İSTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

HYPER-HEURISTICS FOR THE UNIT COMMITMENT PROBLEM

M.Sc. Thesis by
Ali Argun BERBEROĞLU

Department : **Computer Science**

Programme : **Computer Science**

Thesis Supervisor: Asst. Prof. Dr. A. Şima UYAR

JANUARY 2011

İSTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

HYPER-HEURISTICS FOR THE UNIT COMMITMENT PROBLEM

M.Sc. Thesis by
Ali Argun BERBEROĞLU
704041002

Date of submission :	**20 December 2010**
Date of defence examination:	**26 January 2011**

JANUARY 2011

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

ÜNİTE PROGRAMLAMA PROBLEMİ İÇİN ÜST SEZGİSEL YÖNTEMLER

YÜKSEK LİSANS TEZİ
Ali Argun BERBEROĞLU
704041002

Tezin Enstitüye Verildiği Tarih : 20 Aralık 2010
Tezin Savunulduğu Tarih : 26 Ocak 2011

OCAK 2011

FOREWORD

First and foremost, I would like to thank my family for their great support during my whole life. This achievement would not have been possible without their support, encouragement and valuable suggestions.

I would like to express my graditute to Asst. Prof. Dr. Şima Uyar for her patient guidance and support throughout this thesis work. I am truly very fortunate to have the opportunity to work with her. I found her guidance to be extremely valuable.

I am very thankful to my managers and my collagues in Netaş. With their support, encouragement and understanding, I was able to attend the lectures even during some intensive periods of the ongoing projects.

I am also very thankful to the entire faculty and staff members of the Computer Engineering Department for their help during my MSc.

December 2010 Ali Argun BERBEROĞLU

Computer Engineer

TABLE OF CONTENTS

Page

ABBREVIATIONS

ACO	**:** Ant Colony Optimization
AM	**:** All Moves
ANOVA	**:** Analysis of Variance
CF	**:** Choice Function
DE	**:** Differential Evolution
DP	**:** Dynamic Programming
EDP	**:** Economic Dispatch Problem
ES	**:** Evolutionary Strategies
GA	**:** Genetic Algorithm
GD	**:** Great Deluge
GR	**:** Greedy
GRA	**:** Greedy Randomized Search Algorithm
HH	**:** Hyper-heuristic
ICGA	**:** Integer Coded Genetic Algorithm
IE	**:** Improving and Equal
MA	**:** Memetic Algorithm
LR	**:** Lagrangian Relaxation
PL	**:** Priority List
PSO	**:** Particle Swarm Optimization
RD	**:** Random Descent
RP	**:** Random Permutation
RPD	**:** Random Permutation Descent
SR	**:** Simple Random
SSGA	**:** Steady State Genetic Algorithm
OI	**:** Only Improving
UCP	**:** Unit Commitment Problem

LIST OF TABLES

Page

LIST OF FIGURES

LIST OF SYMBOLS

P_i^{max}	: The maximum power which can be generated by unit *i*
P_i^{min}	: The minimum power which can be generated by unit *i*
CS_{cold}	: The cost of a cold start-up
CS_{hot}	: The cost of a hot start-up
$t_{coldstart}$	: The number of hours a generator needs to stay offline for a coldstart
t_{up}	: The minimum up-time
t_{down}	: The minimum down-time
$v_{j,g}^{(t)}$	: The velocity of particle *j* at iteration *t* with respect to the g^{th} dimension
$x_{j,g}^{(t)}$	: The position of particle *j* at iteration *t* with respect to the g^{th} dimension
$pbest_{j,g}$	: The best fitness value of particle *j* at iteration *g*
$gbest_g$	: The best fitness value of the group at iteration *g*
w	: The inertia weight factor
$c1$	: The cognitive acceleration factor
$c2$	: The social acceleration factor
τ_{ij}	: Pheromone levels between node *i* and node *j*
η_{ij}	: Heuristic information between node *i* and node *j*
p^k_{ij}	: The probability of ant *k* to choose next node depending on the edge in between node *i* and node *j* of the graph
α	: The effect of pheromone level
β	: The effect of pheromone level
ρ_0	: The pheromone evaporation rate
$V_{i,g}$	: The donor vector for individual *i* of generation *g*
$X_{j,i,g}$	: The randomly chosen target vector for the gene location *j* of the individual *i* in generation *g*
$e^{-\delta / t}$	: The Metropolis criterion

HYPER-HEURISTICS FOR THE UNIT COMMITMENT PROBLEM

SUMMARY

As the power demand varies in different periods of a day, power generation companies need to plan the operation periods of the generators accordingly. The power demand is especially high during the daytime, since the factories consume most of the generated electricity in that time period. However, this demand decreases significantly at weekends or during early morning and late evening, when people spend their times at home. Due to these cycles in the required power, startup and shutdown costs of the generating units take a huge amount in the total production costs. Violating generator specific operation constraints brings additional maintenance cost as well.

The problem of selecting the generators to be in service and determining for how long they will operate over a predefined time horizon is called the Unit Commitment Problem (UCP). The online units must fulfill the forecasted power demand and reserve requirements for each time slot at minimum operating cost without violating any of the problem specific constraints.

An improvement in the unit commitment schedule of the generators result in great economic savings in power generation cost and energy usage. Therefore, the UCP has attracted great commercial and academic interest and many optimization techniques have been applied to this problem. New algorithms have been implemented to obtain efficient results in large-scale power systems within a reasonable computation time. Several numerical optimization techniques, such as priority list method, dynamic programming, branch-and-bound, benders decomposition, tabu search, greedy algorithm, Lagrangian relaxation have been used for that purpose. Aside from these methods, nature insipired computing methods, such as ant colony optimization, particle swarm optimization, simulated annealing, genetic algorithms, artificial neural networks have been employed to solve the UCP. Operational constraints are integrated into the methods of the second group more effectively and the solution quality is increased.

In this thesis, hyper-heuristic algorithms are implemented to solve the UCP. Hyper-heuristics differ from heuristics in the set, on which they are operating. Heuristics are directly applied to the candidate solutions, but hyper-heuristics are employed to select a heuristic from a set of heuristics at each iteration of the search process. This selection is made either randomly or using some performance indicators about the heuristics. Hyper-heuristics can be succesfully applied to a broad range of optimization problems, since they do not require problem specific information.

Hyper-heuristic algorithms consist of two different mechanisms:

1. Heuristic selection
2. Move acceptance

For the heuristic selection mechanism, six different selection strategies are used in this study:

1. Simple random
2. Random descent
3. Random permutation
4. Random permutation descent
5. Greedy
6. Choice function

In the move acceptance step, four different strategies are used:

1. Only improving
2. Improving and equal
3. Great deluge
4. All moves

Twenty four combinations of the above listed heuristic selection and move acceptance strategies are applied to the UCP and their performances are investigated using two problem instances.

In the first part of the experiments, random permutation descent heuristic selection and only improving move acceptance strategy combination achieves the best results among the twenty four combinations. Therefore, the performance of this algorithm is compared with other optimization techniques using seven problem instances taken from literature.

The steps of the proposed hyper-heuristic algorithm with random permutation descent heuristic selection and only improving move acceptance strategy combination are defined as follows:

1. An initial solution is created randomly and its fitness value is calculated.
2. A permutation array containing the order of mutational heuristics and hill climbers is created randomly.
3. A heuristic is selected with respect to the order in the permutation array and it is applied to the solution.
4. If the selected heuristic is a mutational heuristic, a predefined hill climbing operator is applied to the solution after mutational heuristic.
5. The fitness value of the resultant solution is calculated.
6. The resultant solution replaces the current solution, if its fitness value is better than the previous one.

In the second part of the experiments, the performance of the most efficient hyper-heuristic algorithm is compared with the performance of a genetic algorithm, since these two algorithms use the same genetic operators. Hyper-heuristic method achieves better results than the genetic algorithm. Its superiority becomes more obvious with increasing problem data size. Additionally, hyper-heuristic method does not require system monitoring and parameter tuning during the search process.

In the third part, a set of experiments are performed to compare the eficiency of the hyper-heuristic method with previously published results. Experimental results show that the hyper-heuristic method achieves either the best fitness value or the second best fitness value in all test sets. Based on the results, it can be easily noticed that the hyper-heuristic algorithm is a robust and effective optimization method for varying data sizes.

Consequently, it is recommended that the hyper-heuristic approach can be enhanced by incorporating more effective heuristics and hill climbers. The initial solution can be also improved using the priority list method and advanced learning techniques can be used for heuristic selection and move acceptance mechanisms.

ÜNİTE PROGRAMLAMA PROBLEMİ İÇİN ÜST SEZGİSEL YÖNTEMLER

ÖZET

Elektrik enerjisine olan ihtiyaç günün farklı saatlerinde büyük değişim gösterdiğinden, enerji üreten şirketlerin generatör çalışma sürelerini bu değişime uygun olarak planlamaları gerekmektedir. Üretilen elektriğin önemli bir kısmı fabrikalar tarafından tüketildiğinden enerji ihtiyacı gündüz saatlerinde daha fazla olmaktadır. Fakat bu ihtiyaç hafta sonları ve insanların vakitlerini evlerinde geçirdikleri gece geç saatlerde ve günün erken saatlerinde azalmaktadır. Enerji ihtiyacında görülen bu periyodik dalgalanmalardan dolayı generatörlerin devreye alınma ve devreden çıkarılma maliyetleri toplam enerji üretim maliyetleri içerisinde önemli bir pay tutar. Generatörlere ait çalışma limitlerinin ihlali de ek bir bakım maliyetini beraberinde getirmektedir.

Önceden belirlenmiş bir süreç içerisinde hangi generatörlerin devreye alınacağı ve ne kadar süre çalışacaklarının belirlenmesi problemine ünite programlama problemi denir. Çalışmakta olan üniteler, herbir saat için önceden belirlenmiş olan enerji talebi ve rezerve ihtiyaçlarını, probleme ait hiçbir limit koşulu ihlal etmeden minimum çalışma maliyeti ile sağlamalıdırlar.

Generatörlerin çalışma tarifesinde yapılacak bir iyileştirme elektrik üretim maliyetlerinde ve enerji kullanımında önemli oranda bir tasarrufa neden olacaktır. Bundan dolayı ünite programlama problemi ticari ve akademik yönden büyük ilgi çekmiş ve birçok optimizasyon yöntemi bu probleme uygulanmıştır. Büyük ölçekli enerji üretim sistemlerinde makul hesaplama sürelerinde verimli sonuçlar alabilmek için yeni algoritmalar geliştirilmiştir. Öncelik listesi metodu, dinamik programlama, dallanma ve sınırlama, Bender ayrıştırma yöntemi, tabu araması, açgözlü arama algoritması, Lagrange gevşetme yaklaşımı gibi sayısal optimizasyon teknikleri bu amaçla kullanılmışlardır. Bu metodlardan başka, karınca kolonisi optimizasyonu, parçacık sürü optimizasyonu, tavlama benzetimi, genetik algoritmalar, yapay sinir ağları gibi doğa esinli hesaplama yöntemleri de ünite programlama problemine uygulanmıştır. Probleme ait kısıt koşullar ikinci gruba ait yöntemlere daha iyi adapte edilebildiğinden, elde edilen çözümün kalitesi de artmıştır.

Bu tez çalışması kapsamında ünite programlama probleminin çözülebilmesi için üst sezgisel algoritmalar gerçeklenmiştir. Üst sezgiselleri sezgisel yöntemlerden ayıran en önemli fark üzerlerinde çalıştıkları elemanlardır. Sezgiseller, çözüm adaylarına doğrudan uygulanırlarken, üst sezgiseller arama sürecinin herbir yinelemesinde bir sezgisel yöntem seçmek için kullanılırlar. Bu seçim rastgele yapılabileceği gibi, sezgisellerle ilgili birtakım performans gösterge araçları kullanılarak da yapılabilir. Üst sezgiseller probleme özgü herhangi bir veriye ihtiyaç duymadıklarından optimizasyon problemlerinin önemli bir bölümüne başarıyla uygulanabilirler.

Üst sezgisel algoritmalar iki farklı mekanizmadan oluşurlar:

1. Sezgisel seçimi

2. Hareket kabulü

Sezgisel seçimi mekanizması için bu çalışmada altı farklı seçim stratejisi kullanılmıştır:

1. Basit rastgele
2. Rastgele iniş
3. Rastgele permütasyon
4. Rastgele permütasyon iniş
5. Açgözlü yöntemi
6. Seçim fonksiyonu

Hareket kabulü aşamasında dört farklı strateji kullanılmıştır:

1. Sadece iyileştiren hareketler
2. İyileştiren ve denk hareketler
3. Büyük sel
4. Bütün hareketler

Yukarıda belirtilen sezgisel seçimi ve hareket kabulü yöntemleri ile yirmi dört farklı strateji kombinasyonu oluşturulmuştur. Bu kombinasyonların gösterdiği performanslar ünite programlama problemine ait iki farklı veri örneği üzerinde incelenmiştir.

Yapılan deneylerin ilk aşamasında, rastgele permütasyon iniş sezgisel seçim yöntemi ve sadece iyileştiren hareketleri kabul etme strateji kombinasyonu, yirmi dört kombinasyon arasında en iyi sonuçları elde etmiştir. Bundan dolayı, bu algoritmanın performansı, literatürde geçen yedi problem örneği kullanılarak diğer optimizasyon teknikleri ile karşılaştırılmıştır. Bu çalışmada önerilen üst sezgisel algoritmanın aşamaları aşağıda belirtilmiştir:

1. Başlangıç çözümü rastgele bir biçimde yaratılarak, başarım değeri hesaplanır.
2. Mutasyonel sezgisellerin ve tepe tırmanma metodlarının çözüme uygulanma sırasını içeren permütasyon dizisi rastgele olarak oluşturulur.
3. Permütasyon dizisindeki sıralama doğrultusunda bir sezgisel seçilerek çözüme uygulanır.
4. Seçilen sezgisel, mutasyonel sezgisel bir yöntem ise önceden tanımlanan bir tepe tırmanma algoritması, mutasyonel sezgiselin ardından çözüme uygulanır.
5. Elde edilen çözümün başarım değeri hesaplanır.
6. Elde edilen çözümün başarım değeri, önceki çözümün başarım değerinden daha iyi ise, yeni çözüm önceki çözümün yerine geçer.

Deneylerin ikinci aşamasında en verimli üst sezgisel algoritmanın performansı aynı genetik operatörleri kullanan bir genetik algoritmanın performansı ile karşılaştırılmıştr. Üst sezgisel yöntem genetik algoritmadan daha iyi sonuçlar elde ederken, genetik algoritmaya olan üstünlüğü büyük veri içeren problemlerde daha belirgin hale gelmiştir. Bununla birlikte üst sezgisel yöntemler arama süreci esnasında sistem izleme ve parametre ayarlarına ihtiyaç duymamaktadırlar.

Üçüncü aşamada üst sezgisel metodun verimliliğini literatürde yayınlanmış sonuçlarla karşılaştırmaya yönelik testler koşulmuştur. Test sonuçları üst sezgisel metodun bütün test gruplarında en iyi birinci veya ikinci sonucu aldığını göstermektedir. Bu sonuçlardan yola çıkarak, üst sezgisel algoritmanın farklı büyüklükteki problemler için etkin bir optimizasyon yöntemi olduğu görülmektedir.

Sonuç olarak, üst sezgisel yaklaşımın daha etkin sezgiseller ve tepe tırmanma yöntemleri kullanılarak geliştirilmesi önerilmektedir. Başlangıç çözümü, öncelik listesi yöntemi kullanılarak iyileştirilebilir; sezgisel seçimi ve hareket kabulü mekanizmalarında ileri öğrenme teknikleri kullanılarak bu yaklaşımın verimi arttırılabilir.

1. INTRODUCTION

This study puts forth a hyper-heuristic approach for solving the Unit Commitment Problem (UCP). This approach combines heuristics with local search operators. The UCP is a constrained optimization problem, and the aim of this problem is to determine start-up and shut-down schedules for a predefined number of power generators over a given time period with respect to several operational constraints and hourly varying power demands. The objective is to minimize the power generation costs, while providing the hourly forecasted amount of power and reserve requirements [1].

In the first step of the optimization process, operating units are determined for each time slot without violating any constraints. In the next step, the load demand is assigned to online power units. The second part of the problem is called the Economic Dispatch Problem and it is solved using the λ-iteration method [2]. Experimental results show that the hyper-heuristic approach can create cost effective schedules.

In this study, hyper-heuristics are applied to the UCP, and a comprehensive analysis of different hyper-heuristic techniques is performed. Tests are executed using benchmark data taken from literature and real world data obtained from the Turkish interconnected power network system. The results achieved by hyper-heuristics are compared with the results from other optimization methods, which are applied previously to the UCP, to determine which algorithms are suitable for this problem. Reasons are investigated, why and how an algorithm achieves better results in this problem, and the changes in the performances of different methods are analyzed with respect to the increased problem data size.

An optimal scheduling of the generators decreases the power generation costs and energy usage significantly. Therefore, the UCP has attracted great interest and many approaches have been applied to solve the problem. Lagrangian relaxation [3], priority lists [4], simulated annealing [5], dynamic programming [6], tabu search [7], branch and bound [8], benders decomposition [9], greedy algorithms [10], particle swarm optimization techniques [11], evolutionary algorithms [1,12,13,14,15,16], ant colony optimization techniques [17] are among these approaches.

It is nearly impossible to create a heuristic which can be successful in solving a broad range of optimization problems, since it requires problem specific information, test effort and fine tuning of certain parameters during the search [18]. Hyper-heuristics are proposed to overcome these limitations. Hyper-heuristics operate on a set of heuristics rather than on solution candidates and they select a heuristic from this set to apply to the individual with respect to certain criteria. Heuristic selection can be made either according to a feedback from previous runs about the performance and the elapsed time of each heuristic, according to a probability distribution or randomly.

1.1 Contribution of the Thesis

The successful application of hyper-heuristic methods to scheduling and timetabling problems, e.g. as in [19] has been a motivation for this study. In this study, hyper-heuristic methods achieve good results in all test sets. Furthermore, it is also verified that hyper-heuristics are robust methods for large-scale problems and they obtain consistent results without the need of parameter tuning when compared with other optimization techniques. Based on the promising results, research will continue for further improvements.

1.2 Outline of the Thesis

The rest of the thesis is organized as follows. In the next section, the definition of the UCP is given together with its mathematical formulations. Section 3 contains information on the previous related work to solve the UCP. In section 4, the evolution of the hyper-heuristic approach is mentioned and the details of various hyper-heuristic methods and strategies are given. In section 5, implemented hyper-heuristic methods and genetic operators, which are incorporated into these methods, are explained. Section 5 also contains the experimental results and the comprehensive analysis of these results. Finally, in section 6, conclusion and future work are stated.

2. THE UNIT COMMITMENT PROBLEM

The demand for electricity varies in different time periods of a day. During daytime, this demand increases; whereas, during the late evening and early morning it decreases. Therefore, power generation companies should plan the generation of power on an hourly basis. In the first step, a decision needs to be made as to which of the available units to turn on; and in the second step, an economical dispatch schedule of the units should be determined.

The objective of the UCP is to minimize the power generation costs over a given time period, while providing the hourly forecasted power demand [1]. Operational constraints should be also considered when creating the online/offline schedule of each generator. A solution to the UCP contains binary decision variables representing the online/offline status of the generators for each time slot.

Two main factors are forming the power generation costs, namely fuel costs and start-up costs [1]. Penalty factors are related to the quality of a solution. One of these penalty factors is defined as the demand penalty, which is taken into account, when a predefined hourly demand is not fulfilled by a candidate solution. The other penalty factor is the up/down penalty, which has a negative effect on the fitness value, when the up/down constraint is violated for at least one generator. The following parameters are used to formulate the UCP:

- $P_i(t)$ is the generated power by unit *i* at time *t*,
- $F_i(p)$ is the cost of producing *p* MW power by unit *i*,
- PD(t) is the power demand at time *t*,
- PR(t) is the power reserve at time *t*,
- $CS_i(t)$ is the start-up cost of unit *i* at time *t*,
- $x_i(t)$ is the duration for which unit *i* has stayed online/offline since hour *t*,
- $v_i(t)$ is the status of unit *i* at time *t* (on-off),

- P_i^{max} is the maximum power which can be generated by unit *i*,
- P_i^{min} is the minimum power which can be generated by unit *i*.

The first cost factor is the fuel cost which depends on the power generated by each online unit for a given hour. While solving the UCP, not only the hourly forecasted power demand should be fulfilled, but the power produced by each unit needs to be kept within its minimum and maximum values. Following objective function and operational constraints are taken into consideration to solve the UCP. For N power units at time t, the objective function is defined as given in Eq. (2.1).

$$\min \quad F_{total}(t) = \sum_{i=1}^{N} F_i(P_i(t)) \tag{2.1}$$

subject to following constraints shown in Eq. (2.2) and Eq. (2.3):

$$\sum_{i=1}^{N} P_i(t) = PD(t) \tag{2.2}$$

$$P_i^{\min} \leq P_i(t) \leq P_i^{\max} \tag{2.3}$$

According to the second formula, the total power must be equal to the demand, and the third formula shows that the produced power by each online unit must be within its maximum and minimum capacities.

When a generator changes its status from offline to online, this brings an additional cost, which is called the start-up cost. The effect of the start-up cost on the fitness value depends on both the generator type and the amount of time a generator has stayed offline. This cost is calculated using the Eq. (2.4).

$$CS_i(t) = \begin{cases} CS_{hot} & if \quad x_i(t) \leq t_{coldstart} \\ CS_{cold} & otherwise \end{cases} \tag{2.4}$$

where $t_{coldstart}$ is the number of hours that it takes for the generator to cool down and defines the threshold for a cold or a hot start-up depending on the generator type. CS_{cold} is the cost of a cold start-up and this value is used, if the thermal unit has been off for a number of hours, which is larger than $t_{coldstart}$; otherwise, CS_{hot} is applied as the cost of a hot start-up.

The penalty value of the up/down constraint, formulated in Eq. (2.5), is determined by using the minimum up/down values of each generator. The minimum up-time t_{up} value defines the number of hours a generator has to stay online after it is turned-on, and the minimum down-time t_{down} defines the number of hours a generator has to stay offline after it is turned-off. Additional penalty cost is added to the power generation costs with respect to the number of up/down constraint violations. This constraint is based on both physical and economic considerations to prevent equipment fatigue and excessive maintenance costs due to frequent unit cycling.

$$\begin{array}{ll} if\ v_i(t) = 1 & x_i(t-1) \geq t_{down} \\ else & x_i(t-1) \geq t_{up} \end{array} \tag{2.5}$$

According to the fuel and start-up costs, demand and up/down penalty values, the objective function of the UCP for N units and T hours can be formulated as given in Eq. (2.6).

$$\min \quad F_{total} = \sum_{t=1}^{T} \sum_{i=1}^{N} \left[F_i(P_i(t)).v_i(t) + CS_i(t) \right] \tag{2.6}$$

subject to constraints:

$$\sum_{i=1}^{N} P_i(t).v_i(t) = PD(t) \tag{2.7}$$

$$v_i(t).P_i^{\min} \leq P_i(t) \leq v_i(t) P_i^{\max} \tag{2.8}$$

$$\sum_{i=1}^{N} P_i^{\max}(t).v_i(t) \geq PD(t) + PR(t) \tag{2.9}$$

$$\begin{array}{ll} if\ v_i(t) = 1 & x_i(t-1) \geq t_{down} \\ else & x_i(t-1) \geq t_{up} \end{array} \tag{2.10}$$

The fuel cost of generating p MW power for the i-th generator is calculated using Eq. (2.11). Fuel cost for the generator i depends on three parameters, a_{0i}, a_{1i} and a_{2i}, which are predefined for each generator type.

$$F_i(p) = a_{0i} + a_{1i}.p + a_{2i}.p^2 \quad \textbf{(2.11)}$$

This part of the UCP is called the Economic Dispatch Problem (EDP), which is solved by lambda iteration, whose aim is to allocate the required load demand between the available generators while minimizing the total power generation costs [2]. With the lambda iteration technique an optimal lambda value is searched for.

Algorithm 1 Lambda Iteration

```
 1:  select initial $\lambda$ and $\mu$;
 2:  **repeat**
 3:      calculate $P_i$ for each generator using $dF_i / dP_i = \lambda$;
 4:      calculate $P_{total}$ ;
 5:      diff = PD - $P_{total}$;
 6:      **if** (diff < 0) **then**
 7:          $\lambda = \lambda - \mu$;
 8:      **else**
 9:          $\lambda = \lambda + \mu$;
10:      **endif;**
11:      $\mu = \mu / 2$;
12:  **until** ($|diff| < \varepsilon$)
```

The initial values of λ and μ are determined as given in Eq. (2.12) and in Eq. (2.13), where λ_{max} and λ_{min} are calculated by inserting P_{max} and P_{min} as the values of P respectively, after taking the derivative of F_i (p) with respect to p.

$$\lambda = (\lambda_{max} + \lambda_{min}) / 2 \quad \textbf{(2.12)}$$

$$\mu = (\lambda_{max} - \lambda_{min}) / 2 \quad \textbf{(2.13)}$$

3. RELATED WORK ON THE UCP

The most widely used optimization techniques for the UCP are mentioned in this section. These techniques are dynamic programming, priority list, particle swarm optimization, ant colony optimization, branch and bound, benders decomposition and evolutionary algorithms. Evolutionary algorithms are also divided into four groups with respect to the solution representation and genetic operators. Each of these four groups is explained in detail.

The dynamic programming (DP) technique decomposes a multivariable decision problem as a sequence of single variable decision problems [6]. Therefore, an n variable problem is represented as a sequence of n single variable problems and each of them is solved respectively. DP separates the UCP into time slots, so that online units are determined one hour at a time. At the end of the time schedule, all hour-state pairs are stored for further calculations and an array is obtained that keeps the continuous online and offline periods of all units to prevent the up/down constraint violation.

In the first step of the DP algorithm, the minimum total production cost to reach the current state from the first hour of the schedule is calculated. Secondly, the state of the previous hour that minimizes the cost of the transition to the current state is determined. After collecting this information, a cost effective schedule is created by moving from the state with the least total cost at the final hour to the state of the initial hour using the optimum transition at each step [6].

The most difficult part of applying DP methods to the UCP is storing all possible state combinations for each hour. Therefore, heuristics are used to limit the number of combinations, but they produce suboptimal solutions and decrease the effectiveness of the DP. In certain problem instances, some of the problem constraints are replaced with penalty coeffiecients to obtain a feasible solution.

Priority list (PL) is a fast and simple optimization technique, but it provides suboptimal solutions. In this method, power generators are ranked in ascending order with respect to the average full load cost, so that cost effective units are put in service first to fulfill the power demand of each time slot without violating any operational constraints [4]. However, resultant schedules have high power generation costs. Average full load cost is calculated with Eq. (3.1).

$$Cost_i = (a + b * P_i^{max} + c * P_i^{max} * P_i^{max}) / P_i^{max} \qquad \textbf{(3.1)}$$

The particle swarm optimization (PSO) algorithm is a population based search algorithm, as described in Algorithm 2, which makes use of a group of particles corresponding to the individuals of a genetic algorithm. The initial population is generated randomly. Candidate solutions are represented with particles containing a position vector and a velocity vector [11]. After each time step, the fitness value of each particle is calculated. Each particle keeps its own best position with the best fitness value it has achieved so far. Additionally, the best fitness value of the whole population obtained so far is also recorded. Using the individual best position and the global best position, the velocity vector and the position vector of the corresponding particle are updated as given in Eq. (3.2) and Eq. (3.3) respectively.

$$v_{j,g}^{(t+1)} = w * v_{j,g}^{(t)} + c_1 * \eta() * (pbest_{j,g} - x_{j,g}^{(t)}) + c_2 * \eta() * (gbest_g - x_{j,g}^{(t)}) \qquad \textbf{(3.2)}$$

$$x_{j,g}^{(t+1)} = x_{j,g}^{(t)} + v_{j,g}^{(t+1)} \qquad \textbf{(3.3)}$$

with j=1,2,…,n and g=1,2,…,m where n is the number of particles in a group and m is the number of members in a particle.

$v_{j,g}^{(t)}$ is the velocity of particle j at iteration t, $x_{j,g}^{(t)}$ is the position of particle j at iteration t, w is the inertia weight factor, c1 and c2 are cognitive and social acceleration factors, r1 and r2 are random numbers uniformly distributed in the range (0,1), $pbest_{j,g}$ is the best fitness value of particle j at iteration g and $gbest_g$ is the best fitness value of the group at iteration g.

Algorithm 2 Particle Swarm Optimization Algorithm

```
1:  randomly create initial population;
2:  repeat
3:      calculate the fitness value of each particle;
4:      determine the particle with the best fitness value of the generation;
5:      update the position and the velocity vectors of each particle;
6:  until stopping criterion is met
```

Ant colony optimization (ACO) algorithm is created using the behavior of real ant colonies. Therefore, it is a population based search algorithm containing a learning mechanism. The real ants lay down a substance, called a pheromone, on the way to food [17]. The quantity of pheromone depends both on the length of the path and the quality of the food source, since the intensity of pheromone increases, when more ants choose the same way to reach the source [17].

In the first step of the ACO algorithm, the ACO parameters need to be determined. After initialization of these parameters, an iterative solution construction process starts with an empty partial solution s^p. At each construction step, a solution component is added to the partial solution without violating any of the problem specific constraints. A solution component is selected probabilistically as given in Eq. (3.4).

$$p(q_{ij}|\, s^p) = \frac{{\tau_{ij}}^{\alpha} * n(q_{ij})^{\beta}}{\sum_{q_{il} \in N(s^p)} \tau_{il}^{\alpha} * n(q_{il})^{\beta}}\,, \quad \forall q_{ij} \in N(s^p) \tag{3.4}$$

where τ_{ij} is the pheromone value for component q_{ij}, n(.) is a function which assigns a heuristic value to the feasible solution at each iteration, α and β take positive values, which determine the relation between the pheromone value and the heuristic value [17].

At the end of the construction process, the fitness value of the complete solution is calculated. The pheromone values of the solution components included in a good solution are increased, and the pheromone values of the remaining solution components are decreased with respect to Eq. (3.5) to increment the possibility of selecting a suitable solution component at the next iteration. To prevent rapid convergence of the ACO algorithm, the concept of pheromone evaporation is applied, where all pheromone values are decreased using a predefined evaporation rate between 0 and 1 [17].

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho) * \tau_{ij} + \rho * \triangle \tau & \text{if } \tau_{ij} \in s_{ch} \\ (1-\rho) * \tau_{ij} & \text{otherwise} \end{cases} \quad \textbf{(3.5)}$$

An effective branching method is proposed in [8] which makes use of a simple bounding rule. In the branch-and-bound algorithm, the online/offline schedule of the generators is represented with the commitment matrix. Each row of the matrix shows the schedule of a single generator and each column represents the status of each generator for the corresponding time slot. This algorithm also involves the economic dispatch problem as a subproblem. After determining the online generators, λ iteration method is used to compute the dispatch cost.

In the initial step of the branch-and-bound algorithm, the search space contains all feasible solutions. This space is repeatedly partitioned into smaller subsets and the lower bound of the cost value is calculated for each subset by ignoring the lower power generation limit constraint and the start-up cost [8]. After each partitioning, subsets with a lower bound which is higher than a known feasible solution are discarded. The remaining subsets are partitioned again until a feasible solution is found whose fitness value is smaller than the lower bound of any subset with at least two solutions. In other words, the lower bound belongs only to a single solution at the end, since it is the optimum solution of this problem.

Benders decomposition technique is applied to the UCP in [9]. This algorithm consists of two levels. These levels are called as master and slave. The master level deals with the unit commitment of the generators to fulfill the forecasted power demand and the second level deals with the operational constraints, such as generator limits, minimum up and down time of each unit [9].

This decomposition method uses an iterative search process between these two levels. The resultant schedule of the master problem is conveyed to the slave problem. The slave problem is divided into 24 subproblems and these subproblems are solved sequentially. The slave subproblem calculates the power generation cost with respect to the operational constraints. The result of the subproblem is used by the master problem through the Benders cut to improve the current solution. This iterative search process stops when the fitness values achieved in the master and slave levels become nearly the same except for a small predefined tolerance value.

In literature, there are many successful evolutionary techniques to solve the UCP. They can be divided into four main groups with respect to the solution representation and genetic operators, which are applied to candidate solutions.

In the first group, a binary chromosome is used as the candidate solution which represents the on/off schedule of the generators. Genetic operators are applied to these chromosomes. To solve the EDP in the second step of the problem, an iterative technique, such as lambda iteration [2], is used. Genetic algorithms [1,12], binary differential evolution algorithms [16] and memetic algorithms [14] are examples for the first group.

Since genetic algorithms (GA), developed by Holland in 1975 [21], are commonly working on binary solution representation, various GAs are applied to solve the UDP. GA techniques have common steps. First step is the initialization, where individuals are created randomly with binary digits. In the second step, individuals, which will undergo reproduction, are selected and genetic operators are applied. In the last step, population replacement is performed. A basic genetic algorithm is given in Algorithm 3.

There are two most commonly used population replacement methods. Either the whole population is replaced, or only one individual is generated and replaced with another individual of the population in each iteration [1]. This loop continues until the stopping criterion is met. This stopping criterion can be defined either as the number of iterations, which determines how many times selection, reproduction and population replacement operations take place, or as the predefined fitness value range.

Algorithm 3 Basic Genetic Algorithm

```
1:  randomly create initial population;
2:  repeat
3:      select one mating pair;
4:      generate one offspring through reproduction;
5:      evaluate offspring;
6:      if offspring better than current worst individual
7:          offspring replaces the worst individual;
8:      endif;
9:  until stopping criterion is met
```

Memetic algorithms (MAs) are defined as hybrid algorithms that combine genetic algorithms and local search operators [14]. A meme is defined as a contagious piece of information [22]. The memetic approach provides the evolution of information and the unit of a solution is referred to as a meme rather than a gene, since genes can not be changed with the experience of an individual [14].

In this algorithm, a randomly generated population of individuals evolves towards an optimal solution by undergoing a set of genetic operators, namely crossover, mutation and selection. Memetic algorithms incorporate the concept of memes by allowing individuals to change before the next population is produced. Therefore, a hill climbing operator is applied after the mutation to improve the fitness value of the resulting individual.

In a generic MA, each candidate solution consists of a binary bit string. In the first step, mates are selected to reproduce new candidate solutions. After applying crossover and mutation operators, hill climbing operator is applied to the new candidate solution, which is also called as offspring. This iterative process continues until the stopping criterion is met as shown in Algorithm 4.

Algorithm 4 Basic Memetic Algorithm

```
1:  randomly generate initial population;
2:  calculate fitness of each individual;
3:  repeat
4:      select mates;
5:      apply crossover & mutation;
6:      apply hill climbing;
7:      calculate fitness;
8:  until termination condition is met
9:  return the best solution;
```

In the second approach, the chromosome consists of integers or floating points. These genes represent the on/off cycles of the units. The integer value can be either positive or negative, which corresponds to the duration of the on or off status of each generator. The minimum up and down constraints are preserved by using specialized genetic operators. The EDP also occurs in this approach, and it is commonly solved through lambda-iteration. Differential evolution and evolutionary strategies algorithms are examples for this approach.

Differential evolution (DE) algorithm was introduced by Storn and Price in 1996 [23]. This algorithm is used in continuous search spaces and contains four main operations as given in Algorithm 5: Initialization, mutation, recombination and selection. These operators are applied to all individuals. In DE, each individual in the population, called the target vector, consists of real valued genes, $X_{j,i,g}$, where j is the gene location on the chromosome, i is the index of the individual and g represents the generation number [1,16].

Algorithm 5 Basic Differential Evolution Algorithm

```
 1:  randomly generate initial population;
 2:  evaluate population;
 3:  repeat
 4:      for population size times do
 5:          select next target vector;
 6:          randomly choose base vector;
 7:          donor vector = mutate (base vector);
 8:          trial vector = crossover (target vector, donor vector);
 9:          if trial vector better than target vector
10:              select trial vector;
11:          else
12:              select target vector;
13:          endif;
14:      endfor;
15:  until stopping criterion is met
```

In the mutation step, a donor vector is created using a mutation factor and different target vectors which are selected randomly from the population [1,16]. An example is given in Eq. (3.6).

$$V_{i,g} = X_{r0,g} + F(X_{r1,g} - X_{r2,g}) \tag{3.6}$$

where $V_{i,g}$ is the donor vector and $X_{j,i,g}$ is the randomly chosen target vector.

In the recombination step, which is also known as crossover, a new vector is created using both the donor and the target vectors. This new vector is called the trial vector [1,16]. A Cr parameter is used to determine the length of the segment taken from the target vector. This parameter takes on values in the range [0,1]. If the uniformly distributed random number is less than Cr, the corresponding parameter of the trial vector is taken from the donor vector; otherwise, it is taken from the target vector as shown in Eq. (3.7).

$$U_{j,i,g} = \begin{cases} V_{j,i,g} & if\,(rand_j(0,1) \le Cr \;\; or \;\; j = j_{rand}) \\ X_{j,i,g} & otherwise \end{cases} \quad \textbf{(3.7)}$$

In the selection step, either the target vector or the trial vector is chosen with respect to their fitness values as in Eq. (3.8). These steps continue until a predefined number of iterations have been run or until a solution in a predefined range has been achieved.

$$X_{i,g+1} = \begin{cases} U_{i,g} & if\,(\;fitness(V_{i,g}) \le \; fitness(X_{i,g})) \\ X_{i,g} & otherwise \end{cases} \quad \textbf{(3.8)}$$

Evolutionary strategies (ES) was introduced by Rechenberg in 1973 [24]. In the problem representation, three different parts constitute the individuals chromosome. First part of the chromosome is encoded as a vector of real numbers, the second part contains mutation step size parameters which are associated with each gene. The third part represents the rotation angles of each gene. A sample chromosome in ES looks like the following:

$$< p_1 \;\; p_2 \;\ldots\; p_n \;,\;\; \sigma_1 \;\; \sigma_2 \;\ldots\; \sigma_n \;,\;\; \alpha_1 \;\; \alpha_2 \;\ldots\; \alpha_n >$$

The aim of the ES algorithm is to optimize an objective function with respect to a set of control parameters. These strategy parameters are used to control statistical properties of the genetic operators. Since these parameters are also adapted during the evolution process, the genetic operators in ES are called as self-adaptive operators [1,25].

In ES, all individuals have equal probability of being selected as parents. These parents first go through crossover and then they go through mutation. During the crossover operation, each object variable of the offspring is selected usually among one of the two parental values for the corresponding gene location. For strategy parameters, the arithmetic average value of these parameters is taken. During the mutation operation, mutation step sizes are mutated first, then each resulting mutation step size is used to mutate the corresponding object variable of the chromosome. Algorithm 6 shows the algorithmic flow of a basic ES method.

Algorithm 6 Basic Evolutionary Strategies Algorithm

```
1: randomly generate initial population;
2: calculate fitness of each individual;
3: repeat
4:     for population size times do
5:         randomly select one mating pair;
6:         generate one offspring through reproduction;
7:     endfor;
8:     select individuals with respect to the population size;
9: until stopping criterion is met
```

After applying crossover and mutation, λ children are created from μ parents. In the last step of ES, one of the two different methods is used to determine the individuals for the next generation. In the plus strategy, the best λ individuals are selected from both parents and children. In the comma strategy, individuals are selected only from children [25]. This loop continues until a predefined number of iterations have been executed.

The third approach uses the Lagrangian relaxation (LR) technique along with a genetic algorithm to update the Lagrangian multipliers [3]. The LR method solves the UCP as if operational constraints do not exist. Therefore, the LR decomposition procedure creates a separate problem by embedding some constraints into the objective function through penalty coefficients. These penalty coefficients are called Lagrangian multipliers and they are determined iteratively. However, the dual problem has a lower dimension than the original problem. The difference between the original and the dual problems is defined as the duality gap, which measures the suboptimality of the solution [3].

Many studies have spent some effort to update the Lagrangian multipliers in an appropriate way to minimize the duality gap. Therefore, different genetic algorithms are integrated into the LR method to update the Lagrangian multipliers and to increase the efficiency of the LR method.

The dual problem is solved in two steps. In the first step, the Lagrangian function is minimized under constant Lagrangian multipliers by using a two-state dynamic programming technique. In the second step, the Lagrangian function is maximized using Lagrangian multipliers, which are updated by genetic algorithms. This process, described in Algorithm 7, continues until the duality gap reaches a predefined value or until a predefined number of iterations have been run.

Algorithm 7 Lagrangian Relaxation Method with GA

```
1:  repeat
2:      minimize the LR function by using two-state dynamic programming
    with constant Lagrangian multipliers;
3:      maximize the LR function by using updated Lagrangian multipliers with
    a genetic  algorithm;
4:  until stopping criterion is met
```

In the last approach, each candidate solution is represented with a floating point chromosome. Each gene shows the load for the corresponding generator. The initial population is created using Lagrangian relaxation. Evolutionary programming method is used only for online generators, which are not working at their maximum load capacities. Therefore, the aim of this approach is to improve the already dispatched power and to minimize the penalty values which are added to the computations due to the operational constraints.

4. HYPER-HEURISTICS

Heuristic methods are very successful in solving complex optimization problems. However, heuristics require problem domain knowledge and parameter tuning during the search, so that it is very difficult to apply a heuristic to a new problem, or even to a new instance of a previously solved problem [18, 26]. To overcome this difficulty, hyper-heuristic methods are introduced. Hyper-heuristics do not need any problem specific information, since they are not directly applied to the problem; instead they operate on a set of heuristics to find the most suitable heuristic, or the sequence of heuristics during the optimization process [18, 26]. Therefore, they are using some performance indicators to decide which heuristic to call at each iteration.

4.1 Background

Two fundamental ideas about the concept of hyper-heuristics are expressed in [26]. According to the first fundamental idea, selecting a heuristic or creating a sequence of heuristics is also a search problem [26]. Due to this, the second fundamental idea proposes the usage of a learning mechanism to improve the search process on the set of heuristics [26]. Different types of hyper-heuristics are implemented based on these two fundamentals.

The ideas constructing the hyper-heuristic approach are firstly used in production scheduling problem in 1961 by Fisher and Thompson. They proposed to combine different scheduling rules in a probabilistic learning method. Their study concluded that a random combination of scheduling rules result in better solutions than any of them applied separately to the problem instance [26].

In 1992, Storer firstly mentioned the importance of creating a sequence of heuristics for an efficient search algorithm [26]. He also defined the concept of neighborhoods within the search space, which constructs the basis of local search. In 1993, Feng applied a genetic algorithm, which aims at searching the space of sequences of heuristic selections to solve the open-shop scheduling problem [26]. Each heuristic selection is represented with a pair (j, h), where j is an uncompleted job and h is a heuristic, which is used to select a task from the job j and to insert it into the schedule. The sequence of pairs provides a complete schedule. This method obtained very good results on different benchmark problems. In 1997, Norankov and Goodman made use of evolutionary algorithms to search the space of heuristic sequences [26]. This method is applied to the multistage flow-shop scheduling problem. In the first step, job orderings are determined; and in the second step, jobs are assigned to the machines. Experimental results showed that there was a strong dependency between the solution quality and the heuristic sequence applied to the solution.

The term hyper-heuristic is firstly used in 2000 by Cowling, and he described the concept of “heuristics to choose heuristics” to solve optimization problems [27]. In following years, new techniques are proposed to improve hyper-heuristics. In 2002, the incorporation of the choice function into hyper-heuristics are investigated by Kendall, Cowling and Soubeiga [28]. Better results are achieved when compared with the results obtained by random hyper-heuristics. In 2003, tabu search is employed by Burke, Kendall and Soubeiga as the heuristic selection strategy [29]. Good results are achieved on the university course timetabling problem. In 2004, Kendall and Mohamad incorporated the Great Deluge algorithm as a move acceptance method and obtained very good results in examination timetabling [30].

Hyper-heuristic techniques are applied to optimization problems in two different ways. In the first approach, hyper-heuristics are used to select the most suitable heuristic from a set of heuristics for the corresponding problem state. In the second approach, hyper-heuristics are used to create heuristics for the purpose of obtaining more efficient results by specializing heuristics to the problem instance. This chapter includes a detailed description of these two hyper-heuristic approaches.

4.2 Heuristics to Choose Heuristics

When the term hyper-heuristics was first introduced in the early 2000s, this approach was applied to optimization problems by selecting the most suitable heuristic from a set of heuristics to increase the efficiency of the search algorithm. This type of hyper-heuristics are divided into two groups as constructive and perturbative hyper-heuristics according to the structure of the initial candidate solution [26].

Constructive hyper-heuristics start with an empty solution; whereas, perturbative hyper-heuristics begin to work on a complete initial solution. These complete initial solutions are created either randomly or by satisfying some of the problem specific constraints to obtain better results.

4.2.1 Constructive Hyper-heuristics

Constructive hyper-heuristic methods start with an empty solution. At each iteration, a constructive heuristic is selected to build a part of the solution. This process continues until a complete solution is achieved. Several methods with constructive heuristics have been applied to timetabling, scheduling, constraint satisfaction, cutting and packing problems.

Evolutionary algorithms were firstly employed in examination timetabling problems in 1999 by Terashina-Marin [26]. In this approach, a chromosome representation was used and this approach aimed at evolving the configuration of constraint satisfaction methods. In 2003, Ahmadi applied a variable neighborhood search algorithm to examination timetabling, where he combined different low-level heuristics during exam, period and room selections [26].

Graph-coloring heuristics are also used in timetabling problems, where nodes represent events and edges represent conflicts between events. In examination or course timetabling problems, two events have a conflict, if they contain the same student. The difficulty of an event is proportional to the number of conflicts, the event has with others. The most conflicting events are scheduled first into appropriate time slots when constructing a timetable.

A constructive hyper-heuristic framework was implemented by Burke in 2007, which included the following graph coloring heuristics: Largest Saturation Degree, Largest Color Degree, Largest Degree, Largest Enrollment, Largest Weighted Degree [26]. Tabu search was used as heuristic selection strategy to create efficient sequences of low-level heuristics. This approach achieved promising results in course and examination timetabling problems.

In 2008, Qu and Burke compared the performances of the new implemented heuristic selection strategies with the previously applied tabu search method [26]. These strategies are steepest descent method, iterated local search method and variable neighborhood search. Results showed that iterated local search and variable neighborhood search methods were more effective than steepest descent and tabu search methods. The authors also investigated the effects of sequences of heuristics on the solution quality, and they stated that early heuristic choices in a heuristic sequence have a higher impact on the quality of the solution than the late heuristic selections. In a further study, the combinations of graph coloring heuristics in examination timetabling were investigated by Pillay in 2008, where each individual consists of a variable length string with characters representing one of the five low-level graph coloring heuristics. This study also showed that this method was able to create feasible exam schedules.

Constructive hyper-heuristics were also used in production scheduling to determine which dispatching rule to call at each iteration [26]. In this problem, when a machine completes its task, a dispatching rule calculates the priorities of each waiting job and assigns the job with the highest priority to this machine. Each dispatching rule has a different priority calculation method. Minimum release time, shortest processing time, longest processing time, earliest due date, latest due date, less work remaining, more work remaining are among these rules. Many studies showed that methods combining several rules or heuristics were more efficient than other methods using a single rule or a single heuristic. This statement was experimentally verified in different problem domains.

4.2.2 Perturbative Hyper-heuristics

Perturbative hyper-heuristics operate on a set of perturbative low-level heuristics. In a constructive hyper-heuristic approach, the process continues until a complete solution is obtained. However, perturbative hyper-heuristics operate on a complete solution and this process continues until the predefined stopping criterion is met. Perturbative approach has been successfully applied to channel assignment, personnel scheduling, timetabling and vehicle routing problems. In this approach, heuristics are mostly applied to a single candidate solution at each iteration; therefore, they are called as single point search based hyper-heuristics. Population based perturbative hyper-heuristics are also employed in optimization problems, especially in scheduling and timetabling problems, where each individual contains a sequence of heuristic selections.

Learning mechanism plays an important role to increase the efficiency of the decision making process. To incorporate the learning mechanism into the hyper-heuristic approach, scores are assigned to each heuristic with respect to their performances on the quality of the solution, when they are applied to the candidate solution.

4.2.2.1 Single Point Search Based Hyper-heuristics

Single point search based hyper-heuristics consist of two mechanisms: Heuristic selection and move acceptance.

For the heuristic selection process, different strategies are proposed. Simple random heuristic selection strategy chooses a low-level heuristic randomly at each iteration. Random descent has a similar usage, except that the selected heuristic is applied to the solution repeatedly, until no improvement is achieved. In random permutation strategy, a random permutation of low-level heuristics is created and each one of the heuristics is applied to the solution in the provided order. Random permutation descent also uses the randomly generated permutation, but heuristics are applied repeatedly, until they do not improve the solution. Greedy methods apply all low-level heuristics to the candidate solution at each iteration and select the heuristic that creates the best solution.

In the choice function heuristic selection method, each low-level heuristic is given a score. This score is determined using the following three performance criteria [28]. First criterion is the individual performance of a heuristic. Second criterion is the performance of a heuristic when combined with other heuristics. The last criterion is the elapsed time since the last heuristic has been called. At each iteration, scores are computed for each low-level heuristic. The formulations of the performance computation are given in Eqs. (4.1-4.4).

$$f_1(h_i) = \sum_n \alpha^{n-1}(I_n(h_i)/T_n(h_i)) \qquad \textbf{(4.1)}$$

$$f_2(h_i,h_k) = \sum_n \beta^{n-1}(I_n(h_i,h_k)/T_n(h_i,h_k)) \qquad \textbf{(4.2)}$$

$$f_3(h_i) = elapsedTime(h_i) \qquad \textbf{(4.3)}$$

$$score(h_i) = \alpha * f_1(h_i) + \beta * f_2(h_i, h_k) + \delta * f_3(h_i) \qquad \textbf{(4.4)}$$

where $I_n(h_i)$ and $T_n(h_i)$, $I_n(h_i,h_k)$ and $T_n(h_i,h_k)$ are the change in the fitness function and the amount of time taken, respectively, when the nth last time the heuristic h_k was applied either alone or after the heuristic h_i. In Eq. (4.4), α, β and δ are the relative weight factors of each function which are used to compute the overall scores for each heuristic.

In the tabu search method proposed by Burke in 2003 [29], low-level heuristics are ranked with respect to their scores. Additionally, this method includes a tabu list, which is used to exclude some of the low-level heuristics temporarily, since they did not improve the candidate solution in their last application. If a heuristic improves the solution, its score is increased; otherwise, its score is decreased. The heuristic with the highest score, which is not in the tabu list, is applied to the solution at each iteration.

Nareyek (2003) used reinforcement learning as a heuristic selection strategy. In this learning process, each heuristic starts with the same initial score. The scores of the heuristics are changed with respect to the quality of the resulting solution when they are applied to the individual. During heuristic selection, either the heuristic with the highest score is selected or the scores are converted into probabilities and a heuristic is selected using the roulette wheel strategy.

The second mechanism of single point search based hyper-heuristics is the move acceptance method. For the move acceptance phase, two main strategies are used. These are deterministic and non-deterministic strategies. In the deterministic strategy, the same move acceptance decision is given for the same candidate solution regardless of the current point in the search process. However, in the non-deterministic strategy, different move acceptance decisions can be given for the same candidate solution with respect to the decision point. Therefore, time or iteration number is important for the decision making process.

Three deterministic approaches are used commonly: All Moves, Only Improving Moves, Improving and Equal Moves. In all moves strategy, all improving and non-improving moves are accepted; whereas, in the other two approaches either only improving moves or improving and equal moves are accepted.

One of the non-deterministic move acceptance strategies is the Monte Carlo method, which was proposed in 2003 by Ayob and Kendall [32]. This method accepts all improving moves. However, non-improving moves are accepted with respect to a probability function. An exponential probability function is used in this strategy as given in Eq. (4.5), where δ is the change in quality, t is time in minutes and Q is the number of successive non-improving moves.

$$\beta^{-\delta t/Q} \qquad \textbf{(4.5)}$$

In 2004, the Great Deluge move acceptance algorithm was experimented in a hyper-heuristic approach by Kendall and Mohamad [30]. In this method, the fitness of the initial solution is calculated and this value is set as the initial level value. Then, the down rate value is determined using the Eq. (4.6).

$$DownRate = (f(s_0) - Best\,Result)/NumberofIterations \qquad \textbf{(4.6)}$$

where BestResult is the best result found in literature for this problem and $f(s_0)$ is the fitness value of the initial solution.

After applying one of the low-level heuristics to the candidate solution, if the fitness value of the resultant solution is better than the level value, the level is decremented by the DownRate value and the resultant solution is replaced with the current solution; otherwise, the current solution is kept and the algorithm continues to run by applying another heuristic to this solution as shown in Algorithm 7.

Algorithm 8 Great Deluge Method

```
 1:  create randomly an initial candidate solution $s_0$;
 2:  calculate the fitness value of the initial solution $f(s_0)$;
 3:  set the initial level to $f(s_0)$;
 4:  set the DownRate value;
 5:  **repeat**
 6:      select & apply a heuristic to the candidate solution;
 7:      calculate the fitness of the resultant candidate solution $f(s_n)$;
 8:      **if** ($f(s_n)$ < Level)
 9:          Level = Level – DownRate;
10:          $s_0 = s_n$ ;
11:      **endif;**
12:  **until** stopping criterion is met
```

Simulated annealing is another method used as a non-deterministic move acceptance strategy. This method was proposed by Bai and Kendall in 2005 [26]. In this method, all improving moves are accepted, but non-improving moves are accepted according to the Metropolis criterion $e^{-\delta / t}$, where δ is the change in quality and t is the temperature. The temperature is decreased at each iteration using a cooling schedule. This criterion shows that a probabilistic decision is made to accept even a worsening solution. This probability does not only depend on how much worse the resultant solution is but also on how long the search process has been continuing.

Late acceptance strategy was incorporated by Burke and Nykov into the hyper-heuristic approach in 2008 [26]. This method contains a memory to keep fitness values of previous candidate solutions in a list of size L. At each iteration, the resulting candidate solution is compared with the last element of the list. If the fitness value of the new solution is equal to or better than the fitness value of the last element of the list, the new solution is added to the list as the first element and the last element is removed from the list. This method does not have a high computational expense when compared with simulated annealing and great deluge methods. Additionally, it also accepts worsening moves to prevent getting stuck at local minima.

Another study field in a hyper-heuristic approach is how heuristic selection and move acceptance strategies are combined and in which order mutational heuristics and hill climbers are executed in a hyper-heuristic method, since they have different impacts on the search process. To increase the solution quality, different regions of the search space need to be explored and the highest points of these areas should be reached. This is possible, if a mutational heuristic and a hill climber operator are employed sequentially. Four different frameworks are defined for that purpose and their performances are compared in [18]. These frameworks are called as F_A, F_B , F_C and $F_{D.}$

In F_A and F_B , a heuristic is selected from a set of mutational heuristics and hill climbers. However, the F_B framework extends F_A by employing a predefined hill climbing operator, if the selected low-level heuristic is a mutational heuristic; otherwise, only the selected low-level hill climbing operator is applied to the candidate solution before the move acceptance step. In the F_C framework, a mutational heuristic is selected at the first step, since the heuristic set only contains mutational heuristics. After that, a predefined hill climbing operator is applied to the candidate solution. In the F_D framework, firstly a mutational heuristic is selected and applied to the solution. If the resultant solution is accepted, a hill climber will be selected and applied to this new candidate solution; otherwise, the selected hill climber will be applied to the previous solution. All these four frameworks are depicted in Figures 4.1-4.4.

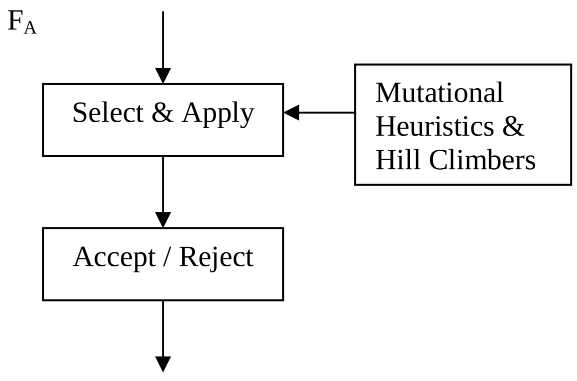
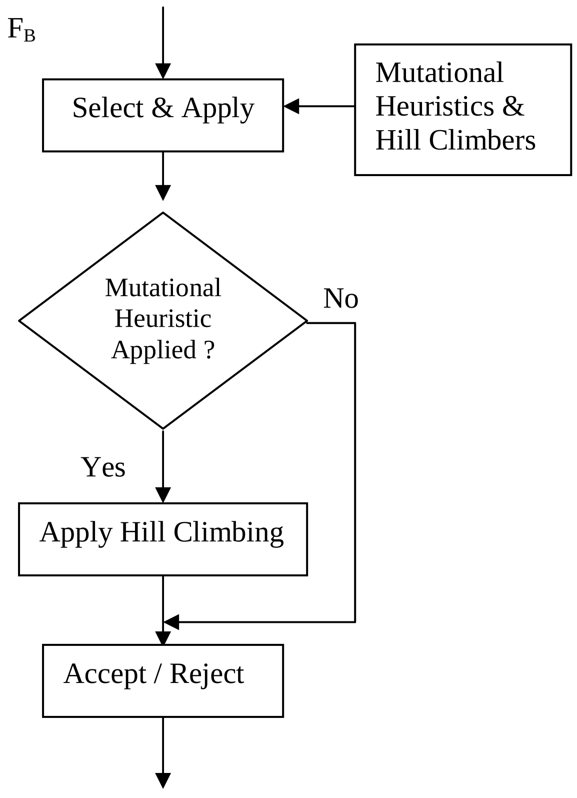


Figure 4.1 : Hyper-heuristic framework F_A



F_B
Select & Apply
Mutational Heuristics & Hill Climbers
Mutational Heuristic Applied ?
No
Yes
Apply Hill Climbing
Accept / Reject

Figure 4.2 : Hyper-heuristic framework F_B

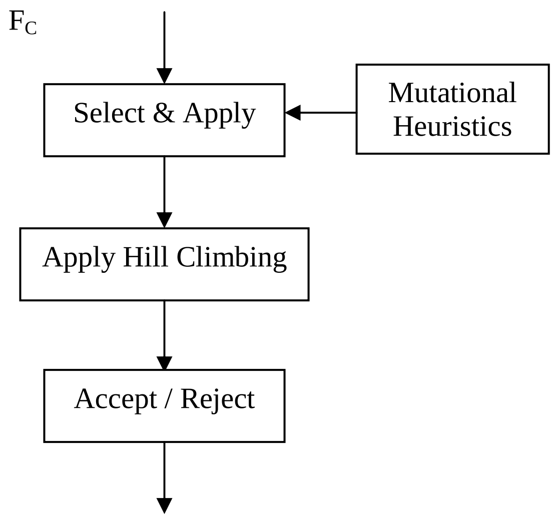


Figure 4.3 : Hyper-heuristic framework F_C

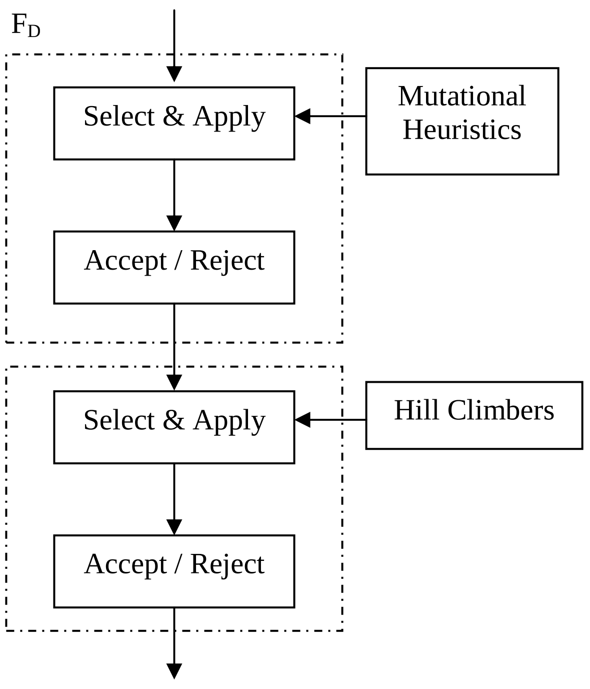


Figure 4.4 : Hyper-heuristic framework F_D

Experimental results showed that the usage of hill climbers has a positive impact on the quality of the solution, and it was also observed that applying a single efficient hill climber produces better results than using a set of hill climbers with a selection strategy. In this study, the F_C framework achieved the best results among these four hyper-heuristic frameworks [18].

4.2.2.2 Population Based Hyper-heuristics

Population based perturbative hyper-heuristics differ from single point search based hyper-heuristics, in which each individual in the population consists of a sequence of integers, where each integer represents a single low-level heuristic to indicate in which order the heuristics are called. In 2003, Cowling applied this approach to a personnel scheduling problem, and he used a genetic algorithm as the heuristic selection mechanism [26]. This method achieved better results when compared to a genetic algorithm and a memetic algorithm in trainer scheduling problem.

In 2005, Burke employed the ant colony algorithm as a hyper-heuristic to solve the personnel scheduling problem [26]. Each vertex represents a heuristic, and a number of ants are distributed among the vertices to carry candidate solutions. In this method, each ant applies a low-level heuristic at each encountered node to its solution. Burke achieved good results with this method for previously studied problem instances.

4.3 Heuristics to Generate Heuristics

Hyper-heuristics are not only used to select heuristics, but they are also used to generate heuristics. In this approach, hyper-heuristics operate on a set of components to construct heuristics instead of searching the set of complete heuristics. Therefore, at each iteration a new heuristic is created and a solution is obtained using this heuristic. This approach is applied to the following problem domains: Production Scheduling, Traveling Salesman Problem, Cutting and Packing, Function Optimization, Satisfiability, Constraint Satisfaction [26].

Although heuristics created by humans are designed to be effective on a set of problem instances, a heuristic needs to be specialized to achieve the best result for each problem instance respectively. Therefore, the most important advantage of the automated heuristic generation approach is that the heuristic implementation process is able to specialize a heuristic for each instance in a cost effective way. These instance specific adjusted heuristics would produce better results when compared with human created heuristics. Experimental results also verify this approach.

In 2005, Ho and Tay applied a genetic programming algorithm to the job shop scheduling problem. This algorithm acts as a hyper-heuristic and evolves composite dispatching rules [26]. In 2007, Jakobovic used the same approach in the parallel machine scheduling problem. These dispatching rules are functions, which are responsible for assigning scores to the jobs with respect to the problem state. When a machine completes its job, an evolved dispatching rule works for each job in the machines queue separately, so that each job obtains a result as its score. The job with the highest score in the queue is assigned to the machine [26].

In 2005, Koza and Poli concluded with their experiments, that the best evolved dispatching rule achieves better results on over 85% of all the problem instances [26]. These experimental results also verified that genetic programming is able to create composite heuristics containing multiple heuristic components, which are more efficient than human created heuristics and reusable on different problem instances.

In 2005, Oltean employed a genetic programming hyper-heuristic to generate evolutionary algorithms [26]. This method was successfully applied to the traveling salesman problem and to function optimization. In this method, each individual consists of a series of instructions which calculate values in a memory array with multiple registers. This array represents an evolutionary algorithm population where each register corresponds to a member of the population. Genetic operators are used as the instructions and they are performing their tasks on the memory array. An example for an instruction is given in Eq. (4.7), where the crossover operator is applied to two members from the memory array. This algorithm also achieved successful results when compared with the human made heuristics.

$$reg[1] = crossover(reg[7], reg[3]) \qquad \textbf{(4.7)}$$

$$reg[3] = mutate(reg[5]) \qquad \textbf{(4.8)}$$

In 2007, Burke applied genetic programming as a hyper-heuristic for one dimensional bin packing problem [26]. This hyper-heuristic method produces heuristics containing arithmetic operators and properties of the pieces and bins. At each iteration, a piece is placed into a bin. For each piece, created heuristics are applied to the bins and the bin with the highest score is selected to pack the piece in.

Poli et al (2007) also made use of genetic programming in one dimensional bin packing [26]. However, this study was based on the remaining space size of each bin, when placing pieces into these bins. According to this approach, when a piece is placed into a selected bin, the remaining space should not be smaller than the size of the smallest piece, which is not packed yet.

5. EXPERIMENTAL STUDY

In this study, hyper-heuristic methods with various heuristic selection and move acceptance strategies are applied to the UCP. In the heuristic selection phase, six different strategies are applied. These are:

- Simple Random (SR),
- Random Descent (RD),
- Random Permutation (RP),
- Random Permutation Descent (RPD),
- Choice Function (CF) ,
- Greedy (GR).

As move acceptance criterion, four different strategies are used:

- All Moves (AM),
- Only Improving (OI),
- Improving and Equal (IE),
- Great Deluge (GD).

Totally 24 combinations of the above listed heuristic selection and move acceptance mechanisms are applied to two instances of the UCP, and results obtained by these combinations are analyzed to determine the most efficient strategy combination for this problem.

In the second part of the experiments, the performance of the most efficient hyper-heuristic method is compared with the performance of a genetic algorithm [12] on different data sets, because these two methods use the same genetic operators. In the third part, reported results of six benchmark problem instances are given. An analysis is made to find out how the problem size affects the performance of each optimization method. In the last part, real world data is used to verify that the proposed hyper-heuristic algorithm can be successfully applied to real world problems.

The proposed algorithms are coded in the C language. For all of our experiments, we used a single PC (2.13 GHz quad core processor with 2GBytes of main memory).

5.1 Proposed Approach

In the proposed approach, each candidate solution consists of binary digits with a length of T*N, where N is equal to the number of units and T is equal to the number of time slots. Figure 5.1 illustrates an example of encoding a candidate solution. Each hour contains N binary digits. The values 0 and 1 indicate that the generator is off or on for the corresponding time slot.

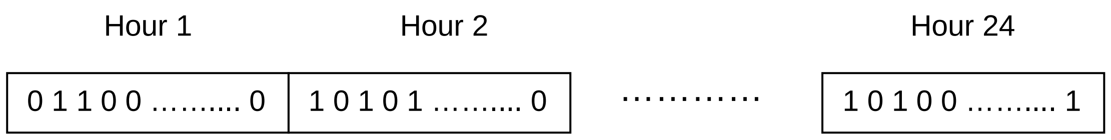


Figure 5.1 : The binary representation of a candidate solution

Seven heuristics are used during this iterative search process. The first two heuristics are classic mutation operators with a probability of 1/L and 2/L, where L is the solution length.

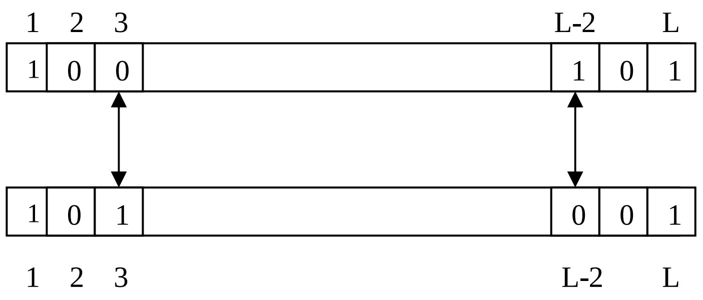


Figure 5.2 : Mutation operator

The third heuristic is the swap-window operator [12]. This operator selects two power units, a time window of width w hours and a window position between 1 and (H-w) randomly. Then, the digits of these two units in this time window are exchanged as shown in Figure 5.3.

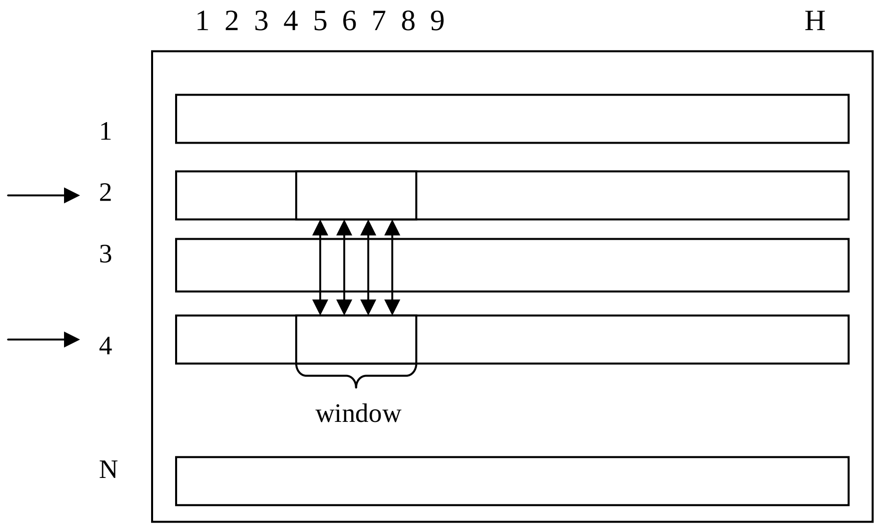


Figure 5.3 : Swap-window operator

Fourth heuristic is the window-mutation operator. This operator selects one unit, a time window of width w and a window position between 1 and (H-w). All 0s in this time window are turned to 1s, and all 1s are turned to 0s [12]. The solutions before mutation and after mutation are depicted in Figure 5.4.

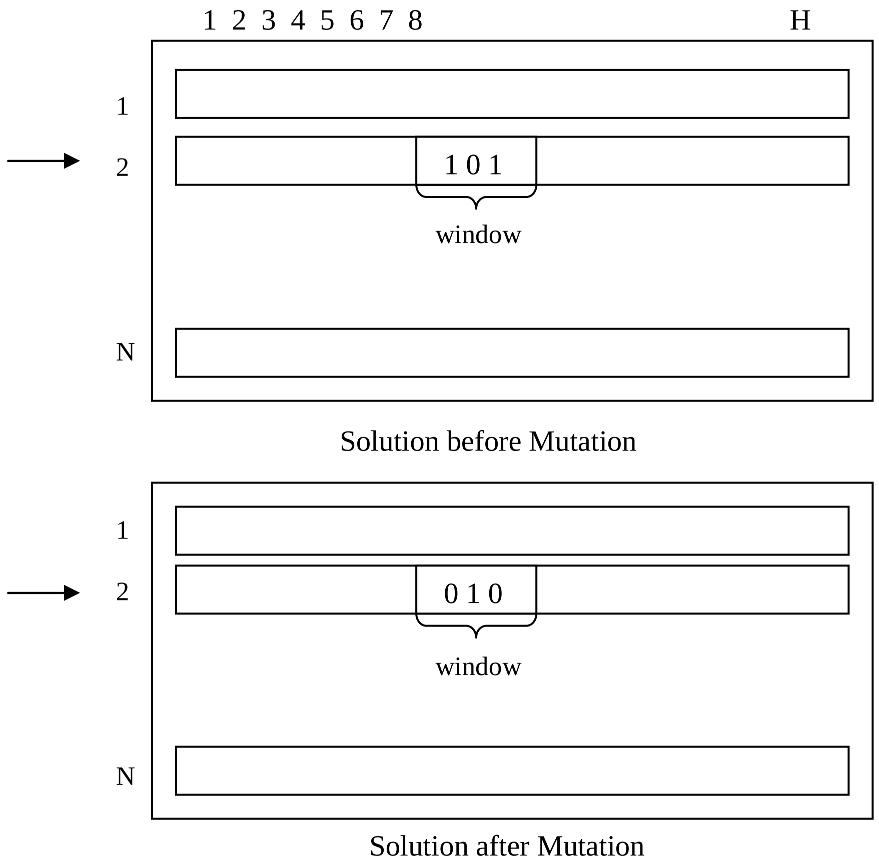


Figure 5.4 : Window-mutation operator

Swap-mutation heuristic is a hill climbing operator. For each time slot of the scheduling period, one of the two operations is performed on the candidate solution with equal probability [12].

- Two units are selected randomly, and the digits for the corresponding hour are exchanged.
- A unit is selected randomly and the corresponding digit for the given hour is turned from 0 to 1 or from 1 to 0.

If the operation results in a better solution, the new solution is replaced with the old one; otherwise, the next operation is performed on the old solution. The behavior of this heuristic is shown in Figure 5.5.

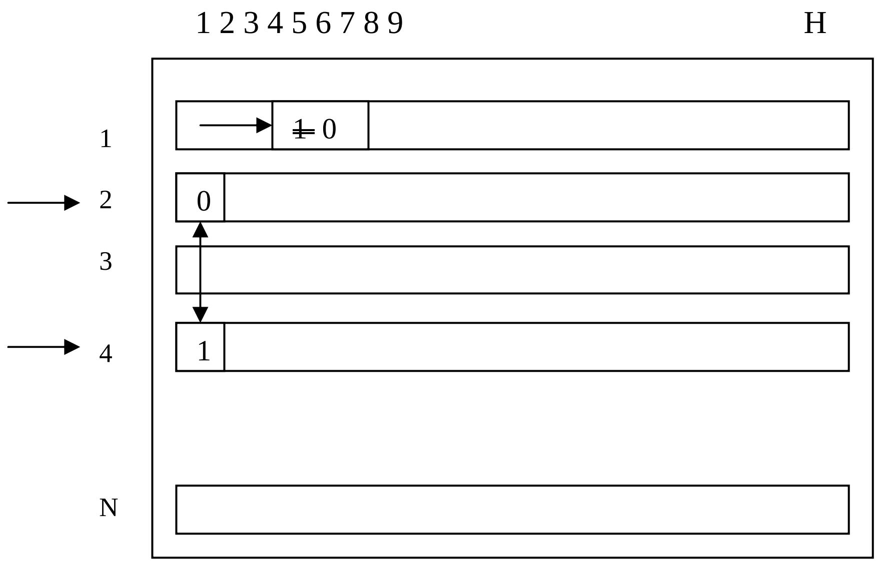


Figure 5.5 : Swap-mutation operator

Swap-window hill-climb heuristic is another hill climbing operator. Two units and a time window of width w hours is selected between 1 and H. The starting point of the time window is the first hour of the schedule. The digits of the two units in the time window are exchanged. If the fitness value of the resultant candidate solution is better, then the solution is kept; otherwise, previous solution is used in the next iteration. For the next iteration, the window is shifted one hour up, and the digit replacement procedure continues until the window reaches the last hour of the schedule [12]. Figure 5.6 shows an example of this heuristic.

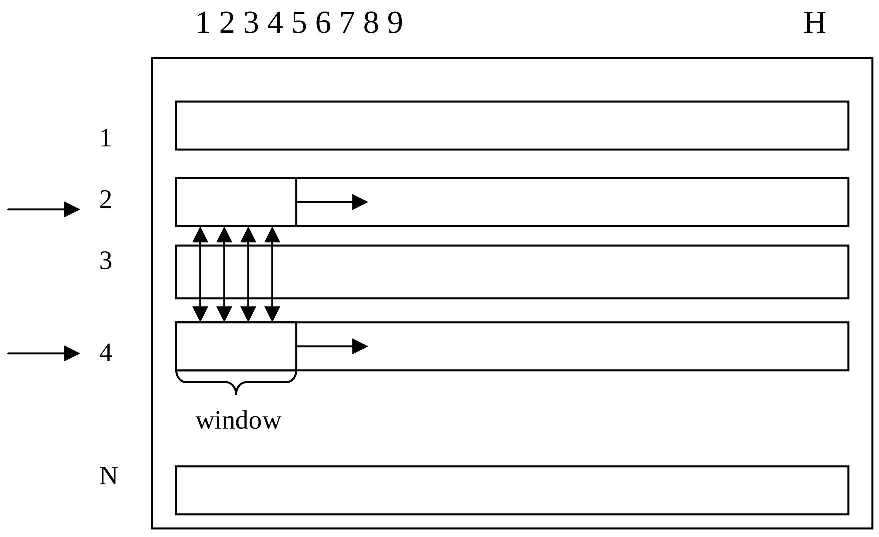


Figure 5.6 : Swap-window hill-climb operator

The last heuristic is the Davis Bit hill climbing operator. A permutation array is created randomly. A power generation unit and a time slot are selected according to this permutation array. Then, the corresponding digit is changed from 0 to 1 or from 1 to 0 [18].

As hyper-heuristic framework, the F_B framework is used [18] as explained in section 4. In this framework, a heuristic is selected from a set of mutational heuristics and hill climbers. If the selected heuristic is a mutational heuristic, then a hill climber is applied to the solution; otherwise, only a hill climber is applied and the fitness value of the resultant solution is calculated. This solution is either accepted or rejected with respect to its fitness value and the selected move acceptance strategy.

In Algorithm 8, RPD heuristic selection strategy is used along with the OI move acceptance criterion. In the first step of this algorithm, an initial solution is created randomly and its fitness value is calculated. After that, the iterative search process starts by applying the first heuristic according to a predefined permutation of seven heuristics. If the heuristic does not have the hill climbing capability, Davis Bit hill climbing operator is applied to the candidate solution. The fitness value of the resulting solution is calculated. This new solution is replaced with the old one and the same heuristic is applied to the solution again, if the fitness value is better; otherwise, the old candidate solution is kept and the next heuristic in the permutation array is applied to the old solution. This process continues until the stopping criterion is met.

Algorithm 9 Proposed Hyper-heuristic Algorithm

```
 1: randomly create an initial solution;
 2: evaluate the initial solution;
 3: create a random permutation of all seven heuristics;
 4: repeat
 5:     select the first heuristic in the permutation;
 6:     repeat
 7:         repeat
 8:             apply the selected heuristic to the solution;
 9:             if heuristic does not contain hill climbing
10:                 apply Davis Bit hill climbing;
11:             endif;
12:             calculate the fitness value;
13:             if the fitness value is better
14:                 accept the new solution;
15:             endif;
16:         until the fitness value is no more improved
17:         select the next heuristic;
18:     until the last heuristic in the permutation is applied
19: until stopping criterion is met
```

5.2 Experimental Setup

Proposed hyper-heuristic algorithm is tested with six benchmark problems taken from literature and with real world data obtained from the Turkish interconnected power system. Additionally, the performance of the hyper-heuristic algorithm is compared with the performance of a genetic algorithm [12] in several problem instances.

In the first step of the genetic algorithm, the initial population consisting of M individuals is generated randomly and the fitness value of each individual is calculated. After two individuals are selected according to the roulette wheel parent selection algorithm, crossover and mutation operators are applied with certain probabilities [12]. This procedure is repeated until M new individuals are created. These new individuals replace the parents except that the best individual of the previous generation is also carried to the next generation [12].

In the genetic algorithm, selection and crossover operators result in population convergence; whereas, mutation is used to maintain diversity. To increase the effectiveness of the search, premature convergence and excessive diversity should be prevented. Therefore, search process is monitored by collecting statistical information from the individuals. The crossover probability is kept between 0.4 and

0.9 and the mutation probability is kept between 0.004 and 0.024 [12]. When premature convergence occurs, the crossover probability is decreased by 0.1 and the mutation probability is increased by 0.004 [12]. To prevent the excessive diversity, the crossover probability is increased by 0.1 and the mutation probability is decreased by 0.004 [12]. Swap-window and window-mutation operators are applied to all the population members with a probability of 0.3. Swap-mutation and swap-window hill-climb operators are also using the same probability rate except that they are only applied to the best individual of every generation [12].

Great Deluge move acceptance strategy is one of the four strategies, which is used in this proposed approach. When applying this strategy to the hyper-heuristic method, fitness value of the initial solution needs to be calculated to determine the down rate [30]. Initial solution is created using the Priority List method [12], which is explained in section 3. The power generation units are ranked in ascending order of the average full load cost, so that cost effective units are committed first, and other units are set to online status according to this order until the load demand is met for each time slot respectively. Operational constraints are satisfied with this method, but resulting schedules contain high power production costs.

In System 1 and the Turkish interconnected power system, as used in [1,14], and in System 2, System 3, System 4 and System 5, as used in [1, 3, 4, 12, 14], the best, the average and the worst case values are reported over 20 runs of the program. For System 6 [3, 12, 14], these values are determined over 10 runs of the program due to time constraints.

System 1 consists of 10 units and 24 hours. Detailed data set of System 1 is given in Appendix A.1. The data for the other problem instances are obtained by repeating the number of power generation units two, four, six, eight and ten times respectively, as also done in [1, 3, 4, 12, 14]. Therefore, System 2 contains 20 units and 24 hours, System 3 contains 40 units and 24 hours, System 4 contains 60 units and 24 hours, System 5 contains 80 units and 24 hours and System 6 contains 100 units and 24 hours.

To increase the efficiency of the hyper-heuristic approach, demand and up/down penalty coefficients are set to 100000 to prevent infeasible candidate solutions. The determination of these values relies on the investigation of the previously obtained experimental results.

Number of iterations per run is set as 1000, 5000, 10000, 15000, 20000, 25000 for System 1, System 2, System 3, System 4, System 5, System 6, respectively. These numbers are determined empirically.

5.3 Experimental Results

In the first part of this section, System 2 and System 3 data sets are used to determine the most efficient strategy pair for heuristic selection and move acceptance phases. Statistical tests are applied to resultant solutions to compare the performance of each combination. These tests are performed at a confidence level of 0,95. The second and the third parts contain performance comparison test results of a hyper-heuristic algorithm with other optimization techniques using seven different problem instances.

5.3.1 Performance Comparison of Different Hyper-heuristic Combinations

Table 5.1 and 5.2 contain experimental results, which are obtained, when the OI move acceptance scheme is applied with 6 different heuristic selection strategies. In these two data sets, RPD heuristic selection strategy achieves the overall best results. RP method obtains the same result for System 2, but it obtains the second best result for System 3. The difference percentage between RP and RPD is only 0,036 %. SR and RD follow these methods and GR obtains the poorest results in these two data sets.

Table 5.1: Cost results for System 2 with the OI move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	RPD	1125997	1128831	1127474
2	RP	1126231	1128931	1127689
3	SR	1127253	1129911	1128435
4	RD	1127253	1129563	1128572
5	CF	1127683	1148563	1133976
6	GR	1129038	1138217	1132815

Figure 5.7 shows the box-whisker plot of the results for System 2 with the OI move acceptance strategy. The one-way analysis of variance (ANOVA) test is used to determine whether the fitness values are the same across different heuristic selection methods. The p-value for this experiment is zero to four decimal places. This value indicates that the fitness values vary from one heuristic selection method to another.

Figure 5.8 shows the multiple comparison results for System 2 with the OI move acceptance strategy. According to the results of this test, the mean values of GR and CF are significantly different from RP, RPD, SR and RD.

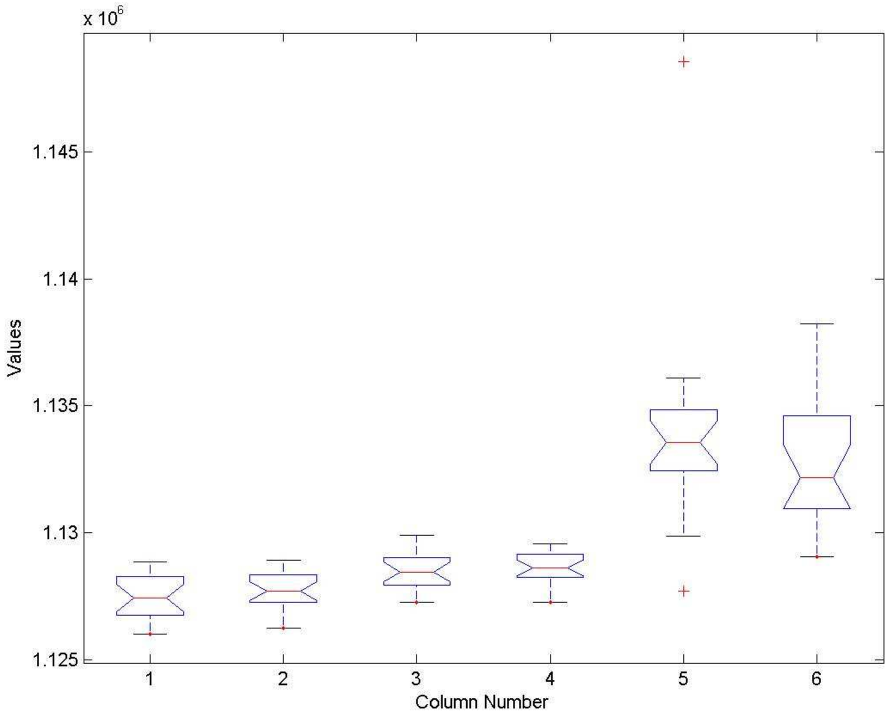


Figure 5.7 : Box-whisker plot for System 2 with OI

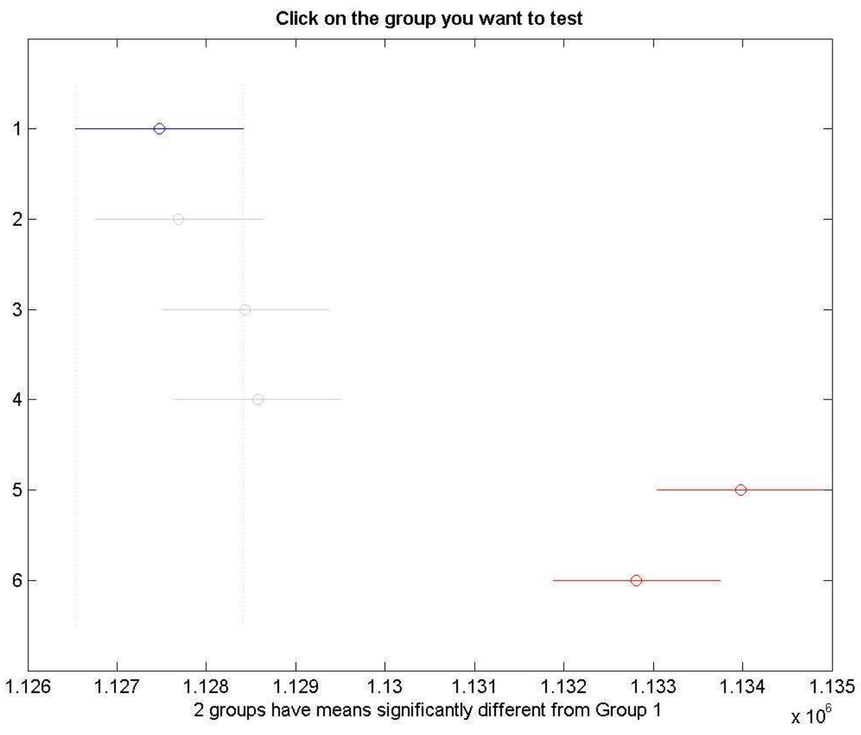


Figure 5.8 : Multiple comparison results for System 2 with OI

Table 5.2: Cost results for System 3 with the OI move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	RPD	2248284	2253971	2250434
2	RP	2249099	2253057	2250835
3	SR	2250116	2255899	2253378
4	RD	2250875	2253851	2252456
5	CF	2253743	2279271	2264473
6	GR	2255837	2267460	2258998

Figures 5.9 and 5.10 show the box-whisker plot and the multiple comparison test results for System 3, when the OI move acceptance strategy is applied. ANOVA function returns 0 as the p-value. From these results, it can be observed that the fitness values from different heuristic selection methods are not the same. The interval between the lowest and the highest fitness values of the CF method is very long, when compared with other methods. According to the multiple comparison test results, CF and GR methods have significantly different mean values from RPD. However, there is no statistically significant difference in terms of the mean values obtained by RP, RPD, SR, RD heuristic selection methods in System 2 and System 3, when either OI or IE is used as the move acceptance criterion.

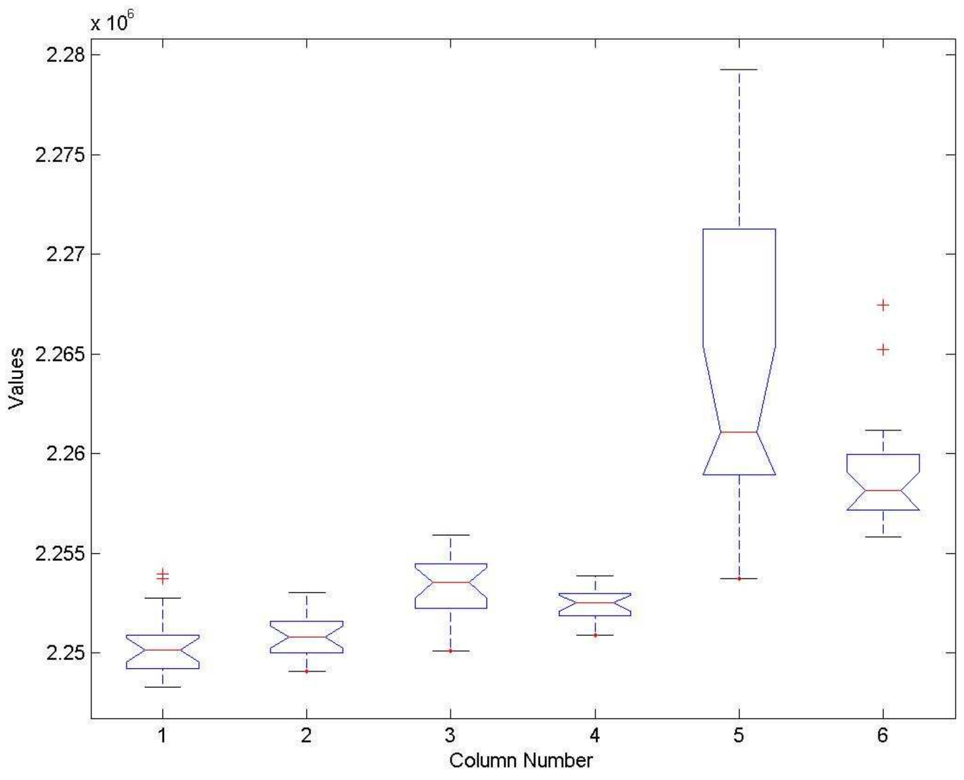


Figure 5.9 : Box-whisker plot for System 3 with OI

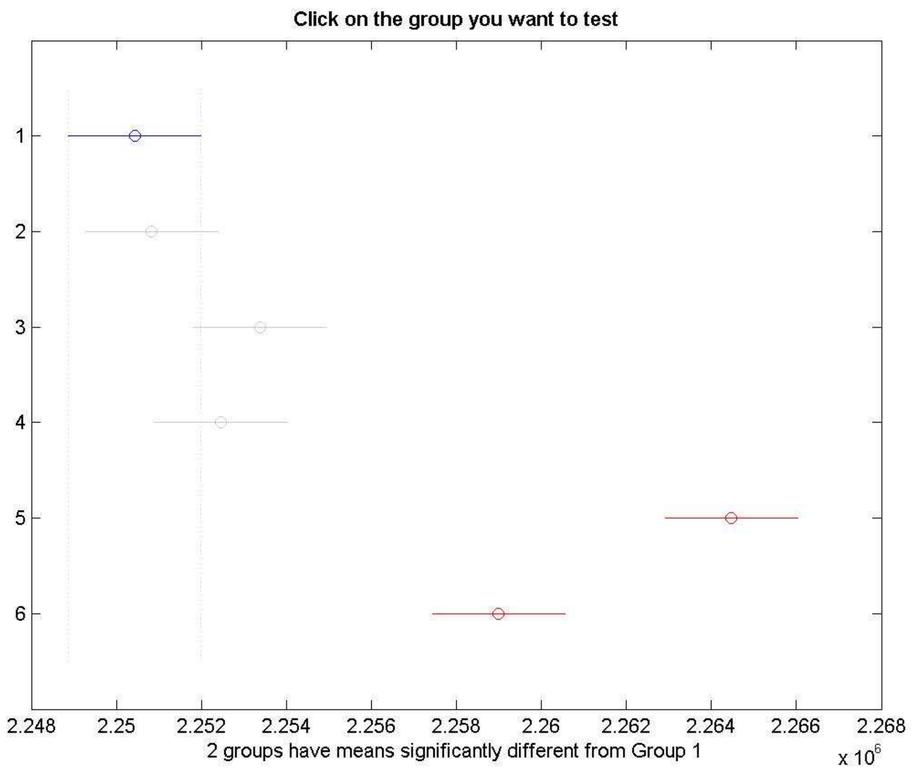


Figure 5.10 : Multiple comparison results for System 3 with OI

The results in the Tables 5.3 and 5.4 are obtained using the IE move acceptance criterion. RPD and SR achieve the best result for System 2, but when the data size of the problem increases, the performance of SR is decreased accordingly. However, RPD proves its consistency by obtaining the best result in System 3 as well. RD achieves the second best result and GR again achieves the poorest results among these six heuristic selection methods. Arithmetic average of the fitness values obtained by the RPD method is better than the average results of other methods in these two data sets.

Table 5.3: Cost results for System 2 with the IE move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	RPD	1126231	1129039	1127381
2	SR	1126231	1129317	1128190
3	RD	1127065	1130338	1128500
4	RP	1127253	1129837	1128510
5	CF	1128041	1147346	1135070
6	GR	1130520	1136545	1133359

Table 5.4: Cost results for System 3 with the IE move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	RPD	2250070	2252741	2251331
2	RD	2250090	2254164	2252311
3	RP	2250837	2253019	2251510
4	SR	2250875	2253881	2251794
5	CF	2252492	2284777	2263464
6	GR	2255599	2263901	2260230

Looking at the results in Figures 5.11 to 5.14, we can see that the fitness values obtained with the IE move acceptance criterion for System 2 and System 3 vary from one heuristic selection method to another. The p-value returned by ANOVA is zero for both of the systems. Multiple comparison test results show, that CF and GR have significantly different mean values from the rest of the methods. Box-whisker plots also show that these two methods have longer intervals between their best and worst results.

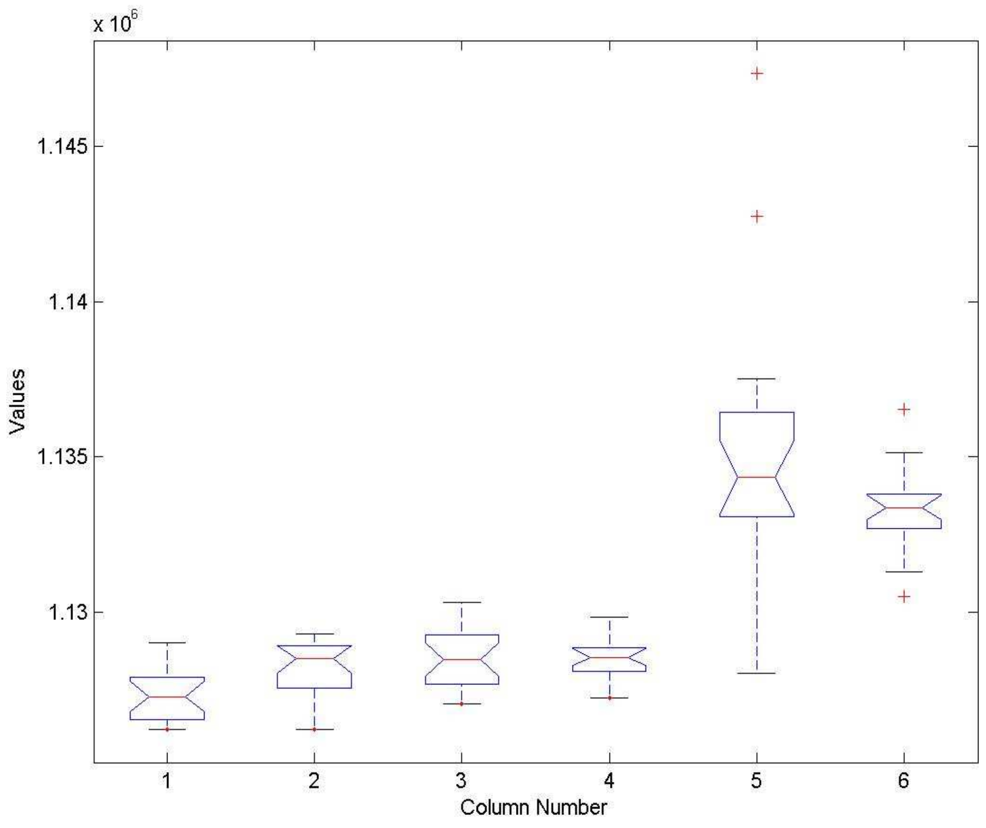


Figure 5.11 : Box-whisker plot for System 2 with IE

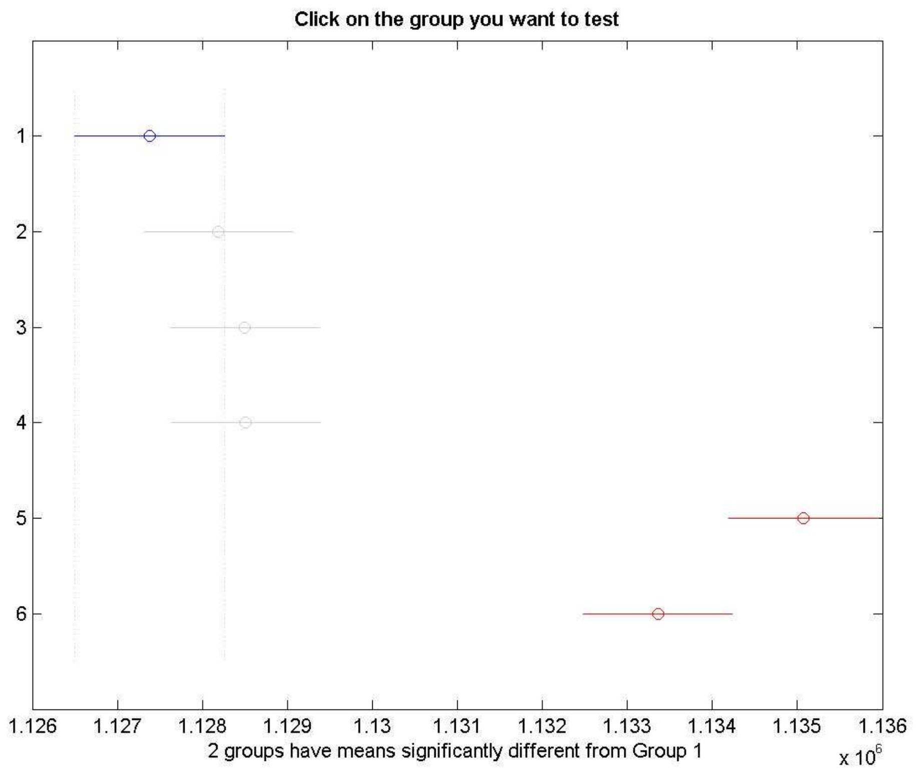


Figure 5.12 : Multiple comparison results for System 2 with IE

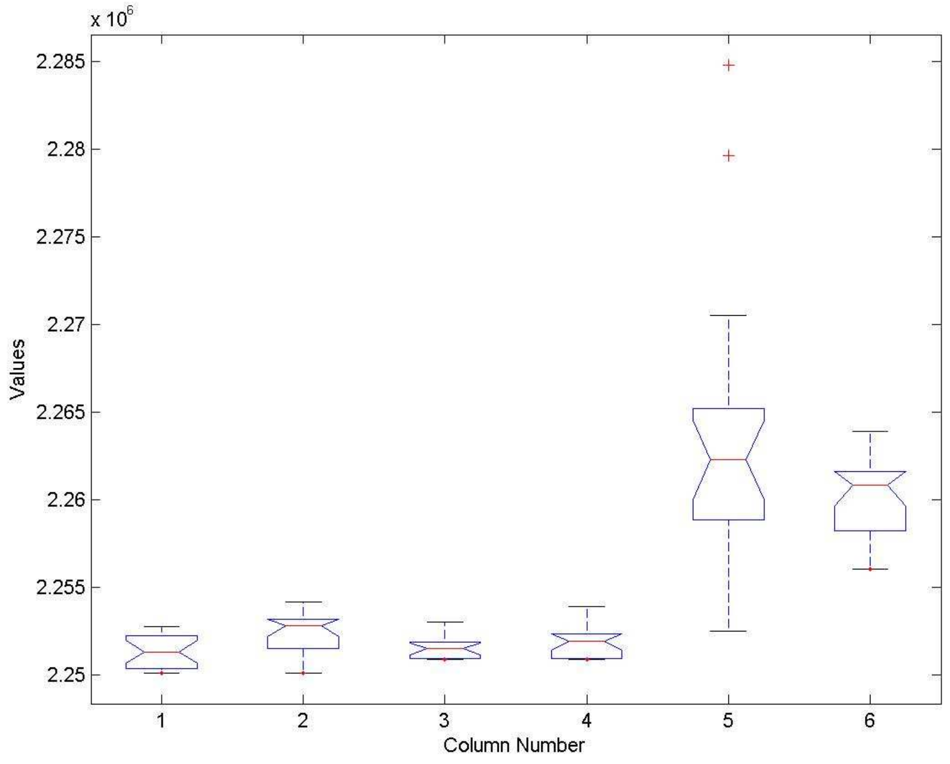


Figure 5.13 : Box-whisker plot for System 3 with IE

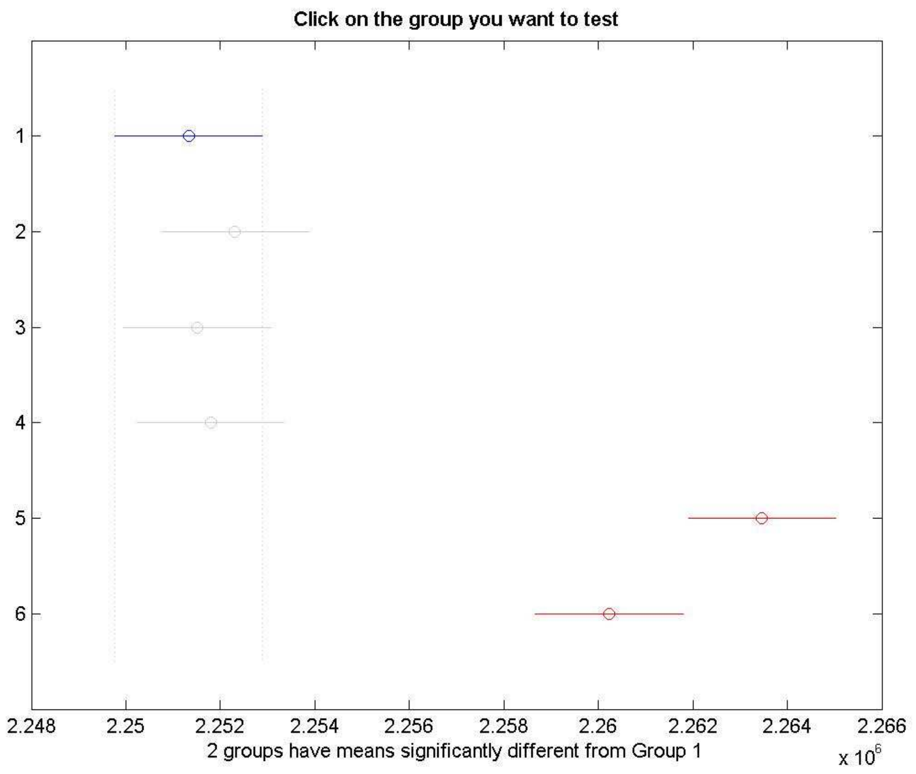


Figure 5.14 : Multiple comparison results for System 3 with IE

When the GD move acceptance strategy is applied, the RPD method obtains the best result in System 2. CF achieves the second best result, but its average result is the poorest one. In System 3, RPD, SR and RD methods achieve the same result; however, CF achieves the poorest best and average results in the increased data set. The difference percentage between RPD and CF is 0,0108% for System 2 and 0,0931% for System 3.

Table 5.5: Cost results for System 2 with the GD move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	RPD	1125997	1129390	1127673
2	CF	1126119	1134568	1128820
3	RP	1126231	1129404	1127944
4	SR	1126231	1129837	1128267
5	RD	1127055	1129837	1128343
6	GR	1127252	1129135	1128345

Figure 5.15 shows the box-whisker plot for System 2, when the GD move acceptance strategy is used. ANOVA function returns 0.0309 as the p-value. This indicates that one heuristic selection method outperforms the other in the fitness values of the solutions it produces.

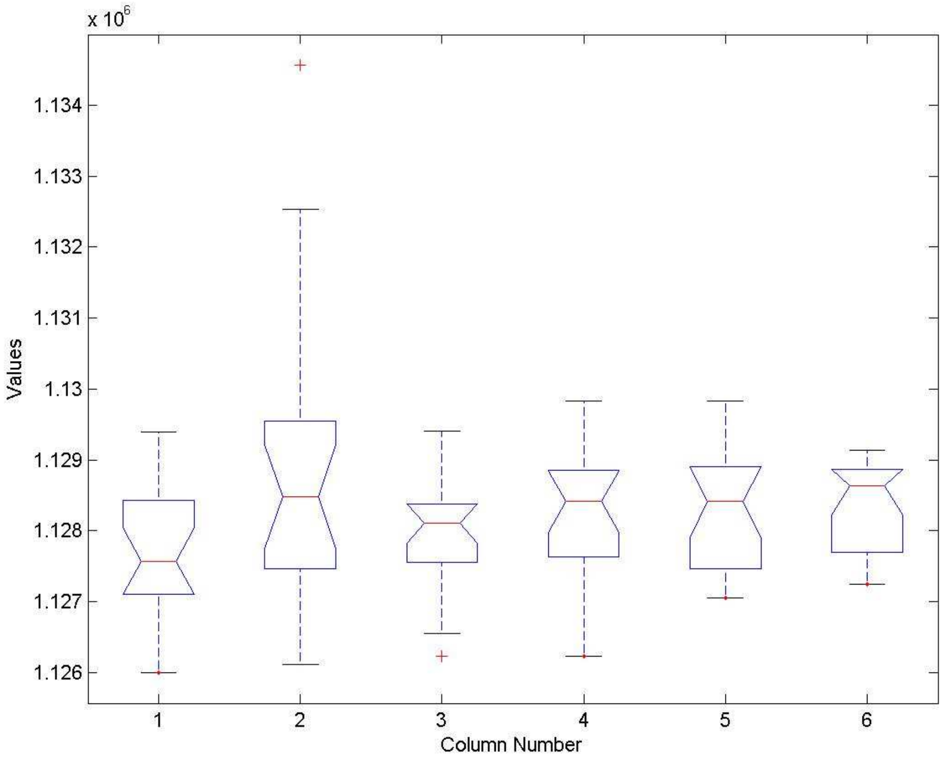


Figure 5.15 : Box-whisker plot for System 2 with GD

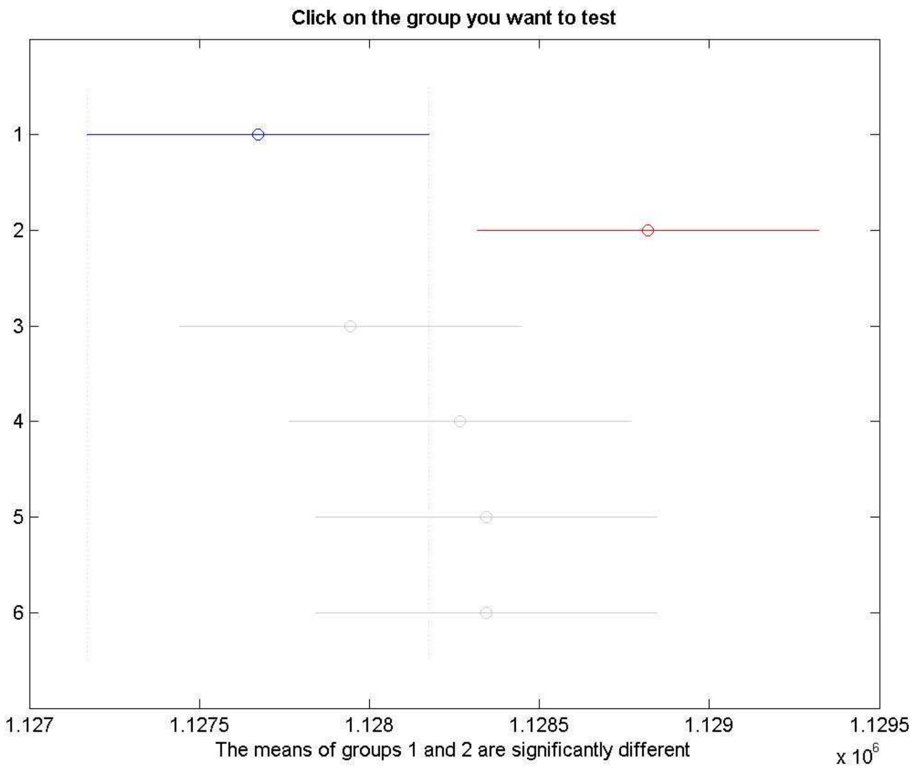


Figure 5.16 : Multiple comparison results for System 2 with GD

Figure 5.16 shows the multiple comparison results. According to these results, RPD and CF have significantly different mean values.

Table 5.6: Cost results for System 3 with the GD move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	RPD	2249099	2252103	2251066
2	RD	2249099	2253712	2251471
3	SR	2249099	2254148	2251906
4	RP	2249576	2253223	2251336
5	GR	2250904	2259784	2254414
6	CF	2251195	2272279	2259073

Figure 5.17 illustrates the box-whisker plot of System 3. In System 3, the p-value returned by ANOVA is 1.1102 e $^{-15}$. Since it is a very small value, it shows that there is a statistical difference between the fitness values of the experimented methods. According to the multiple comparison test results, RPD and CF have significantly different mean values as shown in Figure 5.18.

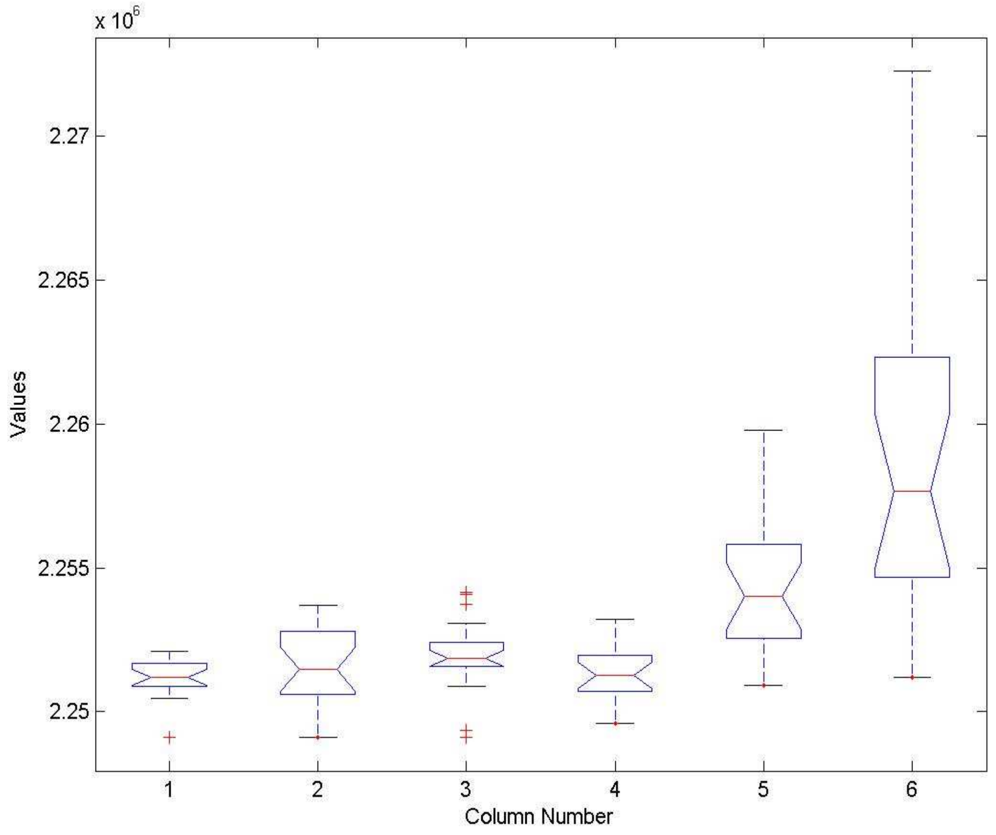


Figure 5.17 : Box-whisker plot for System 3 with GD

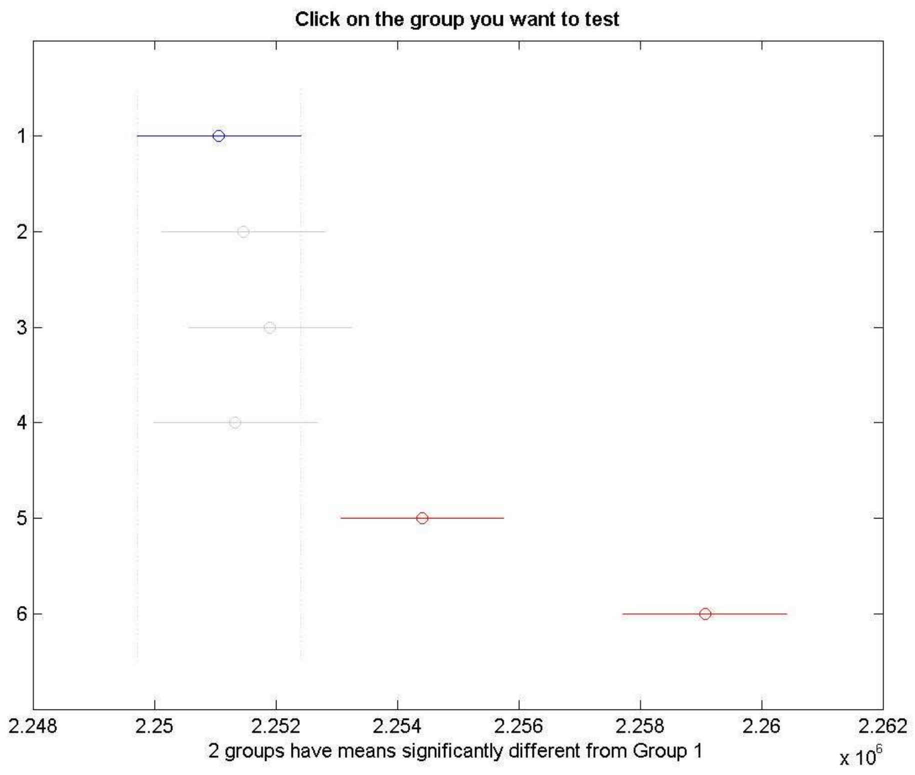


Figure 5.18 : Multiple comparison results for System 3 with GD

GR method produces poor results with the first three move acceptance criteria, since it always selects the most efficient heuristic at each iteration. Heuristics with hill climbing capability are able to obtain better results than other heuristics; therefore, GR method mostly selects hill climbers among seven heuristics at each run of the experiment. Diversity can not be provided effectively with this method and this results in getting stuck at a local optimum.

AM is not an efficient move acceptance strategy, since it accepts all non-improving moves without any limitation. GR achieves the best results with AM in these two data sets, because in GR all heuristics are applied to the solution and the heuristic obtaining the best fitness value is selected. At each iteration hill climbers are applied to the solution. Since hill climbers do not accept a worsening move, AM selects a candidate solution either with a better fitness value or at least with the same fitness value as the previous one.

According to the results in Table 5.7, CF achieves the second best result in System 2. CF method selects a heuristic with respect to the score of each heuristic from previous runs based on the quality of the solution, so that it applies the most efficient heuristic at each iteration during the search. RPD obtains better results than RP and RD obtains better results than SR, since they apply a heuristic to the solution again, if the heuristic causes an improvement in the fitness value of the solution.

Table 5.7: Cost results for System 2 with the AM move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	GR	1135972	1182232	1157148
2	CF	1137093	1180722	1158591
3	RPD	1140067	1180180	1160381
4	RP	1141958	1180711	1161860
5	RD	1142190	1184874	1163611
6	SR	1152371	1183624	1165224

Box-whisker plot of System 2, obtained using the AM move acceptance strategy, is shown in Figure 5.19. The p-value, 0.3145, does not indicate statistically significant differences between the fitness values of different heuristic selection methods. In Figure 5.20, it can be easily observed that the mean values of CF, RPD, RP, RD and SR are not significantly different from GR.

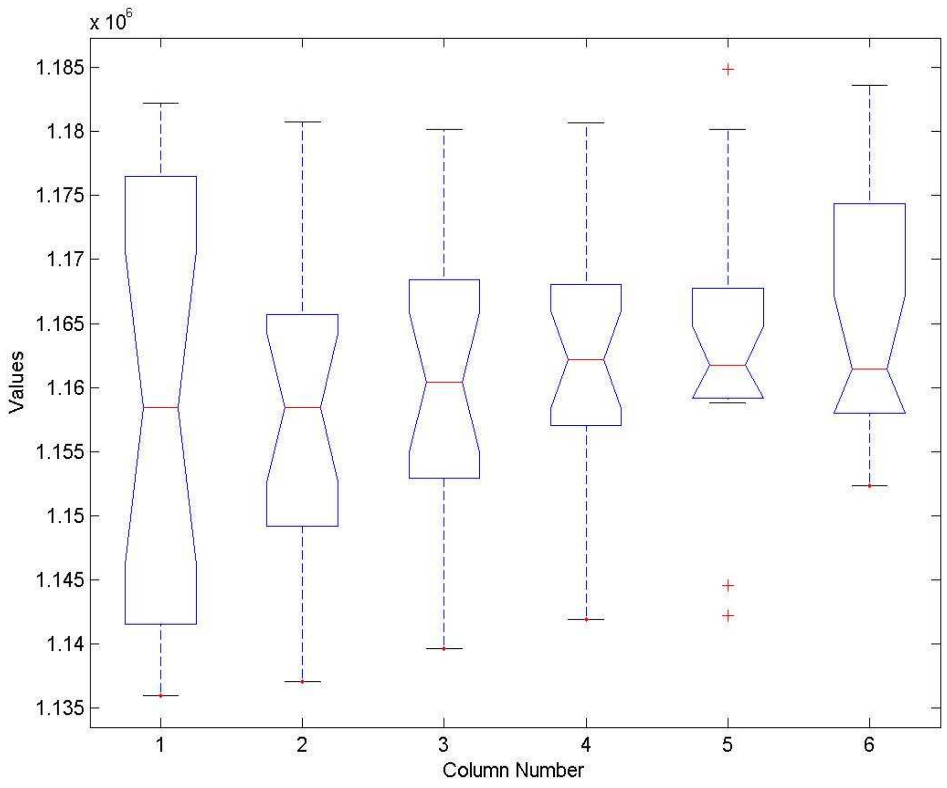


Figure 5.19 : Box-whisker plot for System 2 with AM

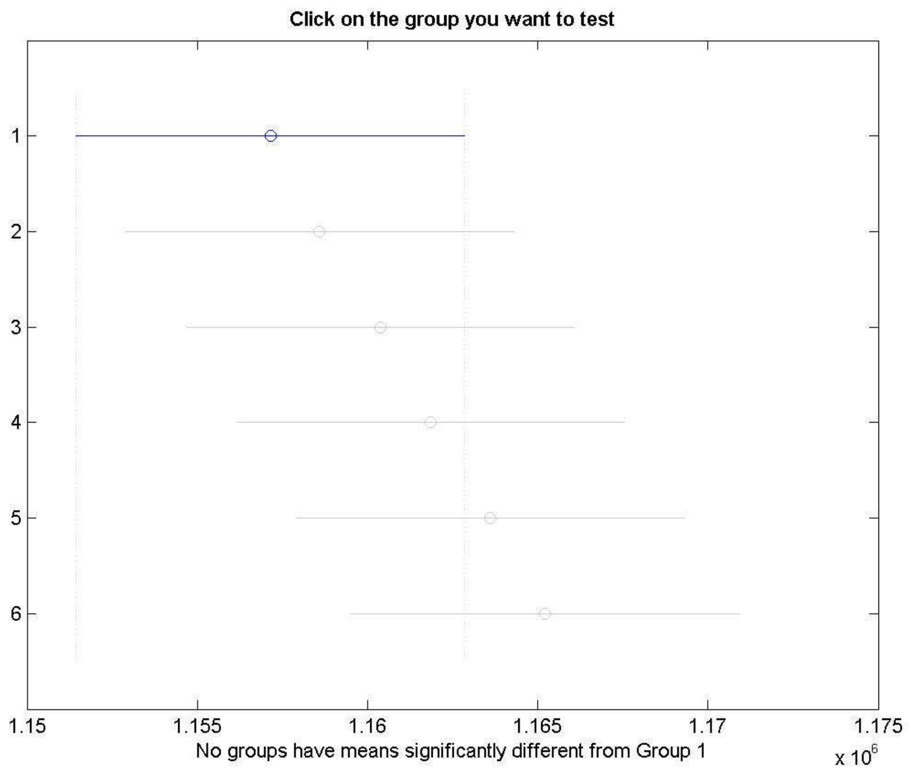


Figure 5.20 : Multiple comparison results for System 2 with AM

Table 5.8 shows the experimental results for System 3, when AM is used as the move acceptance criterion. GR again achieves the best result in this data set in front of the CF method.

Table 5.8: Cost results for System 3 with the AM move acceptance criterion

No	Method	Best Result	Worst Result	Average Result
1	GR	2339024	2478087	2402021
2	CF	2341696	2482374	2404243
3	RPD	2348286	2477003	2406051
4	RD	2354096	2481037	2419543
5	RP	2356811	2483620	2418755
6	SR	2383415	2482194	2427434

Figure 5.21 depicts the box-whisker plot for System 3. The p-value, 0. 1793, also verifies that there is no statistically significant difference between the fitness values obtained by different heuristic selection methods. In the multiple comparison test results, as shown in Figure 5.22, the mean value of GR is not significantly different from other methods.

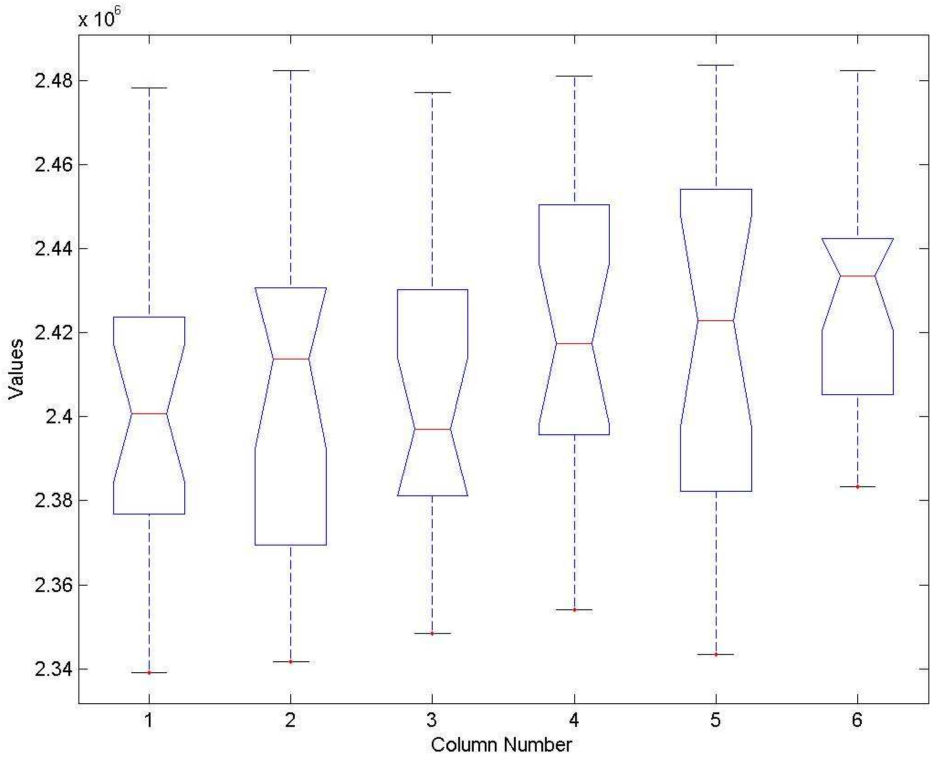


Figure 5.21 : Box-whisker plot for System 3 with AM

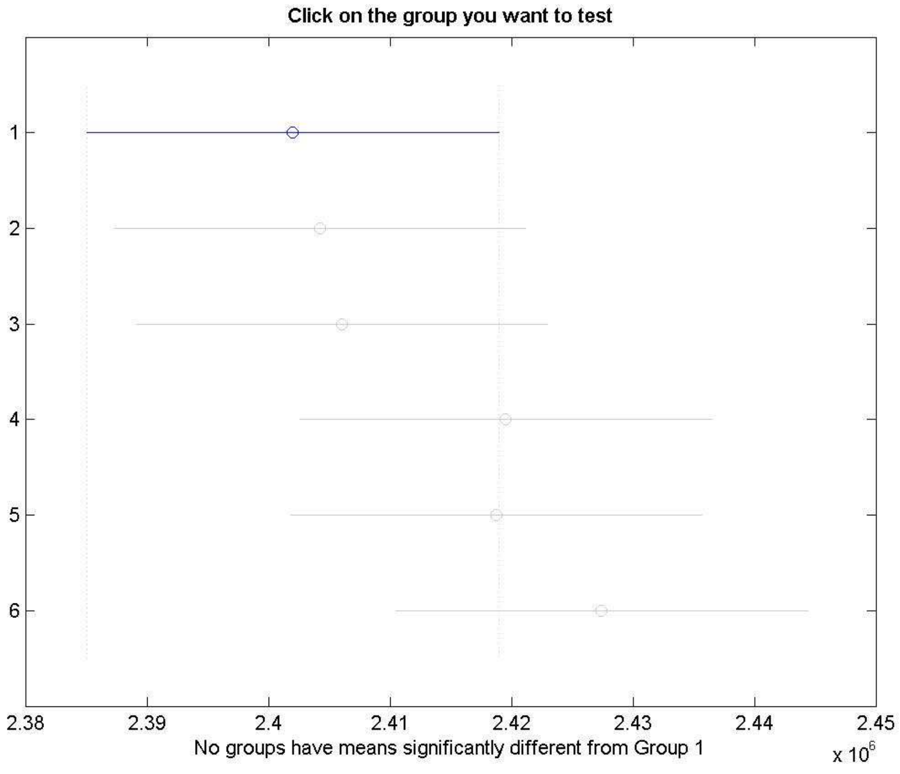


Figure 5.22 : Multiple comparison results for System 3 with AM

In the first part of the experiments, the best result for System 2 is 1125997. This result is obtained with two different heuristic selection and move acceptance strategy pairs. These are RPD-OI and RPD-GD combinations. The best result for System 3 is 2248284, and this result is obtained using RPD-OI strategy pair. Since the data set is increased in System 3, the efficiency of RPD-OI becomes more obvious among all 24 strategy combinations. Four different combinations follow RPD-OI and they produce the same solution with the fitness value of 2249099.

In further parts of this section, this combination is used to compare the performance of the hyper-heuristic (HH) approach with other optimization techniques. Table 5.9 shows the best heuristic selection method and move acceptance criterion combinations for System 2 and System 3.

Table 5.9: The best ten heuristic selection method and move acceptance criterion combinations

Rank	System 2		Rank	System 3	
	Combination	Fitness		Combination	Fitness
1	RPD - OI	1125997	1	RPD - OI	2248284
2	RPD - GD	1125997	2	RP - OI	2249099
3	RPD - OI	1126059	3	RPD - GD	2249099
4	CF - GD	1126119	4	RD - GD	2249099
5	RPD - OI	1126137	5	SR - GD	2249099
6	RP - OI	1126231	6	RPD - GD	2249114
7	RPD - IE	1126231	7	RP - OI	2249118
8	SR - IE	1126231	8	RPD - OI	2249144
9	SR - GD	1126231	9	RD - GD	2249149
10	RP - GD	1126231	10	RPD - OI	2249287

Figure 5.23 shows the distribution of the best fifty solutions of System 2 among several heuristic selection method and move acceptance criterion combinations. Seven of these fifty solutions are produced by RPD-OI pair. RPD-IE, RP-OI, RPD-GD follow this combination with six solutions. Fourteen different combinations obtaining at least one solution are listed in this figure.

Figure 5.24 shows the distribution of the best fifty solutions for System 3. RPD-OI is in the first place with fourteen solutions. RP-OI is the second most effective combination with nine solutions. This figure also verifies that the effectiveness of

RPD-OI becomes more significant with the increased data set. Four of the five combinations listed in Figure 5.23 with at most two solutions are not able to produce a solution which can join the top fifty list of System 3.

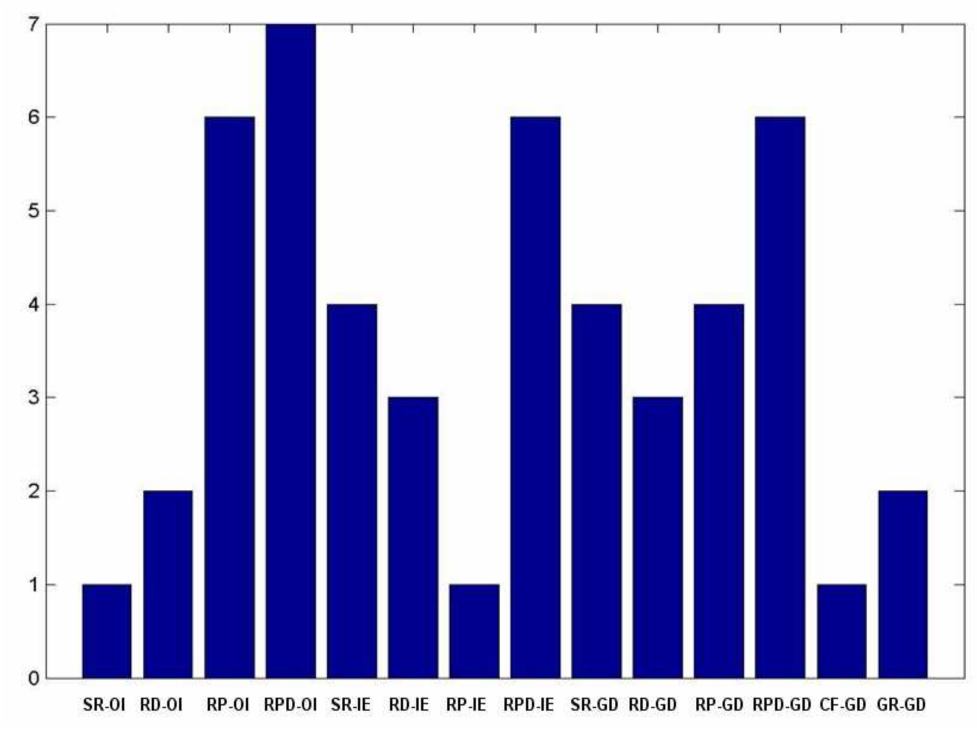


Figure 5.23 : Distribution of the best fifty solutions for System 2

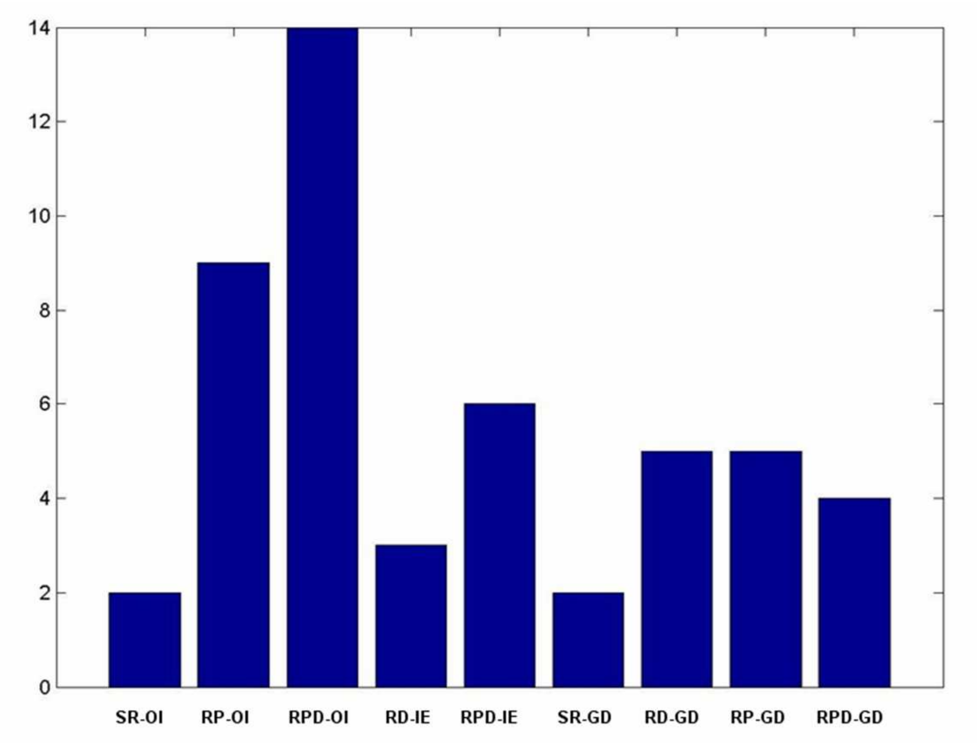


Figure 5.24 : Distribution of the best fifty solutions for System 3

Figure 5.25 shows the iteration number versus the fitness value curves for six different combinations with respect to their best cost results obtained in System 3. CF-IE and GR-IE find the optimum fitness values earlier than other combinations, but they are getting stuck at local optima, since the diversity can not be provided with performance based heuristic selection methods in further parts of the search process. SR-GD, RD-GD and RP-OI achieve the second best result in System 3 and they obtain this solution approximately between the iteration numbers of 3000 and 3600. RPD-OI produces the best solution in System 3 during the 8300th iteration.

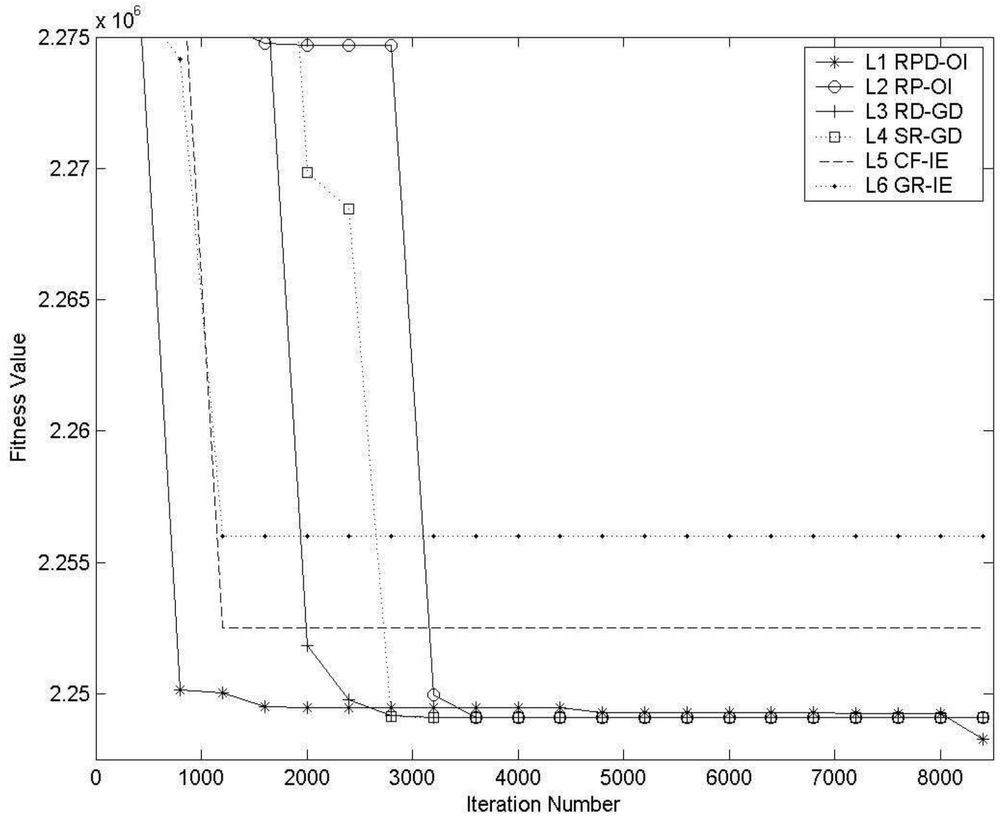


Figure 5.25 : Iteration Number versus Fitness Value curves

Figures 5.26 and 5.27 show multiple comparison test results for System 2 and System 3. These test results are obtained using the cost values produced by six heuristic selection methods along with OI, IE and GD move acceptance schemes. It can be easily observed that CF and GR have significantly different mean values for OI and IE move acceptance schemes in both of the test instances. However, this difference is decreased when these two heuristic selection methods are used along with GD.

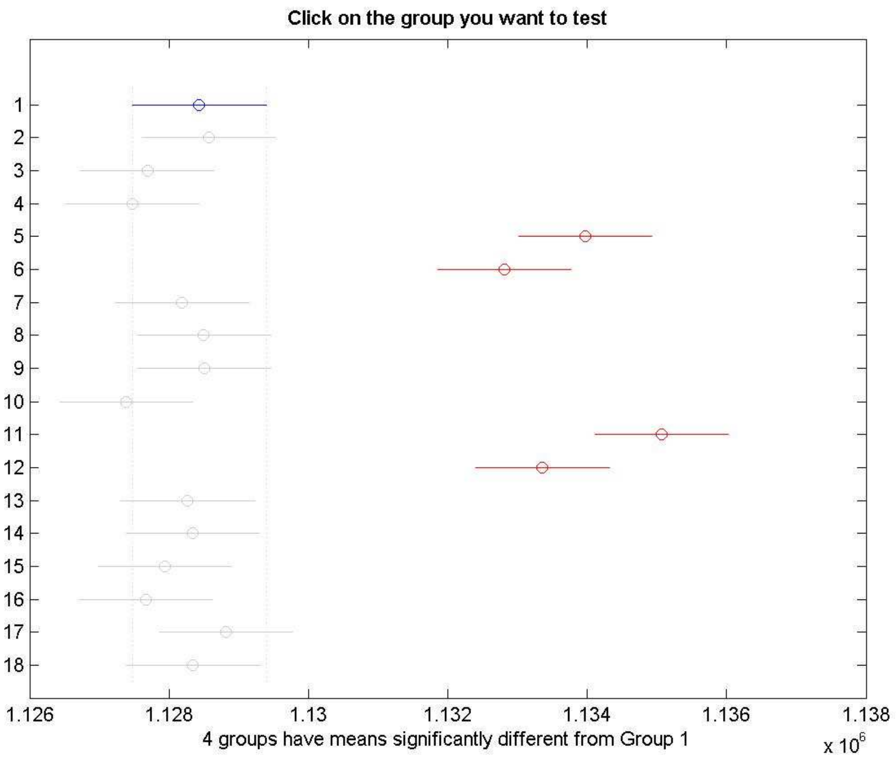


Figure 5.26 : Multiple comparison results for System 2

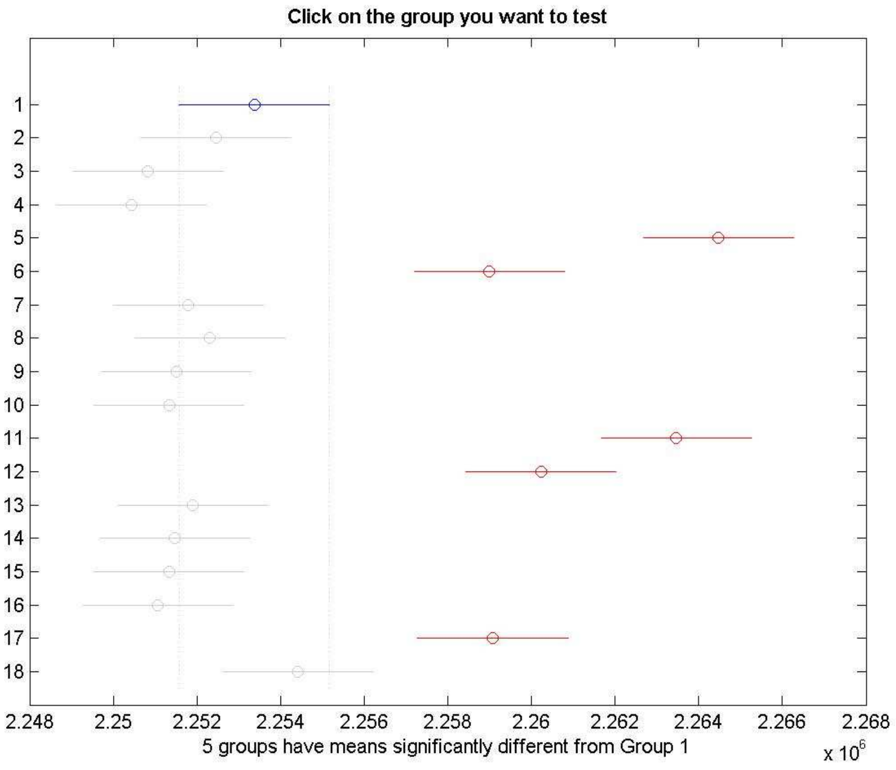


Figure 5.27 : Multiple comparison results for System 3

5.3.2 Performance Comparison of the HH with a Genetic Algorithm

In the second part, the performance of the HH method is compared with the performance of a genetic algorithm (GA2). When applying GA2 to the UCP, premature convergence should be prevented to make the search more efficient. Therefore, the iterative search process needs to be monitored; additionally, mutation and crossover probability rates should be adjusted to prevent the convergence. Although HH method uses the same genetic operators, fine tuning of certain genetic operator probability rates, adaptation and system monitoring are not necessary. This technique creates new candidate solutions with respect to the applied heuristic selection and move acceptance methods.

For System 1, GA2 achieves a better result than HH, but these results are very similar. In the other data sets, HH obtains better results than GA2. When the size of the data set increases, the difference between the fitness values of GA2 and HH becomes more significant. The difference percentage between GA2 and HH is 0,022% for System 2. For System 3, System 4, System 5 and System 6, these values are 0,16%, 0,13%, 0,22% and 0,23% respectively. For larger problems, the solutions using HH are better even in the worst runs than the results obtained by GA2.

Table 5.10: Comparison of the cost results of GA2 and HH methods

Units	GA2		HH	
	best	worst	best	worst
10	565825	570032	565827	567028
20	1126243	1132059	1125997	1128831
40	2251911	2259706	2248284	2253971
60	3376625	3384252	3372040	3376043
80	4504933	4510129	4494452	4499067
100	5627437	5637914	5614360	5620496

5.3.3 Performance Comparison of the HH with other Optimization Techniques

In the third part, the performance of the HH is compared with other optimization algorithms using six benchmark data sets. In literature, only the best, average and worst fitness values are reported for the listed techniques and they are taken from the papers without any modification. The algorithms in the following tables are ranked in decreasing order of their best fitnes values and they are abbreviated as follows:

- LR1 is a Lagrangian Relaxation method as used in [12],
- LR2 is a Lagrangian Relaxation method as used in [14],
- GA1 is a standard genetic algorithm as used in [14],
- GA2 is a genetic algorithm with special operators as used in [12],
- GRA1 and GRA2 are the greedy randomized search methods as used in [10],
- MA and SMA are memetic algorithms as used in [14],
- BDE1 is a binary differential evolution methods as used in [1],
- BDE2 is a binary differential evolution method as used in [16],
- ES is an evolutionary strategies algorithm as used in [1],
- SSGA is steady state genetic algorithm as used in [1],
- ICGA is an integer coded genetic algorithm as used in [20],
- HH is a hyper-heuristic algorithm proposed in this study.

Table 5.11: Cost results for System 1

Algorithm	Best Result	Worst Result	Average Result
LR1	565825	n/a	n/a
GRA1	565825	-	-
GA2	565825	570032	-
BDE2	565827	566650	565965
HH	565827	567028	566243
MA	565827	566861	566453
ES	565827	571312	569199
GA1	565866	571366	567329
BDE1	566166	-	-
ICGA	566404	-	-
SMA	566686	567822	566787
LR2	567663	n/a	n/a

In System 1, LR1, GRA1 and GA2 obtain the best result with 565825. HH achieves the second best result. However, the difference percentage between these results is only 0.0003%. In a small data set, different algorithms obtain very similar results. The difference between the best and the worst value of each algorithm is also low.

In System 2, HH produces the overal best result. GA2 and GRA2 follow the HH method. The difference percentage between the cost values of HH and GA2 is 0.022%. In System 3, HH again finds the best result in front of the SMA method. The difference percentage increases to 0.058% in a larger test instance.

Table 5.12: Cost results for System 2

Algorithm	Best Result	Worst Result	Average Result
HH	1125997	1128587	1127563
GA2	1126243	1132059	-
GRA2	1126805	-	-
ICGA	1127244	-	-
MA	1127254	1130916	1128824
GRA1	1128160	-	-
SMA	1128192	1128403	1128213
GA1	1128876	1131565	1130160
LR2	1129633	n/a	n/a
LR1	1130660	n/a	n/a

Table 5.13: Cost results for System 3

Algorithm	Best Result	Worst Result	Average Result
HH	2248284	2253971	2250534
SMA	2249589	2249589	2249589
LR2	2250223	n/a	n/a
GA2	2251911	2259706	-
GA1	2252909	2269282	2262585
MA	2252937	2270361	2262477
ICGA	2254123	-	-
GRA2	2255416	-	-
LR1	2258503	n/a	n/a
GRA1	2259340	-	-

Although LR1 and GRA1 obtain the best result in System 1, they do not perform well in larger data sets. On the other hand, methods with hybridization techniques, such as repair operators, hill climbers, specialized operators used for reproduction or for initial population generation, produce especially better results with increased problem instances. HH, SMA, LR2 are examples for these methods. In System 3, System 4, System 5 and System 6, HH, SMA and LR2 algorithms take the first three places. At each iteration, HH makes use of a hill-climbing operator; therefore, HH is a robust method for different problem instances with varying data sizes. Additionally, HH also applies mutational heuristics to maintain diversity during the search.

In System 5, SMA finds the best result in front of HH. However, HH finds the best result in System 4 and System 6 and SMA obtains the second best result. In all test sytems, the difference percentage between the best and the worst results of SMA is lower than the difference percentage of the best and the worst values obtained by HH.

Table 5.14: Cost results for System 4

Algorithm	Best Result	Worst Result	Average Result
HH	3369907	3376508	3373251
SMA	3370595	3371272	3370820
LR2	3374994	3374994	3374994
GA2	3376625	3384252	-
GA1	3377393	3401847	3394044
ICGA	3378108	-	-
GRA1	3383290	-	-
MA	3388676	3408275	3394830
LR1	3394066	n/a	n/a

Table 5.15: Cost results for System 5

Algorithm	Best Result	Worst Result	Average Result
SMA	4494214	4494439	4494378
HH	4494452	4499067	4496639
LR2	4496729	4496729	4496729
ICGA	4498943	-	-
MA	4501449	4545305	4527779
GA2	4504933	4510129	-
GA1	4507692	4552982	4525204
LR1	4526022	n/a	n/a

Table 5.16: Cost results for System 6

Algorithm	Best Result	Worst Result	Average Result
HH	5614360	5620496	5618418
SMA	5616314	5616900	5616699
LR2	5620305	5620305	5620305
GA1	5626362	5690086	5669362
GA2	5627437	5637914	-
ICGA	5630838	-	-
MA	5640543	5698039	5665803
GRA1	5669945	-	-
LR1	5657277	n/a	n/a

In the last experiment, real world data from the Turkish Interconnected Power System is used for the performance comparison. This data set only contains eight units and eight hours. According to the experimental results, both HH and BDE2 produce the best result. ES and SSGA obtain the second best result. This experiment also verifies, that the HH is able to find the optimum solution using real world data as well.

Table 5.17: Cost results for Turkish Interconnected Power System

Algorithm	Best Result	Worst Result	Average Result
HH	530346	530346	530346
BDE2	530346	530346	530346
ES	530392	530392	530392
SSGA	530392	530392	530392
BDE1	532142	-	-

Table 5.18 shows the 95% confidence intervals of the fitness values obtained by the HH for each test instance. First column gives the means and the second column gives the 95% confidence intervals of the means.

Table 5.18: Mean and the 95% confidence interval for the fitness values obtained by the HH

	Mean	Confidence interval
System 1	566243	[565819.8 , 566666.2]
System 2	1127563	[1127142.96 , 1127983.04]
System 3	2250534	[2249835.94 , 2251232.06]
System 4	3373251	[3372450.57 , 3374051.43]
System 5	4496639	[4495956.81 , 4497321.19]
System 6	5618418	[5616976.15 , 5619859.85]
System TR	530346	[530346 , 530346]

In System 1, nine different algorithms obtain better results than the upper bound of the confidence interval. In System 2, only four algorithms are able to produce a better result than the upper bound. In System 3 and System 5, SMA and LR2 achieve results within the 0.95 confidence interval. In System 4 and System 6, only SMA is able to find solutions, whose fitness values lie between the upper and the lower bounds of the interval.

6. CONCLUSION

In this study, hyper-heuristic algorithms with different heuristic selection and move acceptance strategy combinations are implemented to solve the UCP. To determine the most effective hyper-heuristic combination for the UCP, experiments are performed using two problem instances. RPD-OI heuristic selection and move acceptance combination achieves the best results in these two test sets. With the increased data size of the problem instance, its effectiveness becomes more significant. Consequently, in the second and third parts of the experiments the performance of this combination is compared with other optimization techniques, which are previously applied to the UCP.

In the first part of the experiments it is also observed, that CF and GR methods obtain good results in short time periods. However, the best and the average results produced by these two methods are not better than the results of RP, RPD, SR and RD. They select a heuristic according to the performance of each heuristic, but some heuristics can outperform the others; therefore, mutational heuristics especially have a small chance of being selected in further parts of the search process. Diversity can not be provided efficiently, and this results in getting stuck at local optima. On the other hand, in the RPD heuristic selection strategy even the mutational heuristics are applied to the solution at later stages of the search with respect to the order of the heuristics in the predefined permutation array, so that different regions of the search space are investigated to find a better solution than the globally best solution which is found so far.

Statistical test results show significant statistical differences between the fitness values produced by different heuristic selection methods for the same move acceptance criterion except AM. This also verifies that one method obtains much better results than at least one of the remaining methods.

After determining the most efficient strategy combination for the hyper-heuristic approach, the proposed hyper-heuristic algorithm is compared with a genetic algorithm explained in [12]. Although these two algorithms contain the same genetic operators, the hyper-heuristic algorithm produces better results than the genetic algorithm. The difference percentage between the results obtained by these two methods increases with the growing data size. When the data size increases, the solutions achieved using the HH approach are better even in the worst runs than the results obtained by GA2. Additionally, the hyper-heuristic method does not require system monitoring and fine tuning of the genetic operator probability rates to prevent the convergence.

Secondly, optimization techniques are ranked in decreasing order of their best fitness values using six benchmark data sets and one real world data set. The HH method finds consistent results in all test sets due to the incorporation of the hill-climbing operators. SMA and LR2 are the other two effective algorithms for the UCP. SMA uses specialized reproduction operators and hill climbers. It also makes use of the LR2 algorithm to create the initial population. HH randomly creates the initial population unlike in SMA, but it combines the usage of mutational heuristics with hill-climbing operators to search different regions of the solution space and to reach the highest point of the selected region.

HH obtains impressive results in all test sets when compared with other optimization algorithms. The performance of this algorithm can be further enhanced by applying more sophisticated heuristics and hill-climbing operators. Additionally, the initial solution can be created using the priority list method to increase the efficiency of this algorithm. Effective techniques including advanced learning mechanisms can be used for heuristic selection and move acceptance parts as well.

REFERENCES

[1] **Uyar, Ş.A., Türkay, B.,** 2008. Evolutionary Algorithms for the Unit Commitment Problem, *Turkish Journal of Electrical Engineering*, 16-3, 239-255.

[2] **Saramourtsis, J., Damousis, A., Bakirtzis, A.G., Dokopoulos, P.,** 1996. Genetic Algorithm Solution to the Economic Dispatch Problem. *IEE Proceedings-C*, 141-4, 377-382.

[3] **Cheng, C., Liu, C.W., Liu, C.C.,** 2000. Unit Commitment by Lagrangian Relaxation and Genetic Algorithms, *IEEE Transactions on Power Systems*, 15-2, 707-714.

[4] **Burns, R.M., Gibson, C.A.,** 1975. Optimization of Priority Lists for a Unit Commitment Program, *In Proceedings of IEEE/PES Summer Meeting*, pp. 1873-1879.

[5] **Zhuang, F., Galiana, F.D.,** 1990. Unit Commitment by Simulated Annealing, *IEEE Transactions on Power Systems*, 5-1, 311-318.

[6] **Lowery, P.G.,** 1996. Generating Unit Commitment by Dynamic Programming, *IEEE Tranasactions on Power Apparatus and Systems*, vol. PAS-85, no. 5, pp. 422-426.

[7] **Mantawy, A.H., Abdel-Magid, Y.L., Selim, S.Z.,** 1998. Unit Commitment by Tabu Search, *IEEE Proceedings - Generation, Transmission and Distribution*, 145-1, 56-64.

[8] **Chen, C.L., Wang, S.C.,** 1993. Branch-and-Bound Scheduling for Thermal Generating Units, *IEEE Transactions on Energy Conversion*, 8-2, 184-189.

[9] **Cote, G., Laughton, M.A.,** 1979. Decomposition Techniques in Power System Planning: the Benders Partitioning Method, *Electrical Power and Energy Systems*, 1-1, 57-64.

[10] **Viana, A., de Sousa, J.P., Matos, M.,** 2003. Using Grasp to Solve the Unit Commitment Problem, *Annals of Operations Research*, vol. 120, pp. 117-132.

[11] **Chen, Y.M., Wang, W.S.,** 2007. Fast Solution Technique for Unit Commitment by Particle Swarm Optimisation and Genetic Algorithm, *International Journal of Energy Technology and Policy*, vol. 5, no. 4, pp. 117-132.

[12] **Kazarlis, S.A., Bakirtzis, A.G., Petridis, V.,** 1996. A Genetic Algorithm Solution to the Unit Commitment Problem, *IEEE Transactions on Power Systems*, 11-1, 83-92.

[13] **Rudolf, A., Bayrleithner, R.,** 1999. A Genetic Algorithm for Solving the Unit Commitment Problem of a Hydro-Thermal Power System, *IEEE Transactions on Power Systems*, 14-4, 1460-1468.

[14] **Valenzula, J., Smith, A.E.,** 2002. A Seeded Memetic Algorithm for Large Unit Commitment Problems, *Journal of Heuristics*, vol. 8, pp. 173-195.

[15] **Dudek, G.,** 2004. Unit Commitment by Genetic Algorithm with Specialized Search Operators, *Electric Power Systems Research*, 72-3, 299-308.

[16] **Keles, A., Etaner-Uyar, A.S., Turkay, B.,** 2007. A Differential Evolution Approach for the Unit Commitment Problem, *In Proceedings of ELECO 2007: 5th International Conference on Electrical and Electronics Engineering*, pp. 132-136.

[17] **Simon, S.P., Padhy, N.P., Anand, R.S.,** 2006. An Ant Colony System Approach for Unit Commitment Problem, *Electrical Power and Energy Systems*, vol. 28, pp. 315-323.

[18] **Ozcan, E., Bilgin, B., Korkmaz, E.E.,** 2008. A Comprehensive Analysis of Hyper-heuristics, *Intelligent Data Analysis*, 12-1, 3-23.

[19] **Ozcan, E., Bilgin, B., Korkmaz, E.E.,** 2006. An Experimental Study on Hyper-Heuristics and Final Exam Scheduling, *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling*, pp. 123-140.

[20] **Damousis, I.G., Bakirtzis, A.G., Dokopoulos, P.S.**, 2004. A Solution for Unit Commitment Problem Using Integer-Coded Genetic Algorithm, *IEEE Transactions on Power Systems*, 19-2, 1165-1172.

[21] **Holland, J.H.**, 1975. Adaptation in Natural and Artificial Systems, *University of Michigan Press*.

[22] **Dawkins, R.**, 1976. The Selfish Genes, *Oxford University Press*.

[23] **Price, K.V.**, **Storn, R.M., Lampinen, J.A.,** 2005. Differential Evolution: A Practical Approach to Global Optimization, *Springer*.

[24] **Rechenberg, I.,** 1973. Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, *Fomman Holzboog Verlag*.

[25] **Beyer, H.G., Schwefel, H.P.,** 2002. Evolution Strategies, A Comprehensive Introduction, *Natural Computing*, vol. 1, pp. 3-52.

[26] **Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.,**2009. A Survey of Hyper-heuristics. *Computer Science Technical Report,* NOTTCS-TR-SUB-0906241418-2747, University of Nottingham.

[27] **Cowling P., Kendall, G., Soubeiga, E.,** 2000. A Hyper-heuristic Approach to Scheduling a Sales Summit, *LNCS 2079 PATAT 2000,* 176-190.

[28] **Kendall, G., Cowling P., Soubeiga, E.,** 2002. Choice Function and Random HyperHeuristics, *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning,* pp. 667-671.

[29] **Burke E.K., Kendall, G., Soubeiga, E.,** 2003. A Tabu Search Hyper-heuristic for Timetabling and Rostering, *Journal of Heuristics*, 9-6, 451-470.

[30] **Kendall, G., Mohamad, M.,** 2004. Channel Assignment in Cellular Communication Using a Great Deluge Hyper-heuristic, *In Proceedings of the IEEE International Conference on Network*, 769-773.

[31] **Nareyek, A.,** 2003. Choosing search Heuristics by non-stationary reinforcement learning, *In: Resende MGC, de Sousa JP (eds) Metaheuristics: Computer Decision-Making*, chap. 9, 523-544.

[32] **Ayob, M., Kendall, G.,** 2003. A Monte Carlo Hyper-Heuristic to Optimise Component Placement Sequencing for Multi Head Placement Machine, *In Proceedings of the Int. Conf. On Intelligent Technologies*, 132-141.

APPENDICES

APPENDIX A.1

Table A.1 : Data set for System 1

	Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
P_{max} (MW)	455	455	130	130	162
P_{min} (MW)	150	150	20	20	25
a_0	1000	970	700	680	450
a_1	16.19	17.26	16.60	16.50	19.70
a_2	0.00048	0.00031	0.00200	0.00211	0.00398
t_{up} (h)	8	8	5	5	6
t_{down} (h)	8	8	5	5	6
S_{hot} ($)	4500	5000	550	560	900
S_{cold} ($)	9000	10000	1100	1120	1800
$t_{coldstart}$ (h)	5	5	4	4	4
Initial state (h)	8	8	-5	-5	-6

	Unit 6	Unit 7	Unit 8	Unit 9	Unit 10
P_{max} (MW)	80	85	55	55	55
P_{min} (MW)	20	25	10	10	10
a_0	370	480	660	665	670
a_1	22.26	27.74	25.92	27.27	27.79
a_2	0.00712	0.00079	0.00413	0.00222	0.00173
t_{up} (h)	3	3	1	1	1
t_{down} (h)	3	3	1	1	1
S_{hot} ($)	170	260	30	30	30
S_{cold} ($)	340	520	60	60	60
$t_{coldstart}$ (h)	2	2	0	0	0
Initial state (h)	-3	-3	-1	-1	-1

Hour	Demand (MW)	Reserve (MW)	Hour	Demand (MW)	Reserve (MW)
1	700	75	13	1400	140
2	750	75	14	1300	130
3	850	85	15	1200	120
4	950	95	16	1050	105
5	1000	100	17	1000	100
6	1100	110	18	1100	110
7	1150	115	19	1200	120
8	1200	120	20	1400	140
9	1300	130	21	1300	130
10	1400	140	22	1100	110
11	1450	145	23	900	90
12	1500	150	24	800	80

Table A.2 : Data set for Turkish Interconnected Power System

	Unit 1	Unit 2	Unit 3	Unit 4
P_{max} (MW)	1120	1350	1432	600
P_{min} (MW)	190	245	318	150
a_0	6995.5	7290.6	6780.5	1564.4
a_1	7.0063	7.2592	5.682	3.1288
a_2	0.0168	0.0127	0.0106	0.0139
t_{up} (h)	8	1	1	10
t_{down} (h)	2	0.5	0.5	3
S_{hot} ($)	800	800	600	400
S_{cold} ($)	1600	1600	1200	800
$t_{coldstart}$ (h)	8	1	1	10
Initial state (h)	-4	-4	-4	-4

	Unit 5	Unit 6	Unit 7	Unit 8
P_{max} (MW)	990	420	630	630
P_{min} (MW)	210	110	140	140
a_0	5134.1	1159.5	1697	1822.8
a_1	6.232	3.3128	3.2324	3.472
a_2	0.0168	0.021	0.013	0.0147
t_{up} (h)	10	10	10	10
t_{down} (h)	3	3	3	3
S_{hot} ($)	500	400	400	400
S_{cold} ($)	1000	800	800	800
$t_{coldstart}$ (h)	10	10	10	10
Initial state (h)	-4	-4	-4	-4

Hour	Demand (MW)	Reserve (MW)
1	2000	200
2	3000	300
3	6500	650
4	1500	150
5	4200	420
6	5100	510
7	2700	270
8	1750	175

APPENDIX A.2

Table A.3 : Best solution for System 2 between Hour 1 and Hour 6

Unit	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6
1	455.00	455.00	455.00	455.00	455.00	455.00
2	245.00	295.00	382.50	455.00	455.00	425.00
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	130.00
5	0.00	0.00	25.00	40.00	25.00	25.00
6	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	245.00	295.00	382.50	455.00	455.00	425.00
13	0.00	0.00	0.00	0.00	0.00	130.00
14	0.00	0.00	0.00	0.00	130.00	130.00
15	0.00	0.00	0.00	40.00	25.00	25.00
16	0.00	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00

Table A.4 : Best solution for System 2 between Hour 7 and Hour 12

Unit	Hour 7	Hour 8	Hour 9	Hour 10	Hour 11	Hour 12
1	455.00	455.00	455.00	455.00	455.00	455.00
2	455.00	455.00	455.00	455.00	455.00	455.00
3	0.00	130.00	130.00	130.00	130.00	130.00
4	130.00	130.00	130.00	130.00	130.00	130.00
5	45.00	30.00	97.50	162.00	162.00	162.00
6	0.00	0.00	20.00	33.00	73.00	80.00
7	0.00	0.00	25.00	25.00	25.00	25.00
8	0.00	0.00	0.00	10.00	10.00	43.00
9	0.00	0.00	0.00	0.00	10.00	10.00
10	0.00	0.00	0.00	0.00	0.00	10.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	455.00	455.00	455.00	455.00	455.00	455.00
13	130.00	130.00	130.00	130.00	130.00	130.00
14	130.00	130.00	130.00	130.00	130.00	130.00
15	45.00	30.00	97.50	162.00	162.00	162.00
16	0.00	0.00	20.00	33.00	73.00	80.00
17	0.00	0.00	0.00	25.00	25.00	25.00
18	0.00	0.00	0.00	10.00	10.00	43.00
19	0.00	0.00	0.00	0.00	10.00	10.00
20	0.00	0.00	0.00	0.00	0.00	10.00

Table A.5 : Best solution for System 2 between Hour 13 and Hour 18

Unit	Hour 13	Hour 14	Hour 15	Hour 16	Hour 17	Hour 18
1	455.00	455.00	455.00	455.00	455.00	455.00
2	455.00	455.00	455.00	310.00	260.00	360.00
3	130.00	130.00	130.00	130.00	130.00	130.00
4	130.00	130.00	130.00	130.00	130.00	130.00
5	162.00	97.50	30.00	25.00	25.00	25.00
6	33.00	20.00	0.00	0.00	0.00	0.00
7	25.00	0.00	0.00	0.00	0.00	0.00
8	10.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	455.00	455.00	455.00	310.00	260.00	360.00
13	130.00	130.00	130.00	130.00	130.00	130.00
14	130.00	130.00	130.00	130.00	130.00	130.00
15	162.00	97.50	30.00	25.00	25.00	25.00
16	33.00	20.00	0.00	0.00	0.00	0.00
17	25.00	25.00	0.00	0.00	0.00	0.00
18	10.00	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00

Table A.6 : Best solution for System 2 between Hour 19 and Hour 24

Unit	Hour 19	Hour 20	Hour 21	Hour 22	Hour 23	Hour 24
1	455.00	455.00	455.00	455.00	455.00	455.00
2	455.00	455.00	455.00	455.00	432.50	345.00
3	130.00	130.00	130.00	0.00	0.00	0.00
4	130.00	130.00	130.00	130.00	0.00	0.00
5	30.00	162.00	105.00	105.00	25.00	0.00
6	0.00	43.00	20.00	20.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	10.00	10.00	0.00	0.00	0.00
9	0.00	10.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	455.00	455.00	455.00	455.00	432.50	345.00
13	130.00	130.00	130.00	0.00	0.00	0.00
14	130.00	130.00	130.00	0.00	0.00	0.00
15	30.00	162.00	105.00	105.00	0.00	0.00
16	0.00	43.00	20.00	20.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00
18	0.00	10.00	0.00	0.00	0.00	0.00
19	0.00	10.00	0.00	0.00	0.00	0.00
20	0.00	10.00	0.00	0.00	0.00	0.00

Table A.7 : Best solution for System 3 between Hour 1 and Hour 6

Unit	Hour 1	Hour 2	Hour 3	Hour 4	Hour 5	Hour 6
1	455.00	455.00	455.00	455.00	455.00	455.00
2	245.00	295.00	388.75	443.75	455.00	455.00
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	130.00
5	0.00	0.00	0.00	25.00	57.50	27.50
6	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	245.00	295.00	388.75	443.75	455.00	455.00
13	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	130.00
15	0.00	0.00	0.00	25.00	57.50	27.50
16	0.00	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00
18	0.00	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00
21	455.00	455.00	455.00	455.00	455.00	455.00
22	245.00	295.00	388.75	443.75	455.00	455.00
23	0.00	0.00	0.00	0.00	0.00	0.00
24	0.00	0.00	0.00	0.00	0.00	130.00
25	0.00	0.00	25.00	25.00	57.50	27.50
26	0.00	0.00	0.00	0.00	0.00	0.00
27	0.00	0.00	0.00	0.00	0.00	0.00
28	0.00	0.00	0.00	0.00	0.00	0.00
29	0.00	0.00	0.00	0.00	0.00	0.00
30	0.00	0.00	0.00	0.00	0.00	0.00
31	455.00	455.00	455.00	455.00	455.00	455.00
32	245.00	295.00	388.75	443.75	455.00	455.00
33	0.00	0.00	0.00	0.00	0.00	130.00
34	0.00	0.00	0.00	130.00	130.00	130.00
35	0.00	0.00	0.00	0.00	57.50	27.50
36	0.00	0.00	0.00	0.00	0.00	0.00
37	0.00	0.00	0.00	0.00	0.00	0.00
38	0.00	0.00	0.00	0.00	0.00	0.00
39	0.00	0.00	0.00	0.00	0.00	0.00
40	0.00	0.00	0.00	0.00	0.00	0.00

Table A.8 : Best solution for System 3 between Hour 7 and Hour 12

Unit	Hour 7	Hour 8	Hour 9	Hour 10	Hour 11	Hour 12
1	455.00	455.00	455.00	455.00	455.00	455.00
2	455.00	455.00	455.00	455.00	455.00	455.00
3	0.00	130.00	130.00	130.00	130.00	130.00
4	130.00	130.00	130.00	130.00	130.00	130.00
5	45.00	30.00	103.75	162.00	162.00	162.00
6	0.00	0.00	20.00	33.00	73.00	80.00
7	0.00	0.00	0.00	25.00	25.00	25.00
8	0.00	0.00	0.00	10.00	10.00	43.00
9	0.00	0.00	0.00	0.00	10.00	10.00
10	0.00	0.00	0.00	0.00	0.00	10.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	455.00	455.00	455.00	455.00	455.00	455.00
13	0.00	130.00	130.00	130.00	130.00	130.00
14	130.00	130.00	130.00	130.00	130.00	130.00
15	45.00	30.00	103.75	162.00	162.00	162.00
16	0.00	0.00	20.00	33.00	73.00	80.00
17	0.00	0.00	25.00	25.00	25.00	25.00
18	0.00	0.00	0.00	10.00	10.00	43.00
19	0.00	0.00	0.00	0.00	10.00	10.00
20	0.00	0.00	0.00	0.00	0.00	10.00
21	455.00	455.00	455.00	455.00	455.00	455.00
22	455.00	455.00	455.00	455.00	455.00	455.00
23	130.00	130.00	130.00	130.00	130.00	130.00
24	130.00	130.00	130.00	130.00	130.00	130.00
25	45.00	30.00	103.75	162.00	162.00	162.00
26	0.00	0.00	20.00	33.00	73.00	80.00
27	0.00	0.00	0.00	25.00	25.00	25.00
28	0.00	0.00	0.00	10.00	10.00	43.00
29	0.00	0.00	0.00	0.00	10.00	10.00
30	0.00	0.00	0.00	0.00	0.00	10.00
31	455.00	455.00	455.00	455.00	455.00	455.00
32	455.00	455.00	455.00	455.00	455.00	455.00
33	130.00	130.00	130.00	130.00	130.00	130.00
34	130.00	130.00	130.00	130.00	130.00	130.00
35	45.00	30.00	103.75	162.00	162.00	162.00
36	0.00	0.00	20.00	33.00	73.00	80.00
37	0.00	0.00	0.00	25.00	25.00	25.00
38	0.00	0.00	0.00	10.00	10.00	43.00
39	0.00	0.00	0.00	0.00	10.00	10.00
40	0.00	0.00	0.00	0.00	0.00	10.00

Table A.9 : Best solution for System 3 between Hour 13 and Hour 18

Unit	Hour 13	Hour 14	Hour 15	Hour 16	Hour 16	Hour 18
1	455.00	455.00	455.00	455.00	455.00	455.00
2	455.00	455.00	455.00	310.00	260.00	360.00
3	130.00	130.00	130.00	130.00	130.00	130.00
4	130.00	130.00	130.00	130.00	130.00	130.00
5	162.00	103.75	30.00	25.00	25.00	25.00
6	33.00	20.00	0.00	0.00	0.00	0.00
7	25.00	0.00	0.00	0.00	0.00	0.00
8	10.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	455.00	455.00	455.00	310.00	260.00	360.00
13	130.00	130.00	130.00	130.00	130.00	130.00
14	130.00	130.00	130.00	130.00	130.00	130.00
15	162.00	103.75	30.00	25.00	25.00	25.00
16	33.00	20.00	0.00	0.00	0.00	0.00
17	25.00	0.00	0.00	0.00	0.00	0.00
18	10.00	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00
21	455.00	455.00	455.00	455.00	455.00	455.00
22	455.00	455.00	455.00	310.00	260.00	360.00
23	130.00	130.00	130.00	130.00	130.00	130.00
24	130.00	130.00	130.00	130.00	130.00	130.00
25	162.00	103.75	30.00	25.00	25.00	25.00
26	33.00	20.00	0.00	0.00	0.00	0.00
27	25.00	0.00	0.00	0.00	0.00	0.00
28	10.00	0.00	0.00	0.00	0.00	0.00
29	0.00	0.00	0.00	0.00	0.00	0.00
30	0.00	0.00	0.00	0.00	0.00	0.00
31	455.00	455.00	455.00	455.00	455.00	455.00
32	455.00	455.00	455.00	310.00	260.00	360.00
33	130.00	130.00	130.00	130.00	130.00	130.00
34	130.00	130.00	130.00	130.00	130.00	130.00
35	162.00	103.75	30.00	25.00	25.00	25.00
36	33.00	20.00	0.00	0.00	0.00	0.00
37	25.00	25.00	0.00	0.00	0.00	0.00
38	10.00	0.00	0.00	0.00	0.00	0.00
39	0.00	0.00	0.00	0.00	0.00	0.00
40	0.00	0.00	0.00	0.00	0.00	0.00

Table A.10 : Best solution for System 3 between Hour 19 and Hour 24

Unit	Hour 19	Hour 20	Hour 21	Hour 22	Hour 23	Hour 24
1	455.00	455.00	455.00	455.00	455.00	455.00
2	455.00	455.00	455.00	455.00	432.50	345.00
3	130.00	130.00	130.00	0.00	0.00	0.00
4	130.00	130.00	130.00	130.00	0.00	0.00
5	30.00	162.00	103.75	0.00	0.00	0.00
6	0.00	41.75	20.00	20.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	10.00	0.00	0.00	0.00	0.00
9	0.00	10.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00
11	455.00	455.00	455.00	455.00	455.00	455.00
12	455.00	455.00	455.00	455.00	432.50	345.00
13	130.00	130.00	130.00	0.00	0.00	0.00
14	130.00	130.00	130.00	130.00	0.00	0.00
15	30.00	162.00	103.75	67.50	25.00	0.00
16	0.00	41.75	20.00	20.00	0.00	0.00
17	0.00	25.00	25.00	25.00	0.00	0.00
18	0.00	10.00	0.00	0.00	0.00	0.00
19	0.00	10.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00
21	455.00	455.00	455.00	455.00	455.00	455.00
22	455.00	455.00	455.00	455.00	432.50	345.00
23	130.00	130.00	130.00	0.00	0.00	0.00
24	130.00	130.00	130.00	130.00	0.00	0.00
25	30.00	162.00	103.75	0.00	0.00	0.00
26	0.00	41.75	20.00	20.00	0.00	0.00
27	0.00	0.00	0.00	0.00	0.00	0.00
28	0.00	10.00	0.00	0.00	0.00	0.00
29	0.00	10.00	0.00	0.00	0.00	0.00
30	0.00	0.00	0.00	0.00	0.00	0.00
31	455.00	455.00	455.00	455.00	455.00	455.00
32	455.00	455.00	455.00	455.00	432.50	345.00
33	130.00	130.00	130.00	0.00	0.00	0.00
34	130.00	130.00	130.00	130.00	0.00	0.00
35	30.00	162.00	103.75	67.50	25.00	0.00
36	0.00	41.75	20.00	20.00	0.00	0.00
37	0.00	0.00	0.00	0.00	0.00	0.00
38	0.00	10.00	0.00	0.00	0.00	0.00
39	0.00	10.00	0.00	0.00	0.00	0.00
40	0.00	0.00	0.00	0.00	0.00	0.00

CURRICULUM VITAE

Candidate's full name:	Ali Argun BERBEROĞLU
Place and date of birth:	Sakarya / Turkey 01/07/1981
Permanent Address:	Kireçburnu Mahallesi Kireçliçeşme Sokak No:14 Daire:3 Sarıyer Istanbul/Turkey
Universities and Colleges attended:	Electrical Engineering, Istanbul Technical University 2000-2004
	Istanbul German High School 1992-2000

Publications:

- **Berberoğlu, A., Uyar, A. Ş.,** "A Hyper-Heuristic Approach for the Unit Commitment Problem", *EvoApplications 2010, Part II, LNCS 6025*, pp. 121-130, Istanbul, Turkey, 2010.
- **Berberoğlu, A., Uyar, A. Ş.,** "Experimental Comparison of Selection Hyper-Heuristics for the Short-Term Electrical Power Generation Scheduling Problem", *EvoApplications 2011.*