ 0.004670 & 0.004670 & 0.004670 & 0.004670 \\ 0.004670 & 0.004670 & 0.004670 & 0.004670 \\ 0.004670 & 0.004670 & 0.004670 & 0.004670 \end{bmatrix}$$

After eighteen fitness evaluations, we have the following pheromone trail values:

$$\begin{bmatrix} 0.004670 & 0.004670 & 0.004670 & 0.004670 \\ 0.004670 & 0.005024 & 0.004746 & 0.005376 \\ 0.004670 & 0.004764 & 0.004670 & 0.004905 \\ 0.004670 & 0.005505 & 0.004942 & 0.004670 \end{bmatrix}$$

In addition, the last selected heuristic is the second low-level heuristic (h_2). In that case, we consider the second row of the matrix. *AbS* selects the fourth low-level heuristic (h_4) with a probability of $q_0 = 0.5$ since h_4 has the highest pheromone trail value.

$$\begin{bmatrix} 0.004670 & 0.004670 & 0.004670 & 0.004670 \\ 0.004670 & 0.005024 & 0.004746 & 0.005376 \\ 0.004670 & 0.004764 & 0.004670 & 0.004905 \\ 0.004670 & 0.005505 & 0.004942 & 0.004670 \end{bmatrix}$$

After selecting h_4, only pheromone trail value between h_2 and h_4 is decreased by a constant factor ($\tau_{h_2,h_4} = (1-0.1) * \tau_{h_2,h_4} = 0.9 * 0.005376 = 0.004838$). Then, only pheromone trail value between h_2 and h_4 is increased using $\tau_{h_2,h_4} = \tau_{h_2,h_4} + \Delta\tau = 0.004838 + 0.00114 = 0.005978$ where $\Delta\tau = 1/(sd * f_s) = 1/(10 * 87.72) = 0.00114$. The resulting pheromone trail values are the following:

$$\begin{bmatrix} 0.004670 & 0.004670 & 0.004670 & 0.004670 \\ 0.004670 & 0.005024 & 0.004746 & 0.005978 \\ 0.004670 & 0.004764 & 0.004670 & 0.004905 \\ 0.004670 & 0.005505 & 0.004942 & 0.004670 \end{bmatrix}$$

4.2 Performance Evaluation of Ant-based Hyper-heuristic

In this section, we perform experiments with our new hyper-heuristic for dynamic environments, combining the Ant-based selection scheme and the Improving and Equal acceptance technique. For comparison, we also experiment with previously used selection mechanisms which incorporate some form of online learning and are shown to be successful in dynamic environments [76], namely the Choice Function (CF) and Reinforcement Learning (RL). We also include an improved version of the Choice Function (ICF) proposed in [64]. These selection mechanisms are also used together with the Improving and Equal acceptance technique.

4.2.1 Experimental design

In the experiments, we use the Moving Peaks Benchmark (MPB) generator [16] to generate the various dynamic environments. For the parameter settings of MPB, we use the ones given in Section 3.1 labeled as EXPSET2. Based on these settings, Δe is taken as 6000 fitness evaluations for low frequency (LF), 1000 for medium frequency (MF) and 126 for high frequency (HF); the *height_severity*, the *width_severity* and *vlength* parameters are taken as given in Table 3.2 which correspond to low severity (LS), medium severity (MS) and high severity (HS) changes.

A real-valued vector corresponds to the coordinates of a point in the search space generated by the MPB. The fitness of a candidate solution at a given time t is given by its *error*, which is calculated as its distance to the optimum in terms of the objective function value at time t. Therefore, the problem becomes that of minimizing the *error* values.

The search algorithm searches through the landscape by perturbing these candidate solutions at each step to obtain a new one using a parameterized Gaussian mutation, $N(0, \sigma^2)$, where σ denotes the standard deviation. We use the same settings for the mutation operators as given in Section 3.1, which are implemented as seven different standard deviations; {0.5, 2, 7, 15, 20, 25, 30}. These mutation operators are used as the low-level heuristics in the hyper-heuristic framework.

The parameters of the proposed Ant-based selection scheme are chosen as follows: ρ and *sd* are set to 0.1 and 1, respectively. Each entry in the pheromone matrix is initialized to $\tau_0 = 1/f_s$ where f_s is the fitness value of the initial solution. We set the lower bound as 0.00001 for each entry in the pheromone matrix. For *AbSrw*, we experiment with seven q_0 values: $\{0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$. For *AbSts*, we consider five tournament size values: $k = \{2, 3, 4, 5, 6\}$ as well as the above given seven q_0 values. We also experiment with another *sd* value: $sd = 10$ for both approaches. In the tables, *AbSrw* with slow decreasing ($sd = 10$) and *AbSts* with slow decreasing ($sd = 10$) are denoted as *sAbSrw* and *sAbSts*, respectively.

For the parameter settings of the other heuristic selection methods, the following settings taken from literature are used. In Reinforcement Learning, the scores of all heuristics are initialized to 15 with lower and upper bounds as 0 and 30 respectively as given in [62]. At each step, the score of a low-level heuristic that improves performance is increased by 1 and otherwise it is decreased by 1. In Choice Function, α, β, and δ are initialized to 0.5 with updates of ± 0.01 at each iteration as given in [64]. In the Improved Choice Function, ϕ andδ are initialized to 0.5. If the low-level heuristic improves performance, the values of ϕ are set to 0.99. Otherwise, the values of ϕ_t at time t are calculated as $\phi_t = \max\{\phi_{t-1} - 0.01, 0.01\}$. In addition, δ is calculated as $\delta_t = 1 - \phi_t$

We assume that all programs are made aware when a change in the environment occurs. For the Reinforcement Learning, Choice Function and the Improved Choice Function selection methods, when a change occurs, the current solution is re-evaluated. For the proposed Ant-based selection scheme, this is not required. The parameters of none of the heuristic selection methods are reset when the environment changes. Due to the nature of the acceptance mechanism, Improving-and-Equal, the first candidate solution generated after each environment change is accepted regardless of its solution quality.

100 runs are performed for each setting where 20 changes occur in each run, i.e. there are 21 consecutive stationary periods. For evaluating the performance of the

approaches, we use the offline error [13] metric. At the end of a run, a lower overall offline error value is desired indicating a *good* performance.

4.2.2 Results and discussion

The results in the tables are provided in terms of average offline error values over 100 runs. The performances of the methods are compared under a variety of change frequency-severity pair settings.

Table 4.1 shows the results of the q_0 tests for both *AbSrw* and *sAbSrw*. In the table, $q_0 = 0.0$ means that the next low-level heuristic is chosen using only the roulette-wheel selection. However, $q_0 = 1.0$ means that roulette wheel selection is not used and always the low-level heuristic with the best score (pheromone value) is chosen to be applied. The results show that there are no statistically significant differences between most cases, however, the best values are provided by different q_0 values for different frequency-severity pairs. Therefore, to avoid overtuning, we use a setting which provides an acceptable performance in most of the cases for both approaches. For the rest of the experiments we continue with a setting of $q_0 = 0.5$ for both *AbSrw* and *sAbSrw*.

Table 4.1 : Final offline error results of various q_0 settings for *AbSrw* and *sAbSrw* under the tested change frequency-severity pairs.

Algorithm	q_0	*LF*			*MF*			*HF*		
		LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
	0.0	3.56	7.67	10.09	4.82	8.96	12.10	12.91	19.73	26.53
	0.1	3.58	7.43	9.51	5.39	8.93	12.34	12.55	19.05	25.62
	0.3	3.74	7.35	9.60	4.79	9.73	11.33	12.06	18.42	25.38
AbSrw	0.5	4.02	8.32	10.38	4.75	9.35	12.87	11.62	17.90	26.42
	0.7	3.93	7.82	11.58	4.55	9.47	13.30	10.85	18.16	25.86
	0.9	4.19	7.51	11.00	5.42	10.18	13.63	12.74	19.52	28.81
	1.0	3.71	8.43	11.85	5.58	11.50	13.44	15.21	23.03	32.24
	0.0	3.80	7.82	10.07	5.22	8.87	12.41	15.13	20.60	28.09
	0.1	3.66	7.03	9.77	5.24	9.04	12.63	14.18	20.22	27.24
	0.3	3.77	8.20	10.07	5.38	10.64	12.39	12.78	18.60	26.18
sAbSrw	0.5	3.74	7.91	10.18	5.18	8.79	11.98	11.60	17.60	25.81
	0.7	3.58	8.44	9.95	4.27	9.71	12.71	10.95	17.58	25.19
	0.9	3.94	8.13	11.48	4.97	10.21	13.10	10.93	18.16	26.63
	1.0	4.30	9.08	11.88	5.35	11.18	14.54	15.62	22.26	30.26

Then, we performed experiments to set the q_0 and tournament size values for the *AbSts* and *sAbSts* variations. The experimental results are provided in Appendix A. The best setting of these two parameters depends on the dynamics of the environment.

The best values are provided by different q_0 and tournament size values for different frequency-severity pairs. For this study, we choose a simpler approach. For those cases where tournament selection is applied, each time we let the tournament size to be determined randomly with equal probability from among the five pre-determined tournament size levels. We performed the q_0 analysis for *AbSts* and *sAbSts* based on this scheme. Table 4.2 shows the final offline error results for various q_0 settings for *AbSts* and *sAbSts* when the tournament sizes are determined randomly. We choose $q_0 = 0.5$ for both *AbSts* and *sAbSts*, since each approach delivers an acceptable performance in most of the cases with this setting which are used for the rest of the experiments.

Table 4.2 : Final offline error results of various q_0 settings for *AbSts* and *sAbSts* using random tournament size under the tested change frequency-severity pairs.

Algorithm	q_0	*LF*			*MF*			*HF*		
		LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
	0.0	4.25	8.42	10.59	4.97	9.57	13.01	15.86	20.18	26.59
	0.1	4.30	7.51	10.89	5.49	10.21	13.23	15.20	20.04	26.48
	0.3	4.06	8.73	11.09	5.37	9.81	13.12	13.80	18.90	25.84
AbSts	0.5	3.93	8.11	10.83	5.25	9.80	13.25	13.64	19.43	26.86
	0.7	3.74	8.73	10.58	4.73	10.89	13.43	12.94	19.91	26.84
	0.9	3.98	10.68	12.67	5.20	10.71	13.90	13.26	21.27	29.31
	1.0	3.82	9.06	12.63	5.24	10.91	14.75	14.85	23.56	31.82
	0.0	4.21	8.15	10.71	5.11	10.11	13.50	14.49	19.66	26.15
	0.1	4.12	7.44	10.86	5.06	10.63	12.71	13.81	18.84	25.31
	0.3	3.81	8.94	10.53	4.94	9.49	13.17	13.17	19.05	25.08
sAbSts	0.5	3.55	8.81	11.67	4.68	10.47	13.37	12.17	18.85	25.10
	0.7	3.76	9.02	11.43	4.70	10.09	12.71	12.23	19.14	26.37
	0.9	3.93	8.87	11.94	4.92	11.67	14.33	12.93	20.69	28.33
	1.0	4.26	9.75	12.00	5.38	9.87	13.83	14.32	21.67	29.81

Table 4.3 summarizes the results of *AbS* variants with the proposed settings in the previous part in which for both *AbSrw* and *sAbSrw*, q_0 is set to 0.5, for both *AbSts* and *sAbSts*, q_0 is set to 0.5 with randomly determined tournament size settings. It can be seen that *sAbSrw* provides the better results in most cases among the versions of the proposed heuristic selection scheme. *AbSts* delivers poor performance in the cases for which the change frequency is high.

Table 4.4 provides a summary of the statistical comparisons for *AbS* variants. According to the results, there are no statistically significant differences between them in most cases. The counts in the table show that *sAbSrw* has the same $s+$ counts as *AbSrw* and it also has the most $\geq$ counts.

Table 4.3 : Final offline error results for the proposed heuristic selection schemes. Here, for both *AbSrw* and *sAbSrw* $q_0 = 0.5$, for both *AbSts* and *sAbSts* $q_0 = 0.5$ with random tournament size settings.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
AbSrw	4.02	8.32	10.38	4.75	9.35	12.87	11.62	17.90	26.42
sAbSrw	3.74	**7.91**	**10.18**	5.18	**8.79**	**11.98**	**11.60**	**17.60**	25.81
AbSts	3.93	8.11	10.83	5.25	9.80	13.25	13.64	19.43	26.86
sAbSts	**3.55**	8.81	11.67	**4.68**	10.47	13.37	12.17	18.85	**25.10**

Table 4.4 : Summary of statistical significance comparisons for *AbS variants*.

Algorithm	$s+$	$s-$	$\geq$	$\leq$
AbSrw	1	0	13	13
sAbSrw	1	0	22	4
AbSts	0	2	6	19
sAbSts	0	0	11	16

Finally, we compare *sAbSrw* with those obtained using the heuristic selection methods taken from literature, namely Reinforcement Learning (RL), Choice Function (CF) and the Improved Choice Function (ICF) selection methods. Table 4.5 shows the results of these comparisons. The better results are marked in bold in the table. The results show that *sAbSrw* performs well different combinations of change frequency and severity settings. Choice Function is worse than the others for almost all cases, however, the results are very close. Improved Choice Function also gives better performance. Improved Choice Function aims to emphasize the intensification component of the generic Choice Function by automatically increasing the weight of relevant components as soon as there is improvement. Diversification, on the other hand, is introduced at a gradually increasing rate. This property works in solving stationary combinatorial optimization problems as shown in [64] as well as in dynamic optimization problems.

Table 4.5 : Final offline error results for the proposed heuristic selection schemes and RL, CF and ICF.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
sAbSrw	**3.74**	7.91	10.18	5.18	**8.79**	**11.98**	11.60	17.60	25.81
CF	3.95	9.57	11.91	4.58	10.30	14.20	8.48	**15.38**	24.83
ICF	3.88	9.24	11.24	4.70	10.63	13.01	**8.19**	15.84	**24.35**
RL	4.48	**7.35**	**10.01**	**4.48**	10.44	12.90	8.38	17.68	24.85

One-way ANOVA and Tukey HSD tests at a 95% confidence level are performed to observe whether the pairwise performance variations between the approaches, namely *sAbSrw*, Choice Function, Improved Choice Function, and Reinforcement Learning, are statistically significant or not. The corresponding results are provided in Table 4.6. As seen in the table, there are no statistically significant differences between them for most cases. *sAbSrw* is significantly better than Choice Function for low frequency and high severity and for medium frequency and high severity settings. However, the differences between *sAbSrw* and the other methods are significantly significant for high frequency and low severity setting.

Table 4.6 : Pair-wise comparison of algorithms for each dynamic environment type determined by a given change frequency and severity. Given A vs B, $s+$ ($s-$) denote that A (B) is performing statistically better than B (A), while $\approx$ denotes that there is no statistically significant performance variation between A and B.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
sAbSrw vs CF	$\approx$	$\approx$	$s+$	$\approx$	$\approx$	$s+$	$s-$	$s-$	$\approx$
sAbSrw vs ICF	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$s-$	$\approx$	$\approx$
sAbSrw vs RL	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$s-$	$\approx$	$\approx$
CF vs ICF	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$
CF vs RL	$\approx$	$s-$	$s-$	$\approx$	$\approx$	$\approx$	$\approx$	$s+$	$\approx$
ICF vs RL	$\approx$	$s-$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$

As can be seen, *sAbSrw* generates competitive results for different combinations of change frequency and severity settings. However, the most important issue is the fact that *sAbSrw* (and also all the other proposed variants) is more suitable to be used in dynamic environments than Choice Function, Improved Choice Function, and Reinforcement Learning because the proposed heuristic selection schemes do not require any special actions to be performed when the environment changes, whereas for the others, right after an environment change, the last candidate solution in the previous environment needs to be re-evaluated. This is a drawback for two reasons: it makes change detection necessary and it also wastes fitness evaluations, especially in environments where change frequencies are very high.

4.3 Experiments using a Detection Mechanism

In our previous studies [70, 76], existing heuristic selection mechanisms are tested in various types of dynamic environments and those that incorporate some form of online learning are shown to be successful. One drawback of these approaches for dynamic environments is that they require the re-evaluation of the last candidate solution in the previous environment for score calculation. As well as wasting computing resources for the re-evaluation, this also means that the algorithm needs to detect when the environment changes. The Ant-based selection heuristic selection does not require any special actions when the environment changes. However, due to the nature of the acceptance mechanism, Improving and Equal (IE), the first solution candidate generated after each environment change is accepted regardless of its solution quality. This means that the algorithm needs to know when a change occurs in the environment.

In this study, we consider a simple change detection mechanism which is commonly used in literature [18]. In this strategy, at each iteration the current solution is re-evaluated. If the fitness value of the current solution changes, this means that a change occurs. Thus, none of heuristic selection methods require the re-evaluation of the last candidate solution in the previous environment for score calculation. It should be noted that we assume that the environment is not noisy. Otherwise, a change in the fitness value of a solution candidate cannot be taken to indicate a change in the environment.

4.3.1 Experimental design

In this section, we investigate the performance of the heuristic selection methods using the above explained change detection mechanism. We consider three heuristic selection methods, namely Ant-based selection with roulette wheel, Choice Function and Reinforcement Learning. For this study, we consider Ant-based selection with roulette wheel since it performs better than tournament selection. In this section, from this point on we will use Ant-based selection (*AbS*) to denote Ant-based selection with roulette wheel selection. The selection mechanisms are used together with the Improving and Equal acceptance technique. In the experiments, we investigate the performance of all the algorithms using the same detection mechanism. It should be noted that IEd denotes Improving and Equal with the detection mechanism.

In the experiments, we use the Moving Peaks Benchmark (MPB) generator [16] to generate the various dynamic environments. For the parameter settings of MPB, we use the ones given in Subsection 4.2.1.

Parameterized Gaussian mutations are used as the low-level heuristics in the hyper-heuristic framework. We use the same settings for the mutation operators as in [76], which are implemented as seven different standard deviations; $\{0.5, 2, 7, 15, 20, 25, 30\}$.

For *AbS*, q_0, *sd* and ρ are set to 0.5, 10 and 0.1, respectively. These are the settings chosen in the previous section. For the parameter settings of the other heuristic selection methods, their proposed settings from literature [62, 76] are used.

To evaluate the performance of the approaches, we use the offline error [13] metric. 100 runs are performed for each setting where 20 changes occur, i.e. there are 21 consecutive stationary periods per run.

4.3.2 Results and discussion

Table 4.7 shows the offline errors generated by each approach for different combinations of change frequency and severity settings. As can be seen, performance degrades for all methods as the change frequency and severity values increase. IE and IEd give competitive performance for low and medium frequency. However, for high frequency IE outperforms the IEd. This observation is somewhat expected since the IEd is provided with a very limited time to respond to the changes in the environment.

Table 4.7 : The offline errors generated by each approach for different combinations of change frequency and severity settings.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
AbS - IE	3.76	**7.72**	10.63	5.00	10.04	**12.25**	11.36	17.86	25.18
AbS - IEd	3.98	7.87	10.77	4.96	10.35	14.42	14.43	23.33	34.44
CF - IE	**3.56**	9.89	11.69	**4.47**	10.57	13.28	9.02	**16.80**	25.46
CF - IEd	3.75	9.98	12.74	5.45	11.63	14.79	13.33	20.98	30.77
RL - IE	3.98	8.03	**10.04**	4.81	**10.02**	12.76	**8.39**	17.66	**25.10**
RL - IEd	4.20	8.08	10.92	5.84	11.68	15.85	21.85	26.18	35.79

First, the performance of IEd is compared to IE combined with *AbS*. The results of the ANOVA and Tukey's HSD tests for statistical significance are reported in Table 4.8. In the table, each entry shows the total number of times the corresponding approach achieves the corresponding significance state ($s+$, $s-$, $\geq$ and $\leq$) over the others for different change severity and frequency settings. From the table, it can be seen that IE is better than IEd.

Table 4.8 : Summary of statistical significance comparisons between *AbS*-IE and *AbS*-IEd.

Algorithm	$s+$	$\geq$
AbS - IE	4	4
AbS - IEd	0	1

Second, we investigate the performance of IEd used together with *AbS*, Choice Function and Reinforcement Learning. The results of the ANOVA and Tukey's HSD tests for statistical significance are reported in Table 4.9. The results show that *AbS* and Choice Function give better performance when combined with the IEd.

Table 4.9 : Summary of statistical significance comparisons between *AbS*, CF and RL combined with IEd.

Algorithm	$s+$	$s-$	$\geq$	$\leq$
AbS - IEd	5	2	9	2
CF - IEd	5	4	6	3
RL - IEd	2	6	0	10

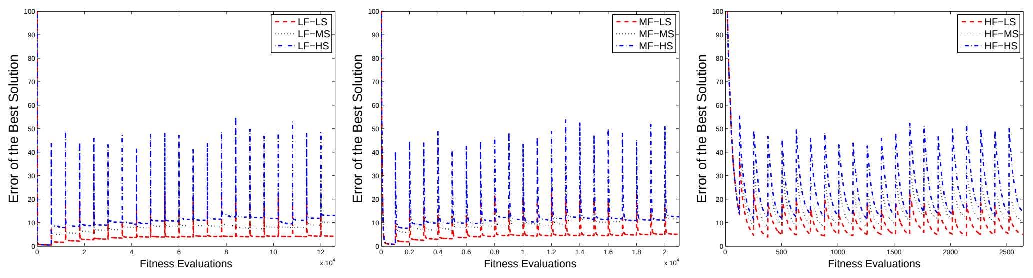
4.4 Analysis of the Components of *AbS*

In this section, we investigate the behavior of our approach on dynamic environment. We also perform the sensitivity analysis of each component of *AbS*.

In the previous set of experiments, *sAbSrw* gives better performance than the other variants. Therefore, we consider *sAbSrw* during the rest of the experiments in which the pheromone values decrease more gradually, i.e. $sd = 10$. Unless stated otherwise, the following setting is used for the rest of the experiments in this chapter: ρ and q_0 are set to 0.1 and 0.5, respectively. From this point on, we use *AbS* to denote *sAbSrw*.

4.4.1 The behavior of ant-based selection

In this subsection, we perform exhaustive tests to empirically analyze the behavior of our approach. Firstly, we examine the tracking ability of the proposed approach. To illustrate its tracking ability when a change occurs, the error values of the best candidate solutions versus the number of evaluations for low, medium and high frequencies of change are plotted in Figure 4.1. It can be figured out that *AbS* display a good tracking behavior and is able to recover quickly, following the optimum for all change frequency and severity settings.



(a) LF (b) MF (c) HF

Figure 4.1 : A sample plot of the error values of the best candidate solutions versus the number of evaluations for the combinations of (a) Low, (b) Medium, (c) High frequencies of change for *AbS*.

We investigate the change of pheromone trail value for each heuristic pair during the search. Figure 4.2 illustrates the semilogarithmic plot with logarithmic scale for y-axis for the pheromone trail values versus fitness evaluations for each heuristic pair for the high frequency and medium severity setting. As seen in the figure, the low-level heuristics with the smaller indexes are mostly selected while the others are selected less. The plots for other frequencies are not provided here, however, similar observations are made for low and medium frequency, too. In *AbS*, the heuristic with the highest pheromone trail is selected with a probability of q_0. If there are two or more heuristics with the highest pheromone value, the low-level heuristic with the smallest index is chosen. Therefore, *AbS* may tend to select the first heuristic at the beginning of the search. To avoid this, we handle the ties as follows: If there are two or more heuristics with the highest pheromone value, one is randomly selected among them. The corresponding results are illustrated in Figure 4.3. It can be seen that similar observations are made for this version, too.

As seen in Figure 4.3, mostly the low-level heuristics with the smaller indexes are selected. Based on these results, to evaluate the performance of the low-level heuristics, each low-level heuristic is allowed to run individually. The performance of each individual heuristic is tested under a random mutation hill climbing framework, which perturbs a solution using the corresponding individual parameter setting for the Gaussian mutation and improving and equal moves are accepted. The results, reported as the average offline error, can be seen in Table 4.10. According to the results in the table, LLH1, which corresponds to using Gaussian mutation with a standard deviation of 0.5 is the most successful approach for low and medium frequencies. However, LLH2 is the best performing heuristic for high frequency.

Table 4.10 : The offline errors generated by each individual low-level heuristic for different combinations of change frequency and severity settings.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
LLH1	**4.26**	**9.39**	**11.35**	**5.45**	**12.67**	**13.77**	15.66	22.82	33.31
LLH2	5.28	10.13	13.30	7.30	13.29	15.52	**12.43**	**18.96**	**23.93**
LLH3	10.56	14.66	16.80	15.39	18.61	20.56	25.08	27.63	31.60
LLH4	17.67	17.62	19.80	25.72	23.58	25.47	42.42	36.46	38.61
LLH5	21.32	19.27	21.40	30.60	26.00	28.29	50.61	41.11	44.15
LLH6	24.01	20.82	22.93	34.59	28.77	30.84	56.59	45.05	47.05
LLH7	26.49	22.35	24.25	38.19	31.16	33.26	60.97	48.39	50.92

In the experiments until now, we use seven Gaussian mutation operators as the low-level heuristics based on seven different standard deviations. Based on the results given in Table 4.10, the first two heuristics (LLH1 and LLH2) give better performance. Our previous experiments showed that using the best performing low-level heuristics does not provide good performance. Therefore, we further evaluate the proposed approach using the first four heuristics as the low-level heuristics, namely LLH1, LLH2, LLH3, and LLH4.

Table 4.11 shows the offline errors generated by *AbS* with 7 and 4 low-level heuristics for different combinations of change frequency and severity settings. As seen in the table, the results are very close. We perform statistical significance tests to determine the number of low-level heuristics to be used. The corresponding results are provided in Table 4.12. There are no statistically significant differences between them for most cases. Since using four heuristics decreases the computational requirements, the number of low-level heuristics is taken as four for the rest of the experiments.

Table 4.11 : The offline errors generated by *AbS* with 7 and 4 low-level heuristics for different combinations of change frequency and severity settings.

# of LLHs	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
7	**3.95**	**8.05**	**10.22**	4.57	**9.02**	**12.39**	11.42	**17.88**	24.48
4	4.26	8.46	10.82	**4.20**	9.71	13.07	**10.07**	17.90	**24.34**

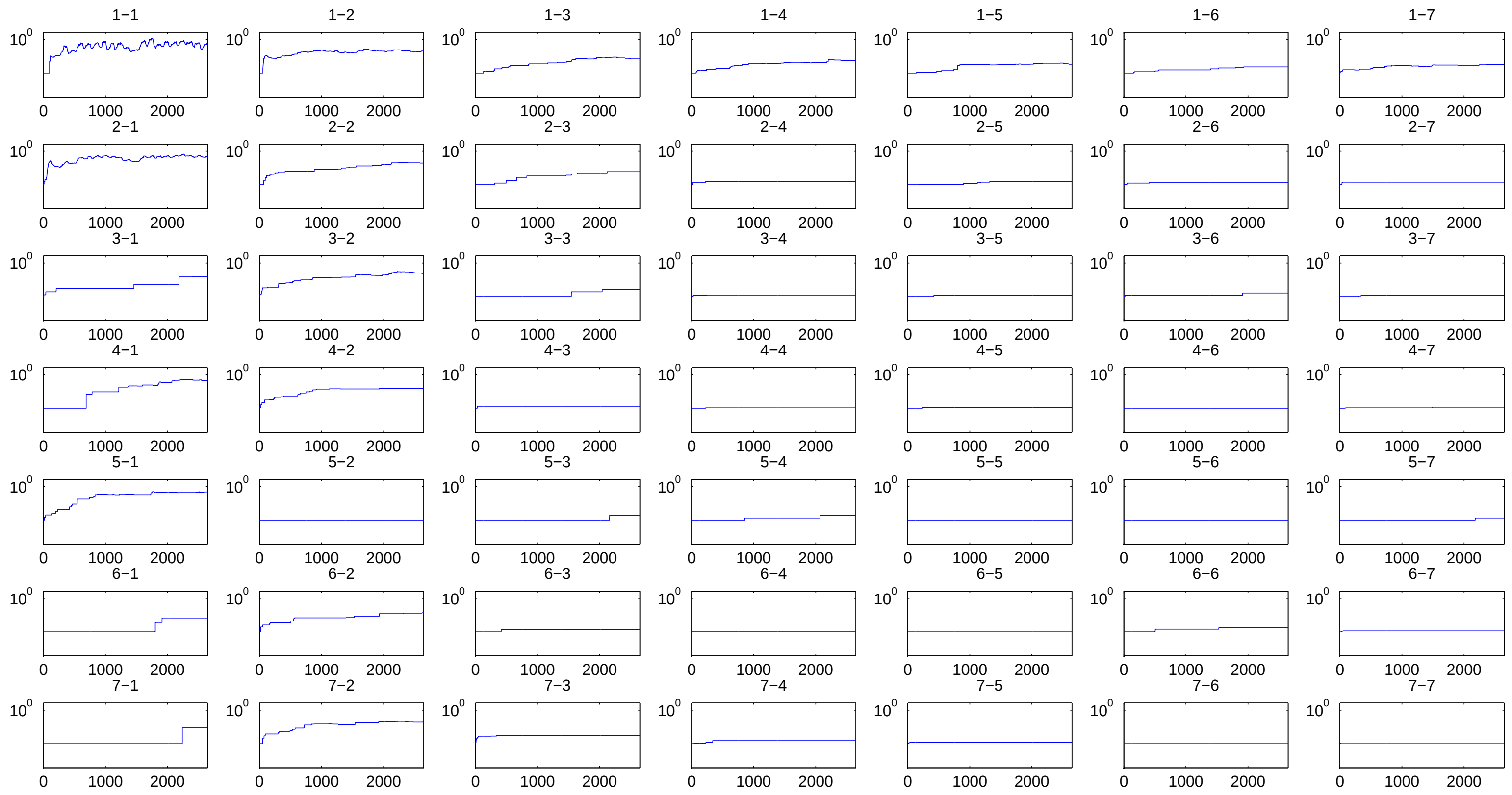


Figure 4.2 : A sample semilogarithmic plot for the pheromone trail values versus fitness evaluations for each heuristic pair based on high frequency and medium severity combination for *AbS*.

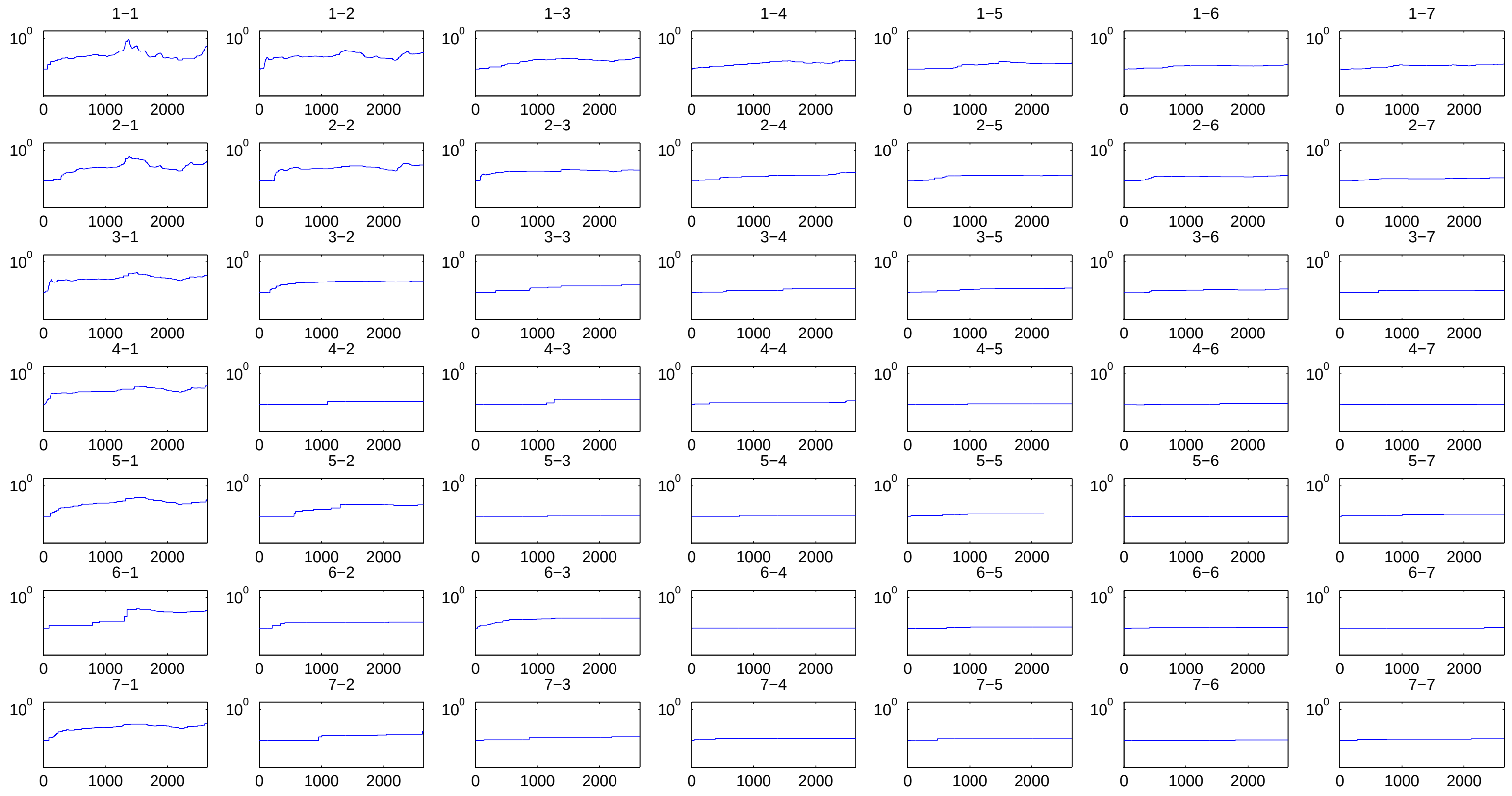


Figure 4.3 : A sample semilogarithmic plot for the pheromone trail values versus fitness evaluations for each heuristic pair based on high frequency and medium severity combination for *AbS* with handling ties.

Table 4.12 : Summary of statistical significance comparisons between *AbS* with 7 and 4 low-level heuristics.

# of LLHs	$s+$	$\geq$
7	0	6
4	1	2

4.4.2 Max-Min ant-based selection hyper-heuristic

Ant-based selection utilizes a matrix of pheromone trail values. When looking into the change of the values in the matrix during the search, we observe that while some pheromone trail values increase considerably, the others remain around their initial values. (See Figure 4.3). Therefore, we decide to experiment with another version of *AbS* (Max-Min *AbS*) which is inspired by the Max-Min Ant Colony Optimization [80] where the pheromone trail values are restricted to vary between certain lower and upper bounds. Unlike Max-Min Ant Colony Optimization, the lower and upper bounds are constant during the search in this method. This version of *AbS* is denoted as *MM AbS*. Both *AbS* and *MM AbS* use 4 low-level heuristics. In *MM AbS*, the lower and upper bounds are set to $\tau_0/50$ and $\tau_0 * 50$ where τ_0 is the initial value of the pheromone trails. This setting is determined empirically as a result of a series of preliminary experiments so that they achieve a good performance. The corresponding offline errors are given in Table 4.13. It can be seen that *MM AbS* delivers good performance for most cases.

Table 4.13 : The offline errors generated by *AbS* and *Max-Min AbS* for different combinations of change frequency and severity settings.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
AbS	3.89	9.05	11.07	5.21	**10.06**	13.31	9.04	**16.39**	24.22
MM AbS	**3.85**	**8.75**	**10.36**	**5.18**	10.30	**13.27**	**8.69**	16.83	**23.48**

Table 4.14 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for *AbS* and *Max-Min AbS*.

Algorithm	$s+$	$s-$	$\geq$	$\leq$
AbS	0	0	2	7
MM AbS	0	0	7	2

An overall comparison of two approaches is provided in Table 4.14. It can be seen that there are no statistically significant differences between them for all cases. However, *MM AbS* performs slightly better than *AbS* for 7 instances. Therefore, we use *MM AbS* as the heuristic selection method for the rest of the experiments.

4.4.3 Re-initialization of pheromone trails with max-min *AbS*

A simple approach to address dynamic optimization problems is to restart the search algorithm when the environment changes. To this end, the pheromone trails values are re-initialized with the same initial value τ_0 whenever a change occurs. Table 4.15 shows the offline errors generated by *MM AbS* and *MM AbS-R* for different combinations of change frequency and severity settings. In this table, *MM AbS-R* denotes the Max-Min *AbS* with re-initialization. It can be observed that *MM AbS-R* gives better performance for medium frequency and high severity settings. On the other hand, *MM AbS* outperforms *MM AbS-R* for all other cases. As expected, the re-initialization of pheromone trails delivers very poor performance for high frequency since it is provided with a limited time for search after re-initialization.

Table 4.15 : The offline errors generated by *MM AbS* and *MM AbS-R* for different combinations of change frequency and severity settings.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
MM AbS	**3.85**	**8.75**	**10.36**	**5.18**	**10.30**	13.27	**8.69**	**16.83**	**23.48**
MM AbS-R	3.99	8.88	10.87	5.80	10.85	**12.80**	17.05	20.39	26.30

4.4.4 The influence of q_0

In this part of the experiment, we explore the influence the settings of q_0 which may affect the performance of our approach. We experiment with seven q_0 values: $\{0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$. For this experiment, we consider *MM AbS* with four low-level heuristics and $sd = 10.0$ which are the good settings obtained in the previous sets of experiments. Table 4.16 shows the results of various q_0 settings for *MM AbS*. It should be note that $q_0 = 0.0$ means that the next heuristic is selected using only the roulette wheel. On the other hand, $q_0 = 1.0$ means that always the heuristic with the best pheromone value is selected. The results show that the best values provided by different q_0 values for different frequency-severity settings. According to results, *MM AbS* delivers very poor performance for $q_0 = 1.0$. Figure 4.4 illustrates this observation for different combination of change frequency and severity settings.

Table 4.16 : Final offline error results of various q_0 settings for *MM AbS* under the tested change frequency-severity pairs.

q_0	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
0.0	3.74	8.30	**10.58**	4.96	10.38	**12.67**	10.93	18.18	24.80
0.1	4.21	**8.18**	10.93	5.01	10.67	13.06	10.33	17.69	25.45
0.3	**3.58**	8.55	11.47	4.86	**9.04**	12.83	9.50	17.07	24.86
0.5	3.84	8.35	11.47	4.67	9.32	13.35	**9.14**	**16.02**	23.41
0.7	3.97	9.56	11.28	4.56	9.61	12.99	9.31	17.43	**22.80**
0.9	4.31	10.09	10.84	**4.34**	10.29	13.58	9.47	16.59	24.27
1.0	7.05	12.06	13.80	9.37	13.21	16.02	18.25	22.37	27.36

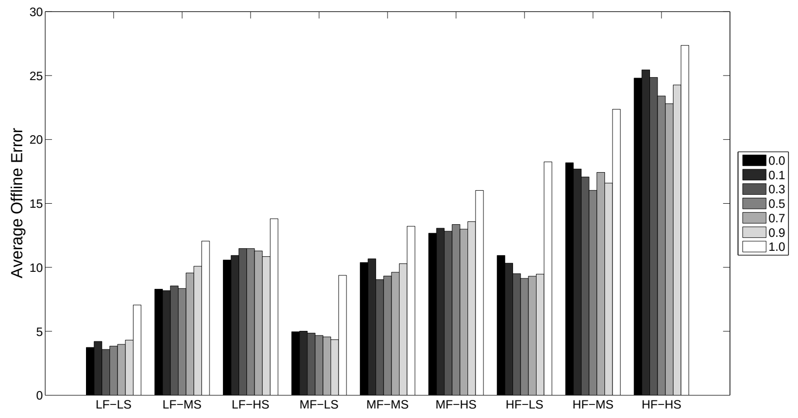


Figure 4.4 : Final offline error versus of different q_0 values for *MM AbS* for different combination of change frequency and severity settings.

We also perform statistical significance tests to determine the best setting of q_0. The statistical comparison summary is provided in Table 4.17. In this table, the results for $q_0 = 1.0$ is not included as it is significantly worse than the rest. Based on the results, the differences between different q_0 values are not statistically significant for most cases

Table 4.17 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts various q_0 settings for *MM AbS*.

q_0	$s+$	$s-$	$\geq$	$\leq$
0.0	0	4	22	19
0.1	0	1	13	31
0.3	1	0	23	21
0.5	1	0	28	16
0.7	2	0	24	19
0.9	1	0	20	24

To be able to decrease the number of parameters needing to be tuned, we try to develop an adaptive version for q_0. However, the results show that the differences between different q_0 values are not statistically significant for different frequency-severity settings. Hence, we observe that an adaptive version of q_0 is not required.

4.4.5 The influence of slow decreasing parameter

In this set of experiment, we look into effect the slow decreasing parameter. To this end, different *sd* values are tested for *MM AbS* with four low-level heuristics. Here, q_0 is set to 0.5. For this set set of experiment, we experiment with seven *sd* values: $\{1, 10, 30, 50, 75, 100, 150\}$. Table 4.18 shows the results of various *sd* settings for *MM AbS*. The results show that the best offline error values provided by different *sd* values for different frequency-severity settings. Figure 4.5 illustrates this observation for different combination of change frequency and severity settings.

Table 4.18 : Final offline error results of various *sd* settings for *MM AbS* under the tested change frequency-severity pairs.

sd	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
1	3.90	9.03	11.34	4.85	10.41	13.15	12.02	18.03	24.45
10	3.53	7.85	11.78	4.71	10.15	13.09	**8.98**	16.86	24.22
30	**3.46**	9.20	11.22	4.96	11.00	13.08	9.24	16.69	**22.93**
50	3.83	8.52	11.02	**4.49**	10.45	12.54	9.05	16.33	24.05
75	4.17	9.16	11.28	4.63	**9.65**	12.35	9.18	**16.30**	23.76
100	3.72	**7.69**	11.03	5.18	10.12	13.38	9.17	16.46	24.57
150	4.10	8.09	**10.88**	5.24	10.24	**12.22**	9.60	16.73	23.35

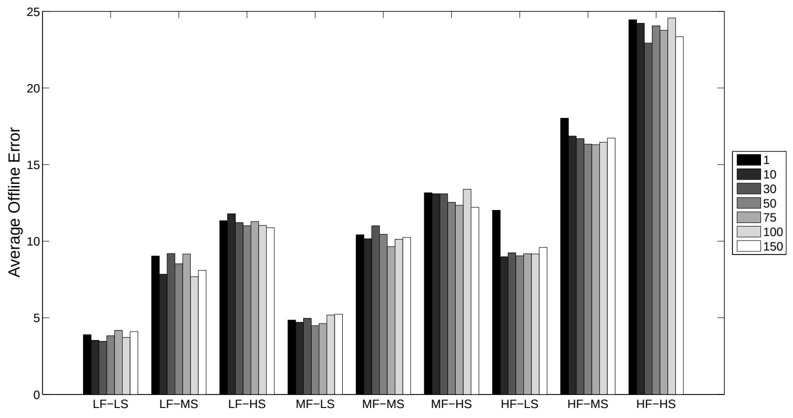


Figure 4.5 : Final offline error versus of different *sd* values for *MM AbS* for different combination of change frequency and severity settings.

The results of statistical significance tests are given in Table 4.19. As can be seen from the results, there are no statistically significant differences between them for most cases. However, $sd = 1$ is significantly worse than the others for 6 cases.

Table 4.19 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts various *sd* settings for *MM AbS*.

sd	$s+$	$s-$	$\geq$	$\leq$
1	0	6	12	36
10	1	0	28	25
30	1	0	24	29
50	1	0	34	19
75	1	0	31	22
100	1	0	27	26
150	1	0	27	26

Based on the results, the setting of slow decreasing parameter is not very critical. There are no statistically significant differences between *sd* values for different frequency-severity pairs. Therefore, we decide to choose a setting which produce acceptable results instead of adaptive version of *sd*.

4.4.6 The influence of evaporation rate

Pheromone evaporation allows the algorithm to forget the bad decisions previously made, which can be seen as exploration mechanism. The evaporation rate (ρ) is an important parameter of Ant Colony Optimization. An approach with small evaporation rate adapt slowly, whereas an approach with high evaporation adapt quickly. In this set of experiment, we investigate the effect of the evaporation rate. To this end, we experiment with four different ρ values: $0.10, 0.15, 0.2, 0.25$. Table 4.20 shows the results of various ρ settings for *MM AbS*. The results show that the performance of *MM AbS* is not much affected by the settings of *rho*. Figure 4.6 illustrates this observation for different combination of change frequency and severity settings.

Table 4.20 : Final offline error results of various ρ settings for *MM AbS* under the tested change frequency-severity pairs.

ρ	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
0.10	3.85	8.14	10.35	4.80	**8.57**	**12.01**	11.95	18.05	25.07
0.15	3.81	7.64	**9.81**	4.88	9.35	12.51	**10.67**	17.49	25.78
0.20	3.89	8.27	10.12	**4.78**	9.72	12.45	11.50	**17.05**	24.77
0.25	**3.63**	**7.63**	10.09	5.01	10.12	12.46	11.73	17.53	**24.25**

Mavrovouniotis and Yang [82] propose an adaptive version for the evaporation rate parameter. In the adaptive approach, if the algorithm approaches the stagnation situation, the evaporation rate is increased by a fixed step size; otherwise, it is decreased by a fixed step size. To detect the stagnation behavior, they consider

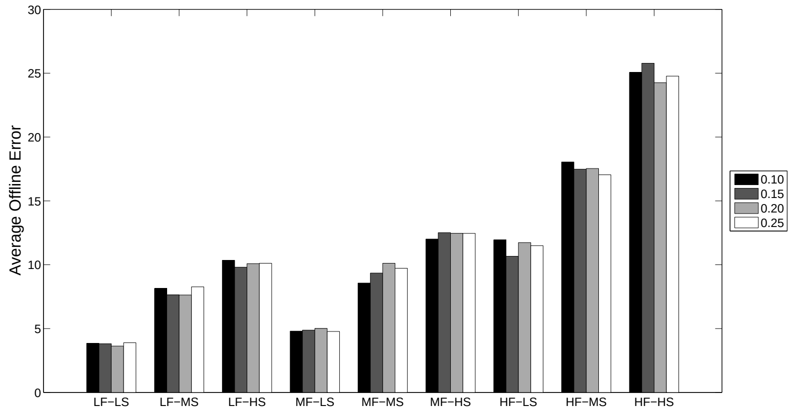


Figure 4.6 : Final offline error versus of different ρ values for *MM AbS* for different combination of change frequency and severity settings.

λ-branching factor which measures the distribution of the pheromone trail values. The λ-branching factor [80] for node i is defined as follows:

$$\lambda_i = \sum_{j=1}^{d} I_{ij} \qquad \textbf{(4.5)}$$

where d is the number of arcs incident to node i and I_{ij} is defined as the following:

$$I_{ij} = \begin{cases} 1 & \text{, if } \tau_{ij} \geq \tau^i_{min} + \lambda(\tau^i_{max} - \tau^i_{min}) \\ 0 & \text{, otherwise} \end{cases} \qquad \textbf{(4.6)}$$

where $\lambda \in [0,1]$ is a constant parameter, τ^i_{min} and τ^i_{max} are the minimal and the maximal pheromone trail values on the arcs incident to node i. The average λ-branching factor ($\bar{\lambda}$) is calculated as the average of the λ-branching factors of all nodes (given in Eq. 4.7)

$$\bar{\lambda} = \frac{1}{2n} \sum_{i=1}^{n} \lambda_i \qquad \textbf{(4.7)}$$

where n is the number of nodes in the corresponding graph.

We use the same measurement, namely average λ-branching factor, to detect the stagnation behavior. According to results, *MM AbS* provides a low λ-branching factor throughout the run. Therefore, the algorithm does not enter the stagnation [80]. Therefore, we do not require an adaptive version of evaporation rate as in [82]. Figure 4.6 also confirms this observation.

5. APPLICATIONS OF THE ANT-BASED SELECTION HYPER-HEURISTICS

In this chapter, we present three applications of the proposed hyper-heuristic, namely Ant-based selection hyper-heuristic. Firstly, we use hyper-heuristics in a multi-population framework, combining offline and online learning mechanisms. We collaborated with Gönül Uludağ in this study. Secondly, we implement the proposed approaches on HyFlex which is an interface to develop hyper-heuristics. Finally, we explore the performance of the proposed approaches on a real-world optimization problem referred to as the Dynamic Traveling Salesman problem.

5.1 Application I: Hyper-heuristics in A Hybrid Multi-population Framework

Estimation of Distribution Algorithms (EDAs) [83] are population based search methodologies in which new candidate solutions are produced using the probabilistic distribution model learned from the current best candidate solutions. There is a growing number of studies which apply *improved* variants of EDAs in dynamic environments [25, 84–89].

There is an emerging field of research in the semi-automated design of search methodologies: hyper-heuristics. This study focuses on the selection hyper-heuristic methodologies. There is strong empirical evidence showing that selection hyper-heuristics are able to quickly adapt without any external intervention in a given dynamic environment providing effective solutions [70, 71].

In this study, in order to exploit the advantages of approaches with learning and those with model-building features in dynamic environments, we propose a hybridization of EDAs with hyper-heuristics in the form of a two-phase framework, combining offline and online learning mechanisms [77–79]. A list of probability vectors for generating good solutions is learned in an offline manner in the first phase. We consider PBIL for the first phase in this study. In the second phase, *two* sub-populations are maintained. A sub-population is sampled using an EDA, while the other one uses a hyper-heuristic for sampling appropriate probability vectors from the previously learned list in an online manner. In this study, we choose a dual population PBIL (PBIL2) as the EDA component.

We perform exhaustive tests to determine a selection method which performs well within the proposed framework. We also compare the proposed framework, incorporating the chosen heuristic selection scheme, to similar methods from literature.

5.1.1 A hybrid framework for dynamic environments

In this subsection, we describe a new multi-phase hybrid framework, referred to as hyper-heuristic based dual population EDA (HH-EDA2), for solving dynamic environment problems.

Although we choose PBIL2 as the EDA component in our studies, the proposed hybrid framework can combine any multi-population EDA with any selection hyper-heuristic in order to exploit the strengths of both approaches.

HH-EDA2 consists of two main phases: offline learning and online learning. In the offline learning phase, a number of masks to be used in the XOR generator are sampled over the search space. The search space is divided into M sub-spaces and a set of masks is generated randomly in each sub-space, thus making the masks distributed well over the landscape. For the XOR generator, each mask corresponds to a different environment. Then, for each environment (represented by each mask) PBIL is executed. As a result of this, good probability vectors $\overrightarrow{P}list$ corresponding to a set of different environments are learned in an offline manner. These learned probability vectors are stored for later use during the online learning phase of HH-EDA2.

In the online learning phase, the probability vectors $\overrightarrow{P}list$, serve as the low-level heuristics, which a selection hyper-heuristic manages. Figure 5.1 shows a simple diagram illustrating the structure and execution of HH-EDA2.

The online learning phase of the HH-EDA2 framework uses the PBIL2 approach. Similar to PBIL2, the population is divided into two sub-populations and two probability vectors, one for each sub-population, are used simultaneously. As seen in Figure 5.1, $pop1$ represents the first sub-population and $\overrightarrow{P}_1$ is its corresponding probability vector; $pop2$ represents the second sub-population and $\overrightarrow{P}_2$ is its

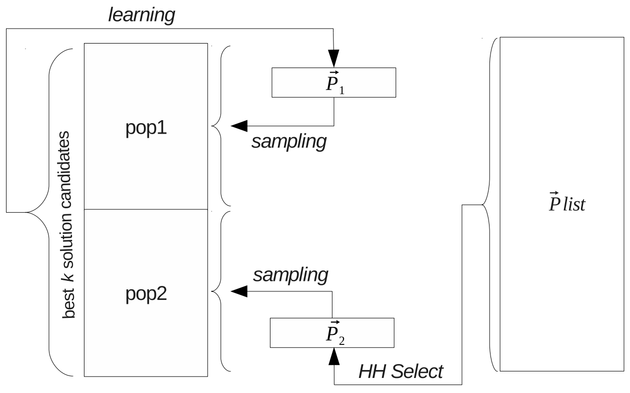


Figure 5.1 : The framework of HH-EDA2.

corresponding probability vector. The pseudocode of the proposed HH-EDA2 is shown in Algorithm 3.

Algorithm 3 Pseudocode of the proposed HH-EDA2 approach.

```
1: $t := 0$
2: initialize $\overrightarrow{P}_1(0) := \overrightarrow{0.5}$
3: $\overrightarrow{P}_2(0)$ is selected from $\overrightarrow{P}list$
4: $S_1(0) := sample(\overrightarrow{P}_1(0))$ and $S_2(0) := sample(\overrightarrow{P}_2(0))$
5: **while** (*termination criteria not fulfilled*) **do**
6:     evaluate $S_1(t)$ and evaluate $S_2(t)$
7:     adjust next population sizes for $\overrightarrow{P}_1(t)$ and $\overrightarrow{P}_2(t)$ respectively
8:     place $k$ best samples from $S_1(t)$ and $S_2(t)$ into $\overrightarrow{B}(t)$
9:     send best fitness from whole/second population to heuristic selection component
10:    learn $\overrightarrow{P}_1(t)$ toward $\overrightarrow{B}(t)$
11:    mutate $\overrightarrow{P}_1(t)$
12:    $\overrightarrow{P}_2(t)$ is selected using heuristic selection
13:    $S_1(t) := sample(\overrightarrow{P}_1(t))$ and $S_2(t) := sample(\overrightarrow{P}_2(t))$
14:    $t := t+1$
15: **end while**
```

In HH-EDA2, the first probability vector $\overrightarrow{P}_1$ is initialized to $\overrightarrow{P}_{central}$, and the second probability vector $\overrightarrow{P}_2$ is initialized to a randomly selected vector from $\overrightarrow{P}list$. Initial sub-populations of equal sizes are sampled independently from their own probability vectors. After the fitness evaluation process, sub-population sample sizes are slightly adjusted within the range $[0.3*n,\ 0.7*n]$ according to their best fitness values. At each iteration, if the best candidate solution of the first sub-population is better than the best candidate solution of the second sub-population, the sample size of the first sub-population, n_1 is determined by $min(n_1+0.05*n, 0.7*n)$; otherwise n_1 is defined by $min(n_1-0.05*n, 0.3*n)$. While, $\overrightarrow{P}_1$ is learned towards the best solution candidate(s) in the whole population and mutation is applied to $\overrightarrow{P}_1$, $\overrightarrow{P}_2$ is selected using the heuristic selection methods from $\overrightarrow{P}list$. No mutation is applied to $\overrightarrow{P}_2$. Then, the two sub-populations are sampled based on their respective probability vectors. The approach repeats this cycle until some termination criteria are met. In the HH-EDA2 framework, different heuristic selection methods can be used for selecting the second probability vector from $\overrightarrow{P}list$.

5.1.2 Computational experiments

In this study, we perform two groups of experiments. In the first group, we investigate the influence of different heuristic selection methods on the performance of the proposed framework, to determine the most suitable one for dynamic environment problems. In the second group of experiments, the proposed framework, incorporating the chosen heuristic selection scheme, is compared to similar methods from literature.

5.1.2.1 Experimental design

In the offline learning phase, first a set of M XOR masks are generated. In order to have the XOR masks distributed uniformly on the search space, an approach similar to stratified sampling is used. Then, for each mask, PBIL is executed for 100 independent runs where each run consists of G generations. During offline learning, each environment is stationary and 3 best candidate solutions are used to learn probability vectors. The population size is set to 100. At the end of the offline learning stage, the probability vector producing the best solution found so far over all runs for each environment, is stored in $\overrightarrow{P}list$. The parameter settings for PBIL used in this stage is given in Table 5.1.

Table 5.1 : Parameter settings for PBILs.

Parameter	Setting	Parameter	Setting
Solution length	100	Mutation rate P_m	0.02
Population size	100	Mutation shift δ_m	0.05
Number of runs	100	Learning rate α	0.25

After the offline learning stage, we experiment with four main types of dynamic environments: randomly changing environments (Random), environments with cyclic changes of type 1 (*Cyclic1*), environments with cyclic changes of type 1 with noise (*Cyclic1-with-Noise*) and environments with cyclic changes of type 2 (*Cyclic2*). In the Cyclic1 type environments, the masks representing the environments, which repeat in a cycle, are selected from among the sampled M masks used in the offline learning phase of HH-EDA2. To construct Cyclic1-with-Noise type environments, we added a random bitwise noise to the masks used in the Cyclic1 type environments. In Cyclic2 type environments, the masks representing the environments, which repeat in a cycle, are generated randomly.

To generate dynamic environments showing different dynamism properties, we consider different change frequencies τ, change severities ρ and cycle lengths CL. We determined the change periods which correspond to low frequency (LF), medium frequency (MF) and high frequency (HF) changes as a result of some preliminary experiments where we execute PBIL on stationary versions of all the Decomposable Unitation-Based Functions. Table 5.2 shows the determined change periods for each Decomposable Unitation-Based Function.

Table 5.2 : The value of the change periods.

Functions	LF	MF	HF
DUF1	50	25	5
DUF2	50	25	5
DUF3	100	35	10

In the Random type environments, the severity of changes are determined based on the definition of the XOR generator and are chosen as 0.1 for low severity (LS), 0.2 for medium severity (MS), 0.5 for high severity (HS), and 0.75 for very high severity (VHS) changes. For all types of cyclic environments, the cycle lengths CL are selected as 2, 4 and 8. Except for Cyclic1-with-Noise type of environments, the environments return to their exact previous locations.

In [78], we explore the effects of restart schemes for HH-EDA2. Our experiments showed that a restart scheme significantly improves the performance of HH-EDA2. In the best performing restart scheme for HH-EDA2, only the first probability vector $\overrightarrow{P}_1$ is reset to the to $\overrightarrow{P}_{central}$, whenever an environment change is detected.

Since HH-EDA2 is a multi-population approach, which also uses a kind of memory, for our comparison experiments, we focus on memory based approaches as well as multi-population ones which are shown in literature to be successful in dynamic environments. Therefore, we use different variants of PBILs with restart schemes and a sentinel-based genetic algorithm which is multi-population approach to dynamic environments. In literature, several PBIL variants are proposed for dynamic environments [25, 31, 90]. In this thesis, we consider PBIL with restart (PBILr), dual population PBIL with restart (PBIL2r), memory-based PBIL with restart (MPBILr), and dual population memory-based PBIL with restart (MPBIL2r). Further details about memory-based PBIL can be found in [25, 90].

Both in PBIL2 and HH-EDA2, each sub-population size is initialized as 50 and adjusted within the range of [30, 70]. For MPBILr and MBIL2r, the population size n is set to 100 and the memory size is fixed to $0.1 * n = 10$. The memory is updated using a stochastic time pattern. After each memory update, the next memory updating time is set as $t_M = t + rand(5, 10)$. For MPBIL2r, initial sub-populations are $0.45 * n = 45$ and sub-population sample sizes are slightly adjusted within the range of [30, 60].

For the sentinel-based genetic algorithm, we use tournament selection where the tournament size is 2, uniform crossover with a probability of 1.0, mutation with a mutation rate of $1/l$ where l is the chromosome length. The population size is set to 100. We test two different values for the number of sentinels: 8 and 16. These values are chosen for two reasons. First of all, [28] suggests working with 10% of the population as sentinels. Secondly, in [78], we experiment with storing $M = 8$ and $M = 16$ probability vectors in $\overrightarrow{P}list$ for HH-EDA2 and found $M = 8$ to be better. At the beginning of the search, sentinels are initialized to locations of the masks representing different parts of the search space. For HH-EDA2, the masks used in the offline learning stage are chosen in such a way as to ensure that they are distributed uniformly on the search space. Therefore $M = 8$ or $M = 16$ masks are used as the sentinels.

In Reinforcement Learning, score of each heuristic is initialized to 15 and is allowed to vary between 0 and 30. If the selected heuristic yields a solution with an improved fitness, its score is increased by 1, otherwise it is decreased by 1. The Reinforcement Learning settings are taken as recommended in [62].

In [91], the results show that Ant-based Selection with roulette wheel selection is better than the version with tournament selection. Therefore, we work Ant-based Selection with roulette wheel selection in this study. For Ant-based Selection, q_0, sd and ρ are set to 0.5, 10 and 0.1, respectively. These are the settings recommended in [91].

For each run of the algorithms, 128 changes occur after the initial environment. Therefore, the total number of generations in a run is calculated as $maxGenerations = changeFrequency * changeCount$.

To compare the performance of approaches over different dynamic environments, the approaches are scored in the same way as in the CHeSC competition [111]. Considering random and cyclic environments, there are 117 problem instances, therefore, 1170 is the maximum overall score that an algorithm can get in this scoring system.

5.1.2.2 Results

In this subsection, we provide and discuss the results of each group of experiments separately.

Comparison of heuristic selection methods

In this set of experiments, we test different heuristic selection methods within the proposed framework. The tested heuristic selection methods are Simple Random, Random Descent, Random Permutation, Random Permutation Descent, Reinforcement Learning and Ant-based Selection. We use all change frequency and severity settings for the Random dynamic environments; we also use all change frequency and cycle length settings for the Cyclic1, Cylic1-with-Noise and Cyclic2 type dynamic environments. Tests are performed on all DUFs, i.e. DUF1, DUF2 and DUF3.

Table 5.3 summarizes the results generated by different heuristic selection methods averaged over 100 runs, on all DUFs for different change severity and frequency settings in randomly changing environments. The results show that all heuristic selection schemes performed well and there were no statistically significant differences between the results for most cases. However, Reinforcement Learning performs the best as a heuristic selection method for high frequency in DUF3.

In the tested cyclic environments, the results for DUF1, DUF2 and DUF3 are provided in Tables 5.4, 5.5 and 5.6, respectively. The results show that for DUF1 and DUF2, in the tested cyclic environments, Random Permutation performs the best as a heuristic selection method in the HH-EDA2 framework. For DUF3, Random Permutation

Table 5.3 : Offline errors generated by different heuristic selection methods averaged over 100 runs, on all DUFs for different change severity and frequency settings in randomly changing environments.

Heuristic	*LF*				*MF*				*HF*			
Selection	*LS*	*MS*	*HS*	*VHS*	*LS*	*MS*	*HS*	*VHS*	*LS*	*MS*	*HS*	*VHS*
						DUF1						
RD	**0.06**	**0.06**	**0.08**	0.09	**0.17**	0.26	0.89	1.05	22.00	23.62	26.82	28.40
RL	**0.06**	**0.06**	**0.08**	0.09	**0.17**	0.26	0.89	1.07	21.95	23.65	26.82	28.41
RP	**0.06**	**0.06**	**0.08**	0.09	**0.17**	**0.25**	**0.86**	**0.99**	**21.94**	**23.60**	26.79	28.26
RPD	**0.06**	**0.06**	**0.08**	0.09	**0.17**	0.26	0.89	1.07	22.00	23.61	26.83	28.39
AbS	**0.06**	**0.06**	**0.08**	0.09	**0.17**	0.26	0.87	1.02	21.98	23.65	26.79	28.31
SR	**0.06**	**0.06**	**0.08**	**0.08**	**0.17**	0.26	**0.86**	1.00	21.95	23.61	**26.78**	**28.30**
						DUF2						
RD	**0.12**	**0.15**	0.54	0.59	0.43	0.85	4.30	4.93	42.92	45.78	50.92	53.14
RL	0.13	0.16	0.56	0.61	**0.42**	**0.83**	4.34	4.98	**42.82**	45.87	50.94	53.27
RP	**0.12**	0.16	**0.49**	**0.53**	0.43	0.85	**4.13**	**4.54**	42.92	**45.74**	**50.86**	**52.95**
RPD	**0.12**	0.16	0.55	0.60	**0.42**	0.85	4.39	4.92	42.95	45.80	50.99	53.12
AbS	0.13	0.16	0.51	0.55	**0.42**	0.87	4.17	4.70	42.88	45.79	50.92	53.06
SR	**0.12**	**0.15**	0.50	0.54	**0.42**	0.86	4.16	4.64	42.92	45.80	50.93	53.00
						DUF3						
RD	19.22	18.29	16.03	14.20	**19.62**	18.96	17.26	15.51	38.33	39.60	40.66	40.25
RL	**19.12**	**18.23**	16.06	14.22	19.63	18.89	17.26	15.50	**38.19**	**39.47**	**40.57**	**39.96**
RP	19.44	18.46	16.04	**14.18**	19.75	18.99	17.26	**15.49**	38.44	39.99	41.29	40.75
RPD	19.32	18.26	**16.03**	14.20	19.63	**18.86**	17.29	15.55	38.37	39.75	40.87	40.27
AbS	19.21	18.35	16.05	**14.18**	19.69	18.90	**17.25**	15.52	38.37	39.81	41.02	40.45
SR	19.45	18.44	16.06	**14.18**	19.78	18.99	**17.25**	15.51	38.35	39.81	41.07	40.43

Descent seems to produce better results than Random Permutation, however this performance difference is not statistically significant and actual offline error values from Random Permutation are close to the ones produced by Random Permutation Descent.

AbS delivers a promising performance for all DUFs in randomly changing environments and the tested cyclically changing environments. However, it performs the best on all DUFs for Cyclic1 with noise when the changes occur at a high frequency and the cycle length is low (2). $CL = 2$ means that the change repeats between two environments. *AbS* acts similar to a memory scheme for this case. It is able to select the most appropriate the probability vector (serve as the low-level heuristic) to sample the population at each step.

We perform statistical significance tests to determine the best heuristic selection method. The statistical comparison summary is given in Table 5.7. As can

Table 5.4 : Offline errors generated by different approaches averaged over 100 runs, on the DUF1 for different cycle length and change frequency settings in different cyclic dynamic environments.

Heuristic	*LF*			*MF*			*HF*		
Selection	*CL=2*	*CL=4*	*CL=8*	*CL=2*	*CL=4*	*CL=8*	*CL=2*	*CL=4*	*CL=8*
					Cylic1				
RD	0.04	0.04	0.04	0.15	0.13	0.14	15.76	15.67	15.77
RL	0.04	0.04	0.04	0.21	0.16	0.19	16.02	17.18	16.79
RP	**0.03**	**0.02**	**0.02**	**0.05**	**0.04**	**0.05**	14.20	13.82	**14.59**
RPD	**0.03**	0.03	0.03	0.07	0.06	0.06	14.60	14.51	14.65
AbS	0.04	0.04	0.04	0.11	0.39	0.43	**4.12**	**12.62**	18.48
SR	**0.03**	0.03	0.03	0.09	0.08	0.08	15.01	14.89	15.27
					Cylic1-with-Noise				
RD	0.04	0.04	0.04	0.16	0.12	0.13	15.88	15.59	15.94
RL	0.03	0.04	0.04	0.21	0.16	0.18	15.74	17.33	17.07
RP	**0.02**	**0.02**	**0.02**	**0.05**	**0.04**	**0.05**	14.48	13.86	**14.66**
RPD	0.03	0.03	0.03	0.07	0.07	0.06	14.72	14.87	14.74
AbS	0.03	0.04	0.04	0.12	0.40	0.42	**4.27**	**12.34**	18.54
SR	0.03	0.03	0.03	0.09	0.08	0.09	15.14	15.08	15.42
					Cylic2				
RD	**0.08**	**0.08**	**0.08**	0.90	0.88	0.90	25.86	26.83	27.00
RL	**0.08**	**0.08**	**0.08**	0.90	0.88	0.91	25.85	26.85	27.00
RP	**0.08**	**0.08**	**0.08**	0.85	**0.86**	0.89	**25.83**	26.80	26.98
RPD	**0.08**	**0.08**	**0.08**	0.90	0.89	0.90	25.84	26.80	26.99
AbS	**0.08**	**0.08**	**0.08**	**0.82**	**0.86**	**0.85**	25.87	26.80	**26.96**
SR	**0.08**	**0.08**	**0.08**	0.87	**0.86**	0.88	25.86	**26.79**	26.97

be seen, Random Permutation generates the best average performance across all dynamic environment problems, performing significantly/slightly better than the rest for 238/195 instances. The second best approach is Random Permutation Descent on average.

Table 5.8 shows the ranking results obtained based on median, best and average offline error values. Random Permutation is still the best approach if the median and best performances are considered as well (Table 5.8) based on the Formula 1 ranking. It can be seen from the table that Random Permutation scores 925 and 905, respectively. Learning via the PBIL process helps, but using an additional learning mechanism on top of that turns out to be misleading for the search process. For example, the use of reinforcement learning in the selection hyper-heuristic (Reinforcement Learning) yields the worst average performance. Random Permutation as a non-learning heuristic

Table 5.5 : Offline errors generated by different approaches averaged over 100 runs, on the DUF2 for different cycle length and change frequency settings in different cyclic dynamic environments.

Heuristic	*LF*			*MF*			*HF*		
Selection	*CL=2*	*CL=4*	*CL=8*	*CL=2*	*CL=4*	*CL=8*	*CL=2*	*CL=4*	*CL=8*
				Cylic1					
RD	0.07	0.08	0.07	0.43	0.38	0.43	29.59	29.83	29.45
RL	0.07	0.07	0.07	0.60	0.49	0.55	30.37	32.35	30.77
RP	**0.04**	**0.04**	**0.04**	**0.09**	**0.08**	**0.08**	27.33	27.38	**26.53**
RPD	0.06	0.06	0.05	0.11	0.10	0.11	27.61	28.06	26.96
AbS	0.07	0.13	0.19	0.25	1.84	1.95	**7.71**	**24.04**	35.37
SR	0.06	0.05	0.05	0.23	0.20	0.20	28.77	29.16	29.38
				Cylic1-with-Noise					
RD	0.07	0.07	0.07	0.43	0.33	0.43	29.40	29.75	29.51
RL	0.07	0.08	0.08	0.57	0.51	0.57	29.64	32.79	30.83
RP	**0.04**	**0.04**	**0.05**	**0.08**	**0.09**	**0.09**	26.96	26.37	27.34
RPD	0.06	0.06	**0.05**	0.11	0.11	0.11	27.68	28.57	**27.26**
AbS	0.07	0.12	0.19	0.26	1.63	1.91	**7.38**	**24.10**	34.95
SR	0.06	0.06	0.06	0.24	0.18	0.22	28.77	29.15	29.45
				Cylic2					
RD	0.49	0.51	0.53	4.09	4.21	4.36	49.36	50.87	51.21
RL	0.49	0.52	0.51	4.16	4.24	4.38	49.39	50.93	51.27
RP	**0.45**	**0.46**	0.51	**3.93**	4.06	4.25	49.34	**50.82**	51.20
RPD	0.48	0.52	0.53	4.07	4.27	4.37	49.36	50.91	51.28
AbS	0.48	0.47	**0.50**	4.01	**4.02**	**4.22**	49.40	**50.82**	**51.16**
SR	0.46	0.49	0.52	3.98	4.08	**4.22**	**49.31**	50.90	51.22

selection combines the learnt probability vectors effectively yielding an improved performance which outperforms Simple Random.

Comparisons to selected approaches from literature

In this set of experiments, we compare the proposed approach to some well known and successful previously proposed approaches from literature. As a result of the first group of experiments, we fix the heuristic selection component as Random Permutation during these experiments and used the same problems, change settings and dynamic environment types.

An overall comparison of all approaches are provided in Tables 5.9 and 5.10. HH-EDA2 generates the best average performance across all dynamic environment problems (Table 5.9) performing significantly/slightly better than the rest for 578/42 instances. The second best approach is PBIL using a single population and restart.

Table 5.6 : Offline errors generated by different approaches averaged over 100 runs, on the DUF3 for different cycle length and change frequency settings in different cyclic dynamic environments.

Heuristic	*LF*			*MF*			*HF*		
Selection	*CL=2*	*CL=4*	*CL=8*	*CL=2*	*CL=4*	*CL=8*	*CL=2*	*CL=4*	*CL=8*
					Cylic1				
RD	10.22	11.36	11.36	11.22	12.14	12.14	23.49	24.49	23.63
RL	10.44	11.49	11.50	12.04	12.98	12.89	23.99	28.69	26.86
RP	**10.09**	11.36	11.33	**10.35**	11.60	11.58	22.23	22.42	22.76
RPD	10.11	**11.33**	**11.31**	10.36	**11.51**	**11.48**	21.36	22.08	**21.67**
AbS	10.07	11.35	11.44	10.20	11.87	12.75	**13.97**	**20.38**	25.30
SR	10.16	11.37	11.35	11.11	12.10	12.15	24.32	24.02	24.47
					Cylic1-with-Noise				
RD	10.21	11.37	11.37	11.24	12.18	12.14	23.31	24.33	23.67
RL	10.43	11.50	11.50	12.02	13.02	12.80	23.88	28.60	26.87
RP	10.09	11.35	11.34	10.35	11.59	11.59	22.21	23.20	23.20
RPD	10.11	**11.33**	**11.32**	10.35	**11.51**	**11.50**	21.20	22.49	**21.67**
AbS	**10.07**	11.35	11.45	**10.21**	11.89	12.74	**13.98**	**20.35**	25.30
SR	10.16	11.37	11.35	11.14	12.08	12.17	24.20	24.11	24.23
					Cylic2				
RD	16.04	**16.55**	16.11	17.38	17.69	17.26	40.65	40.77	**40.64**
RL	16.00	16.62	16.11	17.31	17.75	17.26	40.68	**40.70**	40.65
RP	16.27	16.60	**16.02**	17.47	17.73	17.24	40.67	41.19	41.34
RPD	16.05	16.59	16.11	17.41	17.72	17.25	40.65	40.86	40.79
AbS	**15.84**	16.63	16.14	**17.22**	17.73	17.22	40.77	40.78	41.05
SR	16.20	**16.55**	16.05	17.39	**17.67**	**17.19**	**40.63**	41.04	41.17

Moreover, HH-EDA2 is the top approach if the median and best performances are considered as well (see Table 5.10) based on Formula 1 rankings, scoring 1020 and 995, respectively. The closest competitor accumulates a score of 725 and 649 for its median and best performances, respectively. These results also indicate that the use of a dual population and the selection hyper-heuristic both improves the performance of the overall algorithm. Based on the results, the first population using PBIL serves as the search component, while the second population using the hyper-heuristic acts as a memory in cyclic environments and as a source of diversity in randomly changing environments.

5.1.2.3 Discussion

The empirical results show that the selection scheme that relies on a fixed permutation of the underlying low-level heuristics (Random Permutation) is the most successful one. For the cases when the change period is long enough to allow all the vectors

Table 5.7 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for the different heuristic selection schemes.

Heuristic Selection	$s+$	$s-$	$\geq$	$\leq$
RP	238	65	195	87
RPD	185	70	130	200
AbS	156	152	132	145
SR	135	122	204	124
RD	84	189	148	164
RL	51	251	97	186

Table 5.8 : The overall score according to the Formula 1 ranking based on median, best and average offline error values for the different heuristic selection schemes.

Heuristic Selection	Median	Best	Average
RP	925	905	909
SR	743	691	738
RPD	731	744	730
AbS	677	698	707
RD	606	614	602
RL	530	560	526

in the permutation to be applied at least once, the Random Permutation heuristic selection mechanism becomes equivalent to *Greedy Selection*. In HH-EDA2, the move acceptance stage of a hyper-heuristic is not used. This is the same as using the *Accept All Moves* strategy. This move acceptance scheme is known to perform the best with the *Greedy Selection* method [70].

The overall results also reveal that HH-EDA2 is capable of adapting itself to the changes quickly whether the change is random or cyclic. HH-EDA2 outperforms well

Table 5.9 : Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for the algorithms used.

Algorithm	$s+$	$s-$	$\geq$	$\leq$
HH-EDA2	578	69	42	13
PBILr	400	232	37	33
PBIL2r	343	310	11	38
MPBILr	262	394	28	18
MPBIL2r	251	405	16	30
Sentinel16	242	442	4	14
Sentinel8	229	453	14	6

Table 5.10 : The overall score according to the Formula 1 ranking based on median, best and average offline error values for the algorithms used.

Algorithm	Median	Best	Average
HH-EDA2	1020	995	1020
PBILr	725	649	725
PBIL2r	594	531	594
MPBILr	551	521	550
Sentinel8	527	523	527
MPBIL2r	517	735	518
Sentinel16	512	492	512

know approaches from literature for almost all cases and ranks performance-wise the first among all others.

5.2 Application II: An Implementation on HyFlex

In this section, the proposed selection hyper-heuristic is implemented on HyFlex (Hyper-heuristics Flexible framework). HyFlex provides a number of stationary optimization problems (details are given in Subsection 2.2.3). Therefore, the performance of all variants of the proposed approach, namely *AbSrw*, *AbSts* and *MM AbS*, are explored on stationary optimization problems. These selection mechanisms are also used together with the Improving-and-Equal acceptance technique.

5.2.1 Experimental design

In this thesis, we perform experiments with the proposed approach for six problem domains provided in HyFlex framework, namely maximum satisfiability, one-dimensional bin packing, personnel scheduling, permutation flow shop, the traveling salesman problem and the vehicle routing problem. HyFlex provides a number of instances for each problem domain. In this study, we consider the same 5 instances used in CHESC 2011 competition for each problem domains for a fair comparison. For each problem domain, the crossover heuristics are not used. Each run is repeated 31 times for each setting and is executed 323 seconds running time which is the time in our computer that corresponds to 600 secs on the computer that is used for the competition machine.

The parameters of the proposed Ant-based selection scheme are chosen as recommended in Chapter 4. Each entry in the pheromone matrix is initialized to

$\tau_0 = 1/f_s$ where f_s is the fitness value of initial solution. For all approaches, ρ, q_0 and *sd* are set to 0.1, 0.5 and 10, respectively. For *AbSts*, we let the tournament size to be determined randomly with equal probability from among the five pre-determined tournament size levels: $k = \{2,3,4,5,6\}$. For *MM AbS*, the lower and upper bound are set to $\tau_0 * 50$ and $\tau_0/50$ where τ_0 is the initial value of the pheromone trails.

5.2.2 Results and discussion

Table 5.11 shows the overall score of *AbS* variants among the competing hyper-heuristics in CHeSC2011 according to the Formula 1 ranking based on median value. Considering all problem domains and instances, there are 30 different problems. Therefore, 300 is the maximum overall score an algorithm can get.

As can be seen from the results, *AbSrw* ranks 13th out of 23 algorithms overall with the score of 28, *MM AbS* gets the score of 25 ranking 14th overall, and *AbSts* ranks 22th out of 23 algorithms overall with the score of 0. The proposed method has a number of parameters and the performance of the proposed heuristic selection method is sensitive to the initial setting of those parameters for stationary optimization problems.

Table 5.11 : The overall Formula 1 scores of our approaches compared to competing hyper-heuristics in CHeSC2011.

Rank	Algorithm	Score	Rank	Algorithm	Score
1	AdapHH [92]	178	**13**	***AbSrw***	28
2	VNS-TW [93]	132	**14**	***MM AbS***	25
3	ML [94]	125.5	15	SA-ILS	22.25
4	PHUNTER [95]	93.25	16	DynILS	22
5	EPH [96]	84.75	17	AVEG-Nep [97]	21
6	NAHH [98]	75	18	XCJ	18.5
7	HAHA [99]	74.75	19	GISS [100]	16.75
8	ISEA [101]	65	20	SelfSearch [102]	6
9	KSATS-HH [103]	59.5	21	MCHH-S [104]	4.75
10	HAEA [105]	50.5	**22**	***AbSts***	0
11	ACO-HH [106]	37	23	Ant-Q [107]	0
12	GenHive [108]	30.5			

Table 5.12 presents the score of the *AbS* variants across six problem domains. It can be seen that, *AbSrw* and *MM AbS* get the scores from three problem domains, namely Bin Packing, Personnel Scheduling , and Vehicle Routing Problem. For Bin Packing, *MM AbS* and *AbSrw* rank 3nd and 6th, respectively. *AbSts* gets zero point for all problem domain. *AbS* with roulette wheel gives better performance when compared to *AbS* with tournament selection.

Table 5.12 : The overall Formula 1 scores of *AbS* variants for six problem domains.

Algorithm	BP	MAX-SAT	FS	PS	TSP	VRP	Overall
AbSrw	18	0	0	6	0	4	28
MM AbS	20	0	0	2	0	3	25
AbSts	0	0	0	0	0	0	0

5.3 Application III: Dynamic Traveling Salesman Problem

Benchmark generators are important research tools for creating problem instances which enabled us to control the characteristics of those instances in a given domain. These problem instances are mainly used for performance comparisons of different algorithms. In the experiment, we use the Moving Peaks Benchmark and XOR dynamic problem generator to test our approaches. On the other hand, real-world problem instances could still vary from the artificially generated instances. Testing an algorithm on the artificial instances might not reflect the actual performance of a given algorithm in a real-world setting. Hence, in this study, we also investigate the performance of our approaches, Ant-based selection, on a real-world instance of a problem. We use the Dynamic Traveling Salesman Problem (DTSP) as a real-world problem. DTSP has been mostly studied permutation-encoded problem in dynamic environments. In addition, classic Traveling salesman was implemented on HyFlex. There are are many variants of DTSP. In this thesis, we consider Dynamic Traveling Salesman Problem with traffic factor proposed in [18] (see Subsection 2.1.1.3). In this problem, the costs of a number of edges are changed at every Δe iterations as given in Equation 2.14.

5.3.1 Comparisons of selection hyper-heuristics

DTSP is implemented on HyFlex interface. The implementation of DTSP is based on that of Traveling Salesman Problem (TSP) in HyFlex. We use the same initialization method to generate an initial solution. A candidate solution is represented by a permutation of the cities which represents a complete tour. To generate initial candidate solution, the greedy heuristic in which a solution is constructed in an incremental way is used. It starts from a randomly selected city. Then, it chooses the closest among the remaining cities and it adds to the solution until a complete solution is generated.

We consider the same low-level heuristics implemented for TSP in HyFlex. There are 13 low-level heuristics across the four categories for TSP. These heuristics are described as follows:

Mutational heuristics

h_1: A randomly selected city is reinserted into a randomly selected place in the permutation. Then, the rest of the cities are shifted as required.

h_2: Two randomly selected cities are swapped.

h_3: The permutation is randomly shuffled .

h_4: A number of randomly selected cities are shuffled. Here, the number of cities to shuffle is determined by the mutation density.

h_5: A number of edges is selected and substituted with randomly selected ones. The number of edge is determined by the mutation density.

Ruin and recreate heuristics

h_6: A number of cities in the permutation are removed and reinserted using greedy procedure.

Local search heuristics

h_7: This heuristic is the *2-opt* local search that accepts the first improvement.

h_8: This heuristic is the *2-opt* local search that accepts the best improvement.

h_9: This heuristic is the *3-opt* local search that accepts the first improvement.

Crossover heuristics

h_{10}: Order Crossover [81]

h_{11}: Partially mapped crossover [81]

h_{12}: Precedence preservative crossover [109]

h_{13}: One-point crossover [81]

In the experiments, the mutation density and the depth of hill-climbing are set to 0.2.

In this study, we experiment with Max-Min Ant-based selection with roulette wheel (*MM AbS*) and Ant-based selection with roulette wheel (*AbS*) combined with Improving and Equal. For both approaches, q_0 and sd are set to 0.5 and 10, respectively. Each entry in the pheromone matrix is initialized to $\tau_0 = 1/f_s$ where f_s is the fitness value of initial solution. ρ is set to 0.1. For *MM AbS*, the lower and upper bound are set to $\tau_0 * 50$ and $\tau_0/50$ where τ_0 is the initial value of the pheromone trails.

For all methods, the selection probability of each low-level heuristic are the same at the beginning of the search. To manage the crossover operators, the five randomly initialized solution are stored in a memory. If the selected heuristic is a crossover operator, a solution is selected randomly from this memory. Then, the crossover operator use the current solution and the selected solution to generate one offspring. Whenever the best-so-far solution is changed, the randomly selected solution is replaced with the best-so-far solution.

The performance of our approach is compared to state of the art selection hyper-heuristic, namely learning heuristic selection method with adaptive dynamic heuristic set combined with adaptive iteration limited list-based threshold accepting [92] (AdapHH). AdapHH is chosen since it is the winner of the CHeSC2011 competition. It also ranks first for Traveling Salesman Problem.

AdapHH include an adaptive heuristic subset selection, a pairwise heuristic hybridization method and adaptive parameter setting of low-level heuristics. The adaptive dynamic heuristic set strategy adaptively determines the best heuristic subset at each phase composed of specific number of iterations. This method can eliminate the heuristics performing the worse and keep the best ones according to quality index. A weighted sum of different performance metrics is used to compute the quality index for each heuristic. Some of these performance metrics include the number of new best solution, the total fitness improvement and worsening during the run and a phase, the time spent and the remaining time. If the quality index of a heuristic is less than the average of the quality indexes of all heuristic, the heuristic is excluded from the heuristic subset. This method also uses Tabu list to store the number of phases, called tabu duration, in which a heuristic is consecutively excluded. Whenever the tabu duration reaches its upper bound, this heuristic is permanently excluded. At each step, an appropriate heuristic is selected from the heuristic subset with a selection probability. A relay hybridization method is also used to determine effective pairs of heuristics that are applied successively. In addition, the parameters of low-level heuristics, namely mutation density and depth of hill-climbing, are dynamically adapted using reinforcement learning. Adaptive iteration limited list-based threshold accepting method accepts the worsening solution according to the fitness values of the previous best solutions which is used as the threshold value. If it does not explore new best solution within adaptively adjusted number of steps, the higher value from the list is used as the threshold value. In this study, we use the same settings as recommended in [92]. We include two additional variants of AdapHH to deal with the dynamism. In the first variant, denoted as AdapHH-I, a new initial solution is randomly generated whenever a change occurs in the environment. In the second variant, denoted as AdapHH-E, the current solution is re-evaluated when a change occurs.

We experiment with random DTSP. To generate dynamic environments showing different dynamism properties, we consider different change frequencies and change severities. For both types of DTSP, we determine the change periods which correspond to low frequency (LF), medium frequency (MF) and high frequency (HF) changes as a result of some preliminary experiments where we executed Simple Random - Improving and Equal on stationary versions of kroA150 instance. Based on the resultant convergence behavior, we determine the change period to be approximately 2.91 secs for low frequency (LF), 0.48 secs for medium frequency, and 0.06 secs for high frequency. Moreover, the severity of changes are controlled by m in DTSP and chosen as 0.1 for low severity (LS), 0.2 for medium severity (MS) and 0.5 for high severity (HS). The lower and upper bounds of traffic factor are set to $R_L = 0$ and $R_U = 5$.

To generate the dynamic instances of DTSP, we use four stationary TSP instances from TSPLIB [113], namely kroA100, kroA150, and kroA200 which are used in [18] and u2152 provided in HyFlex.

All trials are repeated for 31 times using each approach for each test case. The algorithms are executed 323 seconds running time which is the time in our computer that corresponds to 600 secs on the computer that is used for the CHeSC competition. The performance of the algorithms is compared based on the offline performance (see Equation 2.17). Here, we take into account the total number of iterations to calculate the offline performance instead of the evaluation counters. The performances of the approaches are compared under a variety of change frequency-severity pair settings under random environments.

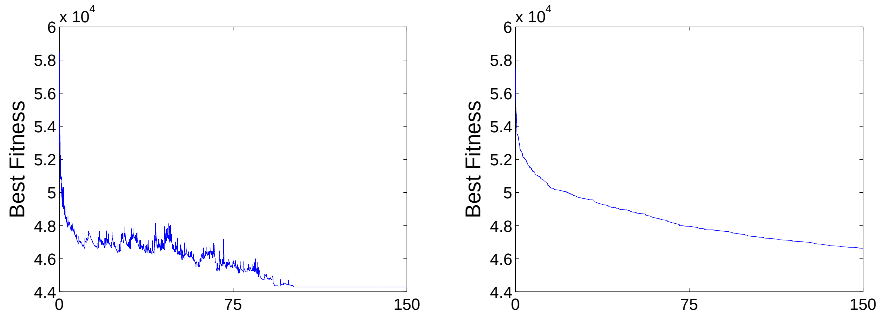
The results are provided in terms of average offline performance values in the tables. The performances of the algorithms are compared under a variety of change frequency-severity pair settings for random DTSP. In the tables, the best performing approach is marked in bold.

5.3.1.1 Results

Table 5.13 summarizes the average offline performance generated by AdapHH, AdapHH-I, AdapHH-E, *MM AbS*, and *AbS* on kroA100 for random DTSP. The performance of all approaches degrades as the change frequency increases. The performance of all approaches also degrades as the change severity increases. Moreover, all algorithms seem to be more affected from the increase in change severity. *AbS* is the best performing approach for both low frequency-medium severity setting and medium frequency-medium severity setting. When compare *AbS* and *MM AbS* with AdapHH variants, they give comparable results for most cases except for high severity. For high severity, AdapHH-I performs the best. In dynamic environments, the restart of the process after a change is more useful when the change is too severe. In AdapHH-I, the current solution is re-initialized whenever a change occurs. Therefore, AdapHH-I delivers the best average performance for high severity

Table 5.14 and 5.15 show the average offline performance generated by AdapHH, AdapHH-I, AdapHH-E, *MM AbS*, and *AbS* on kroA150, and kroA200 for random DTSP, respectively. Similar phenomena as on kroA100 are observed for these instances. The methods deteriorate in performance as the change frequency and severity increase. However, *AbS* does not perform the best in any of the change frequency-severity settings. When compare *AbS* and *MM AbS* with AdapHH variants, the results are very close for low and medium severity. For AdapHH, the re-initialization (AdapHH-I) improves its performance especially for high severity. In addition, *AbS* is slightly better than AdapHH and AdapHH-E for most cases. We also experiment with the large instance of DTSP. Table 5.16 shows the average offline performance for random DTSP for the instance with 2152 cities. *AbS* and *MM AbS* give comparable results for low and medium frequency. In addition, AdapHH-I performs the best for all change frequency and severity settings.

To compare the performance of AdapHH and *AbS*, we allow AdapHH and AbS to run for long periods without any change in the environment. Figure 5.2 illustrates the convergence behavior of AdapHH and *AbS* on the stationary version of kroA200. As seen in the figure, for *AbS*, the improvement continues gradually and it has not yet fully converged at the end of the process. However, AdapHH has been converged in approximately 80 seconds. Moreover, the better improvement has been observed for AdapHH. As a result, AdapHH obtains the better solution more quickly than *AbS*.



(a) AdapHH (b) *AbS*

Figure 5.2 : A sample plot of the fitness values of the best candidate solutions versus time for (a) AdapHH and (b) *AbS*.

To manage the crossover operator, we also include a second version. In this version, the solution memory is implemented as a queue. If the selected heuristic is a crossover operator, the head of queue is taken and placed at the tail of the queue. Whenever the best-so-far solution is changed, the first added to the queue is replaced with the best-so-far solution. When compared to the first version, this strategy slightly improves the performance of the algorithm, however, the results of these two strategies are close for all instances. Although this strategy improves the performance of the method, it is still outperformed by AdapHH-I for high severity.

Table 5.13 : Offline performance generated by different approaches averaged over 31 runs, on the kroA100 for random DTSP.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
AdapHH	23275.81	30116.53	52625.51	23711.82	32464.97	59017.32	24718.20	34972.34	69343.56
AdapHH-I	23110.09	28236.67	**35494.11**	23746.47	29492.83	**37102.84**	23961.29	**30241.23**	**38423.33**
AdapHH-E	**23039.82**	28690.12	44939.58	**23576.88**	30723.64	52841.19	**23916.42**	31606.82	62549.62
MMAbS	23335.19	28079.42	48781.61	23741.00	29086.39	51990.71	24426.87	30589.97	56534.80
AbS	23325.59	**28052.71**	48754.19	23738.94	**29056.41**	52036.54	24431.39	30583.61	56525.44

Table 5.14 : Offline performance generated by different approaches averaged over 31 runs, on the kroA150 for random DTSP.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
AdapHH	29599.83	39008.22	70978.70	30188.39	41851.82	79279.69	31576.96	45941.80	91501.08
AdapHH-I	**29188.16**	**35740.94**	**43897.20**	29877.92	**37214.28**	**45496.29**	**30129.97**	**38412.14**	**47254.63**
AdapHH-E	29248.07	37345.92	71487.01	**29834.54**	39685.22	80583.15	30319.14	41155.93	91219.25
MMAbS	29564.04	36539.80	67724.77	30036.12	37681.16	71778.99	30836.27	39628.81	77725.15
AbS	29557.42	36523.53	67729.86	30039.35	37657.06	71782.59	30841.55	39636.08	77704.55

Table 5.15 : Offline performance generated by different approaches averaged over 31 runs, on the kroA200 for random DTSP.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
AdapHH	33459.83	45227.76	83942.79	34285.63	47415.36	90903.18	35345.42	52427.44	102564.33
AdapHH-I	32941.82	**41348.27**	**50760.89**	**33316.29**	**42688.63**	**51911.74**	34242.69	**43949.12**	**53661.87**
AdapHH-E	**32892.08**	43245.42	83330.11	33380.21	45375.80	93502.31	**34194.92**	47561.57	104836.04
MMAbS	33235.58	42112.43	80709.44	33767.79	43402.95	85048.23	34687.21	45705.47	92081.09
AbS	33218.15	42091.78	80661.30	33766.63	43409.02	85084.60	34687.32	45696.52	92064.20

Table 5.16 : Offline performance generated by different approaches averaged over 31 runs, on the u2152 for random DTSP.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
AdapHH	83478.30	115952.09	227902.88	83343.41	114015.04	221514.78	82604.42	110926.79	202104.03
AdapHH-I	**82000.39**	**99287.51**	**122620.92**	**82229.37**	**99096.48**	**122119.15**	**81691.60**	**98423.85**	**121557.53**
AdapHH-E	82483.81	112020.69	212799.09	82426.76	109351.28	209712.07	82017.43	108356.74	205885.35
MMAbS	83590.96	111051.77	231799.33	83288.48	110279.16	222559.07	82941.03	107345.38	182491.93
AbS	83531.15	111114.30	231867.10	83292.49	110321.52	223206.21	82921.95	107458.24	182372.33

5.3.2 Comparisons to problem specific approaches

In this set of experiments, we compare the proposed approach to some well known and successful previously proposed problem specific approaches from literature, namely Random Immigrants Ant Colony Optimization, Elitism-based Immigrants Ant Colony Optimization, and Memory-based Immigrants Ant Colony Optimization [18].

The ACO algorithms with immigrants are inspired from population-based ACO (P-ACO) which has the long-term memory storing the best ant at every iteration [18]. The pheromone trails are generated according to ants stored in the memory and the pheromone evaporation is not included. However, the ACO algorithms with immigrants use the short-term memory instead of long-term memory in P-ACO. Short-term memory stores a number of best ants of the current iteration. Then, the worst ants in the memory are replaced by a number of immigrants. The solution is constructed in the same way as traditional ACO, however, the pheromone trail values are updated according to ants in short-term memory. There are three variants of ACO with immigrants, namely Random Immigrants Ant Colony Optimization (RIACO), Elitism-based Immigrants Ant Colony Optimization (EIACO), and Memory-based Immigrants Ant Colony Optimization (MIACO). In RIACO, the immigrants are randomly generated. In EIACO, the elitism-based immigrants are generated based on the best (elite) ant from previous environment using inver-over operations in which the segment between two cities are reversed. MIACO uses both short-term and long-term memories. The ants in the long-term memory are initialized randomly and updated as follows: If there are randomly generated ants in the memory, any one of the randomly initialized ants is replaced with the best so far; otherwise, the closest ant in the memory is replaced with best so far if it is worse than the best so far. In this method, the immigrants are generated based on the ants in long-term memory.

The implementations of these algorithms [112] are adapted to use the corresponding methods in the implementation of DTSP. The settings of ACO algorithms with immigrants are taken as recommended in [18]. The parameter settings are given in Table 5.17. In this table, K_s and r the short-term memory size and the migration replacement rate, respectively. In MIACO, the long-term memory size is set to 3. For EIACO and MIACO, the immigrants mutation probability is set to 0.02.

Table 5.17 : Parameter settings for ACO with immigrants.

Parameters	RIACO	EIACO	MIACO
$\#of ants$	28	28	25
q_0	0.0	0.0	0.0
α	1	1	1
β	5	5	5
K_s	6	6	6
r	0.0	0.4	0.4

As a result of the first group of experiments, we consider *AbS* and AdapHH-I during these experiments and used the three instances, namely kroA100, kroA150 and kroA200, change settings and dynamic environment type (random DTSP). For this set of experiments, we also consider cyclic DTSP since MIACO is proposed for this type of DTSP.

5.3.2.1 Results

Table 5.18 shows the average offline performance generated by AdapHH-I, *AbS*, RIACO, EIACO and MIACO on the kroA100 for random and cyclic DTSPs. EIACO performs the best for the most cases for random DTSP. AdapHH-I and *AbS* are outperformed by ACO algorithms with immigrants for most cases. This is expected since ACO with immigrants use problem specific information. For cyclic DTSP, MIACO delivers the best performance for all cases. This is because MIACO uses the memory that stores the best solutions in previously visited environments and reuses them to generate memory-based immigrants.

Table 5.19 and 5.20 show the average offline performance generated by AdapHH-I, *AbS*, RIACO, EIACO and MIACO on the kroA150 and kroA200 for random and cyclic DTSPs, respectively. Similar results are observed as on kroA100. EIACO and AdapHH-I deliver good performance for random DTSP and MIACO performs the best for most cases for cyclic DTSP.

Overall, *AbS* gives comparable results when compared with AdapHH and AdapHH-E for most frequency-severity settings. AdapHH-I is better than *AbS* for most cases, especially for high severity. If the change is too severe, the restart of process is more useful. AdapHH-I re-initializes the current solution whenever the environment changes. Therefore, it is aware of time when a change occurs and acts on this. However, *AbS* does not require any special actions when a change occurs. AdapHH is implemented on HyFlex. It adapts the parameters of mutation and hill-climber heuristics and re-initializes the current solution in some conditions. It can not be used in our hybrid methods (see Section 5.1) without requiring any modifications.

Table 5.18 : Offline performance generated by different approaches averaged over 31 runs, on the kroA100 for random and cyclic DTSP.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
DTSPs with random traffic factor									
AdapHH-I	23110.09	28236.67	35494.11	23746.47	29492.83	37102.84	**23961.29**	30241.23	38423.33
AbS	23325.59	28052.71	48754.19	23738.94	29056.41	52036.54	24431.39	30583.61	56525.44
RIACO	23357.06	26049.75	32087.11	23852.51	26771.10	33237.98	24824.71	28230.79	35362.84
EIACO	**23082.23**	**25672.55**	**31404.28**	**23617.37**	**26510.10**	**32687.57**	24656.16	**28129.19**	**35146.06**
MIACO	23115.52	25713.59	31495.59	23640.65	26538.45	32811.07	24674.43	28175.58	35277.05
DTSPs with cyclic traffic factor									
AdapHH-I	22909.35	27721.32	34657.29	23465.21	28741.21	35879.40	23907.82	30125.22	37475.67
AbS	23239.53	28370.84	48511.20	23728.43	29105.11	52480.95	24358.19	30797.25	56643.11
RIACO	23314.76	26103.29	32045.26	23854.88	26738.92	33295.03	24686.47	28212.18	35352.59
EIACO	22988.93	25717.28	31390.09	23555.15	26462.83	32733.18	24333.77	28072.87	35133.65
MIACO	**22814.76**	**25182.02**	**30477.38**	**23202.40**	**25534.79**	**31269.48**	**23850.96**	**26606.33**	**33112.68**

Table 5.19 : Offline performance generated by different approaches averaged over 31 runs, on the kroA150 for random and cyclic DTSP.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
				DTSPs with random traffic factor					
AdapHH-I	**29188.16**	35740.94	43897.20	**29877.92**	37214.28	45496.29	**30129.97**	38412.14	47254.63
AbS	29557.42	36523.53	67729.86	30039.35	37657.06	71782.59	30841.55	39636.08	77704.55
RIACO	29539.79	33107.41	41263.44	30473.13	34436.04	43007.32	32233.05	**36874.90**	**46160.10**
EIACO	29323.74	**32747.99**	**40470.06**	30315.51	**34229.06**	**42538.69**	32230.15	36963.56	46237.52
MIACO	29376.27	32837.00	40598.01	30364.17	34315.66	42721.73	32222.26	36999.64	46365.86
				DTSPs with cyclic traffic factor					
AdapHH-I	29004.89	35037.45	42982.53	**29447.92**	36685.34	44861.42	**30087.31**	37887.95	46865.99
AbS	29559.49	36416.02	67221.52	29973.92	37755.76	71707.93	30786.97	39832.65	77712.77
RIACO	29567.62	33033.67	41273.82	30379.29	34407.25	43059.94	31935.98	36825.20	46310.00
EIACO	29334.25	32676.67	40475.63	30112.72	34200.78	42616.50	31657.12	36818.65	46334.47
MIACO	**28987.46**	**31756.56**	**38991.97**	29482.94	**32585.93**	**40253.21**	30906.28	**34952.71**	**44287.43**

Table 5.20 : Offline performance generated by different approaches averaged over 31 runs, on the kroA200 for random and cyclic DTSP.

Algorithm	*LF*			*MF*			*HF*		
	LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
DTSPs with random traffic factor									
AdapHH-I	**32941.82**	41348.27	50760.89	**33316.29**	42688.63	51911.74	**34242.69**	43949.12	**53661.87**
AbS	33218.15	42091.78	80661.30	33766.63	43409.02	85084.60	34687.32	45696.52	92064.20
RIACO	33452.27	38073.72	47998.00	34864.93	39897.63	50307.28	37256.45	**42984.75**	54275.11
EIACO	33258.23	**37711.66**	**47230.69**	34748.46	**39773.53**	**50005.37**	37358.22	43235.43	54508.59
MIACO	33296.32	37798.37	47376.31	34806.47	39893.08	50248.12	37241.84	43139.69	54560.92
DTSPs with cyclic traffic factor									
AdapHH-I	32653.77	40624.89	49644.74	**33099.91**	42383.64	51317.10	**34169.92**	43844.27	**52859.18**
AbS	33165.12	42162.62	80272.08	33682.09	43270.70	84827.87	34621.91	45681.38	92410.97
RIACO	33380.58	38080.80	47948.90	34652.64	40010.87	50472.76	36186.57	42383.64	54440.95
EIACO	33147.02	37684.23	47134.78	34361.02	39829.61	50141.15	35937.63	42511.39	54614.21
MIACO	**32555.93**	**36306.30**	**45276.84**	33460.05	**37678.89**	**47722.77**	35052.22	**40685.49**	53232.24

6. CONCLUSION AND FUTURE WORK

In this thesis, we worked on the applicability of selection hyper-heuristics for dynamic environments. First, the performances of well-known selection hyper-heuristics in literature were investigated on continuous dynamic environments exhibiting various change dynamics, produced by the Moving Peaks Benchmark generator. Second, we proposed a new heuristic selection method for solving dynamic optimization problems. In addition, we examined the performance of the proposed method using not only the benchmark functions, but also real-world optimization problems in dynamic environments.

In the first phase of the thesis, we investigated the performance of thirty five hyper-heuristics combining five heuristic selection methods {Simple Random, Greedy, Choice Function, Reinforcement Learning, Random Permutation Descent} and seven move acceptance methods {All Moves, Only Improving, Equal and Improving, Exponential Monte Carlo With Counter, Great Deluge, Simulated Annealing, Simulated Annealing with Reheating}. A hypermutation based single point search method, combined with these seven acceptance schemes, (1+λ)-ES and the state of-the-art real valued optimization approach (μ,λ)-Covariance Matrix Adaptation Evolution Strategy were also included in the experiments. The Moving Peaks Benchmark, a multidimensional dynamic function generator, was used for the experiments. Different dynamic environments were produced by changing the height, width and location of the peaks in the landscape with desired change frequencies and severities. The empirical results showed that learning selection hyper-heuristics incorporating compatible component perform well in dynamic environments. This study also shows that learning selection hyper-heuristics generalize well, which make them suitable approaches to solve dynamic optimization problems.

In the second phase of the thesis, we proposed a novel heuristic selection scheme for selection hyper-heuristics, namely Ant-based selection hyper-heuristic, for dynamic environments. In the first phase of the thesis, existing heuristic selection methods were tested in dynamic environments and the learning selection methods were shown to be successful. However, we assumed that these algorithms were made aware when a change occurs in the environment. For these methods, the current solution was re-evaluated when the environment changes. For the proposed Ant-based selection scheme, this was not required. The parameters of the proposed heuristic selection methods were not reset when the environment changes. In the experimental study, we experimented with the proposed heuristic selection method combined with Improving and Equal acceptance method for dynamic optimization problems generated by Moving Peaks Benchmark. We considered two different variants of Ant-based Selection which use Roulette Wheel and Tournament Selection to determine the next heuristic. When compared Roulette Wheel with Tournament Selection, Roulette Wheel delivered better performance. To assess the performance of our approach, we compared our experimental results with the ones obtained using Choice Function, Reinforcement Learning and an improved version of the Choice Function. These selection mechanisms were also used together with the Improving-and-Equal acceptance technique. The results showed that the proposed heuristic selection method provides comparable results.

The proposed heuristic selection method does not need to know the time and nature of the changes in the environment. Nevertheless, the acceptance mechanism accepts the first solution generated after each environment change regardless of its quality. Therefore, the algorithm requires the detection of environment changes. To detect a change in the environment, we used a simpler approach in which the current solution is re-evaluated at each step. The empirical results showed that the re-evaluation scheme provides a slightly poorer performance. However, however, the approach is suitable for cases where changes cannot be made known to the optimization algorithm. As a future work, acceptance schemes in hyper-heuristics can be developed which are more suitable to dynamic environments.

Furthermore, we performed a comprehensive analysis of the proposed approach. We explored the influence of the parameters on the performance of the algorithms. According to our experimental results, the proposed approach was in general capable of adapting itself to the changes rapidly. Moreover, its performance is not much affected by the settings of the parameters.

In the last phase of the thesis, we examined the performance of the proposed scheme in three different applications. First, we proposed a multi-population framework using the hyper-heuristics. The framework enables hybridization of EDAs and selection hyper-heuristics based on online and offline learning mechanisms for solving dynamic environment problems. A dual population approach was implemented, referred to as HH-EDA2 which uses PBIL2 as the EDA. The performance of the overall algorithm was tested using different heuristic selection methods to determine the best one for HH-EDA2. The results revealed that the selection scheme that relies on a fixed permutation of the underlying low-level heuristics (Random Permutation) was the most successful one. HH-EDA2 was in general capable of adapting itself to the changes rapidly whether the change is random or cyclic. Even though the hybrid method provides good performance in the overall, it generates an outstanding performance particularly in cyclic environments. This is somewhat expected, since the hybridization technique based on a dual population acts similar to a memory scheme, which is already known to be successful in cyclic dynamic environments [25]. Furthermore, HH-EDA2 outperforms well know approaches from literature for almost all cases, except for some deceptive problems.

In the last application, we tested our approach on real-world problems. Even though Ant-based selection was proposed for dynamic environments, we wanted to see its performance in stationary environment too. Therefore, first, the proposed approach was implemented on HyFlex. The Java implementation of HyFlex was used in CHeSC2011 competition and provides six stationary optimization problems. The performance of the proposed approaches were compared to that of competitors in CHeSC2011. The results showed that the proposed method was among the midst ranking algorithms. Then, to assess the performance of the proposed method on a dynamic real-world problem, we chose the Dynamic Traveling Salesman Problem. The instances of the Dynamic Traveling Salesman Problem were generated from stationary Traveling Salesman Problem instances by introducing a traffic factor as proposed in [18]. We compared our experimental results with the ones obtained using the best performing approach on the stationary optimization problems provided by HyFlex. The proposed methods were also compared with problem specific approaches proposed for the Dynamic Traveling Salesman Problem. The results showed that the proposed approaches provided good results for the Dynamic Traveling Salesman Problem except for high severity change cases.

In this thesis, we investigated a single point based selection hyper-heuristics in dynamic environments. Hyper-heuristics were directly employed in various dynamic environment problems. The empirical results showed that hyper-heuristic did not depend on the change properties. However, in literature, different approaches were used for different change properties. For example, if the changes are severe and the change frequency is relatively high, the approaches which maintain diversity at all times are preferred. The approaches increasing diversity after a change are preferred for environments where changes are not too severe. Memory-based approaches are particularly more useful for cyclic environment where a change occurs periodically.

This thesis presented the Ant-based selection hyper-heuristic for solving dynamic optimization problems. This method is based on the simple ant colony optimization algorithm and maintains a matrix of pheromone values between all pairs of low-level heuristics. In Ant colony optimization, pheromone trail values and heuristic information are used together. As a future work, we can introduce a heuristic information in the proposed method as in Ant Colony Optimization. The heuristic information may be the time spent by the low-level heuristics, heuristics types, i.e. mutational, crossover, hill-climbing, or use frequency of the low-level heuristics.

The proposed approach was applied to several benchmark functions and real-world problems without any modifications. All results showed that the proposed approach provided good and competitive results to existing methods. These findings emphasized the general nature of hyper-heuristics also in dynamic environments.

In this thesis, the proposed approaches were applied to benchmark functions and real-world optimization problems in dynamic environments. Another future work can be to design Ant-based selection hyper-heuristics to solve multi-objective optimization problems in dynamic environments.

REFERENCES

[1] **Burke, E.**, **Hart, E.**, **Kendall, G.**, **JimNewall**, **Ross, P. and Schulenburg, S.** (2003). Hyper-Heuristics: An emerging Direction in Modern Search Technology. F. Glover and G. Kochenberger, Eds., *Handbook of Metaheuristics*, Kluwer.

[2] **Ross, P.** (2005). Hyper-heuristics. E.K. Burke and G. Kendall, Eds., *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer.

[3] **Chakhlevitch, K. and Cowling, P.** (2008). Hyperheuristics: Recent developments. C. Cotta, Ed., *Adaptive and Multilevel Metaheuristics*, Springer.

[4] **Burke, E.K.**, **Hyde, M.R.**, **Kendall, G.**, **Ochoa, G.**, **Özcan, E. and Woodward, J.R.** (2009). Exploring Hyper-heuristic Methodologies with Genetic Programming. J. Kacprzyk, L.C. Jain, C.L. Mumford and L.C. Jain, Eds., *Computational Intelligence* (Vol.1, pp. 177–201), Springer.

[5] **Denzinger, J.**, **Fuchs, M. and Fuchs, M.** (1997). High Performance ATP Systems by Combining Several AI Methods. *4th Asia-Pacific Conf. on SEAL*, (pp.102–107).

[6] **Cowling, P.**, **Kendall, G. and Soubeiga, E.** (2000). A hyper-heuristic approach to scheduling a sales summit. *Practice and Theory of Automated Timetabling III : Third International Conference, PATAT 2000*, (Vol. 2079 of LNCS), Springer.

[7] **Crowston, W.B.**, **Glover, F.**, **Thompson, G.L. and Trawick, J.D.** (1963). Probabilistic and parametric learning combinations of local job shop scheduling rules. *ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh*, (117).

[8] **Fisher, H. and Thompson, G.L.** (1963). Probabilistic Learning combinations of local job-shop scheduling rules. *J.F. Muth and G.L. Thompson, Eds.,* Industrial Scheduling, (pp.225–251), Prentice-Hall.

[9] **Soubeiga, E.** (2003). *Development and Application of Hyperheuristics to Personnel Scheduling* (Ph.D. thesis). School of Computer Science and Information Technology, The University of Nottingham, Nottingham.

[10] **Burke, E.K.**, **Hyde, M.R.**, **Kendall, G.**, **Ochoa, G.**, **Özcan, E. and Woodward, J.R.** (2010). A Classification of Hyper-heuristic Approaches. M. Gendreau and J.Y. Potvin, Eds., *Handbook of Metaheuristics* (Vol. 146, pp.449–468), Springer.

[11] **Burke, E.K.**, **Hyde, M.**, **Kendall, G.**, **Ochoa, G.**, **Özcan, E. and Rong, Q.** (2009). A Survey of Hyper-heuristics (Report No. NOTTCS-TR-SUB-0906241418-2747). School of Computer Science and Information Technology, The University of Nottingham.

[12] **Burke, E.K. and Kendall, G.** (2005). *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer.

[13] **Branke, J.** (2002). *Evolutionary optimization in dynamic environments*. Kluwer.

[14] **Cruz, C.**, **Gonzalez, J. and Pelta, D.** (2011). Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, *15*, 1427–1448.

[15] **Yang, S.**, **Ong, Y. and Jin, Y.** (2007). Evolutionary Computation in Dynamic and Uncertain Environments. *Studies in Computational Int.* (Vol. 51). Springer.

[16] **Branke, J.** (1999). Memory enhanced evolutionary algorithms for changing optimization problems. *In Congress on Evolutionary Computation CEC 99*, (Vol. 3, pp.1875–1882), IEEE.

[17] **Ochoa, G.**, **Hyde, M.R.**, **Curtois, T.**, **Vázquez-Rodríguez, J.A.**, **Walker, J.D.**, **Gendreau, M.**, **Kendall, G.**, **McCollum, B.**, **Parkes, A.J.**, **Petrovic, S. and Burke, E.K.** (2012). HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search. *EvoCOP*, (pp.136–147).

[18] **Mavrovouniotis, M. and Yang, S.** (2013). Ant Colony Optimization with Immigrants Schemes for the Dynamic Travelling Salesman Problem with Traffic Factors. *Appl. Soft Comput.*, *13* (10), 4023–4037.

[19] **Jin, Y. and Branke, J.** (2005). Evolutionary Optimization in Uncertain Environments-a Survey. *Trans. Evol. Comp*, *9* (3), 303–317.

[20] **Grefenstette, J.J.** (1992). Genetic algorithms for changing environments. *Proceedings of Parallel Problem Solving from Nature*, (pp.137–1446).

[21] **Cobb, H.G.** (1990). An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments (Report No. AIC-90-001), Naval Research Lab., Washington, DC.

[22] **Vavak, F.**, **Jukes, K. and Fogarty, T.C.** (1997). Adaptive Combustion Balancing in Multiple Burner Boiler Using a Genetic Algorithm with Variable Range of Local Search. *ICGA* (pp.719–726). Morgan Kaufmann.

[23] **Lewis, J.**, **Hart, E. and Ritchie, G.** (1998). A comparison of dominance mechanisms and simple mutation on nonstationary problems. *Proc. of Parallel Problem Solving from Nature* (pp.139–148).

[24] **Uyar, Ş. and Harmanci, E.** (2005). A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments. *Soft Comput.*, *9* (11), 803–814.

[25] **Yang, S. and Yao, X.** (2008). Population-based incremental learning with associative memory for dynamic environments. *IEEE Trans. on Evolutionary Comp.*, *12* (5), 542–561.

[26] **Yang, S.** (2008). Genetic algorithms with memory and elitism based immigrants in dynamic environments. *Evolutionary Computation*, *16* (3), 385–416.

[27] **Ursem, R.K.** (2000). Multinational GA optimization techniques in dynamic environments. *Proc. of the Genetic Evol. Comput. Conf.*, (pp.19–26).

[28] **Morrison, R.W.** (2004). *Designing evolutionary algorithms for dynamic environments*, Springer.

[29] **Ghosh, A. and Muehlenbein, H.** (2004). Univariate marginal distribution algorithms for non-stationary optimization problems. *Int. J. Know.-Based Intell. Eng. Syst.*, *8* (3), 129–138.

[30] **Kobliha, M.**, **Schwarz, J. and Očenášek, J.** (2006). Bayesian Optimization Algorithms for Dynamic Problems. *EvoWorkshops*, (Vol. 3907 of Lecture Notes in Computer Science, pp.800–804). Springer.

[31] **Yang, S. and Yao, X.** (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Comput.*, *9* (11), 815–834.

[32] **Simões, A. and Costa, E.** (2008). Evolutionary Algorithms for Dynamic Environments: Prediction Using Linear Regression and Markov Chains. *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, (pp.306–315). Springer-Verlag, Berlin, Heidelberg.

[33] **Simões, A. and Costa, E.** (2008). Evolutionary Algorithms for Dynamic Environments: Prediction using Linear Regression and Markov Chains (Report No. TR 2008/01), Coimbra, Portugal.

[34] **Simões, A. and Costa, E.** (2009). Prediction in evolutionary algorithms for dynamic environments using markov chains and nonlinear regression, *GECCO '09, Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, (pp.883–890). ACM, New York, NY, USA.

[35] **Simões, A. and Costa, E.** (2009). Improving prediction in evolutionary algorithms for dynamic environments, *GECCO '09, Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, New York, NY, USA.

[36] **Baluja, S.** (1994). Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning (Report No. CMU-CS-94-163). Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA.

[37] **Yang, S.** (2004). Constructing dynamic test environments for genetic algorithms based on problem difficulty. *CEC 2004, Proc. of the 2004 Congress on Evolutionary Computation*, (pp.1262–1269).

[38] **Yang, S.** (2005). Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems. *CEC 2004, Proc. of the 2005 Congress on Evol. Comput*, (pp.2560–2567).

[39] **Guntsch, M. and Middendorf, M.** (2002). Applying Population Based ACO to Dynamic Optimization Problems. *ANTS '02, Proceedings of the Third International Workshop on Ant Algorithms*, (pp.111–122). Springer-Verlag.

[40] **Eyckelhof, C.J. and Snoek, M.** (2002). Ant Systems for a Dynamic TSP. *Ant Algorithms*, (Vol. 2463).

[41] **Younes, A.**, **Basir, O. and Calamai, P.** (2003). Benchmark generator for dynamic optimization. *Proceedings of the 3rd International Conference on Soft Computing, Optimization, Simulation & Manufacturing Systems*, (pp. 273–278).

[42] **Mavrovouniotis, M. and Yang, S.** (2012). A benchmark generator for dynamic permutation-encoded problems. *PPSN XII, Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN XII), Part II*, (Vol. 7492 of LNCS, pp.508–517). Springer-Verlag.

[43] **Branke, J.**, **Salihoğlu, E. and Uyar, Ş. A.** (2005). Towards an analysis of dynamic environments. *GECCO '05, Proc. of the 2005 conference on genetic and evolutionary computation*, (pp.1433–1440). ACM.

[44] **Salihoğlu, E.** (2005). *Towards an analysis of dynamic environments* (Doctoral dissertation). Retrieved from `http://web.itu.edu.tr/etaner/thesis-salihoglu-05.pdf`.

[45] **Mavrovouniotis, M. and Yang, S.** (2010). Ant colony optimization with immigrants schemes in dynamic environments. *PPSN XI, Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN XI), Part II*, (Vol. 6238 of LNCS, pp.371–380). Springer-Verlag.

[46] **Mavrovouniotis, M. and Yang, S.** (2011). A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, *15* (7), 1405–142.

[47] **Mavrovouniotis, M. and Yang, S.** (2011). Memory-Based Immigrants for Ant Colony Optimization in Changing Environments. *Proc. of EvoApplications 2011*, (Vol. 6624 of LNCS, pp.324–333). Springer-Verlag.

[48] **Burke, E.K.**, **Gendreau, M.**, **Hyde, M.R.**, **Kendall, G.**, **Ochoa, G.**, **Özcan, E. and Qu, R.** (2012). Hyper-heuristics: A Survey of the State of the Art. *to appear in the Journal of the Operational Research Society.*

[49] **Özcan, E.**, **Bilgin, B. and Korkmaz, E. E.** (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, *12*, 3–23.

[50] **Bai, R.** (2005). *An Investigation of Novel Approaches for Optimizing Retail Shelf Space Allocation* (Doctoral dissertation). School of Computer Science and Information Technology, The University of Nottingham.

[51] **Cowling, P.**, **Kendall, G. and Soubeiga, E.** (2001). A parameter-free hyper-heuristic for scheduling a sales summit. *Proceedings of the 4th Metaheuristic International Conference*, (pp.127–131).

[52] **Nareyek, A.** (2001). Choosing Search Heuristics by Non-Stationary Reinforcement Learning. *Metaheuristics: Computer Decision-Making*, (pp.523–544). Kluwer Academic Publishers. Retrieved from `http://www.ai-center.com/publications/nareyek-mic01.pdf`.

[53] **Bilgin, B.**, **Özcan, E. and Korkmaz, E. E.** (2006). An experimental study on hyper-heuristics and exam timetabling. *PATAT 2006, Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling*, (pp.123–140).

[54] **Kendall, G. and Mohamad, M.** (2004). Channel Assignment in Cellular Communication Using a Great Deluge Hyperheuristic. *IEEE Int. Conf. on Network*, (pp.769–773).

[55] **Ayob, M. and Kendall, G.** (2003). A Monte Carlo Hyper-Heuristic to Optimise Component Placement Sequencing for Multi Head Placement Machine. *Proceedings of the Int. Conf. on Intelligent Technologies*, (pp.132–141).

[56] **Bai, R. and Kendall, G.** (2005). An investigation of automated planograms using a simulated annealing based hyper-heuristics. T. Ibaraki, K. Nonobe and M. Yagiura, Eds., *Metaheuristics: Progress as Real Problem Solver* (Operations Research/Computer Science Interface Series, Vol.32), Springer.

[57] **Bai, R.**, **Blazewicz, J.**, **Burke, E.K.**, **Kendall, G. and McCollum, B.** (2007). A Simulated Annealing Hyper-Heuristic Methodology for Flexible Decision Support (Report No. NOTTCS-TR-2007-8), School of CSiT, University of Nottingham.

[58] **Lundy, M. and Mees, A.** (1986). Convergence of An Annealing Algorithm. *Mathematical Programming*, *34*, 111–124.

[59] **Cowling, P.**, **Kendall, G. and Soubeiga, E.** (2002). Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. *EvoWorkShops*, (Vol. 4193 of Lecture Notes in Computer Science, pp.1–10). Springer.

[60] **Dowsland, K.A.**, **Soubeiga, E. and Burke, E.K.** (2007). A Simulated Annealing Hyper-Heuristic for Determining Shipper Sizes. *European Journal of Operational Research*, *179* (3), 759–774.

[61] **Burke, E.K.**, **Kendall, G.**,**Mısır, M. and Özcan, E.** (2010). Monte Carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 1–18.

[62] **Özcan, E.,Mısır, M., Ochoa, G. and Burke, E.K.** (2010). A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling. *International Journal of Applied Metaheuristic Computing*, *1* (1), 39–59.

[63] **Gibbs, J., Kendall, G. and Özcan, E.** (2011). Scheduling English Football Fixtures over the Holiday Period Using Hyper-heuristics. R. Schaefer, C. Cotta, J. Kolodziej and G. Rudolph, Eds., *Parallel Problem Solving from Nature - PPSN XI*. Springer Berlin / Heidelberg.

[64] **Drake, J.H., Özcan, E. and Burke, E.K.** (2012). An Improved Choice Function Heuristic Selection for Cross Domain Heuristic Search, C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia and M. Pavone, Eds., *Parallel Problem Solving from Nature - PPSN XII*. Springer Berlin Heidelberg.

[65] **Curtois, T., Ochoa, G., Hyde, M. and Vázquez-Rodríguez, J.A.** (2010). A HyFlex Module for the Personnel Scheduling Problem. Retrieved from `http://www.hyflex.org/chesc2014/wp-content/uploads/2013/09/PersonnelSchedulingHyFlex.pdf`.

[66] **Hyde, M., Ochoa, G., Curtois, T. and Vázquez-Rodríguez, J.A.** (2010). A HyFlex Module for the One Dimensional Bin Packing Problem. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/reports/BinPackingHyFlex.pdf`.

[67] **Hyde, M., Ochoa, G., Curtois, T. and Vázquez-Rodríguez, J.A.** (2010). A HyFlex Module for the Maximum Satisfiability (MAX-SAT) Problem. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/reports/BoolenSatisfiabilityHyflex.pdf`.

[68] **Vázquez-Rodríguez, J.A., Ochoa, G., Curtois, T. and Hyde, M.** (2010). A HyFlex Module for the Permutation Flow Shop Problem. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/reports/PermutationFlowshopHyFlex.pdf`.

[69] **Özcan, E., Etaner-Uyar, S. and Burke, E.** (2009). A Greedy Hyper-heuristic in Dynamic Environments. *GECCO 2009 Workshop on Automated Heuristic Design: Crossing the Chasm for Search Methods*, (pp.2201–2204).

[70] **Kiraz, B., Uyar, Ş. and Özcan, E.** (2011). An Investigation of Selection Hyper-heuristics in Dynamic Environments. *Proc. of EvoApplications 2011*, (Vol. 6624 of LNCS). Springer.

[71] **Kiraz, B. and Topcuoglu, H.R.** (2010). Hyper-heuristic approaches for the dynamic generalized assignment problem. *ISDA 2010, 2010 10th International Conference on Intelligent Systems Design and Applications (ISDA)*, (pp.1487–1492).

[72] **Beyer, H. and Schwefel, H.** (2002). Evolution strategies - A comprehensive introduction. *Natural Computing*, *1*, 3–52.

[73] **Hansen, N. and Ostermeier, A.** (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, *9* (2), 159–195.

[74] **Hansen, N.**, **Müller, S. and Koumoutsakos, P.** (2003). Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evol*, *11* (1), 1–18.

[75] **Hansen, N.** (2011). The CMA Evolution Strategy: A Tutorial. Retrieved from `https://www.lri.fr/~hansen/cmatutorial.pdf`.

[76] **Kiraz, B.**, **Etaner-Uyar, A.Ş. and Özcan, E.** (2013). Selection Hyper-heuristics in Dynamic Environments. *Journal of the Operational Research Society*, *64* (12), 1753–1769.

[77] **Uludağ, G.**, **Kiraz, B.**, **Etaner-Uyar, Ş. and Özcan, E.** (2012). A Framework to Hybridise PBIL and a Hyper-heuristic for Dynamic Environments, *PPSN 2012: 12th International Conference on Parallel Problem Solving from Nature*, (Vol. 7492, pp.358–367). Springer.

[78] **Uludağ, G.**, **Kiraz, B.**, **Etaner-Uyar, Ş. and Özcan, E.** (2012). Heuristic Selection in a Multi-phase Hybrid Approach for Dynamic Environments, *UKCI '12, 12th UK Workshop on Computational Intelligence*. Edinburgh, Scotland.

[79] **Uludağ, G.**, **Kiraz, B.**, **Etaner-Uyar, Ş. and Özcan, E.** (2013). A Hybrid Multi-population Framework for Dynamic Environments Combining Online and Offline Learning. *Soft Computing*, *17* (12), 2327–2348.

[80] **Dorigo, M. and Stützle, T.** (2004). *Ant Colony Optimizations*. MIT Press.

[81] **Eiben, A.E. and Smith, J.E.** (2003). *Introduction to Evolutionary Computing*. Spring.

[82] **Mavrovouniotis, M. and Yang, S.** (2013). Adapting the Pheromone Evaporation rate in Dynamic Routing Problem, *Proc. of EvoApplications 2013*, (Vol. 7835 of LNCS, pp.606–615). Springer-Verlag.

[83] **Larrañaga, P.** (2002). *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Boston, MA.

[84] **Barlow, G.J. and Smith, S.F.** (2009). Using Memory Models to Improve Adaptive Efficiency in Dynamic Problems. *IEEE Symposium on Computational Intelligence in Scheduling, CISCHED*.

[85] **Fernandes, C.M.**, **Lima, C. and Rosa, A.C.** (2008). UMDAs for dynamic optimization problems. *GECCO '08, Proc. of the 10th conference on genetic and evolutionary computation*, (pp.399–406). ACM.

[86] **Wu, Y.**, **Wang, Y.**, **Liu, X. and Ye, J.** (2010). Multi-population and diffusion UMDA for dynamic multimodal problems. *Journal of Systems Engineering and Electronics*, *21* (5), 777–783.

[87] **Yang, S. and Richter, H.** (2009). Hyper-learning for population-based incremental learning in dynamic environments. *Proc. 2009 Congr. Evol. Comput*, (pp.682–689).

[88] **Peng, X., Gao, X. and Yang, S.** (2011). Environment identification-based memory scheme for estimation of distribution algorithms in dynamic environments. *Soft Comput.*, *15*, 311–326.

[89] **Yuan, B., Orlowska, M.E. and Sadiq, S.W.** (2008). Extending a class of continuous estimation of distribution algorithms to dynamic problems. *Optimization Letters*, *2* (3), 433–443.

[90] **Yang, S.** (2005). Population-based incremental learning with memory scheme for changing environments. *GECCO '05, Proceedings of the 2005 conference on Genetic and evolutionary computation*, (pp.711–718). ACM, New York, NY, USA.

[91] **Kiraz, B., Uyar, Ş. and Özcan, E.** (2013). An Ant-based Selection Hyper-heuristic for Dynamic Environments. *EvoApplications 2013*, (Vol. 7835 of Lecture Notes in Computer Science, pp.626–635). Springer.

[92] **Mısır, M., Verbeeck, K., De Causmaecker, P. and Berghe, G.V.** (2012). An intelligent hyper-heuristic framework for CHeSC 2011. *Learning and Intelligent Optimization*, Springer.

[93] **Hsiao, P.C., Chiang, T.C. and Fu, L.C.** (2012). A VNS-based hyper-heuristic with adaptive computational budget of local search. *CEC' 12, 2012 IEEE Congress on Evolutionary Computation*, (pp.1–8).

[94] **Larose, M.** (2011). A Hyper-heuristic for the CHeSC 2011. *LION6*.

[95] **Chan, C., Xue, F., Ip, W. and Cheung, C.** (2012). A Hyper-Heuristic Inspired by Pearl Hunting. Y. Hamadi and M. Schoenauer, Eds., *Learning and Intelligent Optimization*. Springer Berlin Heidelberg.

[96] **Meignan, D.** (2012). An Evolutionary Programming Hyper-heuristic with Co-evolution for CHeSC'11. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/meignan-chesc.pdf`.

[97] **Di Gaspero, L. and Urli, T.** (2011). A Reinforcement Learning approach for the Cross-Domain Heuristic Search Challenge. *The IX Metaheuristics International Conference*.

[98] **Mascia, F. and Stützle, T.** (2012). A Non-adaptive Stochastic Local Search Algorithm for the CHeSC 2011 Competition. Y. Hamadi and M. Schoenauer, Eds., *Learning and Intelligent Optimization*. Springer Berlin Heidelberg.

[99] **Lehrbaum, A. and Musliu, N.** (2012). A New Hyperheuristic Algorithm for Cross-Domain Search Problems. Y. Hamadi and M. Schoenauer, Eds., *Learning and Intelligent Optimization*. Springer Berlin Heidelberg.

[100] **Acuña, A., Parada, V. and Gatica, G.** (2011). Cross-domain Heuristic Search Challenge: GISS Algorithm presentation. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/acuna-chesc.pdf`.

[101] **Kubalík, J.** (2012). Hyper-Heuristic Based on Iterated Local Search Driven by Evolutionary Algorithm. J.K. Hao and M. Middendorf, Eds., *Evolutionary Computation in Combinatorial Optimization*. Springer Berlin Heidelberg.

[102] **Elomari, J.** (2011) Self-Search (Extended Abstract). Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/elomari-chesc.pdf`.

[103] **Sim, K.** (2011) KSATS-HH: A Simulated Annealing Hyper-Heuristic with Reinforcement Learning and Tabu-Search. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/sim-chesc.pdf`.

[104] **McClymont, K. and Keedwell, E.C.** (2011). Markov Chain Hyper-heuristic (MCHH): An Online Selective Hyper-heuristic for Multi-objective Continuous Problems. *GECCO '11, Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, (pp.2003–2010). ACM, New York, NY, USA.

[105] **Gómez, J.** (2011) Hybrid adaptive evolutionary algorithm hyper heuristic. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/gomez-chesc.pdf`.

[106] **Núñez, J, C. and Ceballos, A.** (2011) A general purpose Hyper-Heuristic based on Ant colony optimization. Retrieved from `http://www.asap.cs.nott.ac.uk/external/chesc2011/entries/nunez-chesc.pdf`.

[107] **Khamassi, I.**, **Hammami, M. and Ghedira, K.** (2011). Ant-Q hyper-heuristic approach for solving 2-dimensional Cutting Stock Problem. *SIS '11, 2011 IEEE Symposium on Swarm Intelligence*, (pp.1–7).

[108] **Cichowicz, T.**, **Drozdowski, M.**, **Frankiewicz, M.**, **Pawlak, G.**, **Rytwiński, F. and Wasilewski, J.** (2012). Five Phase and Genetic Hive Hyper-Heuristics for the Cross-Domain Search. Y. Hamadi and M. Schoenauer, Eds., *Learning and Intelligent Optimization*, Springer Berlin Heidelberg.

[109] **Bierwirth, C.**, **Mattfeld, D.C. and Kopfer, H.** (1996). On Permutation Representations fro Scheduling Problems, *PPSN IV, Proceeding of the Parallel Problem Solving from Nature Conference*, (Vol. 1141 of Lecture Notes in Computer Science). Springer.

[110] **HyFlex.** (2014). Retrieved March 01, 2014, from `www.hyflex.org`.

[111] **CHeSC.** (2011). Retrieved December 01, 2011, from `http://www.asap.cs.nott.ac.uk/external/chesc2011/`.

[112] **The ACO with Immigrants.** (2013). Retrieved September 15, 2013, from `www.tech.dmu.ac.uk/$\sim$mmavrovouniotis/`.

[113] **TSP Instances.** (2013). Retrieved November 15, 2013, from `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`.

APPENDICES

APPENDIX A : Results of Ant-Based Selection with Tournament Selection

APPENDIX : Results of Ant-Based Selection with Tournament Selection

Table A.1 and Table A.2 show the results q_0 and tournament size tests for *AbSts* and *sAbSts*, respectively.

Table A.1 : Final offline error results of various q_0 settings for *AbSts* under the tested change frequency-severity pairs.

q_0	ts	*LF*			*MF*			*HF*		
		LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
0.0	2	4.13	8.63	10.95	6.86	10.54	13.09	23.14	24.99	29.16
	3	4.33	8.26	10.81	6.03	10.19	13.21	19.88	23.12	27.89
	4	4.13	7.85	11.17	6.44	10.80	13.68	19.80	23.03	28.10
	5	4.29	8.52	11.14	6.55	11.00	13.80	20.44	23.08	29.03
	6	5.55	8.23	11.37	8.28	11.96	14.48	25.72	25.73	31.28
0.1	2	3.94	8.19	10.38	6.30	9.91	13.24	22.76	24.57	29.94
	3	3.98	8.39	10.77	6.38	9.98	13.09	19.34	22.19	28.03
	4	4.12	8.58	10.78	6.01	10.08	12.99	18.36	22.13	27.78
	5	4.59	8.22	10.73	6.67	10.77	12.99	19.89	23.66	29.00
	6	4.59	8.94	11.76	8.57	11.21	14.56	23.35	24.71	30.59
0.3	2	4.34	7.63	11.04	5.83	10.72	13.26	19.20	23.16	28.70
	3	4.36	8.41	10.93	5.76	10.27	12.66	17.75	21.91	27.43
	4	4.11	8.59	10.52	5.86	10.00	13.27	16.72	20.96	26.97
	5	4.36	8.06	10.64	5.64	10.02	14.08	19.17	22.34	28.51
	6	4.63	8.76	10.90	7.34	10.66	14.03	20.95	23.55	29.79
0.5	2	3.95	8.73	10.90	5.74	9.47	12.74	17.34	21.52	28.42
	3	3.81	7.88	10.85	5.42	10.17	13.11	14.76	20.05	27.40
	4	3.99	8.21	10.35	5.46	10.31	13.49	15.26	20.74	27.23
	5	3.79	8.49	11.10	5.64	10.69	12.90	16.21	21.89	28.14
	6	4.30	8.63	11.65	6.12	10.09	14.03	17.87	22.09	29.43
0.7	2	3.90	7.84	10.26	5.15	9.83	13.70	15.41	20.51	27.24
	3	4.29	8.11	11.24	5.04	9.90	13.17	13.85	21.10	27.79
	4	4.19	8.54	10.96	4.88	10.33	13.47	13.43	19.08	26.74
	5	3.82	8.73	10.70	5.10	11.40	14.05	14.59	20.61	28.16
	6	4.11	9.15	10.96	5.77	10.63	13.81	17.60	23.00	30.37
0.9	2	4.17	8.64	11.12	5.08	10.12	14.14	13.69	20.74	28.01
	3	4.27	8.98	10.82	5.21	11.10	13.97	14.00	20.16	27.90
	4	3.66	8.72	11.97	5.02	10.31	14.80	14.42	21.42	28.09
	5	4.23	9.30	12.05	5.33	10.38	14.50	14.79	22.04	29.93
	6	4.03	9.31	11.18	5.71	11.37	14.30	15.30	22.15	30.76
1.0	2	3.94	8.81	11.58	5.63	11.82	14.23	16.44	24.22	33.01
	3	4.04	9.84	12.29	5.24	10.51	13.71	15.66	22.72	32.45
	4	4.11	8.82	12.29	5.82	10.69	14.31	14.98	24.89	33.35
	5	4.00	10.29	12.12	5.43	10.83	13.95	16.01	23.15	33.09
	6	3.96	10.33	11.12	5.27	11.47	14.76	15.70	23.54	31.57

Table A.2 : Final offline error results of various q_0 settings for *sAbSts* under the tested change frequency-severity pairs.

q_0	ts	*LF*			*MF*			*HF*		
		LS	*MS*	*HS*	*LS*	*MS*	*HS*	*LS*	*MS*	*HS*
	2	4.35	7.81	10.19	6.41	9.77	13.30	21.40	23.44	28.61
	3	4.26	7.91	11.33	5.97	9.93	12.99	17.27	20.62	26.63
0.0	4	3.93	7.84	11.32	5.96	10.27	13.43	16.12	20.82	27.06
	5	4.25	9.16	11.70	5.75	10.34	13.58	16.97	21.16	26.90
	6	4.47	9.06	12.18	7.01	10.53	14.65	18.89	21.94	28.20
	2	4.12	8.27	10.58	6.17	10.65	13.29	19.53	22.57	27.99
	3	4.20	8.46	10.85	5.47	9.39	13.03	15.65	19.89	26.42
0.1	4	4.05	8.66	10.88	5.64	9.46	13.01	15.14	19.84	26.38
	5	3.78	8.81	11.44	5.50	9.89	13.11	16.64	20.04	26.32
	6	4.39	8.88	11.43	6.92	10.30	14.00	17.00	21.61	27.74
	2	3.93	7.21	10.85	6.13	9.30	13.48	18.03	21.42	27.22
	3	3.98	8.53	10.88	5.53	10.76	13.66	15.25	20.14	27.08
0.3	4	4.19	8.43	11.58	5.34	9.90	12.97	13.66	18.87	26.26
	5	3.95	9.14	11.18	5.43	11.03	13.22	14.50	19.92	25.37
	6	4.20	8.84	11.30	5.95	10.39	13.65	16.41	21.08	27.70
	2	3.89	8.65	10.62	5.63	9.79	13.09	16.24	21.03	27.57
	3	4.16	7.56	11.29	5.29	10.98	13.28	14.19	19.81	25.67
0.5	4	4.00	9.27	11.73	5.26	9.93	13.51	12.55	18.34	25.12
	5	3.66	8.82	10.85	5.75	9.53	13.61	12.42	19.00	25.62
	6	3.91	8.86	11.58	6.18	9.78	13.65	14.98	21.24	28.24
	2	3.92	9.21	10.89	5.33	10.06	12.35	14.31	19.50	26.72
	3	4.24	8.59	11.31	5.02	10.68	13.20	13.45	20.04	26.94
0.7	4	3.95	8.27	11.25	4.65	9.02	13.75	12.05	18.38	26.28
	5	3.95	8.65	11.14	5.35	11.10	13.74	13.30	19.49	27.08
	6	4.48	9.01	11.54	5.22	10.99	13.25	14.03	19.43	26.94
	2	3.99	8.25	11.48	4.87	9.78	13.55	12.05	18.94	26.13
	3	4.10	8.85	11.72	5.30	11.65	14.43	13.42	20.41	28.19
0.9	4	4.31	8.16	11.24	5.35	10.51	15.07	13.11	19.70	27.68
	5	4.33	7.82	11.41	4.99	10.66	14.09	13.96	20.41	28.31
	6	4.35	10.12	12.09	5.70	10.55	13.93	14.97	21.57	29.20
	2	4.11	9.46	12.59	5.59	10.13	13.70	16.11	23.47	31.33
	3	4.08	8.97	11.70	5.56	11.79	15.11	16.32	23.04	30.80
1.0	4	4.22	8.18	12.40	5.92	11.67	13.43	14.93	21.88	30.34
	5	4.30	10.66	12.56	5.59	11.02	14.32	17.05	23.06	31.20
	6	3.83	8.92	11.66	5.04	10.53	15.09	14.84	21.96	30.11

CURRICULUM VITAE

Name Surname: Berna KİRAZ

Place and Date of Birth: Ordu - 1982

Adress: Koç Üniversitesi Lojmanları 13/1 Rumelifeneri Yolu Sarıyer - İstanbul

E-Mail: berna.kiraz@marmara.edu.tr

B.Sc.: June 2004

M.Sc.: October 2008

Professional Experience and Rewards:

▪ 2007 - Present: Research & Teaching Assistant, Marmara University, Computer Engineering Department

▪ ISDA 2010, Best Paper Award, "Hyper-Heuristic Approaches for the Dynamic Generalized Assignment Problem", B. Kiraz, H. Topçuoğlu, 2010, Cairo.

▪ TUBITAK 2211-National Scholarship Programme for PhD students

List of Publications and Patents:

▪ Köle M., **Kiraz B.**, Etaner-Uyar A. Ş. and Özcan E., 2012: Heuristics for Car Setup Optimisation in TORCS. *12th Annual Workshop on Computational Intelligence (UKCI 2012)*, September 5-7, 2012 Edinburgh, Scotland.

▪ **Kiraz B.** and Topçuoğlu H. R., 2010: Hyper-Heuristic Approaches for the Dynamic Generalized Assignment Problem. *ISDA 2010*, November 29 - December 1, 2010 Cairo, Egypt.

List of Papers in Preparation

▪ Asta S., **Kiraz B.**, Özcan E., Etaner-Uyar A. Ş. and Köle M., Experimental Comparison of Modern Heuristics for TORCS based Car Setup Optimisation *to be submitted to IEEE Transactions on Computational Intelligence and AI in Games*

PUBLICATIONS/PRESENTATIONS ON THE THESIS

▪ **Kiraz B.**, Etaner-Uyar A. Ş. and Özcan E., 2013: Selection Hyper-heuristics in Dynamic Environments. *Journal of the Operational Research Society*, 64 (12), pp. 1753-1769, 2013.

▪ Uludağ G., **Kiraz B.**, Etaner-Uyar A. Ş. and Özcan E., 2013: A Hybrid Multi-population Framework for Dynamic Environments Combining Online and Offline Learning, *Soft Computing*, Volume 17, Issue 12, pp. 2327-2348, 2013.

▪ **Kiraz B.**, Etaner-Uyar A. Ş. and Özcan E., 2013: An Ant-based Selection Hyper-heuristic for Dynamic Environments. *EvoStar - EvoApplications*, April 3-5, 2013 Vienna, Austria.

▪ Uludağ G., **Kiraz B.**, Etaner-Uyar A. Ş. and Özcan E., 2012: Heuristic Selection in a Multi-phase Hybrid Approach for Dynamic Environments. *12th Annual Workshop on Computational Intelligence (UKCI 2012)*, September 5-7, 2012 Edinburgh, Scotland.

▪ Uludağ G., **Kiraz B.**, Etaner-Uyar A. Ş. and Özcan E., 2012: A Framework to Hybridise PBIL and a Hyper-heuristic for Dynamic Environments. *12th International Conference on Parallel Problem Solving from Nature (PPSN 2012)*, September 1-5, 2012 Taormina, Italy.

▪ **Kiraz B.**, Etaner-Uyar A. Ş. and Özcan E., 2011: An Investigation of Selection Hyper-heuristics in Dynamic Environments. *EvoStar - Applications*, April 27-29, 2011 Torino, Italy.