ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE
ENGINEERING AND TECHNOLOGY

DISTRIBUTED RESOURCE SCHEDULING ON VIRTUAL MACHINES BY USING BIG BANG - BIG CRUNCH ALGORITHM

M.Sc. THESIS

Ercan EROL

Department of Computer Engineering

Computer Engineering

JANUARY 2014

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

DISTRIBUTED RESOURCE SCHEDULING ON VIRTUAL MACHINES BY USING BIG BANG - BIG CRUNCH ALGORITHM

M.Sc. THESIS

Ercan EROL
(504031509)

Department of Computer Engineering

Computer Engineering

Thesis Advisor: Assoc. Prof. Dr. Osman Kaan EROL

JANUARY 2014

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BÜYÜK PATLAMA - BÜYÜK ÇÖKÜŞ YÖNTEMİNİ KULLANARAK SANAL MAKİNELERDE DAĞITIK KAYNAK PLANLAMA

YÜKSEK LİSANS TEZİ

Ercan EROL
(504031509)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Osman Kaan EROL

OCAK 2014

Ercan EROL, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504031509 successfully defended the thesis entitled **"DISTRIBUTED RESOURCE SCHEDULING ON VIRTUAL MACHINES BY USING BIG BANG - BIG CRUNCH ALGORITHM"**, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Osman Kaan EROL** Istanbul Technical University	....
Jury Members :	**Assoc. Prof. Dr. Osman Kaan EROL** Istanbul Technical University	....
	Assoc. Prof. Dr. Şima Etaner UYAR Istanbul Technical University	....
	Asst. Prof. Dr. Taner ARSAN Kadir Has University	....

Date of Submission : **11 December 2013**
Date of Defense : **24 January 2014**

To my Snow White and family,

FOREWORD

I would like to express my sincere gratitude to my advisor Assoc.Prof.Dr. Osman Kaan Erol for his continuous support on my thesis study, for his patience and attention. I am really glad to find the chance of working with one of two academics who introduced Big Bang-Big Crunch optimization method which constitutes also basis the of this study.

Besides my advisor, I would like to thank Dr. Tülin Kaman from the Department of Theoretical Computer Science at ETH Zurich who received her M.Sc. degree from Computational Science and Engineering at Istanbul Technical University with 1st rank in 2004. Without her valuable guidance and help I could not complete my thesis and article in time for submission.

Last but not the least, I would like to thank my family for their support and trust during my whole life to overcome difficulties and obstacles; my parents Kaniye and İsmail Erol, my brother Erdeniz Erol and my Snow White Aleksandra Mykhailova for putting her endless faith in me.

January 2014 Ercan EROL

TABLE OF CONTENTS

Page

ABBREVIATIONS

ANOVA	: Analysis of Variance
BB-BC	: Big Bang-Big Crunch
CPU	: Central Processing Unit
CAPEX	: Capital Expenditure
DRS	: Distributed Resource Scheduling
GA	: Genetic Algorithms
HSD	: Honestly Significant Difference
HW	: Hardware
I/O	: Input/Output
IT	: Information Technology
Mem	: Memory
NSGA	: Nondominated Sorting Genetic Algorithm
OPEX	: Operational Expenditure
PAES	: Pareto Archived Evolution Strategy
PESA	: Pareto Envelope-based Selection Algorithm
QoS	: Quality of Service
SPEA	: Strength Pareto Evolutionary Algorithm
VM	: Virtual Machine

LIST OF TABLES

LIST OF FIGURES

DISTRIBUTED RESOURCE SCHEDULING ON VIRTUAL MACHINES BY USING BIG BANG - BIG CRUNCH ALGORITHM

SUMMARY

Hype Cycle, announced annually by Gartner, defines the maturity, adoption and application of specific technologies. According to 2013 Hype Cycles of IT Operations Management and Cloud Computing reports, Private Cloud Computing and Virtual Machine Resilience are the technologies that are expected to gain more importance. Server Virtualization, one of the technologies that is widely used in Cloud Computing, needs resilience when shared underlying resources are redistributed among virtual machines on-demand. Standing at the peak of inflated expectations has brought some problems such as virtual machine sprawl and managing underlying shared resources efficiently. Distributed Resource Scheduling (DRS) on virtual infrastructures mitigates the administration duties and performance monitoring workload of systems management.

In order not to suffer from CPU bottleneck, DRS aims to balance the CPU load among the physical hosts meanwhile caring the resource allocation policies by powering on and migrating the running virtual machines to the correct hosts. Intervention on underlying shared resources will help to decrease the operational and capital expenditures in IT infrastructure investments. Virtualization resilience is supplied by presenting physical resources as a pool to the virtual machines. A few problems must be handled while managing such resilience. It is required to define how to balance the workload by migrating right virtual machines to convenient physical hosts when physical hosts in the resource pool are insufficient to comply the demands of virtual machines. Physical hosts in the cluster must be configured equally and must reach the same shared disk area in order to migrate the virtual machines without any disruption sensed by end users. Live Migration of virtual machines between physical hosts is just changing the owner host of virtual machine just after the active memory is copied from source to the destination host in order to synchronize the states in both source and destination hosts.

Artificial intelligence techniques are widely used for resource management in real-time dynamic environments. To comply the demand of the virtual machines, best resource allocation in the physical domain is a kind of problem where genetic algorithms are formerly considered to be successful. To cope with this problem, another optimization method, namely Big Bang - Big Crunch optimization method is used. Big Bang - Big Crunch (BB-BC) is an optimization method which is inspired by the natural event in evolution of the universe and is introduced by Erol and Eksin in 2006. BB-BC preserves the randomness of the solution population and it keeps track to the best solution without sticking into any local optimum. A cluster with a definite number of physical servers will host some definite number of virtual machines by assigning CPU and memory resources to them. It is seen that CPU demand of virtual machines running on one physical host may not be equal to the other hosts in the cluster according to the

initial distribution. This causes some of the hosts to suffer from lack of CPU while resources in other hosts are wasted at that moment.

VMware, one of the pioneers in virtualization technology, prefers the Greedy-Hill Climbing algorithm as state of art for resource scheduling in the clusters. Although VMware claims that it is not necessary to find best distribution of virtual machines, their preference may be replaced by one of the Artificial Intelligence methods which will not disturb the layout in the cluster frequently.

Since Greedy-Hill does not have a claim to find the best allocation, its solutions' fitness values can not be a comparison element. The candidate method can have a challenge to Greedy-Hill by execution durations. The modified BB-BC and classical BB-BC algorithms take less than 3 minutes to complete until the end of 200 generations for one CPU worksheet which is convenient to apply in the time interval of imbalance checks.

Main focus of the study is to improve classical BB-BC performance by differentiating speed of the convergence including also migration costs. Environment is assumed as clusters of 4 physical hosts with 20 virtual machines and 8 physical hosts with 40 virtual machines. Initial distribution of virtual machines on physical hosts, virtual machines' properties such as number of cores, assigned memory amounts and processor needs in specific time intervals are supplied as data input to solve the problem. Cells of the individuals in the population represent the virtual machines and their contents are the indices of the physical hosts. Individuals in the population carry the possible distributions of virtual machines.

Tests are executed according to two different scenarios. Algorithms are expected to solve the problem according to the same inital distribution for all CPU workloads in 24 different time intervals in the first scenario. It is aimed to get results over the same problem set while calculating the averages of the values. In the second scenario CPU workloads are supplied to algorithms one after the other and algorithms try to find the best distribution according to the distribution found in the previous time interval. Thus differences created by the successive solutions of the algorithms are compared. The behaviours of algorithms are shown in graphs by plotting averages for all time intervals and for a specific time interval.

The Fitness function is calculated as the weighted sum of normalized standard deviations of physical hosts' cpu load, virtual machine cores and memories including also the amount of migrated virtual machine memories for a specific resource allocation while fitting inside the limitations on physical memory of each node. After the best distribution of Virtual Machines is found it is applied to the cluster by migrating the virtual machines between hosts. Since it is required to copy the migrated virtual machine's memory between hosts without going beyond the limits, migrations are reflected to the fitness function as cost.

BB-BC, as a better optimization method, has been examined before by comparing to traditional genetic algorithms in many papers. The algorithms mentioned in this study are coded using MATLAB 7 (R14) environment and they are compared by their best fitness values with the same input data. Same fitness function is used for all methods and weights of normalized standard deviations are selected in advance to reflect the intense of preference according to resource utilizations' importance. Algorithms are differentiated by their speed of convergence rates. ANOVA and Tukey's HSD test as post-hoc test are used to analyse the significance of the values statistically. It is derived that periodically resetting the convergence rate in Big Bang phase revealed better fitness values compared to constantly decreasing the convergence rate with the confidence rate of 90% for all time intervals while number of time intervals that shows significantly difference is only 9 among 24 with 95% confidence coefficient.

BÜYÜK PATLAMA - BÜYÜK ÇÖKÜŞ YÖNTEMİNİ KULLANARAK SANAL MAKİNELERDE DAĞITIK KAYNAK PLANLAMA

ÖZET

Her yıl Gartner araştırma ve danışmanlık şirketince açıklanan Hype Cycle, belli başlı teknolojilerin olgunluğunu, ne ölçüde benimsenilir ve uygulanır olduğunu belirten grafiksel bir rapor aracıdır. Bilgi Teknolojileri Operasyon Yönetimi ve Bulut Bilişim konulu 2013 Hype Cycle grafiklerine göre Özel Bulut Bilişim ve Sanal Makine Esnekliği önem kazanması beklenen teknolojiler arasında yer almaktadır. Bulut Bilişimde kullanılan teknolojilerin başında gelen Sunucu Sanallaştırma, sanal sunucuların ihtiyaçlarına göre ortak altyapıda yer alan kaynakların dağıtılmasında esnekliğe ihtiyaç duyar. Beklentilerin en üst noktaya ulaştığı bu teknolojilerin yoğun kullanımından ötürü sanal sunucu dağınıklığı, paylaşılan altyapı kaynaklarının verimli yönetilememesi gibi problemler doğmaktadır. Dağıtık Kaynakların Planlanması sanal altyapılardaki yönetim görevlerini ve kaynakların verimlerini gözleme yükünü hafifletir.

Dağıtık Kaynak Planlama, işlemci darboğaz sıkıntısı yaşamamak için sanal sunucuları kaynak tahsis kurallarına göre doğru fiziksel sunuculara taşıyarak fiziksel sunucular arasında işlemci yükünü dengelemeyi hedefler. Ortak altyapıda yer alan kaynaklara müdahale BT yatırımlarındaki operasyonel ve mali giderleri azaltır. Fiziksel kaynakların sanallaştırılarak sanal sunuculara bir havuz şeklinde sunulmasıyla ihtiyaç duyulan esneklik sağlanmaktadır. Ancak bu esnekliğin yönetilmesi de bazı problemler içermektedir. Kaynak havuzunu oluşturan fiziksel sunucular istekleri karşılamada yetersiz kaldığı anlarda üzerlerindeki yükün dağıtılıp dağıtılmaması gerektiği, dağıtılacaksa hangi sanal sunucuların hangi fiziksel sunuculara aktarılacağı gibi noktaların belirlenmesi gerekmektedir. Sanal sunucuların son kullanıcılara hissettirmeden taşınabilmesi için kümeyi oluşturan fiziksel sunucuların konfigürasyonlarının eşlenik ve ortak bir disk alanına erişiyor olması gerekmektedir. Bir sanal sunucunun bir fiziksel sunucudan başka bir fiziksel sunucuya canlı olarak aktarılması, sanal sunucunun belleğindeki etkin verilerin kopyalanmasıyla her iki fiziksel sunucuda da eşlenik olduğu anda diğer fiziksel sunucu üzerinden çalışmaya devam etmesinden ibarettir.

Yapay Zeka teknikleri gerçek zamanlı dinamik ortamlarda kaynak yönetimi için yaygın olarak kullanılır. Sanal sunucuların isteklerini karşılamak için fiziksel kaynakların en iyi dağılımı genetik algoritmaların evvelden beri başarılı kabul edildiği bir problem türüdür. Bu problemi çözmede başka bir en iyileme yöntemi olan Büyük Patlama-Büyük Çöküş (BP-BÇ) de kullanılmıştır. Evrenin evrim sürecindeki doğa olayından esinlenilen BP-BÇ, Erol ve Eksin tarafından 2006'da tanıtılan bir en iyileme yöntemidir. BP-BÇ, yalnızca çözüm popülasyonunun rastlantısallığını korumayıp aynı zamanda yerel en iyilere takılmadan en iyi çözümün izini sürer. Bir kümede yer alan fiziksel sunucular üzerinde işlemci ve bellek kaynakları atanarak belli adette sanal sunucu barındırılabilir. Bir fiziksel sunucu üzerinde çalışan sanal

sunucuların işlemci ihtiyaçlarının kümedeki diğer fiziksel sunuculardaki yüke eşit olmadığı görülebilmektedir. Bu da bazı fiziksel sunucuların işlemci ihtiyaçlarına cevap vermede sıkıntı yaşarken diğer fiziksel sunucuların üzerindeki kullanılmayan kaynakların israfına yol açmaktadır.

Sanallaştırma teknolojilerinin öncülerinden olan VMware, kümelerde kaynak planlama yöntemi olarak Açgözlü-Yüksek Tırmanış algoritmasını kullanmaktadır. Her sanal sunucunun diğer olası fiziksel sunucuların üzerine taşınmasının, o fiziksel sunucu üzerine yüklenen kaynakların kullanımını azaltıp azaltmayacağını deneyerek belirli bir eşik değerinin altına inildiğinde bunu gerçekleyen bu tip yöntemlerin yerini daha erken müdahale ve kontrol yetkinliğindeki yöntemlere bırakması beklenmektedir. Bunun için en uygun yöntemlerden olan Yapay Zeka teknikleri en iyi dağılımları bularak kümedeki düzeni daha az rahatsız edecektir.

Açgözlü-Yüksek Tırmanış algoritmasının en iyi dağılıma ulaşmak gibi bir amacı olmadığından burada elde edilecek uygunluk değerleri karşılaştırmada kriter olarak kullanılmayacaktır. Ancak aday yöntemin çalışma süresi Açgözlü-Yüksek Tırmanış algoritmasına göre karşılaştırılabilir. Tezde birbirleri ile karşılaştırılan BP-BÇ yöntemleri tek bir yük dağılımı için 200 neslin sonuna dek çalıştırıldığında 3 dakikadan daha kısa bir sürede sonuçlanmaktadır. Kümedeki dengesizliklerin kontrol edildiği 5 dakikalık zaman dilimleri için uygun bir çalışma süresi olarak değerlendirilebilir.

Bu çalışmanın ana amacı BP-BÇ en iyileme yöntemini sanal sunucuları dağıtık kaynaklar üzerine yerleştirirken yakınsama hızını farklılaştırarak performansının iyileştiğini sanal sunucu göç maliyetlerini de içerecek biçimde karşılaştırmaktır. Ortam 4 fiziksel sunuculu bir küme üzerinde çalışan 20 sanal sunucu ve 8 fiziksel sunucu üzerinde çalışan 40 sanal sunuculu ayrı bir küme şeklinde varsayılmıştır. Problemi çözmek için gereken veriler sanal sunucuların fiziksel sunucular üzerindeki ilk dağılım durumu, çekirdek adetleri, atanmış bellek miktarları ve belirli zaman aralıklarındaki işlemci gücü ihtiyaçları şeklinde sağlanmıştır. Popülasyon içindeki bireyler sanal sunucuları temsil ederken taşıdıkları değerler de bulundukları fiziksel sunucuların indeksleridir. Böylece bir popülasyon içindeki bireyler olası dağılımları barındırmaktadır.

Testler iki farklı senaryoya göre gerçekleştirilmiştir. İlk senaryoya göre 24 farklı zaman aralığında ihtiyaç duyulan işlemci güçleri ile oluşan yük dengesizliklerini algoritmaların hepsinde aynı ilk dağılıma göre çözmeleri sağlanmıştır. Böylece ortalamaları alırken hep aynı problem kümesi üzerinden elde edilen sonuçlar hedeflenmiştir. İkinci senaryoya göre ise 24 farklı zaman aralığında sanal sunucuların ihtiyaç duyacağı işlemci güçleri algoritmalara bu kez seri olarak verilmiş; bir önceki zaman aralığında elde edilen çözüm kümesi yeni zaman aralığının ilk dağılımı gibi ele alınmıştır. Böylece algoritmaların ardı ardına kendi çözümleri üzerinden yaratacakları farklılıklar da karşılaştırılmıştır. Elde edilen sonuçlardan algoritmaların tüm zaman aralıklarının sonucundaki ve bazı zaman aralıklarındaki davranışları grafik olarak sunulmuştur.

Uygunluk fonksiyonu fiziksel sunucularda koşan sanal sunucuların işlemci yükünün, çekirdek adedinin ve bellek miktarının normalize edilmiş standart sapmalarının ağırlıklı toplamı biçiminde hesaplanır. Sanal sunucuların fiziksel sunucular üzerindeki en iyi dağılımı bulunduktan sonra çözüm ilgili sanal sunucuları fiziksel sunucular arasında taşıyarak uygulanır. Her fiziksel sunucudaki kaynak sınırlarını aşmadan sunucular arası taşınan sanal sunucuların bellek miktarları da fiziksel sunucular arası kopyalanması gerektiğinden maliyet olarak uygunluk fonksiyonuna yansıtılır. Böylece fiziksel sunucular üzerindeki kaynak kullanımlarının standart sapmalarının en düşük düzeye eriştiği görülür.

Daha başarılı sonuçlar veren BP-BÇ en iyileme yöntemi daha evvel birçok çalışmada geleneksel genetik algoritmalara karşı incelenmiştir. Bu çalışmada algoritmalar MATLAB 7 (R14) sürümünde gerçeklenmiş ve aynı veri girdileri üzerinde ulaştıkları en iyi uygunluk değerleri karşılaştırılmıştır. Tüm algoritmalar için aynı uygunluk fonksiyonu kullanılmış ve normalize edilmiş standart sapmaların ağırlıkları da kaynakların kullanım önemlerini yansıtacak biçimde seçilmiştir. Algoritmalar yakınsama hızları açısından farklılıklar göstermektedir. Elde edilen verilerin istatiksel açıdan anlamlı olup olmadığını görmek için varyans analizi ve sonrasında Tukey'in çoklu karşılaştırma testleri uygulanmıştır. Büyük Patlama fazı sırasında yakınsama oranını belirli aralıklarda yeniden ayarlamanın sürekli azalan bir yakınsama oranına göre %90 güven aralığında 24 adet yük dağılımının hepsinde daha iyi sonuçlar verdiği görülürken bu adet %95 güven aralığında 9 adede düşmüştür.

1. INTRODUCTION

1.1 Purpose of Thesis

Enterprises keep implementing efficient computing technologies like cloud computing that provide dynamic flexibility, on-demand services and virtualized resources. Cloud Computing stands in its mature and second generation products' rise era. [1] Virtualization brings a dramatic change to data centers by offering the benefits of consolidation, resource-efficiency, easier management, security, scalability, reliability and power-saving. Since IT infrastructure customer requirements for cloud infrastructure services are varied, infrastructure providers have to ensure that they can be efficient, resilient, reliable and robust in their service delivery while keeping the infrastructure costs in a minimum level for cooling, power consumption, hosting, multi-tenancy and resource management. Intelligent allocation of Virtual Machines (VMs) is a challenge in large virtualized IT infrastructures. Dynamically changing workloads make it difficult to host VMs on shared resources without compromising Quality of Service (QoS) or wasting resources.

This thesis states this problem as an optimization issue and aims to solve this problem using Big Bang-Big Crunch (BB-BC) optimization that is inspired by the natural event in evolution of universe and is introduced by Erol and Eksin in 2006. [2, 3] Load-balanced distribution of VMs on underlying shared resources are compared between traditional Genetic Algorithms and BB-BC methods with varying convergence speeds.

1.2 Literature Review

BB-BC algorithm has been applied to many areas including power flow [4], design of plain truss [5], software testing [6], design of skeletal structures [7], design of complex composite laminates [8], determination of worst case loading margin [9], economic dispatch problem [10] and airport gate assignment problem [11]. In these papers

BB-BC is seen as better alternative to the known heuristic methods comparatively by accuracy, reliability and computation time.

Cloud Infrastructure is also in the focus of other optimization research papers in distinct ways such as scheduling schemes [12–15], resource allocation [16–18], load balancing [19], dynamic provisioning [20], intelligent management [21], energy aware scheduling [22], fuzzy modeling [23], dynamic configuration [24].

As it will be explained in the rest of this thesis load balancing problem has a multi-objective behavior. Multi objective problems like in papers [25, 26] are aimed to be solved through genetic algorithms [27], weighted sums [28, 29], adaptive weighted sum [30] or PAES, PESA, SPEA, NSGA-II, PESA-II methods as compared in [31]. A priori articulation of preferences are reflected to the weights in fitness evaluation of the population for this research.

Algorithms mentioned in this thesis are implemented on MATLAB 7 (R14) environment which includes algorithms also in [32, 33]. Cloud infrastructure to be optimized is assumed as VMware vSphere virtualization environment whose characteristics are explained in the manuals and white papers in [34–37].

Remainder of thesis is organized as follows. Virtualization and especially Server Virtualization are overviewed in section 2. Section 3 describes the Resource Scheduling and the issues to be considered. Greedy Hill-Climbing technique, currently state of art method for VMware DRS [38], Genetic Algorithms and Big Bang-Big Crunch algorithms are mentioned in Section 4. In Section 5 it is explained how it is approached to the Distributed Resource Scheduling problem. Section 6 includes the experimental tests and results. Finally Section 7 summarizes the conclusion on the comparison of optimization algorithms.

2. VIRTUALIZATION

2.1 Overview

Virtualization is the comprehensively used component in all types of Cloud Computing, public, private or hybrid. Virtualization Management is a common element in several types of virtualization such as Server Virtualization, Storage Virtualization, Network Virtualization and Services Virtualization. These virtualization types can be applied individually or combined according to the purpose, budget and needs of customers. IT operations should be served with high quality, agility, low fixed costs and minimal risk.

The concept of virtualization has been proven to be very beneficial in saving a lot of costs for companies and generate a better value from their IT investments. The capital expenditure (CAPEX) is reduced as virtualization is used in making a better utilization of the company's infrastructure. The operational expenditure (OPEX) is also reduced as virtualization can be used to reduce the overall number of servers in the infrastructure. This directly reduces the data center costs like power, cooling and datacenter footprint which helps corporates to move to a greener data center. It also provides centralized and easier administration for the resources. The scale of the companies that can benefit from virtualization can vary from small enterprises to large scale enterprises, each according to their needs. [21]

2.2 Server Virtualization

Since modern servers become more powerful as multi-core architecture improves it creates a demand for server consolidation. Virtual machines can host any types of application by providing an abstraction (virtualization layer) which is called hypervisor between the VMs and the actual hardware as seen in Figure 2.1. Hypervisors manage access of VMs to hardware resources, optimizing resource usage and reducing overhead for cache coherence. [15]

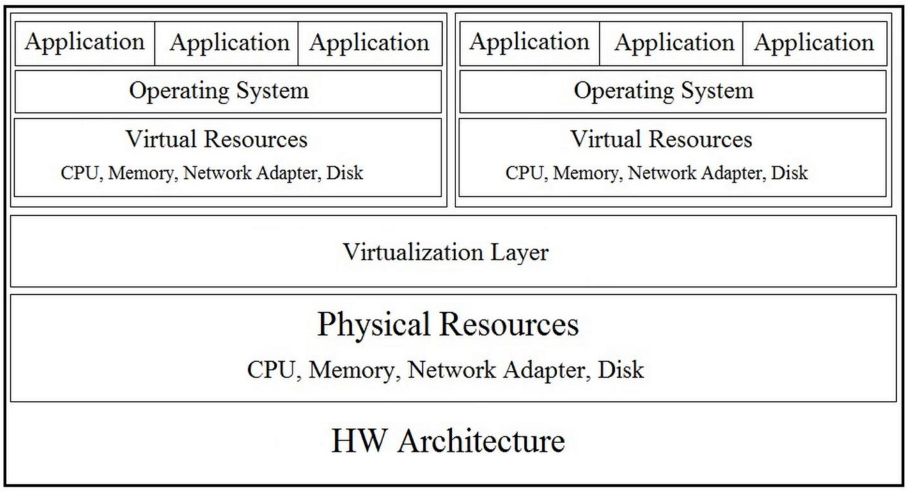

Figure 2.1: Server Virtualization.

Server Virtualization techniques are used for spreading many virtual machines into several physical servers where each VM is run and managed separately from the others. This separation is required for security, supportability of the applications and also for maximizing the utilization of the physical servers.

Hypervisors support a variety of functions for the hosted VMs such as create, delete, restart, suspend, migrate etc. [17] For high availability, physical servers running virtualization software compose clusters to serve as a resource pool. Virtual machines are then created and assigned the relevant resources according to the requirements and the recommended sizing of the applications that will be placed on them. [21]

Server virtualization provides agility by workload balancing, automation and capacity on demand elasticity; QoS by dynamic provisioning, high availability and disaster recovery management; economic savings by consolidation and energy consumption.

3. RESOURCE SCHEDULING

3.1 DRS on Server Virtualization

Distributed Resource Scheduling (DRS) on virtual infrastructures mitigates the administration duties and performance monitoring workload of systems management. In order not to suffer from CPU bottleneck, DRS aims to balance the CPU load among the physical hosts by migrating the running virtual machines to the correct hosts. Different types of evolutionary algorithms can be used in calculating the best placement for the virtual machines on the physical hosts. In addition to resolving resource overcommitment, resource management can help in preventing virtual machines from monopolizing resources and guarantee promised service rates, exploiting underutilized resources and degrading overutilized resources gracefully, controlling the relative importance of virtual machines in absolute service level agreements.

VMware DRS (VMware Distributed Resource Scheduler) is one of the examples among resource management tools. VMware DRS works in the cluster level and load balancing occurs as VM migration (VMware vMotion) between the hosts in the cluster by re-evaluating the cluster in every 5 minutes within the metrics of CPU, Memory and IO resources continuously. It determines the standard deviation from the average loads of CPU, Memory and IO.

VMware DRS has normalized entitlement behaviour based on core load per host metric. For a host j, normalized entitlement $H_{\text{coreent}}[j]$ is calculated as in (3.1).

$$H_{\text{coreent}}[j] = \frac{\sum_{i=0}^{N-1} V_{\text{core}}[i]}{C_{\text{core}}[j]} \tag{3.1}$$

where $C_{\text{core}}[j]$ is the core capacity for host j and $V_{\text{core}}[i]$ is the core entitlement for N number of VM running on host j. If $H_{\text{coreent}}[j] > 1$ then the host seems have insufficient resources to meet the entitlements for VMs running on itself. After normalized entitlements of every host in the cluster is calculated, clusterwide imbalance is aimed to be load balanced by VMware DRS. This entitlement calculations are done not only by core entitlements but also CPU load and memory meanwhile.

3.2 DRS Problems

While VMs are being distributed on the shared resources there are some challenges to be overcomed. VMs to be moved and hosts that are available must be chosen if migration is initiated. These decisions are key to the success of the dynamic management scheme. [17] Virtual machines are assigned with specified number of virtual CPUs (vCPUs) and memory. Therefore normalized entitlements for core and memory are one of the concerns. Administrators have a chance to set some reserved amounts of resources for CPU and memory besides some sharing policies relatively to other VMs in that resource pool. A resource pool represents an aggregate resource allocation that may be consumed by VMs. The process of computing the entitled reservation, limit and shares of its sub-pools and VMs is referred as Resource Pool Divvying. Resource Pool Divvy mechanism protects the VMs from unexpected steep workloads of other VMs. Resource Allocation Limit can also be applied to VMs to specify an upper bound for CPU, memory, or storage I/O resources that can be allocated to a virtual machine in order to reserve some resources or prevent other VMs not to take resources unnecessarily. [37] The cost required to migrate a VM from one physical host to another can be calculated by duration elapsed while copying active memory from one host to another. It is obvious that if a VM dirties its pages very often, they must be copied to the destination host multiple times to synchronize the states in both source and destination hosts. [38] A good solution close to the optimal one which has not any bottleneck in resource allocation must be found. Problem environment is formulated in the rest of this section.

$$V[i] = j \text{ },\exists\, 0 \leq j < M,\ \forall\, 0 \leq i < N \tag{3.2}$$

where i: VM index, j: Physical Host index, N: total number of Virtual Machines, M: total number of Physical Hosts and $V[i]$ in (3.2) states that the distribution of N Virtual Machines over M Physical Hosts. Each VM must be placed on one of the hosts.

$$\forall\, 0 \leq j < M \;\; H_{core}[j] = \sum_{i=0}^{N-1} (V_{core}[i], V[i] = j) \tag{3.3}$$

where $H_{core}[j]$ in (3.3) states that sum of Virtual CPU assignments $V_{core}[i]$ on Physical Hosts if their cell content equals the index of that host. Core assignments for each host are calculated by (3.3).

$$\forall\, 0 \leq j < M \;\; H_{mem}[j] = \sum_{i=0}^{N-1} (V_{mem}[i], V[i] = j) \tag{3.4}$$

where $H_{mem}[j]$ in (3.4) states that sum of Virtual memory assignments $V_{mem}[i]$ on Physical Hosts if their cell content equals the index of that host. Memory assignments are calculated for each host by (3.4).

$$\forall\, 0 \leq j < M \;\; H_{cpu}[j] = \sum_{i=0}^{N-1} (V_{cpu}[i], V[i] = j) \tag{3.5}$$

where $H_{cpu}[j]$ in (3.5) states that sum of VM CPU workloads $V_{cpu}[i]$ on Physical Hosts if their cell content equals the index of that host. Dynamic workloads of VMs are calculated for each physical host by (3.5). Standard deviations to be minimized concurrently for balancing the load in the cluster are given in (3.6), (3.7) and (3.8).

$$\sigma(H_{cpu}[M]) = \frac{1}{M} \sqrt{\sum_{j=0}^{M-1} \left(\overline{H_{cpu}[j]} - H_{cpu}[j]\right)^2} \tag{3.6}$$

$$\sigma(H_{core}[M]) = \frac{1}{M} \sqrt{\sum_{j=0}^{M-1} \left(\overline{H_{core}[j]} - H_{core}[j]\right)^2} \tag{3.7}$$

$$\sigma(H_{mem}[M]) = \frac{1}{M} \sqrt{\sum_{j=0}^{M-1} \left(\overline{H_{mem}[j]} - H_{mem}[j]\right)^2} \tag{3.8}$$

where (3.6), (3.7) and (3.8) state the standard deviations of hosts' CPU usage, core entitlement and memory entitlement respectively. While minimizing (3.6), (3.7) and (3.8), memory entitlement on any physical host must not exceed the maximum memory capacity constraint of a physical host (C_{mem}) as denoted in (3.9).

$$\forall\, 0 \leq j < M \quad H_{mem}[j] \; < \; C_{mem} \tag{3.9}$$

The Distributed Resource Scheduling environment is formulated as above. Different optimization algorithms are compared to realize which is more efficient in load balancing. Efficiency is measured by the fitness function which calculates also migration cost of VMs. Migration cost affects the choice of best distribution among possible distributions which have minimum standard deviations in the resource utilizations.

4. OPTIMIZATION ALGORITHMS

4.1 Greedy Hill-Climbing Technique Overview

Greedy Hill-Climbing technique is used as the state of art method in the VMware environments. [38] Since VM resource demand is changing over time, VMware claims that optimizing cluster for a particular dynamic situation is not worthwhile. Rather than optimizing the resources it is preferred to find a single VM migration which has the most reducing impact on the standard deviation considering a few factors like affinity rules, cost-benefit analysis, pending recommendations etc. This migration selection steps are repeated until the imbalance in the cluster is minimized. After algorithm completes, an execution engine performs the recommended migrations, optionally requiring user approval. [38] Algorithm can be depicted as in Figure 4.1.

DRS load balancing rejects a move if it does not produce enough benefit in terms of improvement in the standard deviation value representing imbalance. The threshold used for this filtering is computed dynamically based on the number of hosts and VMs. The threshold is reduced significantly when imbalance is very high and moves to correct it are filtered by the normal threshold, so that many low-impact moves that are found in Greedy-Hill Climbing algorithm can be used to correct high imbalance. VM migrations that are recommended and incomplete in 5 minutes period will not be considered as factors in imbalance.

When there is a sudden steep increase in demand, a reactive operation such as Greedy-Hill can result in undesirably-high latency to obtain resources or difficulty in obtaining the resources needed to respond while those resources are being highly contended. When such sudden steep increases in demand are predictable, proactive operation can allow preparation for the demand spike to occur, to hide the latency of obtaining the resources and to avoid competing for resources with the demand spike itself.

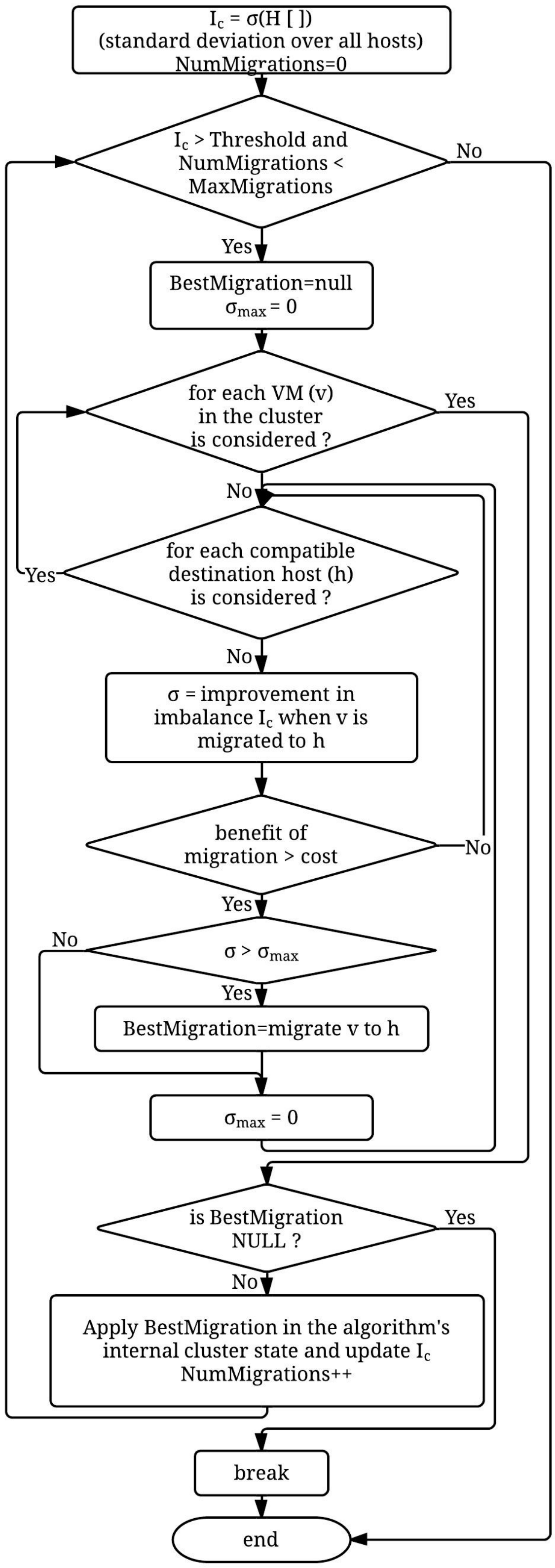

Figure 4.1: Greedy-Hill Climbing Algorithm.

Although Hill-Climbing is currently used as an optimization method in VMware environments other virtualization solutions such as Microsoft and Xen provide a high-level VM-based resource optimization functionality for load balancing. However the details of their approach are not known. [38] Since the future of load balancing promises proactive operations rather than reactive, this Hill-Climbing technique will likely be replaced by an algorithm that aims to find best allocation.

4.2 Genetic Algorithms Overview

Artificial intelligence techniques can be used to determine the best distribution for the virtual machines on the hosts. Genetic Algorithms (GA) are stochastic method for global search and optimization which imitates the evolution of the living beings, described by Charles Darwin. The evolutionary algorithms use the three main principles of the natural evolution: reproduction, natural selection and diversity of the species.

GA are part of the group of Evolutionary Algorithms and work with a set of individuals, representing possible solutions of the problem whose first generation is populated by random. The selection principle is applied by using a criterion, giving an evaluation for the individual with respect to the desired solution. The best-suited individuals create the next generation.

Genes in chromosomes carry the inherited cell information which determines the appearance of different peculiarities in biology.For the genetic algorithms, the chromosomes represent set of genes, which code the independent variables. Every chromosome represents a solution of the given problem. Individual and vector of variables will be used as other words for chromosomes. From other hand, the genes could be Boolean, integers, floating point or string variables, as well as any combination of the above.

A set of different chromosomes (individuals) forms a generation. By means of evolutionary operators, like selection, recombination and mutation an offspring population is created.

In the nature, the selection of individuals is performed by survival of the fittest. The more one individual is adapted to the environment - the bigger are its chances to survive and create an offspring and thus transfer its genes to the next population. In EA the selection of the best individuals is based on an evaluation of fitness function or fitness functions.If the optimization problem is a minimization one, than individuals with small value of the fitness function will have bigger chances for recombination and respectively for generating offspring.

The first step in the reproduction process is the recombination (crossover). In it the genes of the parents are used to form an entirely new chromosome. The typical recombination for the GA is an operation requiring two parents, but schemes with more parents area also possible.

The newly created by means of selection and crossover population can be further applied to mutation. Mutation means, that some elements of the DNA are changed. Those changes are caused mainly by mistakes during the copy process of the parent's genes. In the terms of GA, mutation means random change of the value of a gene in the population. The chromosome, which gene will be changed and the gene itself are chosen by random.

The GA hold a population of individuals (chromosomes), which evolve by means of selection and other operators like crossover and mutation. Every individual in the population gets an evaluation of its adaptation (fitness) to the environment. The selection chooses the best gene combinations (individuals), which through crossover and mutation should drive to better solutions in the next population.

The mechanisms used in GA are listed as follows [32] :

1. Generate initial population – inmost of the algorithms the first generation is randomly generated, by selecting the genes of the chromosomes among the allowed values for the gene. Because of the easier computational procedure it is accepted that all populations have the same number of individuals.
2. Calculation of the values of the function that we want to minimize or maximize.
3. Check for termination of the algorithm – as in the most optimization algorithms, it is possible to stop the genetic optimization by:
    i. Value of the function – the value of the function of the best individual is within defined range around a set value. It is not recommended to use this

criterion alone, because of the stochastic element in the search the procedure, the optimization might not finish within sensible time;

ii. Maximal number of iterations – this is the most widely used stopping criteria. It guarantees that the algorithms will give some results within some time, whenever it has reached the extremum or not;

iii. Stall generation – if within initially set number of iterations (generations) there is no improvement of the value of the fitness function of the best individual the algorithms stops.

4. Selection – between all individuals in the current population are chosen those, who will continue and by means of crossover and mutation will produce offspring population. At this stage elitism could be used – the best n individuals are directly transferred to the next generation. The elitism guarantees, that the value of the optimization function cannot get worst (once the extremum is reached it would be kept).

5. Crossover – the individuals chosen by selection recombine with each other and new individuals will be created. The aim is to get offspring individuals, that inherit the best possible combination of the characteristics (genes) of their parents.

6. Mutation – by means of random change of some of the genes, it is guaranteed that even if none of the individuals contain the necessary gene value for the extremum, it is still possible to reach the extremum.

7. New generation – the elite individuals chosen from the selection are combined with those who passed the crossover and mutation, and form the next generation.

4.3 BB-BC Overview

Big Bang-Big Crunch theory is based on the evolution of the universe which is introduced by Erol and Eksin. [2] It is a population based evolutionary computation method. The algorithm is shown to be fast convergent both in unimodal and multi–modal topologies. [3] There are two phases in this approach which are the Big Bang phase where energy dissipation produces disorder and randomness and the Big Crunch phase where randomly distributed particles are drawn into a single representative point via a center of mass. Representative point can be calculated in three ways:

i. By weighting the individuals with corresponding fitness evaluations as in (4.1) .

$$\overrightarrow{x^c} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{N}} \frac{1}{\mathrm{f}^{\mathrm{i}}} \overrightarrow{x^i}}{\sum_{\mathrm{i}=1}^{\mathrm{N}} \frac{1}{\mathrm{f}^{\mathrm{i}}}} \tag{4.1}$$

In (4.1) , $\overrightarrow{x^i}$ is the position vector for the i^{th} individual and f^i stands for the fitness value of the i^{th} individual.

ii. The fittest individual can be selected as the centre of mass.

iii. Crunching can be performed as the result of Nelder – Mead optimization method.

The randomness is assumed as energy dissipation in nature while convergence to optimum solution is seen as gravity. Energy dissipation creates disorder or chaos from ordered particles by producing new individuals from a converged solution. After a number of sequential Big Bangs and Big Crunches where the distribution of randomness within the search space during generations becomes smaller around the converged point computed during the Big Crunch phase. Convergence takes the population members as a whole in the Big Crunch phase that acts as a squeezing operator and eliminates the necessity for two-by-two combination calculations. The ratio of solution points around the optimum value to points away from optimum value must decrease as the number of iterations increases; but, in no case, it could be equal to zero, which means the end of the search. The convergence or the use of the previous knowledge (center of mass) can be accomplished by spreading new off-springs around this center of mass using a normal distribution operation in every direction where the standard deviation of this normal distribution function decreases as the number of iterations of the algorithm increases. [10]

The BB-BC method has been shown to outperform the enhanced classical Genetic Algorithm for many benchmark test functions [2] in means of low computational time and high convergence speed. [10]

The BB-BC approach takes the following steps:

Step 1. Form an initial generation in a random manner according to the search space.

Step 2. Calculate the fitness function values of all the candidate solutions.

Step 3. Find the center of mass by weighting individuals in the population. The fittest individual can also be selected as preferred in this thesis.

Step 4. Calculate new candidates around the center of mass by adding or subtracting a normally distributed random number whose standard deviation is decreased as the iterations elapse.

Step 5. Return to Step 2 until stopping criterion has been met or number of iterations has been reached.

The steps can be depicted as in Figure 4.2.

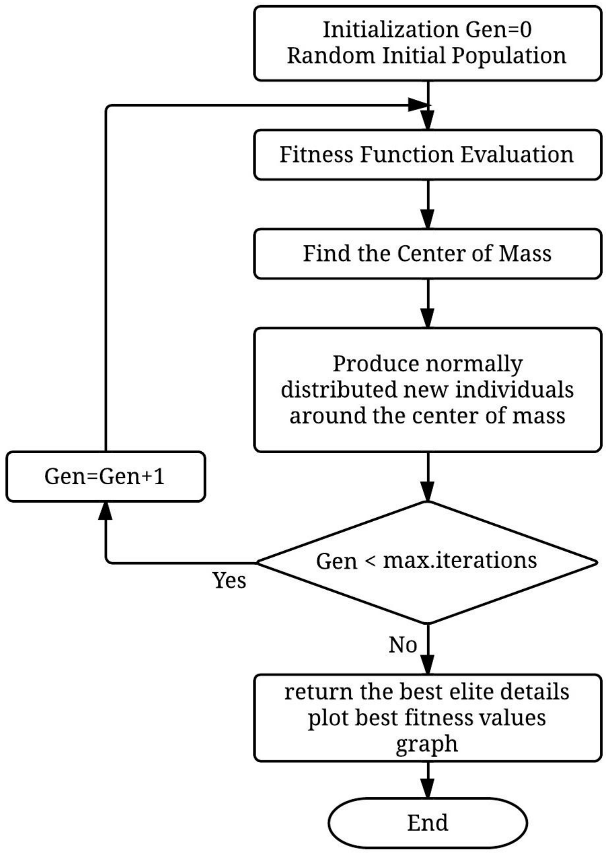

Figure 4.2: BB-BC Algorithm.

5. APPROACH TO THE PROBLEM

5.1 Representation

The objective of this thesis is to compare the optimization algorithms within BB-BC method by different convergence speeds and simple Genetic Algorithm while distributing the virtual machines in the physical hosts. The solution set is represented as each individual in the population is one dimensional array of a possible VMs allocation on the hosts as in Figure 5.1.

	$\mathbf{VM}_0$	$\mathbf{VM}_1$	$\mathbf{VM}_2$	$\cdots$	$\mathbf{VM}_{N-2}$	$\mathbf{VM}_{N-1}$
$\mathbf{VM}_i$:	$\mathbf{H}_j$	$\mathbf{H}_j$	$\mathbf{H}_j$	$\cdots$	$\mathbf{H}_j$	$\mathbf{H}_j$

Figure 5.1: Individual representation while encoding.

Physical hosts are numbered from 0 to $M-1$ and these host index values are randomly placed in the cells of individuals stating each VM is hosted in which host. The index of each cell in the individual is used to represent the virtual machine under consideration namely VM index. The random production of these individuals in the population will have possible allocation sets of VMs.

5.2 Fitness Evaluation

In order to calculate the fitness of each individual in the population, a few factors are taken into consideration. The first one is the CPU workload deviation of the individual from the calculated average load of the hosts. The second factor to be considered is the virtual CPUs assigned to VMs to keep the deviation low not to cause a possible bottleneck while reaching physical CPU resources. Meanwhile memory amounts of VMs also must be considered to be distributed equally in the cluster. The distinguishing factor between good distributions is the total migration cost of VMs in fitness function evaluation. The cost is calculated by the ratio of the migrated VMs' total memory to total memory amount of all VMs in the cluster. The more ratio is the more cost to the solution.

$$V_{cost} = \sum_{i=0}^{N-1} \left(V_{mem}\left[i\right], V_{t+1}\left[i\right] \neq V_t\left[i\right]\right) \tag{5.1}$$

(5.1) shows the sum of the migrated Virtual Machines' memory entitlements $V_{mem}\left[i\right]$ if their host is not the same at time $t+1$ compared to time t. Fitness function is denoted as in (5.2):

$$f = w_1 * \left(\frac{\sigma(H_{cpu}[])}{C_{cpu}}\right) + w_2 * \left(\frac{\sigma(H_{core}[])}{C_{core}}\right) + w_3 * \left(\frac{\sigma(H_{mem}[])}{C_{mem}}\right) + \left(\frac{V_{cost}}{V_{totalm}}\right) \tag{5.2}$$

f fitness value in (5.2) must be minimized where $\sum_{k=1}^{3} w_k = 1$ and $\forall w_k > 0$, C_{cpu} denotes maximum CPU capacity of physical host, V_{cost} denotes sum of migrated Virtual Machines' memory amounts, V_{totalm} denotes sum of Virtual Machines' memory amounts in the resource pool in (5.2).

There may be more than one condition that needs to be concurrently satisfied in many engineering issues. Minimizing the standard deviations of CPU workload, vCPU, memory and migrated VMs are normalized and are transformed into single-objective. So weighted sum of these conditions will be sufficient to find the best allocation of VMs. Many researchers have developed different approaches to select weights. One of the difficulties with the weighted sum method is that varying the weights consistently and continuously may not necessarily result in an accurate, complete representation of the Pareto optimal set. Thus, it is often necessary to incorporate user preferences for various objectives in order to determine a single suitable solution. With methods that incorporate "a priori articulation" of preferences, the user indicates preferences before running the optimization algorithm and subsequently allows the algorithm to determine a single solution that presumably reflects such preferences. Weights are selected in the fitness function as follows to reflect the effect of CPU workload compared to core and memory entitlements: $w_1 = 0.8$, w_2=0.1, w_3=0.1. Consequently, understanding how the weights affect the solution to the weighted sum method has implications concerning other approaches that involve similar method parameters. [29] Fitness formula is built on selected weights in advance in this research.

The normalization plays an important role in ensuring the consistency of optimal solutions with the preferences expressed by the decision maker. The goal of multi-objective optimization is to minimize simultaneously all of the objective functions. Attention is focused mainly on the case of weighted sum method that allows the multi-objective optimization problem to be cast as a single-objective mathematical optimization problem. This single objective function f is constructed as a sum of objective functions multiplied by weighting coefficients w_k whose sum equals to 1. Since these coefficients are normalized to 1, while this is not necessary in general, different objectives are put in the same units.

6. EXPERIMENTAL TESTS AND RESULTS

6.1 Experimental Setup

The BB-BC algorithm and traditional Genetic Algorithm are implemented using MATLAB 7 (R14) programming environment. Codes are presented in the Appendix B. The performance of the algorithms is measured using average of best fitness values in test cases. Experiment is conducted by applying each method 25 times for 24 different cases (CPU workload time sheets) on a system with Intel i7 1.73GHz CPU. Average values are calculated from 25 outputs of each algorithm. In each attempt every algorithm is iterated as 200 generations for each of 25 runs to reach to the best fitness values. Number of individuals in each population is 400.

Greedy-Hill Climbing algorithm, currently state of art method, does not aim to find the best allocation of virtual machines. Hence it does not produce solution sets which have fitness values as high as in the BB-BC algorithms. Execution duration is used as discriminative property compared to Greedy-Hill and fitness value is used to differentiate among modified BB-BC and classical BB-BC. Comparison is made between the best fitness values in each method after 200 generations are iterated. Simple GA is also executed to receive the fitness values as a reference basis line while comparing BB-BC methods among themselves. Analysis of variance (ANOVA) and Tukey's Honestly Significant Difference (HSD) test are applied on the results of the experiments.

Following data are given as input to the algorithms in experimental tests.

- data of Virtual Machines' virtual CPU counts (e.g. VM Cores column in Table 6.5)
- data of Virtual Machines' memory amounts (e.g. VM Mem column in Table 6.5)
- data of CPU workloads of Virtual Machines in 24 different time intervals as time sheets (e.g. VM CPUs column in Table 6.5)
- data of Virtual Machines' initial distribution on hosts (e.g. Initial Host column in Table 6.5)

Example set of input data to solve the problem including all cpu time sheet is given in Table A.1. CPU Workload Data are generated randomly according to the number of cores assigned to the virtual machines.

The algorithms that are compared in this thesis are listed below:

(i) Traditional Genetic Algorithm implemented as in [32]

(ii) BB-BC algorithm with a constantly decreasing rate of convergence by generation count

(iii) BB-BC algorithm with a constantly decreasing rate of convergence by generation count resetting in every 30 generations

(iv) BB-BC algorithm with a constantly decreasing rate of convergence by generation count resetting in every 50 generations

(v) BB-BC algorithm with a constantly decreasing rate of convergence by generation count resetting in every 75 generations

Test cases are built on assumption of 20 virtual machines on 4 physical hosts and assumption of 40 virtual machines on 8 physical hosts for best allocation according to assigned resources and CPU workloads at the specified time sheets.

Parameters used in the codes are represented in Table 6.1. Bold typed values are constant for each run of the algorithms and normal typed values are changed according to the current algorithm that is executed.

Experiments are setup in two different characteristics by composing the test scenarios:

Scenario A. CPU workloads in 24 time intervals are supplied to algorithms with same initial VM host assignments every time

Scenario B. CPU workloads in 24 time intervals are supplied to algorithms with the VM allocations found in previous time interval

6.2 ANOVA and HSD Test on Results

Analysis of Variance (ANOVA) is a hypothesis-testing technique used to test the equality of two or more population (or treatment) means by examining the variances of samples that are taken. ANOVA allows one to determine whether the differences between the samples are simply due to random error (sampling errors) or whether there

Table 6.1: Table of parameters used in the MATLAB implementation.

Parameter Name	Used Values	Parameter Explanation
numberofHost	4, 8	number of physical hosts
representativeBitsofHosts	2, 3	number of bits to represent the host indices
len	40, 120	the length of genomes
popsize	**400**	The size of the population (must be an even number)
maxGens	**200**	The maximum number of generations allowed in a run
probCrossover	**1**	The probability of crossing over
probMutation	**0.003**	The mutation probability (per bit)
BBBCFlag	0,1	0 => Do not use BigBang-BigCrunch 1 => Use BigBang-BigCrunch
BBBCmethod	0-4	0 => use simpleGA 1 => use mod30 2 => use mod50 3 => use mod75 4 => converge according to generation index
crossoverType	**2**	0 => no crossover 1 => 1 point crossover 2 => uniform crossover
verboseFlag	**1**	0 => run quietly 1 => display details of each generation
useMaskRepositoriesFlag	**1**	0 => generate uniform crossover and mutation masks on the fly. Slower. 1 => draw uniform crossover and mutation masks from a pregenerated repository of randomly generated bits. Significantly improves the speed of the code with no apparent changes in the behavior of the SGA
MigCostActive	**1**	0 => vMotion cost is not calculated 1 => vMotion cost of the VMs are calculated in cost function according to VM memories.

are systematic treatment effects that causes the mean in one group to differ from the mean in another.

Most of the time ANOVA is used to compare the equality of three or more means, however when the means from two samples are compared using ANOVA it is equivalent to using a t-test to compare the means of independent samples.

ANOVA is based on comparing the variance (or variation) between the data samples to variation within each particular sample. If the between variation is much larger than the within variation, the means of different samples will not be equal. If the between and within variations are approximately the same size, then there will be no significant difference between sample means.

The thesis tries to improve the performance of classical BB-BC by modifying convergence rates in the resource allocation problem. Since heuristic algorithms may not produce fitness values as good as stochastic methods, traditional GA with a few enhancements is used in ANOVA instead of Greedy-Hill. The execution durations can be a challenge point against Greedy-Hill in order to find a solution in 5 minutes interval between two imbalance controls. It is seen that algorithm executions take less than 3 minutes in order to complete 200 generations for both test scenarios.

In ANOVA, the best fitness values from algorithms in Test scenario A are analysed. Null hypothesis is "There is no significant difference in the best fitness values of modified BB-BC algorithms, classical BB-BC and traditional GA". It is rejected in all time sheets for alpha=0.05. This result is expected since traditional GA is compared with BB-BC algorithms as it will be anticipated when Greedy-Hill algorithm is chosen.

Numerator degrees of freedom is 4 for comparing 5 algorithms, denominator degrees of freedom is 120 since the number of executions for each algorithm is 25. The F distribution table critical F values and the studentized range distribution q values according to (4,120) are given in Table 6.2.

Table 6.2: F critical values and q range values used in ANOVA and Tukey's HSD test.

	alpha=0.10	alpha=0.05	alpha=0.025	alpha=0.01
F critical value (4,120)	1.9923	2.4472	2.8943	3.48
q value (4,120)	3.276	3.685	4.053	4.497

As it will be seen from the Table 6.3 the Null Hypothesis is rejected by these F values found in ANOVA for each time interval and Tukey's HSD is applied as post-hoc test to find how the means differ.

Table 6.3: F values found in ANOVA in 24 time sheets.

Time Sheet	F(4,120)
1	[25.3770]
2	[36.1828]
3	[37.3296]
4	[28.8197]
5	[27.2007]
6	[22.0937]
7	[29.4504]
8	[24.7656]
9	[18.0408]
10	[36.3010]
11	[29.9729]
12	[21.0026]
13	[14.0700]
14	[31.2063]
15	[42.0526]
16	[31.0270]
17	[30.2938]
18	[21.7932]
19	[33.1669]
20	[23.6856]
21	[18.1472]
22	[30.7531]
23	[19.2182]
24	[22.9223]

The MATLAB code used for ANOVA and Tukey's HSD is presented in Appendix B6. Tukey's HSD tests are applied for 24 time sheets according to both alpha=0.1 and alpha=0.05.

The output results are presented in the Appendix C. In Appendix C1, at least one of the three modified BB-BC compared to classical BB-BC is statistically significant with 90% confidence coefficient in all 24 time intervals as seen in Figure 6.1. However it is seen that from Appendix C2 Tukey's HSD tests have statistically significant differences between modified BB-BC and classical BB-BC in 9 time intervals among 24 time intervals with 95% confidence as in Figure 6.2. All results are summarized in Table 6.4.

Table 6.4: HSD values between modified BB-BC and classical BB-BC in 24 time intervals.

Time Sheet	mod75 vs. genindex	mod50 vs. genindex	mod30 vs. genindex	HSD %90 confidence	HSD %95 confidence
1	2.995	1.820	1.858	2.767	3.112
2	3.492	3.561	0.796	3.119	3.509
3	2.339	1.872	3.097	2.708	3.046
4	1.784	1.332	2.897	1.788	2.012
5	1.915	2.188	2.506	2.205	2.480
6	0.532	0.387	0.942	0.795	0.895
7	2.736	3.380	4.012	3.581	4.029
8	2.680	1.771	3.016	2.710	3.049
9	1.349	1.698	1.707	1.557	1.751
10	4.217	4.211	4.639	2.828	3.181
11	0.214	0.529	1.965	1.795	2.019
12	1.378	2.191	1.556	1.979	2.226
13	2.316	1.468	0.088	2.099	2.361
14	0.938	2.755	1.339	2.473	2.782
15	2.160	1.679	2.694	1.527	1.718
16	2.761	4.302	4.696	3.436	3.865
17	4.312	5.220	4.502	3.440	3.870
18	3.096	4.289	6.017	4.038	4.542
19	3.930	4.700	4.444	3.419	3.846
20	3.898	2.125	3.960	3.579	4.026
21	4.785	3.870	2.792	4.256	4.788
22	3.571	4.807	3.567	3.100	3.487
23	0.736	1.924	0.620	1.773	1.995
24	4.272	2.544	3.052	3.092	3.478

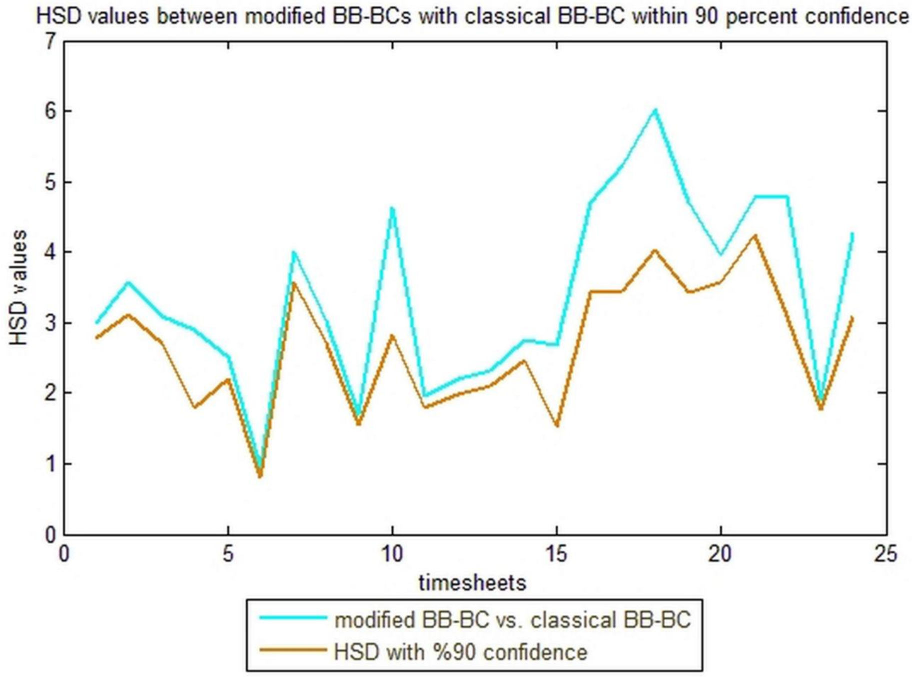

Figure 6.1: HSD values between modified BB-BCs with classical BB-BC within 90% confidence.

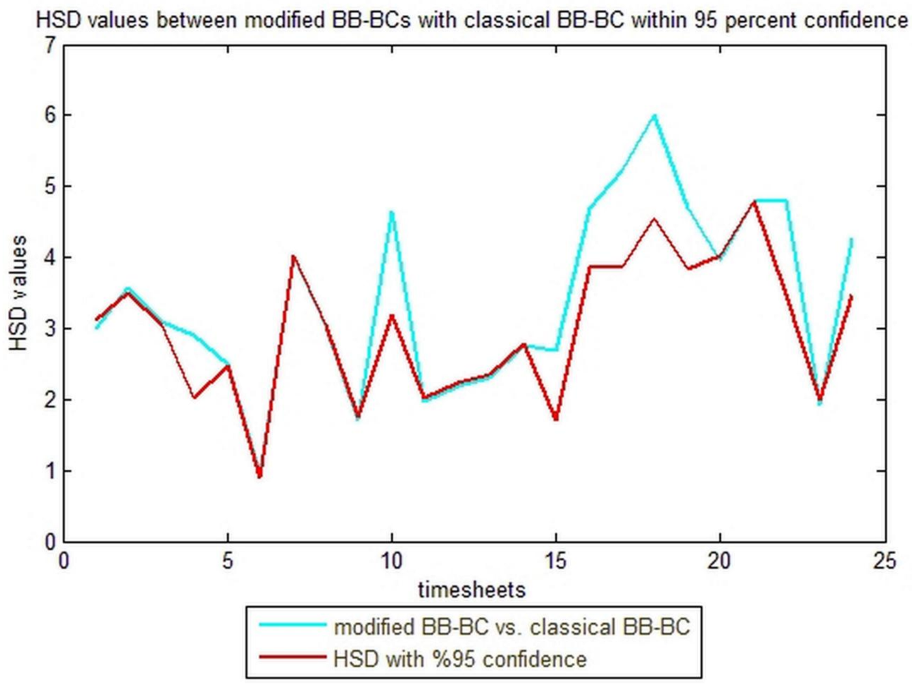

Figure 6.2: HSD values between modified BB-BCs with classical BB-BC within 95% confidence.

6.3 Example Graphs and Results from the Experiments

Scenario A is applied with 20 VMs on 4 hosts and scenario B is applied with 40 VMs on 8 hosts and 20 VMs on 4 hosts. Results are plotted by the best fitness values in the test scenarios and experiments as follows.

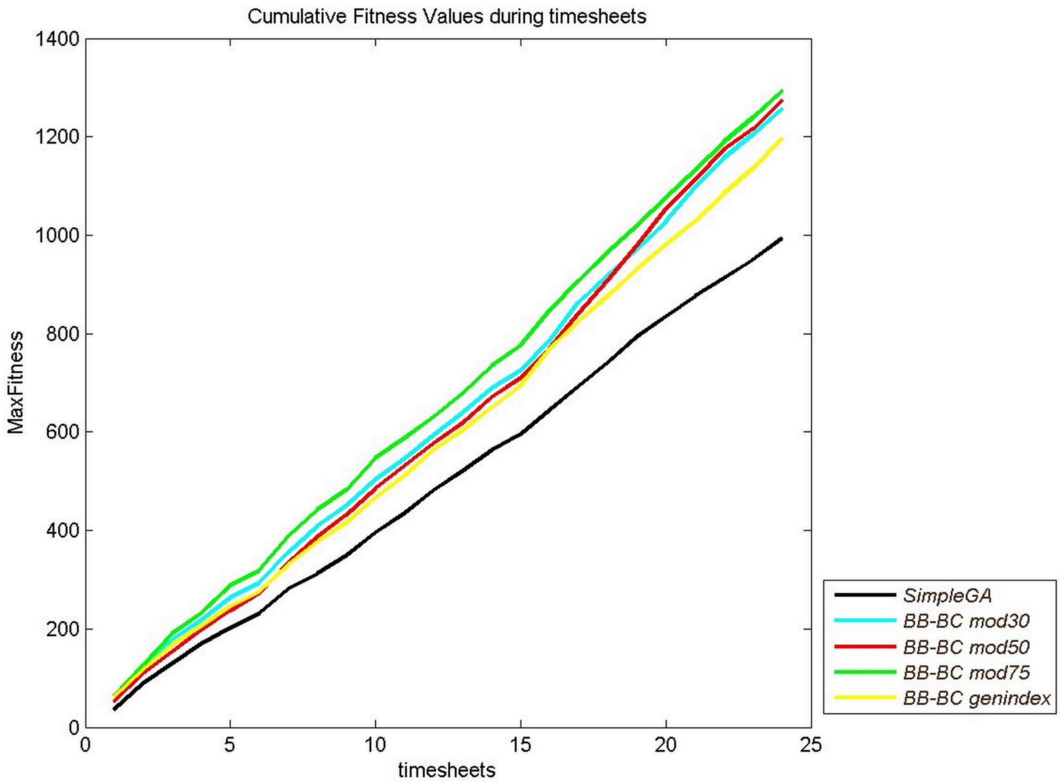

Figure 6.3: Cumulative values are reached in scenario B after an attempt of run with 20 VMs on 4 hosts.

It is seen that resetting convergence rate to a high value in normal distribution gives better results compared to GA and constantly decreasing convergence rate in Figure 6.3. Better results compared to GA in resetting convergence rate to a high value in normal distribution can be seen in Figure 6.4 also. The difference between results of distinct convergence rates with 8 hosts is not so obvious since the problem is easier compared to the one with 4 hosts.

Average of several executions for a specific time sheet in test scenario A is seen in Figure 6.5. In order to realize the performances of the algorithms, best fitness values for 24 time sheets are also compared cumulatively in Figure 6.6.

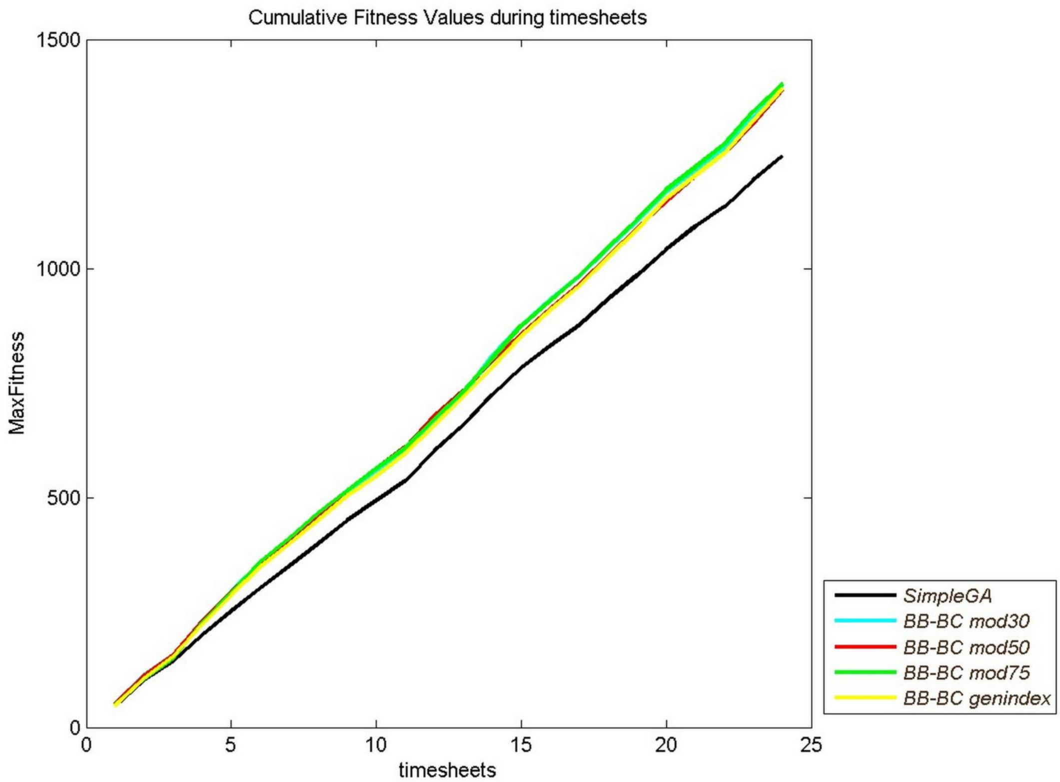

Figure 6.4: Cumulative values are reached in scenario B after an attempt of run with 40 VMs on 8 hosts.

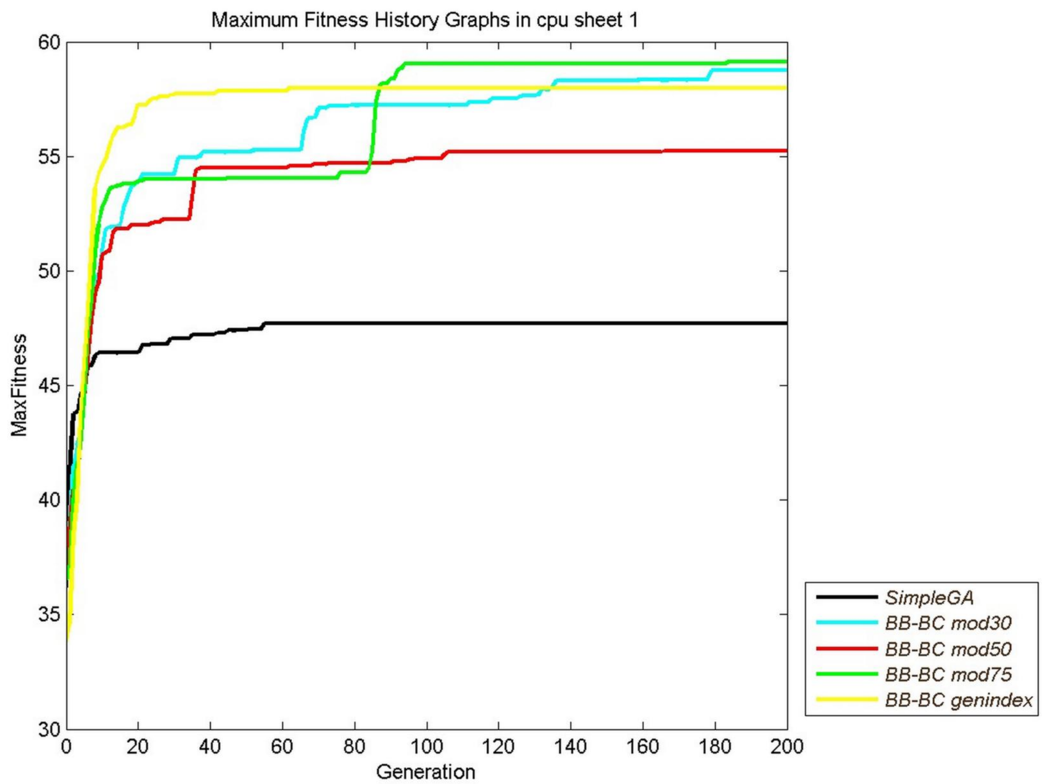

Figure 6.5: Example graph of average outputs out of randomly chosen 5 runs during 200 generations in scenario A with 20 VMs on 4 hosts for time sheet(1).

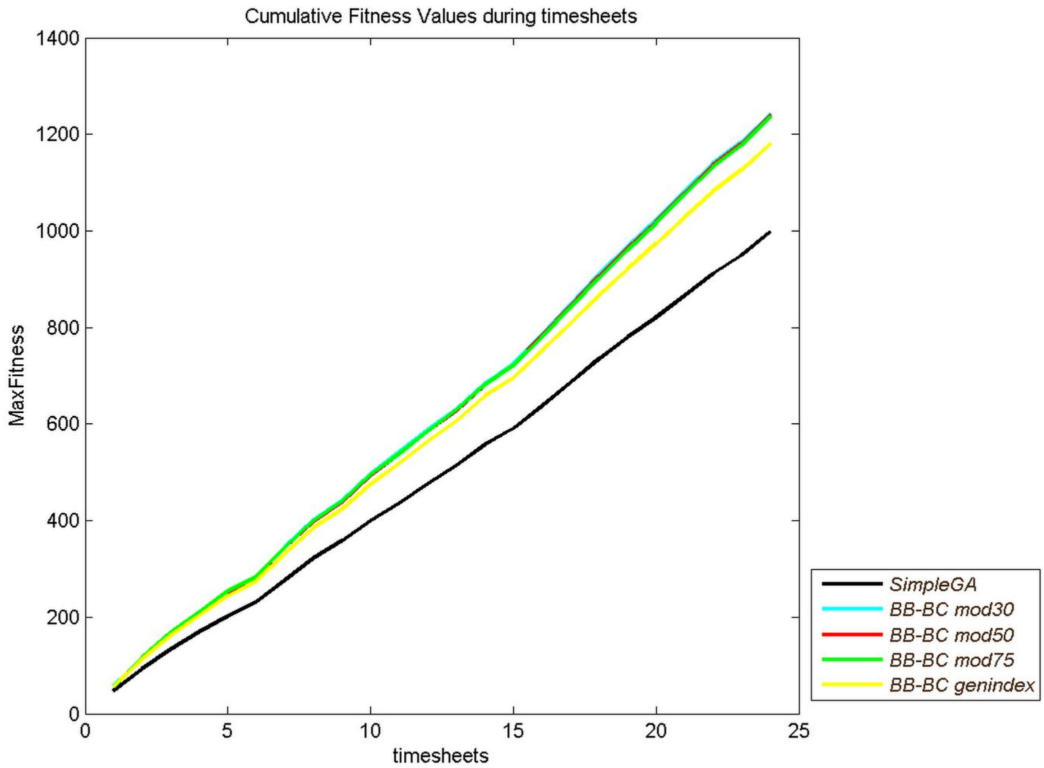

Figure 6.6: Cumulative average values of 25 runs are reached in scenario A after an attempt of run with 20 VMs on 4 hosts.

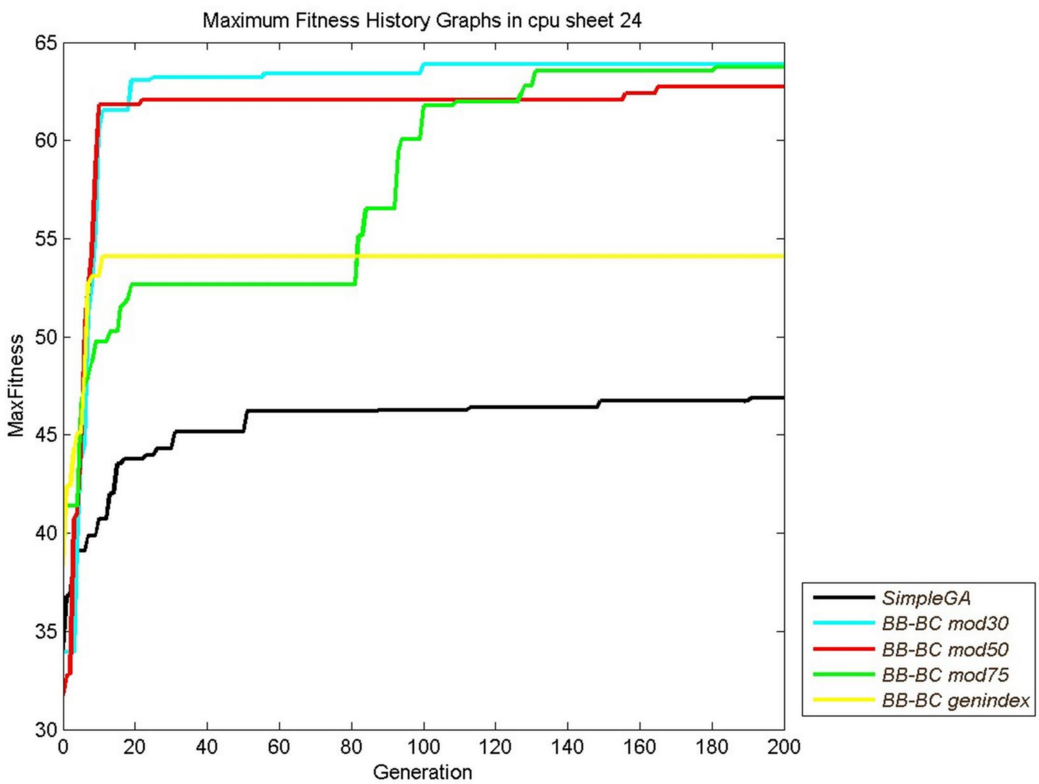

Figure 6.7: Example graph of outputs after an attempt during 200 generations in scenario A with 20 VMs on 4 hosts for time sheet(24).

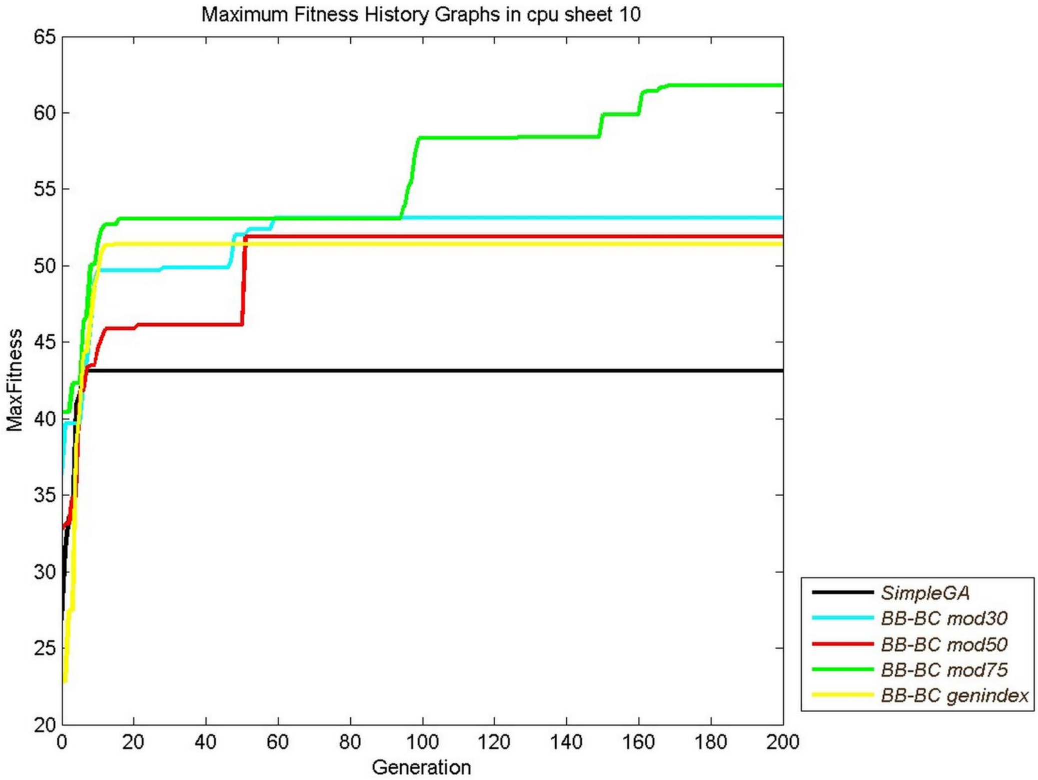

Figure 6.8: Example graph of outputs after an attempt during 200 generations in scenario A with 20 VMs on 4 hosts for time sheet(10).

If a few specific time sheets are taken as examples to examine, it can be seen that the difference of speed while reaching to the best fitness and goodness of these fitness values. In Figure 6.7 Convergence Rate Resetting effect is easily seen in "BB-BC mod75" after the 80^{th} generation and "BB-BC mod50" after the 160^{th} generation. In Figure 6.8 Convergence Rate Resetting effect is seen in "BB-BC mod75" after the 100^{th} and 155^{th} generation and "BB-BC mod50" after the 50^{th} generation.

The Table 6.5 gives the initial and final states of the VM distribution on physical hosts with resource assignments including what is changed after BB-BC algorithm has found a better distribution with a convergence rate resetting in every 75 generations for time sheet(24).

It is also seen that 7 Virtual Machines are replaced from their previous hosts to other hosts to balance the workload with a cost of 208 GB copied memory between hosts. Migrated VMs are written in italic and bold in Table 6.5. The difference of standard deviations in Cores, Memories and CPU workloads in the cluster is seen after "BB-BC mod 75" algorithm is applied for time sheet(24) in Table 6.6.

Table 6.5: Distribution of VMs initially and eventually for time sheet(24).

VM index	VM cores	VM Mem (GB)	VM CPUs (GHz) time sheet:24	Initial Host	Final Host
0	4	24	3	0	0
1	8	64	18	0	0
2	6	32	9	0	0
3	2	16	5	0	0
4	2	8	1	***1***	***0***
5	12	96	18	***1***	***2***
6	10	96	11	1	1
7	8	40	4	1	1
8	4	20	0	***1***	***0***
9	8	28	7	***2***	***3***
10	1	4	1	2	2
11	2	8	4	2	2
12	4	36	0	***2***	***1***
13	6	44	16	2	2
14	1	4	2	3	3
15	2	12	2	***3***	***2***
16	1	2	0	3	3
17	2	8	3	***3***	***1***
18	2	16	0	3	3
19	20	128	18	3	3

Table 6.6: The improvements are reached after BB-BC mod 75 is applied for time sheet(24).

Host index	Host cores		Host memories		Host CPUs	
	before	after	before	after	before	after
0	20	26	136	164	35	33
1	36	24	260	180	34	33
2	21	23	120	164	28	34
3	28	32	170	178	25	33
standard deviation	7.411	4.031	70.95	8.7	4.796	0.5

7. CONCLUSIONS AND RECOMMENDATIONS

In order to improve efficiency a new approach is proposed for BB-BC optimization algorithm in this thesis. Experiments simulated a Cloud Computing environment consists of Virtual Machines and physical hosts. In a typical Resource Management world, metrics are measured by a Global Resource Scheduler tool in every t minutes, that is 5 minutes in VMware environment, and actions are taken to distribute the Virtual Machines properly on the physical hosts in the cluster. Currently VMware uses Hill-Climbing technique since intelligent distribution of VMs with varying workloads is not assumed worthwhile. However some future works aim to operate proactively. Therefore evolutionary algorithms may likely be preferred soon. Since Greedy-Hill does not have a claim to find the best allocation, BB-BC algorithms' execution durations are examined to fit into the 5 minutes time interval used for cluster's workload balance control.

Using BB-BC algorithm with varying convergence speeds is presented in this research. In order to do that, a population of random distributions of VMs is generated initially and the next generations of this population are produced by the best individual using a normal distribution function. The fitness of each solution is calculated based on the values of the deviations of the hosts from the average loads on CPU workload, core and memory that is accompanied with the proposed placement. The migration cost for each virtual machine is also taken into consideration while calculating the fitness for the individuals. This is in order to get the best solution with the least total amount of memory migrations possible. In a real environment there may be some other constraints related to resource sharing policies, limits or reservations. These constraints can easily be reflected to the fitness evaluation of individuals.

Each algorithm is executed 25 times for 24 CPU workloads for comparison. ANOVA and Tukey's HSD test as post-hoc test are used to analyse the significance of the results statistically. It is derived that periodically resetting the convergence rate in Big Bang phase revealed better fitness values compared to constantly decreasing the convergence rate with the confidence rate of 90% for all time intervals while number of time intervals that shows significantly difference is only 9 among 24 with 95% confidence coefficient.

REFERENCES

[1] **LeHong, H. and Fenn, J.** (2013). Emerging Technologies Hype Cycle for 2013: Redefining the Relationship, `http://www.gartner.com/newsroom/id/2575515`.

[2] **Erol, O. and Eksin, I.** (2006). A New Optimization Method: Big Bang - Big Crunch, *Advances in Engineering Software*, ***37***, 106–111.

[3] **Genc, H.**, **Eksin, I. and Erol, O.** (2013). Big Bang - Big Crunch Optimization Algorithm with Local Directional Moves, *Turkish Journal of Electrical Engineering & Computer Sciences*, ***21***, 1359–1375.

[4] **Rao, C.G.K. and Yesuratnam, G.** (2012). Big-Bang and Big-Crunch (BB-BC) and FireFly Optimization (FFO): Application and Comparison to Optimal Power Flow with Continuous and Discrete Control Variables, *International Journal on Electrical Engineering and Informatics*, ***4***(4), 575–583.

[5] **Kripka, M. and Kripka, R.** (2008). Big Crunch Optimization Method, *Proceedings of the International Conference on Engineering Optimization*, Rio de Janeiro, Brazil.

[6] **Singh, S. and Kumar, P.** (2012). Application of Big Bang Big Crunch Algorithm to Software Testing, *International Journal of Computer Science and Communication*, ***3***(1), 259–262.

[7] **Kaveh, A. and Talatahari, S.** (2010). A Discrete Big Bang - Big Crunch Algorithm for Optimal Design Skeletal Structures, *Asian Journal of Civil Engineering (Building and Housing)*, ***11***(1), 103–122.

[8] **Tabakov, P.** (2011). Big Bang - Big Crunch Optimization Method in Optimum Design of Complex Composite Laminates, *World Academy of Science, Engineering and Technology*, ***53***, 835–839.

[9] **Verma, D.H.K. and Manekar, Y.** (2012). Big Bang Big Crunch Optimization for Determination of Worst Case Loading Margin, *IJERA*, ***2***(4), 421–426.

[10] **Labbi, Y. and Attous, D.** (2010). Big Bang-Big Crunch Optimization Algorithm for Economic Dispatch with Valve-Point Effect, *Journal of Theoretical and Applied Information Technology*, ***16***(1), 48–56.

[11] **Genc, H.**, **Erol, O.**, **Eksin, I.**, **Berber, M. and Guleryuz, B.** (2012). A Stochastic Neighborhood Search Approach for Airport Gate Assignment Problem, *Expert Systems with Applications*, ***39***, 316–327.

[12] **Kim, S.**, **Kim, Y. and N. Song, C.K.** (2010). Adaptable Scheduling Schemes for Scientific Applications on Science Cloud, *Proceedings of the IEEE International Conference on Cluster Computing Workshops and Posters*, pp.1–3.

[13] **Lu, X. and Gu, Z.** (2011). A Load-Adaptive Cloud Resource Scheduling Model Based on Ant Colony Algorithm, *Proceedings of the IEEE International Conference on Cloud Computing and Intelligence Systems*, pp.296–300.

[14] **Zhong, H.**, **Tao, K. and Zhang, X.** (2010). An Approach to Optimized Resource Scheduling Algorithm for Open-source Cloud Systems, *Proceedings of the Fifth Annual ChinaGrid Conference*, pp.124–129.

[15] **Chen, H.**, **Jin, H. and K. Hu, J.H.** (2010). Dynamic Switching-Frequency Scaling: Scheduling Overcommitted Domains in Xen VMM, *Proceedings of the 39th International Conference on Parallel Processing*, pp.287–296.

[16] **Warneke, D. and Kao, O.** (2011). Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud, *IEEE Transactions on Parallel and Distributed Dystems*, ***22***(6), 985–997.

[17] **Kochut, A. and Beaty, K.** (2007). On Strategies for Dynamic Resource Management in Virtualized Server Environments, *Proceedings of the 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp.193–200.

[18] **Song, Y.**, **Sun, Y. and Shi, W.** (2011). A Two-Tiered On-Demand Resource Allocation Mechanism for VM-Based Data Centers, *IEEE Transactions on Services Computing*, ***6***(1), 116–129.

[19] **Hu, J.**, **Gu, J.**, **Sun, G. and Zhao, T.** (2010). A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment, *Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms and Programming*, pp.89–96.

[20] **Bi, J.**, **Zhu, Z.**, **Tian, R. and Wang, Q.** (2010). Dynamic Provisioning Modeling for Virtualized Multi-tier Applications in Cloud Data Center, *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, pp.370–377.

[21] **Assaf, E.**, **Badr, A. and Farag, I.** (2010). Application of Computational Intelligence to Virtualized Data Center Management, *International Journal of Computer Applications*, ***55***(10), 42–49.

[22] **Goiri, I.**, **Julia, F.**, **Nou, R.**, **Berral, J.**, **Guitart, J. and Torres, J.** (2010). Energy-Aware Scheduling in Virtualized Datacenters, *Proceedings of the IEEE International Conference on Cluster Computing*, pp.58–67.

[23] **Wang, L.**, **Xu, J.**, **Zhao, M.**, **Tu, Y. and Fortes, J.** (2011). Fuzzy Modeling Based Resource Management for Virtualized Database Systems, *Proceedings of the 19th Annual IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp.32–42.

[24] **Jin, H.**, **Deng, L.**, **Wu, S. and Shi, X.** (2011). Dynamic Processor Resource Configuration in Virtualized Environments, *Proceedings of the IEEE International Conference on Services Computing*, pp.32–39.

[25] **Ngatchou, P.**, **Zarei, A. and El-Sharkawi, M.** (2005). Pareto Multi Objective Optimization, *Proceedings of the 13th International Conference on Intelligent Systems Application to Power Systems*, pp.84–91.

[26] **Grodzevich, O. and Romanko, O.** (2006). Normalization and Other Topics in MultiObjective Optimization, *Proceedings of the Fields-MITACS Industrial Problems Workshop*, pp.89–101.

[27] **Fonsecay, C. and Flemingz, P.** (1993). Genetic Algorithms for Multiobjective Optimization:Formulation, Discussion and Generalization, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp.416–423.

[28] **Stanimirovic, I.**, **Zlatanovic, M. and Petkovic, M.** (2011). On The Linear Weighted Sum Method for Multi-objective Optimization, *Facta Universitatis (Nis) Ser. Math. Inform*, ***26***, 49–63.

[29] **Marler, R. and Arora, J.** (2012). The Weighted Sum Method for Multi-objective Optimization: New Insights, *Structural and Multidisciplinary Optimization*, ***41***(6), 853–862.

[30] **Kim, I. and Weck, O.** (2006). Adaptive Weighted Sum Method for Multiobjective Optimization: A New Method for Pareto Front Generation, *Structural and Multidisciplinary Optimization*, ***31***(2), 105–116.

[31] **Corne, D.W.**, **Jerram, N.R.**, **Knowles, J.D.**, **Oates, M.J. and J, M.** (2001). PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, Morgan Kaufmann Publishers, pp.283–290.

[32] **Popov, A.**, (2005). Genetic Algorithms for Optimization Programs for Matlab User Manual, Hamburg, 1.0 edition.

[33] **Burjorjee, K.** (2009). TurboGA: A Simple Genetic Algorithm With a Powerful Performance Enhancing Tweak, `http://www.xentrik.net/scripts/info/turboga__a_simple_genetic_algorithm_with_a_powerful_performance_enhancing_tweak.html`.

[34] **Url-1** (2010). VMware vSphere: The CPU Scheduler in VMware ESX 4.1, `http://www.vmware.com/files/pdf/techpaper/VMW_vSphere41_cpu_schedule_ESX.pdf`.

[35] **Url-2** (2006). VMware Infrastructure Resource Management with VMware DRS, `http://www.vmware.com/pdf/vmware_drs_wp.pdf`.

[36] **Url-3** (2013). The CPU Scheduler in VMware vSphere 5.1 Performance Study, `http://www.vmware.com/files/pdf/techpaper/VMware-vSphere-CPU-Sched-Perf.pdf`.

[37] **Url-4** (2012). VMware vSphere Resource Management on ESXi 5.1 and vCenter Server 5.1, `http://pubs.vmware.com/vsphere-51/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-51-resource-management-guide.pdf`.

[38] **Gulati, A.**, **Holler, A.**, **Ji, M.**, **Shanmuganathan, G.**, **Waldspurger, C. and Zhu, X.** (2013). VMware Distributed Resource Management: Design, Implementation, and Lessons Learned, *VMware Technical Journal Summer*, pp.45–64.

APPENDICES

APPENDIX A : Example set of input data to solve the problem including all cpu timesheets
APPENDIX B : Example MATLAB codes used for tests
APPENDIX B1 : main.m file for the main program
APPENDIX B2 : stdevHostsMig.m file for calculation of fitness values considering migration costs
APPENDIX B3 : eliteDetails.m file for the details of best fit member
APPENDIX B4 : draw_graphs.m file to plot the graphs of outputs
APPENDIX B5 : draw_avg.m file to plot the averages of the results
APPENDIX B6 : anova.m file used for ANOVA and Tukey's HSD test
APPENDIX C : ANOVA and Tukey's HSD test results
APPENDIX C1 : ANOVA_and_HSD_results_for_alpha_0.1.txt output
APPENDIX C2 : ANOVA_and_HSD_results_for_alpha_0.05.txt output

APPENDIX A

Table A.1: Example set of input data to solve the problem including all cpu timesheets.

VM index	VM Mem	VM Core	Initial Host	VM CPUs (GHz) for 24 timesheets																							
				t:1	t:2	t3	t:4	t:5	t:6	t:7	t:8	t:9	t:10	t:11	t:12	t:13	t:14	t:15	t:16	t:17	t:18	t:19	t:20	t:21	t:22	t:23	t:24
0	24	4	0	6	11	3	11	4	4	9	11	0	3	1	0	7	7	9	2	6	10	7	1	4	11	5	3
1	64	8	0	19	13	3	4	5	22	12	6	2	22	1	8	8	14	0	10	0	10	10	13	2	17	11	18
2	32	6	0	4	17	1	0	16	17	12	5	1	4	5	2	6	8	7	5	15	2	17	12	15	2	1	9
3	16	2	0	5	0	5	1	5	0	1	0	0	2	3	0	4	3	1	3	2	1	1	4	0	5	1	5
4	8	2	1	4	5	1	0	5	1	0	5	5	0	2	3	3	0	4	0	1	3	1	2	4	3	4	1
5	96	12	1	28	22	2	0	30	0	24	2	23	13	12	3	7	15	8	0	6	7	23	24	29	15	16	18
6	96	10	1	27	7	21	23	29	9	8	28	4	22	10	3	7	28	21	9	4	13	11	8	0	16	25	11
7	40	8	1	18	14	18	0	8	0	20	14	8	1	3	21	4	4	22	17	16	14	22	12	2	16	3	4
8	20	4	1	2	2	4	9	0	2	6	0	1	2	2	8	6	3	4	5	9	9	3	1	10	5	8	0
9	28	8	2	4	16	15	15	14	0	6	7	19	12	3	3	12	6	21	19	14	20	4	21	14	15	5	7
10	4	1	2	1	2	1	0	1	1	1	2	0	0	1	0	1	2	2	1	2	1	0	1	2	2	2	1
11	8	2	2	1	0	5	1	3	5	1	5	1	5	3	2	2	3	3	1	2	0	4	2	3	2	3	4
12	36	4	2	1	4	5	7	11	10	5	5	11	9	9	2	4	2	0	8	2	4	9	11	7	7	1	0
13	44	6	2	14	17	17	14	8	1	11	6	5	15	17	5	10	1	7	16	16	10	8	0	9	16	13	16
14	4	1	3	1	0	2	1	0	2	0	1	1	2	2	0	1	0	2	2	0	0	1	2	2	0	0	2
15	12	2	3	0	2	3	3	2	2	4	0	0	1	2	1	0	3	0	5	5	5	2	1	2	4	1	2
16	2	1	3	1	0	2	0	2	0	2	1	0	2	1	1	0	2	1	1	2	2	1	1	1	1	2	0
17	8	2	3	3	3	2	1	0	4	0	5	2	1	2	1	3	5	2	0	4	4	5	4	2	2	1	3
18	16	2	3	1	4	3	2	0	0	3	4	5	5	1	4	5	2	4	0	4	2	4	3	0	4	5	0
19	128	20	3	35	17	38	41	1	49	21	19	42	12	34	2	39	39	52	22	10	31	14	10	25	13	44	18

APPENDIX B

APPENDIX B1

```
% main.m
% This is the main program that runs different convergence rated BB-BC algorithms besides
% a modified version of simpleGA by Keki Burjorjee.
% BSD license for simpleGA codes is attached at the end of the code.

diary('degeroku.txt');
numberofHosts=4           % number of physical hosts
representativeBitsofHosts=2;
len=40                    % The length of the genomes
popSize=400;               % The size of the population (must be an even number)
maxGens=200;              % The maximum number of generations allowed in a run
probCrossover=1;           % The probability of crossing over.
probMutation=0.003;        % The mutation probability (per bit)
BBBCFlag=1;                % BigBang-BigCrunch is described in thesis
                           % 1 => Use BigBang-BigCrunch
                           % 0 => Do not use BigBang-BigCrunch
BBBCmethod=3;              % 0 => use simpleGA
                           % 1 => use mod30
```

```
                           % 2 => use mod50
                           % 3 => use mod75
                           % 4 => converge according to generation index
crossoverType=2;           % 0 => no crossover
                           % 1 => 1pt crossover
                           % 2 => uniform crossover
verboseFlag=1;             % 1 => display details of each generation
                           % 0 => run quietly
useMaskRepositoriesFlag=1; % 1 => draw uniform crossover and mutation masks from
                           %      a pregenerated repository of randomly generated bits.
                           %      Significantly improves the speed of the code with
                           %      no apparent changes in the behavior of
                           %      the SGA
                           % 0 => generate uniform crossover and mutation
                           %      masks on the fly. Slower.
MigCostActive=1;           % vMotion cost of the VMs are calculated in cost function according to VM
                   % memories.


vmCores=csvread('vmCores2.csv');
vmCores
vmMemories=csvread('vmMemories2.csv');
vmMemories
initialvmHostAssignment=csvread('initialvmHostAssignment2.csv');
prevVMassignment=initialvmHostAssignment;
CPUtimesheet=csvread('CPUtimesheet2.csv');
CPUtimesheet
```

```
cumMaxFitness=[];

for tindex=1:24
    tintervalname = num2str(tindex);
    switch BBBCmethod
        case 0
            tintervalname = ['SimpleGAtimeinterval' tintervalname '.txt'];
        case 1
            tintervalname = ['BBBCmod30timeinterval' tintervalname '.txt'];
        case 2
            tintervalname = ['BBBCmod50timeinterval' tintervalname '.txt'];
        case 3
            tintervalname = ['BBBCmod75timeinterval' tintervalname '.txt'];
        case 4
            tintervalname = ['BBBCgenindextimeinterval' tintervalname '.txt'];
        otherwise
            disp('wrong BBBC option');
    end;
    diary(tintervalname);
    vmCPUs=CPUtimesheet(:,[tindex]);
    vmCPUs
    maxFitnessHist=zeros(1,maxGens+1);
    eliteIndiv=[];
    bbbcIndiv=[];
    bbbcMasks=[];
    eliteFitness=-realmax;
```

```
if BBBCFlag < 1
    maskReposFactor=5;
    uniformCrossmaskRepos=rand(popSize/2,(len+1)*maskReposFactor)<0.5;
    mutmaskRepos=rand(popSize,(len+1)*maskReposFactor)<probMutation;
end
% the population is a popSize by len matrix of randomly generated boolean
% values
pop=rand(popSize,len)<.5;
for gen=0:maxGens
% evaluate the fitness of the population. The vector of fitness values
% returned  must be of dimensions 1 x popSize.

    if MigCostActive < 1
        fitnessVals=stdevHosts(pop, vmCores, vmMemories, vmCPUs, numberofHosts);
    else
        fitnessVals=stdevHostsMig(pop, vmCores, vmMemories, vmCPUs, numberofHosts,prevVMassignment);
    end
    [maxFitnessHist(1,gen+1),maxIndex]=max(fitnessVals);

    if eliteFitness<maxFitnessHist(gen+1)
        eliteFitness=maxFitnessHist(gen+1);
        eliteIndiv=pop(maxIndex,:);
        eliteDetails(eliteIndiv,pop,vmCores,vmMemories,vmCPUs,numberofHosts);
    end

    % display the generation number, the average Fitness of the population,
```

```
% and the maximum fitness of any individual in the population
if verboseFlag
    display(['gen=' num2str(gen,'%.3d') '   maxFitness=' ...
    num2str(maxFitnessHist(1,gen+1),'%3.3f')]);
end

% Normalize the fitness values and then create an array with the
% cumulative normalized fitness values (the last value in this array
% will be 1)
cumNormFitnessVals=cumsum(fitnessVals/sum(fitnessVals));
if BBBCFlag < 1
% Use fitness proportional selection with Roulette
% Wheel Sampling to determine the indices of the parents
% of all crossover operations
        markers=rand(1,popSize);
    [temp parentIndices]=histc(markers,[0 cumNormFitnessVals]);
    parentIndices=parentIndices(randperm(popSize));
   % determine the first parents of each mating pair
    firstParents=pop(parentIndices(1:popSize/2),:);
    % determine the second parents of each mating pair
    secondParents=pop(parentIndices(popSize/2+1:end),:);
    % create crossover masks
    if crossoverType==0
        masks=mutationOnlycrossmasks;
    elseif crossoverType==1
        masks=false(popSize/2, len);
```

```
    temp=ceil(rand(popSize/2,1)*(len-1));
    for i=1:popSize/2
        masks(i,1:temp(i))=true;
    end
else
    if useMaskRepositoriesFlag
        temp=floor(rand*len*(maskReposFactor-1));
        masks=uniformCrossmaskRepos(:,temp+1:temp+len);
    else
        masks=rand(popSize/2, len)<.5;
    end
end
% determine which parent pairs to leave uncrossed
reprodIndices=rand(popSize/2,1)<1-probCrossover;
masks(reprodIndices,:)=false;
% implement crossover
firstKids=firstParents;
firstKids(masks)=secondParents(masks);
secondKids=secondParents;
secondKids(masks)=firstParents(masks);
pop=[firstKids; secondKids];

% implement mutation
if useMaskRepositoriesFlag
    temp=floor(rand*len*(maskReposFactor-1));
    masks=mutmaskRepos(:,temp+1:temp+len);
```

```
      else
        masks=rand(popSize, len)<probMutation;
    end
    pop=xor(pop,masks);
    pop([popSize],:) = eliteIndiv;  % elitist model
else
%BBBCFlag
    for bbbcIndiv=1:popSize-1
        switch BBBCmethod
        %case 0
        %    tintervalname = ['SimpleGAtimeinterval' tintervalname '.txt'];
        case 1
            bbbcMasks = rand(1, len)< double(1 / (2*(mod(gen,30) + 1)))*2;
            % bb-bc implementation according to mod30 generation index convergence
        case 2
            bbbcMasks = rand(1, len)< double(1 / (2*(mod(gen,50) + 1)))*2;
            % bb-bc implementation according to mod50 generation index convergence
        case 3
            bbbcMasks = rand(1, len)< double(1 / (2*(mod(gen,75) + 1)))*2;
            % bb-bc implementation according to mod50 generation index convergence
        case 4
            bbbcMasks = rand(1, len)< double(1 / (2*(gen+1)))*2;
            % bb-bc implementation according to generation index convergence
        otherwise
            disp('wrong BBBC option');
    end;
```

```
                pop([bbbcIndiv],:) = xor(eliteIndiv,bbbcMasks);
          end
          pop([popSize],:) = eliteIndiv;  %elitist model
      end
  end
bestElite=eliteDetails(eliteIndiv,pop,vmCores,vmMemories,vmCPUs,numberofHosts)
      numberofVmotions=bestElite==prevVMassignment
      vmotions=0;
      vmotioncost=0;
      for vmindex=1:(len/representativeBitsofHosts)
          if bi2de(numberofVmotions([vmindex],:),'left-msb')<3
              vmotions=vmotions+1;
              vmotioncost=vmotioncost+vmMemories(vmindex);
          end
      end
      fprintf('vmotions: %d vmotioncost: %d \n', vmotions,vmotioncost);
  prevVMassignment=bestElite
  cumMaxFitness(tindex)=max(maxFitnessHist)

       strindex = num2str(tindex);

       switch BBBCmethod
          case 0
              strindexname = ['simpleGAmaxFitGraph_t' strindex '.txt'];
          case 1
              strindexname = ['BBBCmod30maxFitGraph_t' strindex '.txt'];
```

```
        case 2
              strindexname = ['BBBCmod50maxFitGraph_t' strindex '.txt'];
        case 3
            strindexname = ['BBBCmod75maxFitGraph_t' strindex '.txt'];
        case 4
            strindexname = ['BBBCgenindexmaxFitGraph_t' strindex '.txt'];
        otherwise
            disp('wrong BBBC option');
     end;

     dlmwrite(strindexname,maxFitnessHist);
prevVMassignment=initialvmHostAssignment;

end
cumsumMaxFitness=cumsum(cumMaxFitness)

    switch BBBCmethod
        case 0
            dlmwrite('simpleGAcumulative.txt',cumsumMaxFitness);
        case 1
            dlmwrite('BBBCmod30cumulative.txt',cumsumMaxFitness);
        case 2
            dlmwrite('BBBCmod50cumulative.txt',cumsumMaxFitness);
        case 3
            dlmwrite('BBBCmod75cumulative.txt',cumsumMaxFitness);
        case 4
```

```
                dlmwrite('BBBCgenindexcumulative.txt',cumsumMaxFitness);
            otherwise
                disp('wrong BBBC option');
        end;
diary off

%Copyright (c) 2009, Keki Burjorjee
%All rights reserved.

%Redistribution and use in source and binary forms, with or without modification, are permitted
%provided that the following conditions are met:
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the distribution
%THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
%AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
%IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
%ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
%LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
%CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
%SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
%INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
%CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
%ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
%POSSIBILITY OF SUCH DAMAGE.
```

APPENDIX B2

```
% stdevHostsMig.m

function fitness=stdevHostsMig(pop,vmCores,vmMemories,vmCPUs,numberofHosts,prevVMassignment)
    maxMem=192;
    maxCpu=58;
    maxCore=20;
    [popSize len]=size(pop);

    representativeBitsofHosts=2;
    numberofVMs=len/representativeBitsofHosts;
    hostSumCpu=zeros(numberofHosts,1);
    hostSumCore=zeros(numberofHosts,1);
    hostSumMemory=zeros(numberofHosts,1);
    vmHostAssignment=zeros(numberofVMs,representativeBitsofHosts);
    fitnessValues=zeros(1,popSize);
    invfitnessValues=zeros(1,popSize);
    memtotal=0;
    for vmmem=1:numberofVMs
        memtotal=memtotal+vmMemories(vmmem);
    end
```

```
for vmassignment=1:popSize
    hostSumCpu=zeros(numberofHosts,1);
    hostSumCore=zeros(numberofHosts,1);
    hostSumMemory=zeros(numberofHosts,1);
    tableVMassignment=transpose(reshape(transpose(pop([vmassignment],:)),representativeBitsofHosts,
    numberofVMs));

    numberofVmotions=tableVMassignment==prevVMassignment;
    vmotions=0;
    vmotioncost=0;
    for vmindex=1:(len/representativeBitsofHosts)
        if bi2de(numberofVmotions([vmindex],:),'left-msb')<3
            vmotions=vmotions+1;
            vmotioncost=vmotioncost+vmMemories(vmindex);
        end
    end

    for vm=1:numberofVMs
        hostIndex=bi2de(tableVMassignment([vm],:),'left-msb');
        hostSumCpu(hostIndex+1)=hostSumCpu(hostIndex+1)+vmCPUs(vm);
        hostSumCore(hostIndex+1)=hostSumCore(hostIndex+1)+vmCores(vm);
        hostSumMemory(hostIndex+1)=hostSumMemory(hostIndex+1)+vmMemories(vm);
    end
    for host=1:numberofHosts
        if hostSumMemory(host) > maxMem
            hostSumMemory(host)=realmax;
```

```
        end
    end
    stdevvalue=((8*(0.1 + (std2(hostSumCpu)/maxCpu)))+(1*(0.1+(std2(hostSumCore)/maxCore)))
    +(1*(0.1+(std2(hostSumMemory)/maxMem))))+ (1*(vmotioncost/memtotal));
    if stdevvalue == 0
        hostSumCpu
        mean2(hostSumCpu)
        std2(hostSumCpu)
        hostSumCore
        mean2(hostSumCore)
        std2(hostSumCore)
        hostSumMemory
        mean2(hostSumMemory)
        std2(hostSumMemory)
        tableVMassignment
        pause
    end
    invfitnessValues(1,vmassignment)=1./stdevvalue;
    if sum(invfitnessValues) == 0
        invfitnessValues(1,vmassignment)=0.0001;
    end
end
fitness=100 .* invfitnessValues;
```

APPENDIX B3

```
% eliteDetails.m
function details=eliteDetails(eliteGenome,pop,vmCores,vmMemories,vmCPUs,numberofHosts)
[popSize len]=size(pop);

representativeBitsofHosts=length(de2bi((numberofHosts-1),'left-msb'));
numberofVMs=len/representativeBitsofHosts;
hostSumCpu=zeros(numberofHosts,1);
hostSumCore=zeros(numberofHosts,1);
hostSumMemory=zeros(numberofHosts,1);
vmHostAssignment=zeros(numberofVMs,representativeBitsofHosts);

 tableVMassignment=transpose(reshape(transpose(eliteGenome),representativeBitsofHosts,numberofVMs));
    for vm=1:numberofVMs
    hostIndex=bi2de(tableVMassignment([vm],:),'left-msb');
    hostSumCpu(hostIndex+1)=hostSumCpu(hostIndex+1)+vmCPUs(vm);
    hostSumCore(hostIndex+1)=hostSumCore(hostIndex+1)+vmCores(vm);
    hostSumMemory(hostIndex+1)=hostSumMemory(hostIndex+1)+vmMemories(vm);
    end

    hostSumCpu
    mean2(hostSumCpu)
```

```
std2(hostSumCpu)
hostSumCore
mean2(hostSumCore)
std2(hostSumCore)
hostSumMemory
mean2(hostSumMemory)
std2(hostSumMemory)

details=tableVMassignment
```

APPENDIX B4

```
%draw_graphs.m
maxGens=200;
for tindex=1:24
    %Plot graphs of maxFitnessHistory during t
    strindex = num2str(tindex);
    strindexname = ['simpleGAmaxFitGraph_t' strindex '.txt'];
    maxFitnessHistSimpleGA=csvread(strindexname);
    strindexname = ['BBBCmod30maxFitGraph_t' strindex '.txt'];
    maxFitnessHistBBBCmod30=csvread(strindexname);
    strindexname = ['BBBCmod50maxFitGraph_t' strindex '.txt'];
    maxFitnessHistBBBCmod50=csvread(strindexname);
    strindexname = ['BBBCmod75maxFitGraph_t' strindex '.txt'];
    maxFitnessHistBBBCmod75=csvread(strindexname);
    strindexname = ['BBBCgenindexmaxFitGraph_t' strindex '.txt'];
    maxFitnessHistgenindex=csvread(strindexname);

    ff=figure(tindex);
    hold off
    P1=plot([0:maxGens],maxFitnessHistSimpleGA,'k-');
    hold on
    P2=plot([0:maxGens],maxFitnessHistBBBCmod30,'c-');
```

```
    hold on
    P3=plot([0:maxGens],maxFitnessHistBBBCmod50,'r-');
    hold on
    P4=plot([0:maxGens],maxFitnessHistBBBCmod75,'g-');
    hold on
    P5=plot([0:maxGens],maxFitnessHistgenindex,'y-');
    set(P1,'LineWidth',2);
    set(P2,'LineWidth',2);
    set(P3,'LineWidth',2);
    set(P4,'LineWidth',2);
    set(P5,'LineWidth',2);

    str = sprintf('Maximum Fitness History Graphs in cpu sheet %d',tindex);
    title(str);
    xlabel('Generation')
    ylabel('MaxFitness')
    hleg = legend('SimpleGA','BB-BC mod30','BB-BC mod50','BB-BC mod75','BB-BC genindex',...
              'Location','SouthEastOutside');
    % Make the text of the legend italic and color it brown
    set(hleg,'FontAngle','italic','TextColor',[.3,.2,.1]);
    saveas(ff,str,'png');
end

    strindexname = ['simpleGAcumulative.txt'];
    cumFitnessHistSimpleGA=csvread(strindexname);
    strindexname = ['BBBCmod30cumulative.txt'];
```

```
cumFitnessHistBBBCmod30=csvread(strindexname);
strindexname = ['BBBCmod50cumulative.txt'];
cumFitnessHistBBBCmod50=csvread(strindexname);
strindexname = ['BBBCmod75cumulative.txt'];
cumFitnessHistBBBCmod75=csvread(strindexname);
strindexname = ['BBBCgenindexcumulative.txt'];
cumFitnessHistgenindex=csvread(strindexname);

ff=figure(tindex+1);
hold off
P1=plot([1:24],cumFitnessHistSimpleGA,'k-');
hold on
P2=plot([1:24],cumFitnessHistBBBCmod30,'c-');
hold on
P3=plot([1:24],cumFitnessHistBBBCmod50,'r-');
hold on
P4=plot([1:24],cumFitnessHistBBBCmod75,'g-');
hold on
P5=plot([1:24],cumFitnessHistgenindex,'y-');
set(P1,'LineWidth',2);
set(P2,'LineWidth',2);
set(P3,'LineWidth',2);
set(P4,'LineWidth',2);
set(P5,'LineWidth',2);

str = sprintf('Cumulative Fitness Values during timesheets');
```

```
title(str);
xlabel('timesheets')
ylabel('MaxFitness')
hleg = legend('SimpleGA','BB-BC mod30','BB-BC mod50','BB-BC mod75','BB-BC genindex',...
          'Location','SouthEastOutside');
% Make the text of the legend italic and color it brown
set(hleg,'FontAngle','italic','TextColor',[.3,.2,.1]);
saveas(ff,str,'png');
```

APPENDIX B5

```
%draw_avg.m
maxGens=200;
outputCount=5

for outputidx=1:5
    %average values
    strindex = num2str(outputidx);
    strindexname = ['simpleGAmaxFitGraph_t' strindex '.txt'];
    maxFitnessHistSimpleGA=csvread(strindexname);
    strindexname = ['BBBCmod30maxFitGraph_t' strindex '.txt'];
    maxFitnessHistBBBCmod30=csvread(strindexname);
    strindexname = ['BBBCmod50maxFitGraph_t' strindex '.txt'];
    maxFitnessHistBBBCmod50=csvread(strindexname);
    strindexname = ['BBBCmod75maxFitGraph_t' strindex '.txt'];
    maxFitnessHistBBBCmod75=csvread(strindexname);
    strindexname = ['BBBCgenindexmaxFitGraph_t' strindex '.txt'];
    maxFitnessHistgenindex=csvread(strindexname);

for tindex=1:24
    %Plot graphs of maxFitnessHistory during t
    strindex = num2str(tindex);
```

```
strindexname = ['simpleGAmaxFitGraph_t' strindex '.txt'];
 maxFitnessHistSimpleGA=csvread(strindexname);
 strindexname = ['BBBCmod30maxFitGraph_t' strindex '.txt'];
 maxFitnessHistBBBCmod30=csvread(strindexname);
 strindexname = ['BBBCmod50maxFitGraph_t' strindex '.txt'];
 maxFitnessHistBBBCmod50=csvread(strindexname);
 strindexname = ['BBBCmod75maxFitGraph_t' strindex '.txt'];
 maxFitnessHistBBBCmod75=csvread(strindexname);
 strindexname = ['BBBCgenindexmaxFitGraph_t' strindex '.txt'];
 maxFitnessHistgenindex=csvread(strindexname);

 ff=figure(tindex);
 hold off
 P1=plot([0:maxGens],maxFitnessHistSimpleGA,'k-');
 hold on
 P2=plot([0:maxGens],maxFitnessHistBBBCmod30,'c-');
 hold on
 P3=plot([0:maxGens],maxFitnessHistBBBCmod50,'r-');
 hold on
 P4=plot([0:maxGens],maxFitnessHistBBBCmod75,'g-');
 hold on
 P5=plot([0:maxGens],maxFitnessHistgenindex,'y-');
 set(P1,'LineWidth',2);
 set(P2,'LineWidth',2);
 set(P3,'LineWidth',2);
 set(P4,'LineWidth',2);
```

```
    set(P5,'LineWidth',2);

    str = sprintf('Maximum Fitness History Graphs in cpu sheet %d',tindex);
    title(str);
    xlabel('Generation')
    ylabel('MaxFitness')
    hleg = legend('SimpleGA','BB-BC mod30','BB-BC mod50','BB-BC mod75','BB-BC genindex',...
              'Location','SouthEastOutside');
    % Make the text of the legend italic and color it brown
    set(hleg,'FontAngle','italic','TextColor',[.3,.2,.1]);
    saveas(ff,str,'png');
end

    strindexname = ['simpleGAcumulative.txt'];
    cumFitnessHistSimpleGA=csvread(strindexname);
    strindexname = ['BBBCmod30cumulative.txt'];
    cumFitnessHistBBBCmod30=csvread(strindexname);
    strindexname = ['BBBCmod50cumulative.txt'];
    cumFitnessHistBBBCmod50=csvread(strindexname);
    strindexname = ['BBBCmod75cumulative.txt'];
    cumFitnessHistBBBCmod75=csvread(strindexname);
    strindexname = ['BBBCgenindexcumulative.txt'];
    cumFitnessHistgenindex=csvread(strindexname);

    ff=figure(tindex+1);
    hold off
```

```
P1=plot([1:24],cumFitnessHistSimpleGA,'k-');
hold on
P2=plot([1:24],cumFitnessHistBBBCmod30,'c-');
hold on
P3=plot([1:24],cumFitnessHistBBBCmod50,'r-');
hold on
P4=plot([1:24],cumFitnessHistBBBCmod75,'g-');
hold on
P5=plot([1:24],cumFitnessHistgenindex,'y-');
set(P1,'LineWidth',2);
set(P2,'LineWidth',2);
set(P3,'LineWidth',2);
set(P4,'LineWidth',2);
set(P5,'LineWidth',2);

str = sprintf('Cumulative Fitness Values during timesheets');
title(str);
xlabel('timesheets')
ylabel('MaxFitness')
hleg = legend('SimpleGA','BB-BC mod30','BB-BC mod50','BB-BC mod75','BB-BC genindex',...
          'Location','SouthEastOutside');
% Make the text of the legend italic and color it brown
set(hleg,'FontAngle','italic','TextColor',[.3,.2,.1]);
saveas(ff,str,'png');
```

APPENDIX B6

```
% anova.m
diary('ANOVA_and_HSD_results_for_alpha_0.05.txt');
for tindex=1:24
tintervalname = num2str(tindex)
    tintervalname = ['anova' tintervalname '.csv'];
    X=csvread(tintervalname);
    [p,table,stats] = anova1(X)
    [c,m,h,nms] = multcompare(stats,'alpha',0.05,'display','off','ctype','hsd')
end
diary off

diary('ANOVA_and_HSD_results_for_alpha_0.1.txt');
for tindex=1:24
tintervalname = num2str(tindex)
    tintervalname = ['anova' tintervalname '.csv'];
    X=csvread(tintervalname);
    [p,table,stats] = anova1(X)
    [c,m,h,nms] = multcompare(stats,'alpha',0.1,'display','off','ctype','hsd')
end
diary off
```

APPENDIX C

APPENDIX C1

ANOVA_and_HSD_results_for_alpha_0.1.txt output:

```
tintervalname =

1


p =

  2.9976e-015


table =

  Columns 1 through 5

    'Source'     'SS'            'df'     'MS'
'F'
    'Columns'    [1.8098e+003]    [  4]    [452.4529]
[25.3770]
    'Error'      [2.1395e+003]    [120]    [ 17.8293]
       []
    'Total'      [3.9493e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [2.9976e-015]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
```

```
    source: 'anova1'
     means: [57.6089 56.4346 56.4724 54.6142 47.0772]
        df: 120
         s: 4.2225


c =

    1.0000    2.0000   -1.7982    1.1743    4.1467
    1.0000    3.0000   -1.8359    1.1365    4.1090
    1.0000    4.0000    0.0222    2.9947    5.9671
    1.0000    5.0000    7.5593   10.5317   13.5042
    2.0000    3.0000   -3.0102   -0.0378    2.9347
    2.0000    4.0000   -1.1520    1.8204    4.7928
    2.0000    5.0000    6.3850    9.3574   12.3299
    3.0000    4.0000   -1.1143    1.8582    4.8306
    3.0000    5.0000    6.4228    9.3952   12.3676
    4.0000    5.0000    4.5646    7.5370   10.5095


m =

   57.6089    0.8445
   56.4346    0.8445
   56.4724    0.8445
   54.6142    0.8445
   47.0772    0.8445


h =

     []


nms =

1
2
3
4
5


tintervalname =

2
```

```
p =

     0


table =

    'Source'     'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'    [3.2805e+003]    [  4]    [820.1357]
[36.1828]    [      0]
    'Error'      [2.7200e+003]    [120]    [ 22.6665]
       []           []
    'Total'      [6.0005e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [59.2053 59.2742 56.5100 55.7137 45.3714]
        df: 120
         s: 4.7609


c =

    1.0000    2.0000   -3.4204   -0.0689    3.2826
    1.0000    3.0000   -0.6563    2.6952    6.0467
    1.0000    4.0000    0.1401    3.4916    6.8431
    1.0000    5.0000   10.4824   13.8339   17.1854
    2.0000    3.0000   -0.5873    2.7642    6.1157
    2.0000    4.0000    0.2090    3.5605    6.9120
    2.0000    5.0000   10.5513   13.9028   17.2543
    3.0000    4.0000   -2.5551    0.7964    4.1479
    3.0000    5.0000    7.7871   11.1386   14.4901
    4.0000    5.0000    6.9908   10.3423   13.6938


m =

   59.2053    0.9522
   59.2742    0.9522
   56.5100    0.9522
   55.7137    0.9522
```

```
   45.3714    0.9522


h =

     []


nms =

1
2
3
4
5


tintervalname =

3


p =

     0


table =

    'Source'     'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'    [2.5506e+003]    [  4]    [637.6470]
[37.3296]    [      0]
    'Error'      [2.0498e+003]    [120]    [ 17.0815]
       []           []
    'Total'      [4.6004e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [52.5680 52.1016 53.3264 50.2293 41.0558]
        df: 120
         s: 4.1330
```

```
c =

    1.0000    2.0000   -2.4430    0.4664    3.3758
    1.0000    3.0000   -3.6678   -0.7584    2.1510
    1.0000    4.0000   -0.5708    2.3387    5.2481
    1.0000    5.0000    8.6027   11.5122   14.4216
    2.0000    3.0000   -4.1342   -1.2248    1.6846
    2.0000    4.0000   -1.0372    1.8723    4.7817
    2.0000    5.0000    8.1363   11.0458   13.9552
    3.0000    4.0000    0.1876    3.0971    6.0065
    3.0000    5.0000    9.3611   12.2706   15.1800
    4.0000    5.0000    6.2640    9.1735   12.0829


m =

   52.5680    0.8266
   52.1016    0.8266
   53.3264    0.8266
   50.2293    0.8266
   41.0558    0.8266


h =

     []


nms =

1
2
3
4
5


tintervalname =

4


p =

  1.1102e-016
```

```
table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [   858.7402]    [  4]    [214.6851]
[28.8197]
    'Error'      [   893.9094]    [120]    [  7.4492]
       []
    'Total'      [1.7526e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [1.1102e-016]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [42.5132 42.0620 43.6268 40.7296 36.1052]
        df: 120
         s: 2.7293


c =

    1.0000    2.0000   -1.4701    0.4512    2.3725
    1.0000    3.0000   -3.0349   -1.1136    0.8077
    1.0000    4.0000   -0.1377    1.7836    3.7049
    1.0000    5.0000    4.4867    6.4080    8.3293
    2.0000    3.0000   -3.4861   -1.5648    0.3565
    2.0000    4.0000   -0.5889    1.3324    3.2537
    2.0000    5.0000    4.0355    5.9568    7.8781
    3.0000    4.0000    0.9759    2.8972    4.8185
    3.0000    5.0000    5.6003    7.5216    9.4429
    4.0000    5.0000    2.7031    4.6244    6.5457


m =
```

```
   42.5132    0.5459
   42.0620    0.5459
   43.6268    0.5459
   40.7296    0.5459
   36.1052    0.5459


h =

     []


nms =

1
2
3
4
5


tintervalname =

5


p =

  4.4409e-016


table =

  Columns 1 through 5

    'Source'     'SS'             'df'     'MS'
'F'
    'Columns'    [1.2324e+003]    [  4]    [308.0955]
[27.2007]
    'Error'      [1.3592e+003]    [120]    [ 11.3268]
       []
    'Total'      [2.5916e+003]    [124]            []
       []

  Column 6

    'Prob>F'
```

```
    [4.4409e-016]
               []
               []


stats = 

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [42.4612 42.7348 43.0524 40.5464 34.6588]
        df: 120
         s: 3.3655


c =

    1.0000    2.0000   -2.6428   -0.2736    2.0956
    1.0000    3.0000   -2.9604   -0.5912    1.7780
    1.0000    4.0000   -0.4544    1.9148    4.2840
    1.0000    5.0000    5.4332    7.8024   10.1716
    2.0000    3.0000   -2.6868   -0.3176    2.0516
    2.0000    4.0000   -0.1808    2.1884    4.5576
    2.0000    5.0000    5.7068    8.0760   10.4452
    3.0000    4.0000    0.1368    2.5060    4.8752
    3.0000    5.0000    6.0244    8.3936   10.7628
    4.0000    5.0000    3.5184    5.8876    8.2568


m =

   42.4612    0.6731
   42.7348    0.6731
   43.0524    0.6731
   40.5464    0.6731
   34.6588    0.6731


h =

     []


nms =

1
2
```

```
3
4
5


tintervalname =

6


p =

  1.1025e-013


table =

    'Source'     'SS'           'df'     'MS'           'F'
         'Prob>F'
    'Columns'    [130.2079]    [  4]    [32.5520]    [22.
0937]    [1.1025e-013]
    'Error'      [176.8030]    [120]    [ 1.4734]
   []               []
    'Total'      [307.0110]    [124]           []
   []               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [31.5508 31.4052 31.9604 31.0184 29.0460]
        df: 120
         s: 1.2138


c =

    1.0000    2.0000   -0.7089    0.1456    1.0001
    1.0000    3.0000   -1.2641   -0.4096    0.4449
    1.0000    4.0000   -0.3221    0.5324    1.3869
    1.0000    5.0000    1.6503    2.5048    3.3593
    2.0000    3.0000   -1.4097   -0.5552    0.2993
    2.0000    4.0000   -0.4677    0.3868    1.2413
    2.0000    5.0000    1.5047    2.3592    3.2137
    3.0000    4.0000    0.0875    0.9420    1.7965
```

```
    3.0000    5.0000    2.0599    2.9144    3.7689
    4.0000    5.0000    1.1179    1.9724    2.8269


m =

   31.5508    0.2428
   31.4052    0.2428
   31.9604    0.2428
   31.0184    0.2428
   29.0460    0.2428


h =

     []


nms =

1
2
3
4
5


tintervalname =

7


p =

     0


table =

    'Source'     'SS'              'df'     'MS'
'F'          'Prob>F'
    'Columns'    [3.5198e+003]     [  4]    [879.9399]
[29.4504]    [      0]
    'Error'      [3.5854e+003]     [120]    [ 29.8787]
       []           []
    'Total'      [7.1052e+003]     [124]            []
       []           []
```

```
stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [59.3500 59.9948 60.6268 56.6144 46.3292]
        df: 120
         s: 5.4661


c =

    1.0000    2.0000   -4.4927   -0.6448    3.2031
    1.0000    3.0000   -5.1247   -1.2768    2.5711
    1.0000    4.0000   -1.1123    2.7356    6.5835
    1.0000    5.0000    9.1729   13.0208   16.8687
    2.0000    3.0000   -4.4799   -0.6320    3.2159
    2.0000    4.0000   -0.4675    3.3804    7.2283
    2.0000    5.0000    9.8177   13.6656   17.5135
    3.0000    4.0000    0.1645    4.0124    7.8603
    3.0000    5.0000   10.4497   14.2976   18.1455
    4.0000    5.0000    6.4373   10.2852   14.1331


m =

   59.3500    1.0932
   59.9948    1.0932
   60.6268    1.0932
   56.6144    1.0932
   46.3292    1.0932


h =

     []


nms =

1
2
3
4
5
```

```
tintervalname =

8


p =

  5.8842e-015


table =

  Columns 1 through 5

    'Source'     'SS'               'df'     'MS'
'F'
    'Columns'    [1.6952e+003]     [  4]    [423.7918]
[24.7656]
    'Error'      [2.0535e+003]     [120]    [ 17.1121]
       []
    'Total'      [3.7486e+003]     [124]            []
       []

  Column 6

    'Prob>F'
    [5.8842e-015]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [55.0876 54.1780 55.4232 52.4072 45.4472]
        df: 120
         s: 4.1367


c =

    1.0000    2.0000   -2.0024    0.9096    3.8216
    1.0000    3.0000   -3.2476   -0.3356    2.5764
    1.0000    4.0000   -0.2316    2.6804    5.5924
    1.0000    5.0000    6.7284    9.6404   12.5524
```

```
    2.0000    3.0000   -4.1572   -1.2452    1.6668
    2.0000    4.0000   -1.1412    1.7708    4.6828
    2.0000    5.0000    5.8188    8.7308   11.6428
    3.0000    4.0000    0.1040    3.0160    5.9280
    3.0000    5.0000    7.0640    9.9760   12.8880
    4.0000    5.0000    4.0480    6.9600    9.8720


m =

   55.0876    0.8273
   54.1780    0.8273
   55.4232    0.8273
   52.4072    0.8273
   45.4472    0.8273


h =

     []


nms =

1
2
3
4
5


tintervalname =

9


p =

  1.2656e-011


table =

  Columns 1 through 5

    'Source'     'SS'             'df'     'MS'
'F'
```

```
    'Columns'     [   407.4122]    [  4]    [101.8531]
[18.0408]
    'Error'       [   677.4835]    [120]    [  5.6457]
        []
    'Total'       [1.0849e+003]    [124]            []
        []

  Column 6

    'Prob>F'
    [1.2656e-011]
              []
              []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [40.0964 40.4460 40.4548 38.7476 35.7040]
        df: 120
         s: 2.3761


c =

    1.0000    2.0000   -2.0223   -0.3496    1.3231
    1.0000    3.0000   -2.0311   -0.3584    1.3143
    1.0000    4.0000   -0.3239    1.3488    3.0215
    1.0000    5.0000    2.7197    4.3924    6.0651
    2.0000    3.0000   -1.6815   -0.0088    1.6639
    2.0000    4.0000    0.0257    1.6984    3.3711
    2.0000    5.0000    3.0693    4.7420    6.4147
    3.0000    4.0000    0.0345    1.7072    3.3799
    3.0000    5.0000    3.0781    4.7508    6.4235
    4.0000    5.0000    1.3709    3.0436    4.7163


m =

   40.0964    0.4752
   40.4460    0.4752
   40.4548    0.4752
   38.7476    0.4752
   35.7040    0.4752
```

```
h =

     []


nms =

1
2
3
4
5


tintervalname =

10


p =

     0


table =

    'Source'     'SS'            'df'     'MS'
'F'         'Prob>F'
    'Columns'    [2.7058e+003]    [  4]    [676.4622]
[36.3010]    [      0]
    'Error'      [2.2362e+003]    [120]    [ 18.6348]
       []           []
    'Total'      [4.9420e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [55.3664 55.3604 55.7888 51.1496 43.5832]
        df: 120
         s: 4.3168
```

```
c =

    1.0000    2.0000   -3.0328    0.0060    3.0448
    1.0000    3.0000   -3.4612   -0.4224    2.6164
    1.0000    4.0000    1.1780    4.2168    7.2556
    1.0000    5.0000    8.7444   11.7832   14.8220
    2.0000    3.0000   -3.4672   -0.4284    2.6104
    2.0000    4.0000    1.1720    4.2108    7.2496
    2.0000    5.0000    8.7384   11.7772   14.8160
    3.0000    4.0000    1.6004    4.6392    7.6780
    3.0000    5.0000    9.1668   12.2056   15.2444
    4.0000    5.0000    4.5276    7.5664   10.6052


m =

   55.3664    0.8634
   55.3604    0.8634
   55.7888    0.8634
   51.1496    0.8634
   43.5832    0.8634


h =

     []


nms =

1
2
3
4
5


tintervalname =

11


p =

     0
```

```
table =

    'Source'     'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'    [   900.1484]    [  4]    [225.0371]
[29.9729]    [      0]
    'Error'      [   900.9613]    [120]    [  7.5080]
       []           []
    'Total'      [1.8011e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [43.5812 43.8964 45.3328 43.3676 37.5588]
        df: 120
         s: 2.7401


c =

    1.0000    2.0000   -2.2441   -0.3152    1.6137
    1.0000    3.0000   -3.6805   -1.7516    0.1773
    1.0000    4.0000   -1.7153    0.2136    2.1425
    1.0000    5.0000    4.0935    6.0224    7.9513
    2.0000    3.0000   -3.3653   -1.4364    0.4925
    2.0000    4.0000   -1.4001    0.5288    2.4577
    2.0000    5.0000    4.4087    6.3376    8.2665
    3.0000    4.0000    0.0363    1.9652    3.8941
    3.0000    5.0000    5.8451    7.7740    9.7029
    4.0000    5.0000    3.8799    5.8088    7.7377


m =

   43.5812    0.5480
   43.8964    0.5480
   45.3328    0.5480
   43.3676    0.5480
   37.5588    0.5480


h =
```

```
     []


nms =

1
2
3
4
5


tintervalname =

12


p =

  3.8181e-013


table =

  Columns 1 through 5

    'Source'     'SS'             'df'     'MS'
'F'
    'Columns'    [   766.0688]    [  4]    [191.5172]
[21.0026]
    'Error'      [1.0942e+003]    [120]    [  9.1187]
       []
    'Total'      [1.8603e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [3.8181e-013]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
```

```
    source: 'anova1'
     means: [46.3860 47.1988 46.5636 45.0080 40.3636]
        df: 120
         s: 3.0197


c =

    1.0000    2.0000   -2.9386   -0.8128    1.3130
    1.0000    3.0000   -2.3034   -0.1776    1.9482
    1.0000    4.0000   -0.7478    1.3780    3.5038
    1.0000    5.0000    3.8966    6.0224    8.1482
    2.0000    3.0000   -1.4906    0.6352    2.7610
    2.0000    4.0000    0.0650    2.1908    4.3166
    2.0000    5.0000    4.7094    6.8352    8.9610
    3.0000    4.0000   -0.5702    1.5556    3.6814
    3.0000    5.0000    4.0742    6.2000    8.3258
    4.0000    5.0000    2.5186    4.6444    6.7702


m =

   46.3860    0.6039
   47.1988    0.6039
   46.5636    0.6039
   45.0080    0.6039
   40.3636    0.6039


h =

     []


nms =

1
2
3
4
5


tintervalname =

13
```

```
p =

  1.9190e-009


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [   577.7894]     [  4]    [144.4474]
[14.0700]
    'Error'      [1.2320e+003]     [120]    [ 10.2663]
       []
    'Total'      [1.8097e+003]     [124]            []
       []

  Column 6

    'Prob>F'
    [1.9190e-009]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [44.3060 43.4588 42.0788 41.9904 38.0424]
        df: 120
         s: 3.2041


c =

    1.0000    2.0000   -1.4084    0.8472    3.1028
    1.0000    3.0000   -0.0284    2.2272    4.4828
    1.0000    4.0000    0.0600    2.3156    4.5712
    1.0000    5.0000    4.0080    6.2636    8.5192
    2.0000    3.0000   -0.8756    1.3800    3.6356
    2.0000    4.0000   -0.7872    1.4684    3.7240
    2.0000    5.0000    3.1608    5.4164    7.6720
    3.0000    4.0000   -2.1672    0.0884    2.3440
    3.0000    5.0000    1.7808    4.0364    6.2920
```

```
    4.0000    5.0000    1.6924    3.9480    6.2036


m =

   44.3060    0.6408
   43.4588    0.6408
   42.0788    0.6408
   41.9904    0.6408
   38.0424    0.6408


h =

     []


nms =

1
2
3
4
5


tintervalname =

14


p =

     0


table =

    'Source'     'SS'            'df'     'MS'
'F'          'Prob>F'
    'Columns'    [1.7785e+003]    [  4]    [444.6235]
[31.2063]    [      0]
    'Error'      [1.7097e+003]    [120]    [ 14.2479]
       []           []
    'Total'      [3.4882e+003]    [124]           []
       []           []
```

```
stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [52.2200 54.0364 52.6208 51.2816 43.3740]
        df: 120
         s: 3.7746


c =

    1.0000    2.0000   -4.4736   -1.8164    0.8408
    1.0000    3.0000   -3.0580   -0.4008    2.2564
    1.0000    4.0000   -1.7188    0.9384    3.5956
    1.0000    5.0000    6.1888    8.8460   11.5032
    2.0000    3.0000   -1.2416    1.4156    4.0728
    2.0000    4.0000    0.0976    2.7548    5.4120
    2.0000    5.0000    8.0052   10.6624   13.3196
    3.0000    4.0000   -1.3180    1.3392    3.9964
    3.0000    5.0000    6.5896    9.2468   11.9040
    4.0000    5.0000    5.2504    7.9076   10.5648


m =

   52.2200    0.7549
   54.0364    0.7549
   52.6208    0.7549
   51.2816    0.7549
   43.3740    0.7549


h =

     []


nms =

1
2
3
4
5
```

```
tintervalname =

15


p =

     0


table =

    'Source'      'SS'               'df'      'MS'
'F'           'Prob>F'
    'Columns'     [   914.2412]     [  4]     [228.5603]
[42.0526]    [      0]
    'Error'       [   652.2128]     [120]     [  5.4351]
        []           []
    'Total'       [1.5665e+003]     [124]             []
        []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [40.2696 39.7888 40.8032 38.1096 33.3692]
        df: 120
         s: 2.3313


c =

    1.0000    2.0000   -1.1604    0.4808    2.1220
    1.0000    3.0000   -2.1748   -0.5336    1.1076
    1.0000    4.0000    0.5188    2.1600    3.8012
    1.0000    5.0000    5.2592    6.9004    8.5416
    2.0000    3.0000   -2.6556   -1.0144    0.6268
    2.0000    4.0000    0.0380    1.6792    3.3204
    2.0000    5.0000    4.7784    6.4196    8.0608
    3.0000    4.0000    1.0524    2.6936    4.3348
    3.0000    5.0000    5.7928    7.4340    9.0752
    4.0000    5.0000    3.0992    4.7404    6.3816


m =
```

```
   40.2696    0.4663
   39.7888    0.4663
   40.8032    0.4663
   38.1096    0.4663
   33.3692    0.4663


h =

     []


nms =

1
2
3
4
5


tintervalname =

16


p =

     0


table =

    'Source'     'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'    [3.4126e+003]    [  4]    [853.1410]
[31.0270]    [     0]
    'Error'      [3.2996e+003]    [120]    [ 27.4968]
       []           []
    'Total'      [6.7122e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
```

```
    source: 'anova1'
     means: [58.9172 60.4580 60.8524 56.1560 46.7020]
        df: 120
         s: 5.2437


c =

    1.0000    2.0000   -5.2322   -1.5408    2.1506
    1.0000    3.0000   -5.6266   -1.9352    1.7562
    1.0000    4.0000   -0.9302    2.7612    6.4526
    1.0000    5.0000    8.5238   12.2152   15.9066
    2.0000    3.0000   -4.0858   -0.3944    3.2970
    2.0000    4.0000    0.6106    4.3020    7.9934
    2.0000    5.0000   10.0646   13.7560   17.4474
    3.0000    4.0000    1.0050    4.6964    8.3878
    3.0000    5.0000   10.4590   14.1504   17.8418
    4.0000    5.0000    5.7626    9.4540   13.1454


m =

   58.9172    1.0487
   60.4580    1.0487
   60.8524    1.0487
   56.1560    1.0487
   46.7020    1.0487


h =

     []


nms =

1
2
3
4
5


tintervalname =

17
```

```
p =

     0


table =

    'Source'      'SS'               'df'      'MS'
'F'           'Prob>F'
    'Columns'     [3.3408e+003]    [  4]    [835.1909]
[30.2938]    [      0]
    'Error'       [3.3084e+003]    [120]    [ 27.5697]
       []            []
    'Total'       [6.6491e+003]    [124]              []
       []            []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [61.2644 62.1728 61.4540 56.9524 48.3800]
        df: 120
         s: 5.2507


c =

    1.0000    2.0000   -4.6047   -0.9084    2.7879
    1.0000    3.0000   -3.8859   -0.1896    3.5067
    1.0000    4.0000    0.6157    4.3120    8.0083
    1.0000    5.0000    9.1881   12.8844   16.5807
    2.0000    3.0000   -2.9775    0.7188    4.4151
    2.0000    4.0000    1.5241    5.2204    8.9167
    2.0000    5.0000   10.0965   13.7928   17.4891
    3.0000    4.0000    0.8053    4.5016    8.1979
    3.0000    5.0000    9.3777   13.0740   16.7703
    4.0000    5.0000    4.8761    8.5724   12.2687


m =

   61.2644    1.0501
   62.1728    1.0501
   61.4540    1.0501
   56.9524    1.0501
```

```
   48.3800    1.0501


h =

     []


nms =

1
2
3
4
5


tintervalname =

18


p =

  1.5488e-013


table =

  Columns 1 through 5

    'Source'     'SS'              'df'      'MS'
'F'
    'Columns'    [3.3105e+003]     [  4]     [827.6141]
[21.7932]
    'Error'      [4.5571e+003]     [120]     [ 37.9758]
       []
    'Total'      [7.8676e+003]     [124]             []
       []

  Column 6

    'Prob>F'
    [1.5488e-013]
               []
               []
```

```
stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [60.9416 62.1348 63.8632 57.8460 49.3044]
        df: 120
         s: 6.1625


c =

    1.0000    2.0000   -5.5313   -1.1932    3.1449
    1.0000    3.0000   -7.2597   -2.9216    1.4165
    1.0000    4.0000   -1.2425    3.0956    7.4337
    1.0000    5.0000    7.2991   11.6372   15.9753
    2.0000    3.0000   -6.0665   -1.7284    2.6097
    2.0000    4.0000   -0.0493    4.2888    8.6269
    2.0000    5.0000    8.4923   12.8304   17.1685
    3.0000    4.0000    1.6791    6.0172   10.3553
    3.0000    5.0000   10.2207   14.5588   18.8969
    4.0000    5.0000    4.2035    8.5416   12.8797


m =

   60.9416    1.2325
   62.1348    1.2325
   63.8632    1.2325
   57.8460    1.2325
   49.3044    1.2325


h =

     []


nms =

1
2
3
4
5
```

```
tintervalname =

19


p =

     0


table =

    'Source'      'SS'                'df'      'MS'
'F'           'Prob>F'
    'Columns'     [3.6124e+003]     [  4]     [903.0878]
[33.1669]     [      0]
    'Error'       [3.2674e+003]     [120]     [ 27.2286]
       []            []
    'Total'       [6.8798e+003]     [124]             []
       []            []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [58.3112 59.0804 58.8244 54.3808 44.9044]
        df: 120
         s: 5.2181


c =

    1.0000    2.0000   -4.4425   -0.7692    2.9041
    1.0000    3.0000   -4.1865   -0.5132    3.1601
    1.0000    4.0000    0.2571    3.9304    7.6037
    1.0000    5.0000    9.7335   13.4068   17.0801
    2.0000    3.0000   -3.4173    0.2560    3.9293
    2.0000    4.0000    1.0263    4.6996    8.3729
    2.0000    5.0000   10.5027   14.1760   17.8493
    3.0000    4.0000    0.7703    4.4436    8.1169
    3.0000    5.0000   10.2467   13.9200   17.5933
    4.0000    5.0000    5.8031    9.4764   13.1497


m =
```

```
   58.3112    1.0436
   59.0804    1.0436
   58.8244    1.0436
   54.3808    1.0436
   44.9044    1.0436


h =

     []


nms =

1
2
3
4
5


tintervalname =

20


p =

  1.8874e-014


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [2.8273e+003]    [  4]    [706.8185]
[23.6856]
    'Error'      [3.5810e+003]    [120]    [ 29.8418]
        []
    'Total'      [6.4083e+003]    [124]            []
        []

  Column 6

    'Prob>F'
```

```
    [1.8874e-014]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [55.4604 53.6876 55.5228 51.5628 42.7328]
        df: 120
         s: 5.4628


c =

    1.0000    2.0000   -2.0728    1.7728    5.6184
    1.0000    3.0000   -3.9080   -0.0624    3.7832
    1.0000    4.0000    0.0520    3.8976    7.7432
    1.0000    5.0000    8.8820   12.7276   16.5732
    2.0000    3.0000   -5.6808   -1.8352    2.0104
    2.0000    4.0000   -1.7208    2.1248    5.9704
    2.0000    5.0000    7.1092   10.9548   14.8004
    3.0000    4.0000    0.1144    3.9600    7.8056
    3.0000    5.0000    8.9444   12.7900   16.6356
    4.0000    5.0000    4.9844    8.8300   12.6756


m =

   55.4604    1.0926
   53.6876    1.0926
   55.5228    1.0926
   51.5628    1.0926
   42.7328    1.0926


h =

     []


nms =

1
2
```

```
3
4
5


tintervalname =

21


p =

  1.1122e-011


table =

  Columns 1 through 5

    'Source'     'SS'               'df'      'MS'
'F'
    'Columns'    [3.0631e+003]    [  4]    [765.7852]
[18.1472]
    'Error'      [5.0638e+003]    [120]    [ 42.1985]
       []
    'Total'      [8.1270e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [1.1122e-011]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [60.0788 59.1636 58.0860 55.2936 46.4500]
        df: 120
         s: 6.4960


c =
```

```
    1.0000    2.0000   -3.6577    0.9152    5.4881
    1.0000    3.0000   -2.5801    1.9928    6.5657
    1.0000    4.0000    0.2123    4.7852    9.3581
    1.0000    5.0000    9.0559   13.6288   18.2017
    2.0000    3.0000   -3.4953    1.0776    5.6505
    2.0000    4.0000   -0.7029    3.8700    8.4429
    2.0000    5.0000    8.1407   12.7136   17.2865
    3.0000    4.0000   -1.7805    2.7924    7.3653
    3.0000    5.0000    7.0631   11.6360   16.2089
    4.0000    5.0000    4.2707    8.8436   13.4165


m =

   60.0788    1.2992
   59.1636    1.2992
   58.0860    1.2992
   55.2936    1.2992
   46.4500    1.2992


h =

     []


nms =

1
2
3
4
5


tintervalname =

22


p =

     0


table =
```

```
    'Source'     'SS'              'df'      'MS'
'F'           'Prob>F'
    'Columns'    [2.7539e+003]    [  4]    [688.4828]
[30.7531]    [      0]
    'Error'      [2.6865e+003]    [120]    [ 22.3874]
       []           []
    'Total'      [5.4404e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [56.9840 58.2200 56.9800 53.4128 45.3740]
        df: 120
         s: 4.7315


c =

    1.0000    2.0000   -4.5668   -1.2360    2.0948
    1.0000    3.0000   -3.3268    0.0040    3.3348
    1.0000    4.0000    0.2404    3.5712    6.9020
    1.0000    5.0000    8.2792   11.6100   14.9408
    2.0000    3.0000   -2.0908    1.2400    4.5708
    2.0000    4.0000    1.4764    4.8072    8.1380
    2.0000    5.0000    9.5152   12.8460   16.1768
    3.0000    4.0000    0.2364    3.5672    6.8980
    3.0000    5.0000    8.2752   11.6060   14.9368
    4.0000    5.0000    4.7080    8.0388   11.3696


m =

   56.9840    0.9463
   58.2200    0.9463
   56.9800    0.9463
   53.4128    0.9463
   45.3740    0.9463


h =

     []
```

```
nms =

1
2
3
4
5


tintervalname =

23


p =

  3.0734e-012


table =

  Columns 1 through 5

    'Source'     'SS'            'df'     'MS'
'F'
    'Columns'    [   563.1228]    [  4]    [140.7807]
[19.2182]
    'Error'      [   879.0464]    [120]    [  7.3254]
       []
    'Total'      [1.4422e+003]    [124]           []
       []

  Column 6

    'Prob>F'
    [3.0734e-012]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [44.5880 45.7760 44.4720 43.8520 39.5992]
        df: 120
```

```
        s: 2.7065


c =

    1.0000    2.0000   -3.0933   -1.1880    0.7173
    1.0000    3.0000   -1.7893    0.1160    2.0213
    1.0000    4.0000   -1.1693    0.7360    2.6413
    1.0000    5.0000    3.0835    4.9888    6.8941
    2.0000    3.0000   -0.6013    1.3040    3.2093
    2.0000    4.0000    0.0187    1.9240    3.8293
    2.0000    5.0000    4.2715    6.1768    8.0821
    3.0000    4.0000   -1.2853    0.6200    2.5253
    3.0000    5.0000    2.9675    4.8728    6.7781
    4.0000    5.0000    2.3475    4.2528    6.1581


m =

   44.5880    0.5413
   45.7760    0.5413
   44.4720    0.5413
   43.8520    0.5413
   39.5992    0.5413


h =

     []


nms =

1
2
3
4
5


tintervalname =

24


p =
```

```
  4.3632e-014


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [2.0422e+003]    [  4]    [510.5546]
[22.9223]
    'Error'      [2.6728e+003]    [120]    [ 22.2733]
       []
    'Total'      [4.7150e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [4.3632e-014]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [59.3400 57.6120 58.1200 55.0680 48.0484]
        df: 120
         s: 4.7195


c =

    1.0000    2.0000   -1.5943    1.7280    5.0503
    1.0000    3.0000   -2.1023    1.2200    4.5423
    1.0000    4.0000    0.9497    4.2720    7.5943
    1.0000    5.0000    7.9693   11.2916   14.6139
    2.0000    3.0000   -3.8303   -0.5080    2.8143
    2.0000    4.0000   -0.7783    2.5440    5.8663
    2.0000    5.0000    6.2413    9.5636   12.8859
    3.0000    4.0000   -0.2703    3.0520    6.3743
    3.0000    5.0000    6.7493   10.0716   13.3939
    4.0000    5.0000    3.6973    7.0196   10.3419
```

```
m =

   59.3400    0.9439
   57.6120    0.9439
   58.1200    0.9439
   55.0680    0.9439
   48.0484    0.9439


h =

     []


nms =

1
2
3
4
5
```

APPENDIX C2

ANOVA_and_HSD_results_for_alpha_0.05.txt output:

```
tintervalname =

1


p =

  2.9976e-015


table =

  Columns 1 through 5

    'Source'     'SS'             'df'     'MS'
'F'
    'Columns'    [1.8098e+003]    [  4]    [452.4529]
[25.3770]
    'Error'      [2.1395e+003]    [120]    [ 17.8293]
       []
    'Total'      [3.9493e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [2.9976e-015]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [57.6089 56.4346 56.4724 54.6142 47.0772]
```

```
        df: 120
         s: 4.2225


c =

    1.0000    2.0000   -2.1336    1.1743    4.4821
    1.0000    3.0000   -2.1713    1.1365    4.4444
    1.0000    4.0000   -0.3132    2.9947    6.3025
    1.0000    5.0000    7.2239   10.5317   13.8396
    2.0000    3.0000   -3.3456   -0.0378    3.2701
    2.0000    4.0000   -1.4874    1.8204    5.1282
    2.0000    5.0000    6.0496    9.3574   12.6653
    3.0000    4.0000   -1.4497    1.8582    5.1660
    3.0000    5.0000    6.0874    9.3952   12.7030
    4.0000    5.0000    4.2292    7.5370   10.8449


m =

   57.6089    0.8445
   56.4346    0.8445
   56.4724    0.8445
   54.6142    0.8445
   47.0772    0.8445


h =

     []


nms =

1
2
3
4
5


tintervalname =

2
```

```
p =

     0


table =

    'Source'     'SS'            'df'     'MS'
'F'          'Prob>F'
    'Columns'    [3.2805e+003]    [  4]    [820.1357]
[36.1828]    [      0]
    'Error'      [2.7200e+003]    [120]    [ 22.6665]
       []           []
    'Total'      [6.0005e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [59.2053 59.2742 56.5100 55.7137 45.3714]
        df: 120
         s: 4.7609


c =

    1.0000    2.0000   -3.7986   -0.0689    3.6607
    1.0000    3.0000   -1.0344    2.6952    6.4249
    1.0000    4.0000   -0.2381    3.4916    7.2213
    1.0000    5.0000   10.1042   13.8339   17.5635
    2.0000    3.0000   -0.9655    2.7642    6.4938
    2.0000    4.0000   -0.1691    3.5605    7.2902
    2.0000    5.0000   10.1731   13.9028   17.6325
    3.0000    4.0000   -2.9333    0.7964    4.5260
    3.0000    5.0000    7.4090   11.1386   14.8683
    4.0000    5.0000    6.6126   10.3423   14.0719


m =

   59.2053    0.9522
   59.2742    0.9522
   56.5100    0.9522
```

```
   55.7137    0.9522
   45.3714    0.9522


h =

     []


nms =

1
2
3
4
5


tintervalname =

3


p =

     0


table =

    'Source'     'SS'              'df'     'MS'
'F'          'Prob>F'
    'Columns'    [2.5506e+003]    [  4]    [637.6470]
[37.3296]    [      0]
    'Error'      [2.0498e+003]    [120]    [ 17.0815]
       []           []
    'Total'      [4.6004e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [52.5680 52.1016 53.3264 50.2293 41.0558]
```

```
        df: 120
         s: 4.1330


c =

    1.0000    2.0000   -2.7713    0.4664    3.7041
    1.0000    3.0000   -3.9961   -0.7584    2.4793
    1.0000    4.0000   -0.8990    2.3387    5.5764
    1.0000    5.0000    8.2744   11.5122   14.7499
    2.0000    3.0000   -4.4625   -1.2248    2.0129
    2.0000    4.0000   -1.3654    1.8723    5.1100
    2.0000    5.0000    7.8080   11.0458   14.2835
    3.0000    4.0000   -0.1406    3.0971    6.3348
    3.0000    5.0000    9.0328   12.2706   15.5083
    4.0000    5.0000    5.9358    9.1735   12.4112


m =

   52.5680    0.8266
   52.1016    0.8266
   53.3264    0.8266
   50.2293    0.8266
   41.0558    0.8266


h =

     []


nms =

1
2
3
4
5


tintervalname =

4


p =
```

```
  1.1102e-016


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [   858.7402]     [  4]    [214.6851]
[28.8197]
    'Error'      [   893.9094]     [120]    [  7.4492]
       []
    'Total'      [1.7526e+003]     [124]            []
       []

  Column 6

    'Prob>F'
    [1.1102e-016]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [42.5132 42.0620 43.6268 40.7296 36.1052]
        df: 120
         s: 2.7293


c =

    1.0000    2.0000   -1.6869    0.4512    2.5893
    1.0000    3.0000   -3.2517   -1.1136    1.0245
    1.0000    4.0000   -0.3545    1.7836    3.9217
    1.0000    5.0000    4.2699    6.4080    8.5461
    2.0000    3.0000   -3.7029   -1.5648    0.5733
    2.0000    4.0000   -0.8057    1.3324    3.4705
    2.0000    5.0000    3.8187    5.9568    8.0949
    3.0000    4.0000    0.7591    2.8972    5.0353
    3.0000    5.0000    5.3835    7.5216    9.6597
    4.0000    5.0000    2.4863    4.6244    6.7625
```

```
m =

   42.5132    0.5459
   42.0620    0.5459
   43.6268    0.5459
   40.7296    0.5459
   36.1052    0.5459


h =

     []


nms =

1
2
3
4
5


tintervalname =

5


p =

  4.4409e-016


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [1.2324e+003]    [  4]    [308.0955]
[27.2007]
    'Error'      [1.3592e+003]    [120]    [ 11.3268]
       []
    'Total'      [2.5916e+003]    [124]            []
       []

  Column 6
```

```
    'Prob>F'
    [4.4409e-016]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [42.4612 42.7348 43.0524 40.5464 34.6588]
        df: 120
         s: 3.3655


c =

    1.0000    2.0000   -2.9101   -0.2736    2.3629
    1.0000    3.0000   -3.2277   -0.5912    2.0453
    1.0000    4.0000   -0.7217    1.9148    4.5513
    1.0000    5.0000    5.1659    7.8024   10.4389
    2.0000    3.0000   -2.9541   -0.3176    2.3189
    2.0000    4.0000   -0.4481    2.1884    4.8249
    2.0000    5.0000    5.4395    8.0760   10.7125
    3.0000    4.0000   -0.1305    2.5060    5.1425
    3.0000    5.0000    5.7571    8.3936   11.0301
    4.0000    5.0000    3.2511    5.8876    8.5241


m =

   42.4612    0.6731
   42.7348    0.6731
   43.0524    0.6731
   40.5464    0.6731
   34.6588    0.6731


h =

     []


nms =
```

```
1
2
3
4
5


tintervalname =

6


p =

  1.1025e-013


table =

    'Source'     'SS'           'df'      'MS'           'F'
         'Prob>F'
    'Columns'    [130.2079]    [  4]    [32.5520]    [22.
0937]    [1.1025e-013]
    'Error'      [176.8030]    [120]    [ 1.4734]
   []                []
    'Total'      [307.0110]    [124]            []
   []                []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [31.5508 31.4052 31.9604 31.0184 29.0460]
        df: 120
         s: 1.2138


c =

    1.0000    2.0000   -0.8053    0.1456    1.0965
    1.0000    3.0000   -1.3605   -0.4096    0.5413
    1.0000    4.0000   -0.4185    0.5324    1.4833
    1.0000    5.0000    1.5539    2.5048    3.4557
    2.0000    3.0000   -1.5061   -0.5552    0.3957
    2.0000    4.0000   -0.5641    0.3868    1.3377
```

```
    2.0000    5.0000    1.4083    2.3592    3.3101
    3.0000    4.0000   -0.0089    0.9420    1.8929
    3.0000    5.0000    1.9635    2.9144    3.8653
    4.0000    5.0000    1.0215    1.9724    2.9233


m =

   31.5508    0.2428
   31.4052    0.2428
   31.9604    0.2428
   31.0184    0.2428
   29.0460    0.2428


h =

     []


nms =

1
2
3
4
5


tintervalname =

7


p =

     0


table =

    'Source'     'SS'             'df'     'MS'
'F'          'Prob>F'
    'Columns'    [3.5198e+003]    [  4]    [879.9399]
[29.4504]    [      0]
    'Error'      [3.5854e+003]    [120]    [ 29.8787]
       []           []
```

```
    'Total'       [7.1052e+003]    [124]             []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [59.3500 59.9948 60.6268 56.6144 46.3292]
        df: 120
         s: 5.4661


c =

    1.0000    2.0000   -4.9269   -0.6448    3.6373
    1.0000    3.0000   -5.5589   -1.2768    3.0053
    1.0000    4.0000   -1.5465    2.7356    7.0177
    1.0000    5.0000    8.7387   13.0208   17.3029
    2.0000    3.0000   -4.9141   -0.6320    3.6501
    2.0000    4.0000   -0.9017    3.3804    7.6625
    2.0000    5.0000    9.3835   13.6656   17.9477
    3.0000    4.0000   -0.2697    4.0124    8.2945
    3.0000    5.0000   10.0155   14.2976   18.5797
    4.0000    5.0000    6.0031   10.2852   14.5673


m =

   59.3500    1.0932
   59.9948    1.0932
   60.6268    1.0932
   56.6144    1.0932
   46.3292    1.0932


h =

     []


nms =

1
2
3
```

```
4
5


tintervalname =

8


p =

  5.8842e-015


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [1.6952e+003]    [  4]    [423.7918]
[24.7656]
    'Error'      [2.0535e+003]    [120]    [ 17.1121]
       []
    'Total'      [3.7486e+003]    [124]           []
       []

  Column 6

    'Prob>F'
    [5.8842e-015]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [55.0876 54.1780 55.4232 52.4072 45.4472]
        df: 120
         s: 4.1367


c =
```

```
    1.0000    2.0000   -2.3310    0.9096    4.1502
    1.0000    3.0000   -3.5762   -0.3356    2.9050
    1.0000    4.0000   -0.5602    2.6804    5.9210
    1.0000    5.0000    6.3998    9.6404   12.8810
    2.0000    3.0000   -4.4858   -1.2452    1.9954
    2.0000    4.0000   -1.4698    1.7708    5.0114
    2.0000    5.0000    5.4902    8.7308   11.9714
    3.0000    4.0000   -0.2246    3.0160    6.2566
    3.0000    5.0000    6.7354    9.9760   13.2166
    4.0000    5.0000    3.7194    6.9600   10.2006


m =

   55.0876    0.8273
   54.1780    0.8273
   55.4232    0.8273
   52.4072    0.8273
   45.4472    0.8273


h =

     []


nms =

1
2
3
4
5


tintervalname =

9


p =

  1.2656e-011


table =
```

```
  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [   407.4122]     [  4]    [101.8531]
[18.0408]
    'Error'      [   677.4835]     [120]    [  5.6457]
       []
    'Total'      [1.0849e+003]     [124]            []
       []

  Column 6

    'Prob>F'
    [1.2656e-011]
              []
              []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [40.0964 40.4460 40.4548 38.7476 35.7040]
        df: 120
         s: 2.3761


c =

    1.0000    2.0000   -2.2110   -0.3496    1.5118
    1.0000    3.0000   -2.2198   -0.3584    1.5030
    1.0000    4.0000   -0.5126    1.3488    3.2102
    1.0000    5.0000    2.5310    4.3924    6.2538
    2.0000    3.0000   -1.8702   -0.0088    1.8526
    2.0000    4.0000   -0.1630    1.6984    3.5598
    2.0000    5.0000    2.8806    4.7420    6.6034
    3.0000    4.0000   -0.1542    1.7072    3.5686
    3.0000    5.0000    2.8894    4.7508    6.6122
    4.0000    5.0000    1.1822    3.0436    4.9050


m =

   40.0964    0.4752
```

```
   40.4460    0.4752
   40.4548    0.4752
   38.7476    0.4752
   35.7040    0.4752


h =

     []


nms =

1
2
3
4
5


tintervalname =

10


p =

     0


table =

    'Source'     'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'    [2.7058e+003]     [  4]     [676.4622]
[36.3010]    [      0]
    'Error'      [2.2362e+003]     [120]     [ 18.6348]
       []           []
    'Total'      [4.9420e+003]     [124]             []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
```

```
     means: [55.3664 55.3604 55.7888 51.1496 43.5832]
        df: 120
         s: 4.3168


c =

    1.0000    2.0000   -3.3757    0.0060    3.3877
    1.0000    3.0000   -3.8041   -0.4224    2.9593
    1.0000    4.0000    0.8351    4.2168    7.5985
    1.0000    5.0000    8.4015   11.7832   15.1649
    2.0000    3.0000   -3.8101   -0.4284    2.9533
    2.0000    4.0000    0.8291    4.2108    7.5925
    2.0000    5.0000    8.3955   11.7772   15.1589
    3.0000    4.0000    1.2575    4.6392    8.0209
    3.0000    5.0000    8.8239   12.2056   15.5873
    4.0000    5.0000    4.1847    7.5664   10.9481


m =

   55.3664    0.8634
   55.3604    0.8634
   55.7888    0.8634
   51.1496    0.8634
   43.5832    0.8634


h =

     []


nms =

1
2
3
4
5


tintervalname =

11
```

```
p =

     0


table =

    'Source'      'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'     [   900.1484]     [  4]     [225.0371]
[29.9729]    [      0]
    'Error'       [   900.9613]     [120]     [  7.5080]
       []            []
    'Total'       [1.8011e+003]     [124]            []
       []            []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [43.5812 43.8964 45.3328 43.3676 37.5588]
        df: 120
         s: 2.7401


c =

    1.0000    2.0000   -2.4617   -0.3152    1.8313
    1.0000    3.0000   -3.8981   -1.7516    0.3949
    1.0000    4.0000   -1.9329    0.2136    2.3601
    1.0000    5.0000    3.8759    6.0224    8.1689
    2.0000    3.0000   -3.5829   -1.4364    0.7101
    2.0000    4.0000   -1.6177    0.5288    2.6753
    2.0000    5.0000    4.1911    6.3376    8.4841
    3.0000    4.0000   -0.1813    1.9652    4.1117
    3.0000    5.0000    5.6275    7.7740    9.9205
    4.0000    5.0000    3.6623    5.8088    7.9553


m =

   43.5812    0.5480
   43.8964    0.5480
   45.3328    0.5480
```

```
   43.3676    0.5480
   37.5588    0.5480


h =

     []


nms =

1
2
3
4
5


tintervalname =

12


p =

  3.8181e-013


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [   766.0688]     [  4]    [191.5172]
[21.0026]
    'Error'      [1.0942e+003]     [120]    [  9.1187]
       []
    'Total'      [1.8603e+003]     [124]            []
       []

  Column 6

    'Prob>F'
    [3.8181e-013]
```

```
                []
                []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [46.3860 47.1988 46.5636 45.0080 40.3636]
        df: 120
         s: 3.0197


c =

    1.0000    2.0000   -3.1784   -0.8128    1.5528
    1.0000    3.0000   -2.5432   -0.1776    2.1880
    1.0000    4.0000   -0.9876    1.3780    3.7436
    1.0000    5.0000    3.6568    6.0224    8.3880
    2.0000    3.0000   -1.7304    0.6352    3.0008
    2.0000    4.0000   -0.1748    2.1908    4.5564
    2.0000    5.0000    4.4696    6.8352    9.2008
    3.0000    4.0000   -0.8100    1.5556    3.9212
    3.0000    5.0000    3.8344    6.2000    8.5656
    4.0000    5.0000    2.2788    4.6444    7.0100


m =

   46.3860    0.6039
   47.1988    0.6039
   46.5636    0.6039
   45.0080    0.6039
   40.3636    0.6039


h =

     []


nms =

1
2
```

```
3
4
5


tintervalname =

13


p =

  1.9190e-009


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [    577.7894]    [  4]    [144.4474]
[14.0700]
    'Error'      [1.2320e+003]    [120]    [ 10.2663]
       []
    'Total'      [1.8097e+003]    [124]           []
       []

  Column 6

    'Prob>F'
    [1.9190e-009]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [44.3060 43.4588 42.0788 41.9904 38.0424]
        df: 120
         s: 3.2041


c =
```

```
    1.0000    2.0000   -1.6629    0.8472    3.3573
    1.0000    3.0000   -0.2829    2.2272    4.7373
    1.0000    4.0000   -0.1945    2.3156    4.8257
    1.0000    5.0000    3.7535    6.2636    8.7737
    2.0000    3.0000   -1.1301    1.3800    3.8901
    2.0000    4.0000   -1.0417    1.4684    3.9785
    2.0000    5.0000    2.9063    5.4164    7.9265
    3.0000    4.0000   -2.4217    0.0884    2.5985
    3.0000    5.0000    1.5263    4.0364    6.5465
    4.0000    5.0000    1.4379    3.9480    6.4581


m =

   44.3060    0.6408
   43.4588    0.6408
   42.0788    0.6408
   41.9904    0.6408
   38.0424    0.6408


h =

     []


nms =

1
2
3
4
5


tintervalname =

14


p =

     0


table =
```

```
    'Source'     'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'    [1.7785e+003]    [  4]    [444.6235]
[31.2063]    [      0]
    'Error'      [1.7097e+003]    [120]    [ 14.2479]
       []           []
    'Total'      [3.4882e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [52.2200 54.0364 52.6208 51.2816 43.3740]
        df: 120
         s: 3.7746


c =

    1.0000    2.0000   -4.7734   -1.8164    1.1406
    1.0000    3.0000   -3.3578   -0.4008    2.5562
    1.0000    4.0000   -2.0186    0.9384    3.8954
    1.0000    5.0000    5.8890    8.8460   11.8030
    2.0000    3.0000   -1.5414    1.4156    4.3726
    2.0000    4.0000   -0.2022    2.7548    5.7118
    2.0000    5.0000    7.7054   10.6624   13.6194
    3.0000    4.0000   -1.6178    1.3392    4.2962
    3.0000    5.0000    6.2898    9.2468   12.2038
    4.0000    5.0000    4.9506    7.9076   10.8646


m =

   52.2200    0.7549
   54.0364    0.7549
   52.6208    0.7549
   51.2816    0.7549
   43.3740    0.7549


h =

     []
```

```
nms =

1
2
3
4
5


tintervalname =

15


p =

     0


table =

    'Source'     'SS'            'df'     'MS'
'F'          'Prob>F'
    'Columns'    [   914.2412]    [  4]    [228.5603]
[42.0526]    [      0]
    'Error'      [   652.2128]    [120]    [  5.4351]
       []           []
    'Total'      [1.5665e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [40.2696 39.7888 40.8032 38.1096 33.3692]
        df: 120
         s: 2.3313


c =

    1.0000    2.0000   -1.3455    0.4808    2.3071
    1.0000    3.0000   -2.3599   -0.5336    1.2927
```

```
    1.0000    4.0000    0.3337    2.1600    3.9863
    1.0000    5.0000    5.0741    6.9004    8.7267
    2.0000    3.0000   -2.8407   -1.0144    0.8119
    2.0000    4.0000   -0.1471    1.6792    3.5055
    2.0000    5.0000    4.5933    6.4196    8.2459
    3.0000    4.0000    0.8673    2.6936    4.5199
    3.0000    5.0000    5.6077    7.4340    9.2603
    4.0000    5.0000    2.9141    4.7404    6.5667


m =

   40.2696    0.4663
   39.7888    0.4663
   40.8032    0.4663
   38.1096    0.4663
   33.3692    0.4663


h =

     []


nms =

1
2
3
4
5


tintervalname =

16


p =

     0


table =

    'Source'      'SS'                'df'      'MS'
'F'             'Prob>F'
```

```
    'Columns'    [3.4126e+003]    [  4]    [853.1410]
[31.0270]    [      0]
    'Error'      [3.2996e+003]    [120]    [ 27.4968]
       []           []
    'Total'      [6.7122e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [58.9172 60.4580 60.8524 56.1560 46.7020]
        df: 120
         s: 5.2437


c =

    1.0000    2.0000   -5.6487   -1.5408    2.5671
    1.0000    3.0000   -6.0431   -1.9352    2.1727
    1.0000    4.0000   -1.3467    2.7612    6.8691
    1.0000    5.0000    8.1073   12.2152   16.3231
    2.0000    3.0000   -4.5023   -0.3944    3.7135
    2.0000    4.0000    0.1941    4.3020    8.4099
    2.0000    5.0000    9.6481   13.7560   17.8639
    3.0000    4.0000    0.5885    4.6964    8.8043
    3.0000    5.0000   10.0425   14.1504   18.2583
    4.0000    5.0000    5.3461    9.4540   13.5619


m =

   58.9172    1.0487
   60.4580    1.0487
   60.8524    1.0487
   56.1560    1.0487
   46.7020    1.0487


h =

     []
```

```
nms =

1
2
3
4
5


tintervalname =

17


p =

     0


table =

    'Source'     'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'    [3.3408e+003]    [  4]    [835.1909]
[30.2938]    [      0]
    'Error'      [3.3084e+003]    [120]    [ 27.5697]
       []            []
    'Total'      [6.6491e+003]    [124]            []
       []            []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [61.2644 62.1728 61.4540 56.9524 48.3800]
        df: 120
         s: 5.2507


c =

    1.0000    2.0000   -5.0217   -0.9084    3.2049
    1.0000    3.0000   -4.3029   -0.1896    3.9237
    1.0000    4.0000    0.1987    4.3120    8.4253
    1.0000    5.0000    8.7711   12.8844   16.9977
```

```
    2.0000    3.0000   -3.3945    0.7188    4.8321
    2.0000    4.0000    1.1071    5.2204    9.3337
    2.0000    5.0000    9.6795   13.7928   17.9061
    3.0000    4.0000    0.3883    4.5016    8.6149
    3.0000    5.0000    8.9607   13.0740   17.1873
    4.0000    5.0000    4.4591    8.5724   12.6857


m =

   61.2644    1.0501
   62.1728    1.0501
   61.4540    1.0501
   56.9524    1.0501
   48.3800    1.0501


h =

     []


nms =

1
2
3
4
5


tintervalname =

18


p =

  1.5488e-013


table =

  Columns 1 through 5

    'Source'     'SS'            'df'     'MS'
'F'
```

```
    'Columns'    [3.3105e+003]    [  4]    [827.6141]
[21.7932]
    'Error'      [4.5571e+003]    [120]    [ 37.9758]
       []
    'Total'      [7.8676e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [1.5488e-013]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [60.9416 62.1348 63.8632 57.8460 49.3044]
        df: 120
         s: 6.1625


c =

    1.0000    2.0000   -6.0208   -1.1932    3.6344
    1.0000    3.0000   -7.7492   -2.9216    1.9060
    1.0000    4.0000   -1.7320    3.0956    7.9232
    1.0000    5.0000    6.8096   11.6372   16.4648
    2.0000    3.0000   -6.5560   -1.7284    3.0992
    2.0000    4.0000   -0.5388    4.2888    9.1164
    2.0000    5.0000    8.0028   12.8304   17.6580
    3.0000    4.0000    1.1896    6.0172   10.8448
    3.0000    5.0000    9.7312   14.5588   19.3864
    4.0000    5.0000    3.7140    8.5416   13.3692


m =

   60.9416    1.2325
   62.1348    1.2325
   63.8632    1.2325
   57.8460    1.2325
   49.3044    1.2325
```

```
h =

     []


nms =

1
2
3
4
5


tintervalname =

19


p =

     0


table =

    'Source'     'SS'              'df'     'MS'
'F'          'Prob>F'
    'Columns'    [3.6124e+003]    [  4]    [903.0878]
[33.1669]    [      0]
    'Error'      [3.2674e+003]    [120]    [ 27.2286]
      []           []
    'Total'      [6.8798e+003]    [124]            []
      []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [58.3112 59.0804 58.8244 54.3808 44.9044]
        df: 120
         s: 5.2181


c =
```

```
    1.0000    2.0000   -4.8570   -0.7692    3.3186
    1.0000    3.0000   -4.6010   -0.5132    3.5746
    1.0000    4.0000   -0.1574    3.9304    8.0182
    1.0000    5.0000    9.3190   13.4068   17.4946
    2.0000    3.0000   -3.8318    0.2560    4.3438
    2.0000    4.0000    0.6118    4.6996    8.7874
    2.0000    5.0000   10.0882   14.1760   18.2638
    3.0000    4.0000    0.3558    4.4436    8.5314
    3.0000    5.0000    9.8322   13.9200   18.0078
    4.0000    5.0000    5.3886    9.4764   13.5642


m =

   58.3112    1.0436
   59.0804    1.0436
   58.8244    1.0436
   54.3808    1.0436
   44.9044    1.0436


h =

     []


nms =

1
2
3
4
5


tintervalname =

20


p =

  1.8874e-014


table =
```

```
 Columns 1 through 5

    'Source'     'SS'             'df'     'MS'
'F'
    'Columns'    [2.8273e+003]    [  4]    [706.8185]
[23.6856]
    'Error'      [3.5810e+003]    [120]    [ 29.8418]
       []
    'Total'      [6.4083e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [1.8874e-014]
              []
              []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [55.4604 53.6876 55.5228 51.5628 42.7328]
        df: 120
         s: 5.4628


c =

    1.0000    2.0000   -2.5067    1.7728    6.0523
    1.0000    3.0000   -4.3419   -0.0624    4.2171
    1.0000    4.0000   -0.3819    3.8976    8.1771
    1.0000    5.0000    8.4481   12.7276   17.0071
    2.0000    3.0000   -6.1147   -1.8352    2.4443
    2.0000    4.0000   -2.1547    2.1248    6.4043
    2.0000    5.0000    6.6753   10.9548   15.2343
    3.0000    4.0000   -0.3195    3.9600    8.2395
    3.0000    5.0000    8.5105   12.7900   17.0695
    4.0000    5.0000    4.5505    8.8300   13.1095


m =

   55.4604    1.0926
   53.6876    1.0926
```

```
   55.5228    1.0926
   51.5628    1.0926
   42.7328    1.0926


h =

     []


nms =

1
2
3
4
5


tintervalname =

21


p =

  1.1122e-011


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [3.0631e+003]    [  4]    [765.7852]
[18.1472]
    'Error'      [5.0638e+003]    [120]    [ 42.1985]
       []
    'Total'      [8.1270e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [1.1122e-011]
```

```
                  []
                  []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [60.0788 59.1636 58.0860 55.2936 46.4500]
        df: 120
         s: 6.4960


c =

    1.0000    2.0000   -4.1737    0.9152    6.0041
    1.0000    3.0000   -3.0961    1.9928    7.0817
    1.0000    4.0000   -0.3037    4.7852    9.8741
    1.0000    5.0000    8.5399   13.6288   18.7177
    2.0000    3.0000   -4.0113    1.0776    6.1665
    2.0000    4.0000   -1.2189    3.8700    8.9589
    2.0000    5.0000    7.6247   12.7136   17.8025
    3.0000    4.0000   -2.2965    2.7924    7.8813
    3.0000    5.0000    6.5471   11.6360   16.7249
    4.0000    5.0000    3.7547    8.8436   13.9325


m =

   60.0788    1.2992
   59.1636    1.2992
   58.0860    1.2992
   55.2936    1.2992
   46.4500    1.2992


h =

     []


nms =

1
2
3
```

```
4
5


tintervalname =

22


p =

     0


table =

    'Source'      'SS'              'df'      'MS'
'F'          'Prob>F'
    'Columns'     [2.7539e+003]    [  4]    [688.4828]
[30.7531]    [      0]
    'Error'       [2.6865e+003]    [120]    [ 22.3874]
       []           []
    'Total'       [5.4404e+003]    [124]            []
       []           []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [56.9840 58.2200 56.9800 53.4128 45.3740]
        df: 120
         s: 4.7315


c =

    1.0000    2.0000   -4.9426   -1.2360    2.4706
    1.0000    3.0000   -3.7026    0.0040    3.7106
    1.0000    4.0000   -0.1354    3.5712    7.2778
    1.0000    5.0000    7.9034   11.6100   15.3166
    2.0000    3.0000   -2.4666    1.2400    4.9466
    2.0000    4.0000    1.1006    4.8072    8.5138
    2.0000    5.0000    9.1394   12.8460   16.5526
    3.0000    4.0000   -0.1394    3.5672    7.2738
```

```
    3.0000    5.0000    7.8994   11.6060   15.3126
    4.0000    5.0000    4.3322    8.0388   11.7454


m =

   56.9840    0.9463
   58.2200    0.9463
   56.9800    0.9463
   53.4128    0.9463
   45.3740    0.9463


h =

     []


nms =

1
2
3
4
5


tintervalname =

23


p =

  3.0734e-012


table =

  Columns 1 through 5

    'Source'     'SS'              'df'      'MS'
'F'
    'Columns'    [    563.1228]    [  4]     [140.7807]
[19.2182]
    'Error'      [    879.0464]    [120]     [  7.3254]
       []
```

```
    'Total'       [1.4422e+003]    [124]             []
       []

  Column 6

    'Prob>F'
    [3.0734e-012]
               []
               []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
    source: 'anova1'
     means: [44.5880 45.7760 44.4720 43.8520 39.5992]
        df: 120
         s: 2.7065


c =

    1.0000    2.0000   -3.3083   -1.1880    0.9323
    1.0000    3.0000   -2.0043    0.1160    2.2363
    1.0000    4.0000   -1.3843    0.7360    2.8563
    1.0000    5.0000    2.8685    4.9888    7.1091
    2.0000    3.0000   -0.8163    1.3040    3.4243
    2.0000    4.0000   -0.1963    1.9240    4.0443
    2.0000    5.0000    4.0565    6.1768    8.2971
    3.0000    4.0000   -1.5003    0.6200    2.7403
    3.0000    5.0000    2.7525    4.8728    6.9931
    4.0000    5.0000    2.1325    4.2528    6.3731


m =

   44.5880    0.5413
   45.7760    0.5413
   44.4720    0.5413
   43.8520    0.5413
   39.5992    0.5413


h =
```

```
    []


nms =

1
2
3
4
5


tintervalname =

24


p =

  4.3632e-014


table =

  Columns 1 through 5

    'Source'     'SS'              'df'     'MS'
'F'
    'Columns'    [2.0422e+003]    [  4]    [510.5546]
[22.9223]
    'Error'      [2.6728e+003]    [120]    [ 22.2733]
       []
    'Total'      [4.7150e+003]    [124]            []
       []

  Column 6

    'Prob>F'
    [4.3632e-014]
            []
            []


stats =

    gnames: [5x1 char]
         n: [25 25 25 25 25]
```

```
     source: 'anova1'
      means: [59.3400 57.6120 58.1200 55.0680 48.0484]
         df: 120
          s: 4.7195


c =

    1.0000    2.0000   -1.9692    1.7280    5.4252
    1.0000    3.0000   -2.4772    1.2200    4.9172
    1.0000    4.0000    0.5748    4.2720    7.9692
    1.0000    5.0000    7.5944   11.2916   14.9888
    2.0000    3.0000   -4.2052   -0.5080    3.1892
    2.0000    4.0000   -1.1532    2.5440    6.2412
    2.0000    5.0000    5.8664    9.5636   13.2608
    3.0000    4.0000   -0.6452    3.0520    6.7492
    3.0000    5.0000    6.3744   10.0716   13.7688
    4.0000    5.0000    3.3224    7.0196   10.7168


m =

   59.3400    0.9439
   57.6120    0.9439
   58.1200    0.9439
   55.0680    0.9439
   48.0484    0.9439


h =

     []


nms =

1
2
3
4
5
```

CURRICULUM VITAE

Name Surname: Ercan Erol

Place and Date of Birth: Çanakkale,Turkey and 29.04.1979

Adress: Mecidiyeköy Mh. Darcan Sk. Ünal Apt. No:6/7 34387 Şişli-İstanbul, Turkey

E-Mail: ercanerol@gmail.com

B.Sc.: Department of Computer Engineering, Middle East Technical University (2001)

M.Sc.: -

Professional Experience and Rewards: He received B.Sc. degree from Department of Computer Engineering at Middle East Technical University in 2001. He has more than 10 years of experience on VMware, RHEL and Windows systems management besides storage and SAN management including zero data loss designs, high available infrastructures with Disaster Recovery Plans and Strategies. Currently he holds VMware Certified Professional on vSphere 5 certificate and works as Infrastructure Supervisor in an IT company.

List of Publications and Patents: -

PUBLICATIONS/PRESENTATIONS ON THE THESIS